CARL VON OSSIETZKY universität OLDENBURG

Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik

Stochastic Satisfiability Modulo Theories: A Symbolic Technique for the Analysis of Probabilistic Hybrid Systems

Dissertation zur Erlangung des Grades eines
Doktors der Naturwissenschaften

von

Dipl.-Inf. Tino Teige

Gutachter:

Prof. Dr. Martin Fränzle
Prof. Dr.-Ing. Holger Hermanns

Tag der Disputation: 29. August 2012

Niemals ohne Dich!

Acknowledgments

First of all, I would like to express my deepest gratitude to my supervisor Prof. Dr. Martin Fränzle for his excellent words of advice and his extraordinary support during all the steps finally leading to this thesis. He introduced me to the research area of formal verification of complex systems and sparked my interest in symbolic techniques for the analysis of probabilistic hybrid systems, the latter becoming the core topic of this thesis. I particularly appreciate the freedom Martin gave me in organizing my work, and I want to emphasize all the numerous and oftentimes spontaneous discussions with him about various questions not only related to science but also to private matters.

I am furthermore very grateful to Prof. Dr.-Ing. Holger Hermanns for the valuable collaboration on the fundamentals of the approach of this thesis as well as for his willingness of being my co-examiner, to Prof. Dr. Ernst-Rüdiger Olderog and Dr. Sibylle Fröschle for attending my thesis defense as members of the committee, and to my colleagues from the University of Oldenburg, from the OFFIS Institute for Information Technology, from the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (AVACS), and from the European Collaborative Project "Modeling, Verification and Control of Complex Systems: From Foundations to Power Network Applications" (MoVeS) for many fruitful discussions and for the great atmosphere during everyday work as well as during project meetings. Especially, I would like to thank Andreas Eggers, Dr. Christian Herde, Natalia Kalinnik, Stefan Kupferschmid, Karsten Scheibler, and Dr. Tobias Schubert for jointly developing the iSAT tool which establishes one central building block of the SiSAT tool presented in this thesis.

My cordial thanks go to my dear colleagues Dr. Christian Herde and Andreas Eggers for the close and outstanding cooperation during the last years and for the pleasant activities outside job. I gratefully acknowledge Christian's great help to me when I came to Oldenburg and started my job at university. From him, I learned a lot about satisfiability and constraint solving as well as about predicative encodings of hybrid systems. I regard highly Andreas' invaluable technical assistance in performing the experiments of this thesis, which has saved me several hours of manual effort, namely by mechanizing the execution of the experiments on the servers as well as the extraction of relevant data obtained from these experiments.

Last but not least, I am indebted to my family and to my friends for all the good times we spent together and for the support in bad moments.

Tino Teige
Oldenburg, September 2012

Abstract

In today's high-tech world, embedded computer systems interacting with technical, physical, or even biological environments are our permanent companions. While several of these applications are almost free of risk and just contribute to a better quality of life such as the use of cellular phones, washing machines, and refrigerators, other embedded systems operate in a safety-critical context where the health of people might be jeopardized like, for instance, in airplanes, automobiles, and medical devices. It is thus of utmost importance that embedded and, in particular, safety-critical systems never run into unsafe situations causing disastrous consequences.

Such real-world safety-critical applications exhibit an intricate system behavior comprising discrete computations like of embedded digital controllers, continuous dynamics of technical environments like the continuous evolution of the temperature, as well as their interaction by means of sensors and actuators. These systems are commonly known under the term of hybrid discrete-continuous systems.

When permitting a more realistic view on real-world embedded systems, it becomes apparent that their dynamics are frequently influenced by randomness. For instance, the landing maneuver of an airplane is, among others, subject to the wind speed and wind direction. These physical entities however are controlled by nature and can be forecast only with uncertainty. That is to say, statements about safe landing maneuvers should also incorporate stochastic predictions of the evolution of wind. Other sources of randomness can be found in the hardware itself like unpredictable failures of circuits, noise in measurements affecting sensors, and actuators missing their setpoints. Such more realistic systems are referred to as stochastic or probabilistic hybrid systems.

When safety-critical applications are subject to stochastic dynamics, preventing unsafe situations by design usually is impossible or economically infeasible, such that a residual risk has to be accepted. The notion of safety in this context is slightly relaxed, requiring that the probability of reaching the unsafe system states always is below an acceptable threshold like, for instance, 1‰. In order to certify safety of real-world applications, manual inspection becomes more and more impracticable due to their rapid-growing complexity. This motivates the development of computer-aided certification methods. The research area dealing with automatic analysis procedures for stochastic and probabilistic hybrid systems has attracted wide interest in recent years and has yielded diverse analysis approaches which are chiefly based on simulation or finite-state abstractions.

In this thesis, we pioneer a completely different approach, namely a symbolic technique for the safety analysis of probabilistic hybrid automata (PHAs) involving a simple model of randomness, that is, probabilistic events from a finite sample space as is the case with throwing dice. Our approach builds on bounded model checking (BMC), where the step-bounded state reachability problem of non-probabilistic hybrid systems is reduced to the satisfiability problem of logical formulae involving arithmetic constraints. For non-probabilistic hybrid systems, the latter is a quantifier-free satisfiability modulo theories

(SMT) problem. To reflect the semantics of PHAs, we extend SMT by existential as well as randomized quantifiers, the latter known from stochastic propositional satisfiability (SSAT). This extension of SMT to stochastic satisfiability modulo theories (SSMT) facilitates a reduction of probabilistic bounded state reachability of PHAs to the problem of solving SSMT formulae, being referred to as probabilistic bounded model checking (PBMC). Completing the symbolic analysis procedure, algorithms for solving SSMT formulae and, moreover, a number of algorithmic enhancements to improve performance in practice are investigated. This symbolic approach establishes an automatic falsification procedure for probabilistic safety properties of the shape "the probability of reaching the unsafe states always is below 1‰".

Motivated by the fact that industrial applications often call for quantitative measures distinct from classical state reachability probabilities, we further propose a symbolic method for computing expected values of PHAs, being able to verify probabilistic safety requirements like "the mean time to failure always is at least 20 minutes".

We finally suggest approaches that go beyond probabilistic bounded state reachability but are yet restricted to probabilistic finite-state models. Being based on a generalization of the logical concept of Craig interpolation, these symbolic procedures aim at the verification of probabilistic safety properties like "the probability of reaching the unsafe states always is below 1‰", on the one hand, as well as of probabilistic stability properties like "the probability that the system stabilizes within some region always is at least 99.9%", on the other hand.

A significant characteristic of the above symbolic approaches is the direct treatment of concurrency, as well-known from the non-probabilistic case. That is to say, the state explosion problem, arising from an explicit construction of the product automaton (with respect to the discrete state space) as in several other analysis approaches, is alleviated, thus contributing to a better scalability.

Zusammenfassung

In der heutigen hochtechnologisierten Welt sind eingebettete Computersysteme, welche mit technischen, physikalischen oder sogar biologischen Umgebungen interagieren, unsere ständigen Wegbegleiter. Während viele dieser Anwendungen nahezu risikofrei sind und lediglich zu einer besseren Lebensqualität beitragen wie der Einsatz von Mobilfunktelefonen, Waschmaschinen und Kühlschränken, operieren andere eingebettete Systeme in einem sicherheitskritischen Kontext, in welchem die Gesundheit von Menschen gefährdet sein könnte, zum Beispiel in Flugzeugen, Automobilen und medizinischen Geräten. Es ist daher von äußerster Wichtigkeit, dass eingebettete und insbesondere sicherheitskritische Systeme niemals zu unsicheren Situationen mit desaströsen Konsequenzen führen.

Sicherheitskritische Anwendungen in der realen Welt weisen ein kompliziertes Systemverhalten auf, welches diskrete Berechnungen wie in digitalen eingebetteten Steuereinheiten, kontinuierliche Dynamiken der technischen Umgebungen wie die kontinuierliche Entwicklung der Temperatur sowie deren Zusammenspiel mit Hilfe von Sensoren und Aktuatoren beinhaltet. Diese Systeme werden üblicherweise unter dem Begriff der hybrid diskret-kontinuierlichen Systeme zusammengefasst.

Erlaubt man einen realistischeren Blick auf eingebettete Systeme der realen Welt, so wird es augenscheinlich, dass ihre Dynamik häufig dem Zufall unterliegt. Beispielsweise hängt das Landemanöver eines Flugzeugs unter anderem von der Windgeschwindigkeit und Windrichting ab. Diese physikalischen Entitäten werden jedoch von der Natur kontrolliert und können nur mit Ungewissheit vorhergesagt werden. Aussagen über sichere Landemanöver sollten daher auch stochastische Vorhersagen über die Entwicklung des Windes mitberücksichtigen. Andere Quellen von Zufall können in der Hardware selbst gefunden werden wie unvorhersehbare Ausfälle von Schaltkreisen, Störungen in den Messungen von Sensoren und Aktuatoren, die ihre Einstellwerte verfehlen. Solche realistischeren Systeme werden als stochastische oder probabilistische hybride Systeme bezeichnet.

Wenn sicherheitskritische Anwendungen stochastisches Verhalten einschließen, ist das Vermeiden unsicherer Situationen durch konstruktive Maßnahmen in der Regel unmöglich oder ökonomisch nicht realisierbar, so dass ein Restrisiko in Kauf genommen werden muss. Der Sicherheitsbegriff ist demnach in diesem Kontext leicht abgeschwächt, und zwar erfordert dieser, dass die Wahrscheinlichkeit des Erreichens unsicherer Zustände immer unterhalb eines akzeptierbaren Schwellwertes liegt, zum Beispiel 1‰. Um die Sicherheit solcher Anwendungen zu zertifizieren, wird eine manuelle Inspektion aufgrund der schnell wachsenden Systemkomplexität immer unpraktikabler. Durch diesen Umstand ist die Entwicklung computergestützter Zertifizierungsmethoden motiviert. Das Forschungsgebiet, das sich mit der automatischen Analyse stochastischer und probabilistischer hybrider Systeme beschäftigt, ist in den letzten Jahren auf reges Interesse gestoßen und hat diverse Analyseverfahren hervorgebracht, die vor allem auf Simulation oder Abstraktion basieren.

In der vorliegenden Arbeit stellen wir einen andersartigen Ansatz vor, nämlich eine symbolische Technik zur Sicherheitsanalyse probabilistischer hybrider Automaten (PHAs), die ein einfaches Modell von Zufall unterstützen, und zwar Zufallsexperimente mit endlich vielen Ausgängen, wie es der Fall beim Würfeln ist. Dieser Ansatz knüpft an eine Technik an, die sich bounded model checking (BMC), also beschränkte Modellprüfung, nennt. BMC reduziert das schrittbeschränkte Zustandserreichbarkeitsproblem für nichtprobabilistische hybride Systeme auf das Erfüllbarkeitsproblem quantorenfreier logischer Formeln, die arithmetische Ausdrücke umfassen. Letztere Probleme bezeichnet man im Englischen als satisfiability modulo theories (SMT). Um die Semantik von PHAs widerspiegeln zu können, erweitert diese Arbeit den Begriff von SMT um existentielle sowie randomisierte Quantoren, wobei die letztgenannten dem stochastischen propositionalen Erfüllbarkeitsproblem (SSAT) entstammen. Diese Erweiterung von SMT mündet in den neuartigen Begriff von stochastic satisfiability modulo theories (SSMT) und ermöglicht die Reduktion des probabilistischen beschränkten Zustandserreichbarkeitsproblems für PHAs auf das Problem des Lösens von SSMT-Formeln, was als probabilistic bounded model checking (PBMC) bezeichnet wird. Um das symbolische Analyseverfahren zu vervollständigen, werden Lösungsalgorithmen für SSMT-Formeln entwickelt und darüber hinaus eine Vielzahl algorithmischer Optimierungen untersucht, um die Performanz in der Praxis zu verbessern. Dieser symbolische Ansatz begründet ein automatisches Falsifizierungsverfahren für probabilistische Sicherheitseigenschaften der Art „die Wahrscheinlichkeit des Erreichens unsicherer Zustände liegt immer unterhalb der 1‰-Grenze“.

Motiviert durch die Tatsache, dass industrielle Anwendungen häufig nach quantitativen Maßen verlangen, die von klassischen Erreichbarkeitswahrscheinlichkeiten abweichen, stellt die vorliegende Arbeit des Weiteren eine symbolische Methode zur Berechnung von Erwartungswerten probabilistischer hybrider Automaten vor, die in der Lage ist, probabilistische Sicherheitsanforderungen der Gestalt „die mittlere Dauer bis zum Systemausfall beträgt mindestens 20 Minuten“ zu verifizieren.

Der Schlussteil der Arbeit widmet sich dann einer Methodik, die über probabilistische beschränkte Zustandserreichbarkeit hinausragt, bislang jedoch auf probabilistische zustandsendliche Systeme limitiert ist. Basierend auf einer Verallgemeinerung des logischen Konzepts der Craigschen Interpolation zielen diese symbolischen Verfahren auf die Verifikation von probabilistischen Sicherheitseigenschaften wie „die Wahrscheinlichkeit des Erreichens unsicherer Zustände liegt immer unterhalb der 1‰-Grenze“ zum einen sowie von probabilistischen Stabilitätsanforderungen wie „die Wahrscheinlichkeit, dass das System innerhalb einer gewissen Region stabilisiert, beträgt mindestens 99,9%“ zum anderen.

Eine signifikante Charakteristik der obigen Ansätze ist die direkte Handhabung von Parallelität ähnlich dem nichtprobabilitischen Fall. Das Problem der Zustandsexplosion, welches durch die explizite Konstruktion des Produktautomaten in vielen anderen Analyseverfahren entsteht, kann dadurch gelindert werden, was zu einer besseren Skalierbarkeit beiträgt.

Contents

List of Figures

1 Introduction

"It is remarkable that this science, which originated in the consideration of games of chance, should become the most important object of human knowledge. ... The most important questions in life are, for the most part, really only problems of probability."

Pierre-Simon, marquis de Laplace, 1812[1]

1.1 Motivation

200 years ago, Laplace already recognized the omnipresence and the important role of *probability* in our life. The term of probability commonly denotes the degree of certainty that some event will happen, where this degree is given by a numerical measure ranging in between 0 to 1. For instance, if the probability is close to 1 or to 0 then we are almost sure that the event will or will not occur, respectively. A simple example of a probabilistic phenomenon is *throwing a die*: the unknown outcome of one throw is a natural number between 1 and 6 where all results are equiprobable, i.e. each number occurs with probability $\frac{1}{6}$. Throwing dice or other random experiments are an essential ingredient of games of chance which can be traced back some thousands years ago.

Nowadays, randomness is observed and investigated in various scientific disciplines. For instance, the stochastic process of radioactive decay is studied in physics, random genetic mutations are considered in biology, and theories like the random walk hypothesis, assuming a random evolution of stock market prices, are suggested in finance. Another steadily growing area in which random phenomena have to be taken into account is the development of embedded computer systems interacting with technical, physical, or even biological environments. While several of these applications contribute to the quality of life such as the use of cellular phones, washing machines, and refrigerators, other embedded systems operate in a *safety-critical* context where the health of people might be jeopardized. Safety-critical applications are developed, for instance, in the aviation, automotive, and railroad industry but also in medical engineering. When considering assistance systems in an automobile, malfunction of the navigation system will hardly ever cause any disastrous consequences, while a failure of the precrash system, which automatically applies partial or full braking among others, will most likely increase the severity of an accident. It is thus of utmost importance that embedded and, in particular, safety-critical systems work irreproachably.

Due to the growing complexity of embedded systems employed around the world, the development of computer-aided approaches to the validation of their safety has evolved to an active and significant research area. In order to describe the intricate behavior of real-world safety-critical applications in a very precise manner, the expressive model

[1]The original quotation in French has appeared in the book *Théorie Analytique des Probabilités* by Laplace in 1812. The English translation above was published in [Pic09].

of hybrid discrete-continuous systems is frequently utilized. A hybrid system comprises discrete as well as continuous behavior, thus being able to cope with the computations of embedded digital controllers, with the continuous dynamics of technical environments like the continuous evolution of the temperature, as well as with their interaction by means of analog-to-digital and digital-to-analog converters. As mentioned above, randomness is omnipresent in physical, biological, and chemical processes and thus should be covered by a realistic system model. To this end, a wealth of ideas of augmenting hybrid systems with probabilities has been suggested. The resulting models are known under the general term stochastic hybrid systems and particularly vary in the degree to which they support random phenomena.

With regard to automatic analysis procedures for stochastic hybrid systems, several approaches rely on Monte Carlo simulation which is an inherently incomplete technique and allows for approximate results only. Other approaches are based on the application of established methods for probabilistic finite-states systems on finite-state abstractions of the original system. These procedures also yield approximate results in general, while most of them in fact compute upper estimates of the actual results. As a consequence, the latter approaches establish verification procedures for probabilistic safety properties of the shape "the worst-case probability of reaching an unsafe state is at most 0.9‰". That is to say, if an upper estimate of at most 0.9‰ is computed then the latter property is verified.

In this thesis, we propose a completely different approach to the (unsafe) state reachability analysis of stochastic hybrid systems. Motivated by the success of symbolic model checking techniques for non-probabilistic hybrid systems, we aspire to a similar concept for probabilistic hybrid systems. To be more exact, instead of abstracting the original system, we aim at a precise description of the next-state relation of the stochastic system by means of a stochastic logic. Though the symbolic encoding is precise, the resulting stochastic constraint formulae just cover bounded system behavior akin to the bounded model checking, or BMC for short, approach for the non-probabilistic case. With the aid of an algorithm for solving such stochastic constraint formulae, lower estimates of the actual reachability probability are determined. Due to the latter fact, we achieve an automatic and fully symbolic falsification procedure being able to refute above probabilistic safety properties once a lower estimate exceeding 0.9‰ is computed. This approach thus complements the verification procedures mentioned earlier.

In contrast to stochastic hybrid systems in full generality, the model investigated in the thesis is very confined in its stochastic behavior as it only admits probabilistic events from a finite sample space as is the case with throwing dice. Albeit being simple, interesting random phenomena like component failures or message losses are characterizable.

Concerning the issue of concurrent systems, it is important to remark that above symbolic approach does not flatten concurrent systems before model checking which has become known as state explosion in the finite-state case. In fact, the symbolic encoding of concurrent probabilistic hybrid systems accommodates concurrency directly such that the size of the encoding is linear in the number of parallel components, as well-known from the non-probabilistic case. That is to say, this treatment alleviates the state explosion, arising from an explicit construction of the product automaton (with respect to the discrete state space) as in several other analysis approaches, and thus contributes to a better scalability.

Motivated by the fact that industrial applications often call for quantitative measures distinct from classical (unsafe) state reachability probabilities, we further propose a symbolic method for computing expected values of probabilistic hybrid systems, being able to verify probabilistic safety properties of the shape “the mean time to failure is always at least 20 minutes”.

In order to go beyond probabilistic bounded state reachability, we moreover suggest a symbolic technique that is able to compute upper estimates of the reachability probability but is yet restricted to probabilistic finite-state models. This symbolic verification procedure can then be used to validate safety properties of the shape “the worst-case probability of reaching an unsafe state is at most 0.9‰”, namely if an upper estimate of at most 0.9‰ is calculated. In addition to the latter, we finally investigate an approach to probabilistic region stability of probabilistic finite-state systems.

1.2 Contributions and structure of the thesis

In this thesis, we make three contributions to symbolic model checking of probabilistic hybrid and finite-state systems. These contributions are elaborated on in Chapters 3 to 9. An overview of some relevant foundations and notations being used throughout this thesis is given in Chapter 2. In Chapter 10, we conclude with a summary of the achievements and discuss promising directions for future research.

We remark that essential parts of this thesis were already published in proceedings of scientific conferences as well as in academic journals by the author of this thesis together with his co-authors. The relevant publications are cited before the corresponding passages in the thesis.

In the remainder of this section, we outline the three major contributions.

Symbolic falsification procedure for probabilistic safety properties of probabilistic hybrid automata based on SSMT solving. The main analysis approach of this thesis is illustrated in Figure 1.1. The starting point is a given problem from the real world, i.e. a real-world system comprising random phenomena and a probabilistic safety property of the shape “the probability of a fatal system error is at most 1‰ in worst case”. In Chapter 3, we elaborate on such a real-world scenario from the networked automation systems domain. In order to address above problems by means of mathematical methods, the frequently informal descriptions of real-world problems must be phrased in a mathematically exact way. To this end, we introduce the formal model of concurrent discrete-time probabilistic hybrid automata in Chapter 3 as well as the notion of probabilistic bounded state reachability in Chapter 5. As an example, we show how the case study from the networked automation systems domain can be modeled as a system of probabilistic hybrid automata in Chapter 8. In a next step, the formal probabilistic system model and the unsafe states are encoded symbolically, which is described in Chapter 5. This encoding scheme is similar to predicative descriptions of non-probabilistic hybrid systems where the system behavior is mapped to a logical formula involving rich arithmetic constraints, the latter being also known as a satisfiability modulo theories

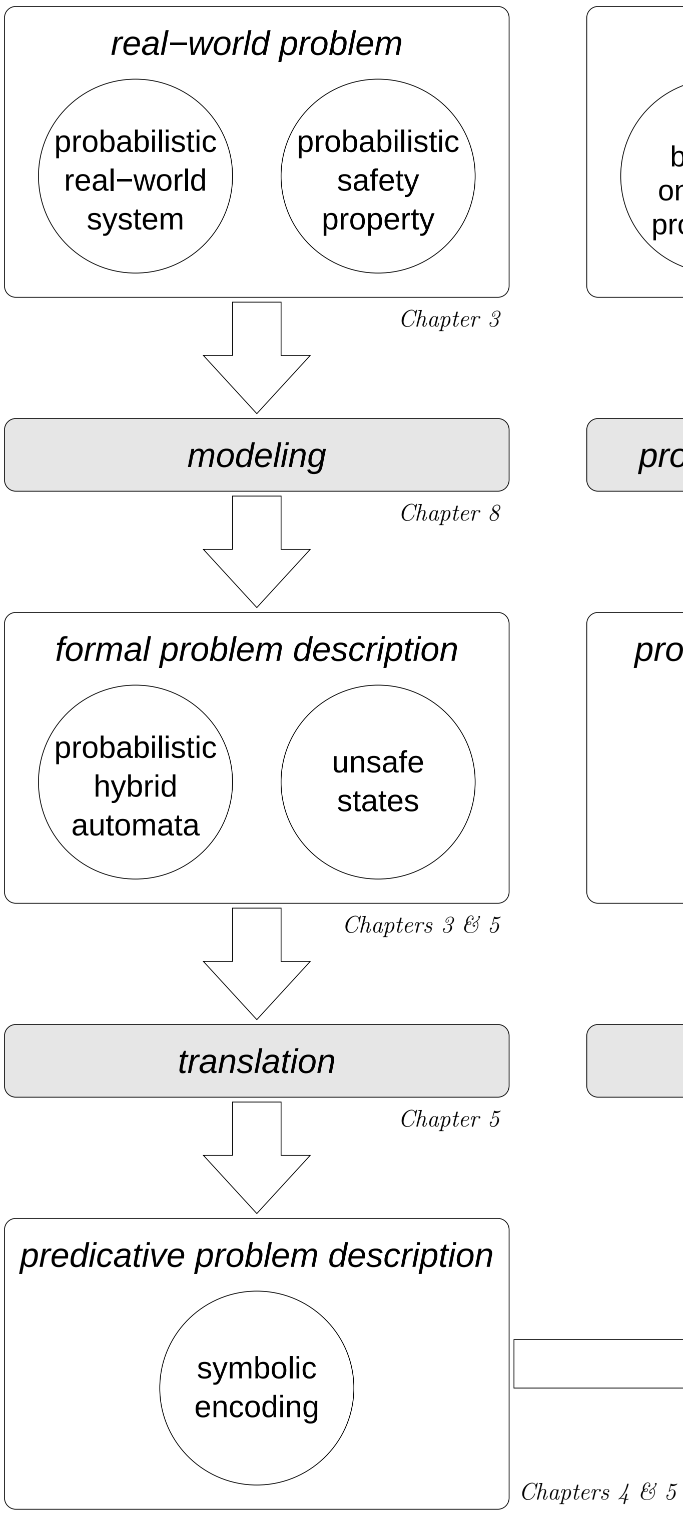

real−world problem
probabilistic real−world system
probabilistic safety property
Chapter 3
modeling
Chapter 8
formal problem description
probabilistic hybrid automata
unsafe states
Chapters 3 & 5
translation
Chapter 5
predicative problem description
symbolic encoding
Chapters 4 & 5
probabilistic BMC
Chapter 5
probabilistic BMC formulae
SSMT formulae
Chapter 4
problem−solving algorithm
Chapter 6
result
lower bounds on failure probability
falsifying prob. safety property
Chapter 8

Figure 1.1: General view of the main symbolic analysis procedure.

(SMT) formula. To further cope with the probabilistic dynamics present in probabilistic hybrid automata, the notion of SMT is enhanced by randomized quantifiers as known from stochastic propositional satisfiability (SSAT). Together with the classical existential quantifiers needed for resolving the non-determinism in the system, this extension of SMT results in the novel concept of stochastic satisfiability modulo theories (SSMT), which is introduced in Chapter 4. From the predicative problem description, concrete SSMT formulae are then achieved using the idea of probabilistic bounded model checking, see Chapter 5. These SSMT formulae reflect the original problem restricted to step-bounded system behavior and, moreover, their quantitative interpretations yield lower bounds on the worst-case probability of reaching the unsafe states. In order to complete the symbolic analysis procedure, Chapter 6 investigates algorithms to solve SSMT formulae as well as a number of algorithmic enhancements to improve performance in practice. As indicated in Figure 1.1, the approach sketched above establishes a falsification procedure for probabilistic safety properties of probabilistic hybrid automata. That is to say, once a lower estimate of the worst-case probability of reaching the unsafe states is computed which exceeds the acceptable threshold value, for instance, 1‰ as above, the probabilistic safety property is falsified. In order to demonstrate practical applicability, the symbolic analysis procedure proposed in this thesis is applied to the concrete case study from the networked automation systems domain in Chapter 8.

Symbolic verification procedure for safety requirements on expected values of probabilistic hybrid automata based on SSMT solving. Motivated by the fact that industrial applications often call for quantitative measures distinct from classical reachability probabilities, namely to gain a more precise insight into the system behavior, Chapter 7 is devoted to a symbolic method for computing expected values of concurrent discrete-time probabilistic hybrid automata like, for instance, mean time to failure (MTTF). This method builds upon SSMT-based probabilistic bounded model checking, and its schematic view is roughly the same as depicted in Figure 1.1. Since the proposed method addresses probabilistic safety properties of the shape “the MTTF is always at least 20 minutes”, it however turns into a verification approach being able to validate that a system of probabilistic hybrid automata meets such above safety requirement once a lower bound on the worst-case MTTF is computed which is at least 20 minutes. This SSMT-based expected-value analysis procedure is also applied to the case study from the networked automation systems domain in Chapter 8.

Symbolic verification procedure for probabilistic safety properties of probabilistic finite-state systems based on generalized Craig interpolation. Both aforementioned analysis procedures are only able to cope with bounded system behavior. In Chapter 9, we pioneer symbolic approaches that go beyond probabilistic bounded state reachability but are yet restricted to probabilistic finite-state models. To this end, we introduce and make use of the novel concept of generalized Craig interpolation for SSAT formulae. Akin to symbolic methods for the non-probabilistic case, generalized Craig interpolation provides an opportunity to compute a symbolic overapproximation of the (backward) reachable state set of probabilistic finite-state systems. This computation relies on a resolution calculus for SSAT formulae which is explained in Chapter 6. Craig interpolation-based model checking for non-probabilistic systems is able to verify safety properties of the shape “the unsafe states are unreachable” whenever the overapproximated set of all reachable states has an empty intersection with the set of unsafe states. As reaching the unsafe states is frequently unavoidable in probabilistic scenarios, a simple check for empty intersection does not suffice in general to verify probabilistic safety properties like “the worst-case probability of reaching the unsafe states is at most 1‰”. To develop such a symbolic verification procedure, we exploit a predicative description of the system as well as a symbolic overapproximation of the backward reachable state set in order to construct SSAT formulae whose quantitative interpretations yield upper bounds on the worst-case probability of reaching the unsafe states. Whenever an upper bound of at most 1‰ is computed then above probabilistic safety property is verified. Chapter 9 furthermore investigates the application of generalized Craig interpolation to probabilistic region stability of probabilistic finite-state systems.

2 Foundations and Notations

This chapter serves as a glossary of general foundations and notations used throughout this thesis.

2.1 General notations

The sets of the *real numbers*, the *integers*, the *natural numbers*, and the *Booleans* are denoted by $\mathbb{R}$, $\mathbb{Z}$, $\mathbb{N}$, and $\mathbb{B}$, respectively. An *interval* $\mathbb{I}$ over $T \in \{\mathbb{R}, \mathbb{Z}, \mathbb{N}\}$ is a subset of T, i.e. $\mathbb{I} \subseteq T$, that satisfies the following property: if $x_1, x_2 \in \mathbb{I}$ with $x_1 < x_2$ then each $x \in T$ with $x_1 < x < x_2$ is contained in $\mathbb{I}$, i.e. $x \in \mathbb{I}$. The *infimum* and *supremum* of an interval $\mathbb{I}$ over T, denoted by $\inf(\mathbb{I})$ and $\sup(\mathbb{I})$, are the greatest number of T that is less than or equal to each number in $\mathbb{I}$ and the smallest number of T that is greater than or equal to each number in $\mathbb{I}$, respectively. Within this thesis, we primarily refer to *bounded* intervals $\mathbb{I}$, i.e. the infimum $\inf(\mathbb{I})$ and the supremum $\sup(\mathbb{I})$ of $\mathbb{I}$ exist within $\mathbb{R}$. We distinguish four cases of bounded intervals. A bounded interval $\mathbb{I}$ over $T \in \{\mathbb{R}, \mathbb{Z}, \mathbb{N}\}$ with $l = \inf(\mathbb{I})$ and $u = \sup(\mathbb{I})$ is called *closed*, denoted by $[l, u]$, *open*, denoted by (l, u), *left-open*, denoted by $(l, u]$, and *right-open*, denoted by $[l, u)$, if $l \in \mathbb{I}$ and $u \in \mathbb{I}$, if $l \notin \mathbb{I}$ and $u \notin \mathbb{I}$, if $l \notin \mathbb{I}$ and $u \in \mathbb{I}$, and if $l \in \mathbb{I}$ and $u \notin \mathbb{I}$, respectively.

Given any formula (or term) φ, we define $\mathit{Var}(\varphi)$ as the set of all variables that occur in φ. The notation $\varphi[v/x]$ denotes usual substitution of v for x in φ. For consecutive substitutions $\varphi[v_1/x_1][v_2/x_2]\ldots[v_k/x_k]$, we occasionally write $\varphi[v_1, v_2, \ldots, v_k/x_1, x_2, \ldots, x_k]$ if the x_i do not mutually occur in the v_i. The *domain* of a variable x, i.e. the set of possible values x can take, is occasionally denoted by $\mathrm{dom}(x)$.

We use the following common abbreviations:

$$\begin{array}{lcl}
\sum_{i=1}^{k} a_i & := & a_1 + a_2 + \ldots + a_k, \\
\prod_{i=1}^{k} a_i & := & a_1 \cdot a_2 \cdot \ldots \cdot a_k, \\
\max_{i=1}^{k} a_i & := & \max(a_1, a_2, \ldots, a_k), \\
\bigtimes_{i=1}^{k} a_i & := & a_1 \times a_2 \times \ldots \times a_k, \\
\bigwedge_{i=1}^{k} a_i & := & a_1 \wedge a_2 \wedge \ldots \wedge a_k, \text{ and} \\
\bigodot_{i=1}^{k} a_i & := & a_1 \odot a_2 \odot \ldots \odot a_k,
\end{array}$$

where $\times$ and $\odot$ denotes Cartesian product and concatenation, respectively. Whenever it is clear from the context, we omit the symbol $\odot$, i.e. for some a and b, we occasionally write ab instead of $a \odot b$. For a set $A = \{a_1, a_2, \ldots, a_k\}$, we further define $\sum_{a \in A} := \sum_{i=1}^{k} a_i$, $\prod_{a \in A} := \prod_{i=1}^{k} a_i$, $\max_{a \in A} := \max_{i=1}^{k} a_i$, $\bigtimes_{a \in A} := \bigtimes_{i=1}^{k} a_i$, $\bigwedge_{a \in A} := \bigwedge_{i=1}^{k} a_i$, and $\bigodot_{a \in A} := \bigodot_{i=1}^{k} a_i$ as usual.

2.2 Propositional logic

Propositional logic is a branch of mathematical logic that investigates propositional formulae and their interpretations. A *propositional formula* consists of *atomic formulae* (*atoms* for short) and *logical connectives* (also known as *logical operators*). Atomic formulae are of no deeper propositional structure and are also called *propositional* or *Boolean variables* or, whenever it is clear from the context, just *variables* as from now. Logical connectives are used to connect propositional formulae. In a formal definition of propositional formulae, these connectives are usually restricted to the unary operator *negation*, denoted as $\neg$, and to the binary operators *conjunction*, denoted as $\wedge$, and *disjunction*, denoted as $\vee$.

Syntactically, propositional formulae are defined inductively: first, each propositional variable is a propositional formula and, second, if φ and ψ are propositional formulae then also $\neg\varphi$, $\varphi \wedge \psi$, and $\varphi \vee \psi$ are propositional formulae. Semantically, such formulae are interpreted by means of truth assignments. A *truth assignment* of a propositional formula φ is a mapping that assigns to each variable that occurs in φ a truth value from the Boolean domain $\mathbb{B}$, i.e. either `true` or `false`. A truth assignment τ then determines the truth value of a propositional formula φ. Formally, we need to extend the concept of a truth assignment to a mapping $\widehat{\tau}$ from propositional formulae to truth values as follows: if φ is a variable then $\widehat{\tau}(\varphi) = \tau(\varphi)$. Otherwise, $\widehat{\tau}(\neg\varphi) = \texttt{true}$ if and only if $\widehat{\tau}(\varphi) = \texttt{false}$, $\widehat{\tau}(\varphi \wedge \psi) = \texttt{true}$ if and only if $\widehat{\tau}(\varphi) = \texttt{true}$ and $\widehat{\tau}(\psi) = \texttt{true}$, and $\widehat{\tau}(\varphi \vee \psi) = \texttt{true}$ if and only if $\widehat{\tau}(\varphi) = \texttt{true}$ or $\widehat{\tau}(\psi) = \texttt{true}$. Slightly abusing notation, for a given formula φ and a truth assignment τ we write $\tau(\varphi)$ to denote $\widehat{\tau}(\varphi)$. We call a truth assignment τ of a formula φ *satisfying assignment*, *solution*, or *model* of φ if and only if $\tau(\varphi) = \texttt{true}$, and occasionally denote it by $\tau \models \varphi$.

If a propositional formula φ has a model, then we call φ *satisfiable* and otherwise *unsatisfiable*. In case that each truth assignment of φ is also a model of φ, the formula φ is called a *tautology* or *tautological* denoted by $\models \varphi$. If φ is not a tautology then we call φ *non-tautological* denoted by $\not\models \varphi$. It clearly holds that φ is tautological if and only if $\neg\varphi$ is unsatisfiable. We sometimes write `true` for a tautological formula and `false` for an unsatisfiable formula. If two formulae φ and ψ have the same models, i.e. $\tau \models \varphi$ if and only if $\tau \models \psi$, then φ and ψ are *semantically equivalent* or just *equivalent* and we write $\varphi \equiv \psi$. In case that φ is satisfiable if and only if ψ is satisfiable, φ and ψ are called *equi-satisfiable*. Please note that based on the logical connectives above, other connectives can be defined. One common logical operation we use in this thesis is *implication*, denoted as $\Rightarrow$, that is defined by $\varphi \Rightarrow \psi \equiv \neg\varphi \vee \psi$.

In addition to the notion of a truth assignment, we further define the concept of a *partial assignment* τ to the variables $\mathit{Var}(\varphi)$ of a formula φ. Such a partial assignment τ is not forced to map each variable in $\mathit{Var}(\varphi)$ to a truth value but just exactly those from some subset $V \subseteq \mathit{Var}(\varphi)$, i.e. $\tau : V \to \mathbb{B}$ is a (total) function. Whenever $x \in V$, we say that $\tau(x)$ is *defined*, and otherwise, i.e. if $x \notin V$, we say that $\tau(x)$ is *not defined*.

A central concept used in this thesis is a normalized syntactical representation of formulae. A *propositional literal* ℓ (or *literal* for short) is a propositional variable x or its negation $\neg x$. In the first case, i.e. $\ell = x$, we call literal ℓ *positive*, and in the second case, i.e. $\ell = \neg x$, *negative*. The *opposite literal* of a literal $\ell \in \{x, \neg x\}$, denoted by $\mathit{neg}(\ell)$, is defined as follows: if $\ell = x$ then $\mathit{neg}(\ell) = \neg x$ and if $\ell = \neg x$ then $\mathit{neg}(\ell) = x$. A *clause* is then a (potentially empty) disjunction of literals. Throughout the thesis and without loss of generality, we assume that a clause does not contain the same literal more than once as $\ell \vee \ell \equiv \ell$. Consequently, we may also identify a clause with its set of literals. We remark that the empty clause, denoted by $\emptyset$, is equivalent to `false`. A propositional formula φ is in *conjunctive normal form* (CNF) if φ is a (potentially empty) conjunction of clauses. As for clauses, we may represent a formula in CNF as a set of clauses. An empty formula in CNF is equivalent to `true`. It is well-known that for each propositional formula φ there is a semantically equivalent propositional formula ψ in CNF. However, the size of ψ may be exponential in the size of φ. In many applications, for instance in symbolic model checking, it is not necessary to obtain an equivalent formula but an equi-satisfiable one. Computing an equi-satisfiable formula in CNF is always possible in linear time by the so-called *Tseitin transformation* [Tse68] that introduces additional variables. We further say that a propositional formula is in kCNF if and only if it is in CNF and each of its clauses contains exactly k literals. For constant $k \in \mathbb{N}$ with $k \geq 3$, there exist linear-time algorithms to transform a propositional formula to an equi-satisfiable formula in kCNF (for $k = 3$ confer [Kar72]), while an efficient procedure to achieve a 2CNF is not known.

For the sake of simplicity, we occasionally use a slightly more general definition of propositional formulae that may additionally contain the constants `true` and `false`. Semantically, we define that the constant `true` is a tautology, i.e. $\models \texttt{true}$, and that the constant `false` is unsatisfiable. We thus have that $\neg\texttt{true} \equiv \texttt{false}$, $\neg\texttt{false} \equiv \texttt{true}$, $\texttt{true} \wedge \varphi \equiv \varphi$, $\texttt{false} \wedge \varphi \equiv \texttt{false}$, $\texttt{true} \vee \varphi \equiv \texttt{true}$, and $\texttt{false} \vee \varphi \equiv \varphi$ hold. By using above equivalence rules, each propositional formula φ that potentially contains the constants `true` and `false` can clearly be rewritten into a semantically equivalent formula φ', i.e. $\varphi' \equiv \varphi$, such that a) φ' does not comprise the constants `true` and `false` or b) φ' coincides with `true`, i.e. $\varphi' = \texttt{true}$, or c) φ' coincides with `false`, i.e. $\varphi' = \texttt{false}$. We call such a formula φ' a *cleaning* of φ. We primarily refer to a cleaning of a propositional formula φ in CNF. Here, a cleaning φ' of φ is simply achieved by removing from all clauses in φ the constant literals `false` and $\neg$`true`, and by excluding all clauses in φ that contain at least one of the constant literals `true` and $\neg$`false`. More formally,

$$\varphi' = \{c \setminus \{\texttt{false}, \neg\texttt{true}\} : c \in \varphi, c \cap \{\texttt{true}, \neg\texttt{false}\} = \emptyset\} \quad .$$

Observe that the cleaning φ' of a propositional formula φ in CNF obtained by above construction does *never* contain the constants `true` or `false`. The rationale is that above case b) $\varphi' = \texttt{true}$ is encoded by the empty formula, i.e. the empty conjunction, that is equivalent to `true`, and above case c) $\varphi' = \texttt{false}$ is represented in such a way that φ' includes the empty clause, i.e. the empty disjunction, that is equivalent to `false`. For technical reasons, we identify a cleaning of a propositional formula in CNF by a cleaning that is achieved by aforementioned construction.

Propositional logic can be generalized by using the idea of *quantification* where propositional variables are bound by some *quantifiers*. Most common are the *existential* quantifier, denoted as $\exists$, and the *universal* quantifier, denoted as $\forall$. A *quantified Boolean formula* (QBF) $\mathcal{Q} : \varphi$ then consists of a propositional formula φ and of a quantifier prefix $\mathcal{Q}$ binding all variables in φ to quantifiers $\exists$ and $\forall$, i.e. $\mathcal{Q} : \varphi$ has no free variables. The semantics of a quantified Boolean formula $\mathcal{Q} : \varphi$ is as follows. If the leftmost quantifier is existential, i.e. $\mathcal{Q} = \exists x \odot \mathcal{Q}'$, then $\mathcal{Q} : \varphi$ is true if and only if *one* of the subformulae

$\mathcal{Q}' : \varphi[\texttt{true}/x]$ or $\mathcal{Q}' : \varphi[\texttt{false}/x]$ is true. In case the leftmost quantifier is universal, i.e. $\mathcal{Q} = \forall x \odot \mathcal{Q}'$, then $\mathcal{Q} : \varphi$ is true if and only if *both* substitutions are true. Since all variables in φ are quantified by $\mathcal{Q}$, the quantifier-free base cases, i.e. if $\mathcal{Q}$ is empty, yield formulae which are equivalent either to `true` or to `false`. For more details on QBF, we refer the reader to [BB09].

2.3 Computational complexity theory

This section recalls the basic concepts of computational complexity which are important for this thesis. For a more detailed overview, we refer the reader to the foundational textbooks [Pap94, GJ90]. Computational complexity investigates the difficulty to solve computational problems on a general computing machine. Usually, *Turing machines* are considered as a mathematical model of such machines. Though Turing machines are simple to some extent and easy to analyze formally, it is believed that these machines are as powerful as any other machine model. The latter conjecture is also known as the *Church-Turing thesis*. In brief, a Turing machine can read and write symbols on a memory with infinite capacity (encoded by so-called infinite tapes) by performing actions from a pre-defined (finite) set. Turing machines are classified by *how* the next action a is chosen. Among others, these are *deterministic* Turing machines, where action a is unambiguously specified by the current state of the machine, *non-deterministic* machines, where a can be freely chosen among some available actions, and *probabilistic* machines, where a is selected randomly according to some probability distribution. When restricting the amount of memory or running time of a Turing machine, this enables the definition of *complexity classes*. We just briefly recall these classes of decision problems that are important for this thesis.

A *decision problem* is a computational problem for which the answer is "yes" or "no". An instance of a decision problem with answer "yes" is called a *true* instance and an instance with answer "no" is said to be a *false* instance. The complexity class P consists of all decision problems that can be solved by a deterministic Turing machine in polynomial time. NP contains all decision problems that are solvable by non-deterministic Turing machines in polynomial time, where solvable means here that all true instances can be decided. The class co-NP is defined by all problems which complements are members of NP, where the complement of a decision problem results from reversing the "yes" and "no" answers. All decision problems that are solved by probabilistic Turing machines in polynomial time with an error probability less than a half establish the class PP. More precisely, for true instances the machine outputs "yes" with probability strictly greater than a half and for false instances the output is "yes" with probability at most a half. More intuitively, PP is the class of all problems whose true instances have the property that the majority, i.e. more than a half, of the computation paths are accepting on non-deterministic Turing machines. Finally, PSPACE comprises all decision problems solvable by deterministic Turing machines using only a polynomial amount of memory. While the inclusion chain

$$\mathrm{P} \subseteq \begin{matrix}\text{co-NP}\\ \text{NP}\end{matrix} \subseteq \mathrm{PP} \subseteq \mathrm{PSPACE}$$

holds, it is still open whether these inclusions are strict or not. It is widely suspected that all inclusions are strict.

All of the above complexity classes have so-called complete problems, i.e. problems that are the “hardest” in the corresponding classes. A problem is *complete* for a complexity class C if it is a member of C and it is hard for C. A problem is *hard* for C if every problem in C can be *reduced* to that problem. The actual properties of such a reduction depends on the particular complexity class. In Subsection 4.2.1, we prove PSPACE-hardness of a decision problem. To show that a problem P is PSPACE-hard, *polynomial-time many-one reductions* from a suitable PSPACE-complete problem P' to P are usually considered. The latter reduction converts each instance I' of P' into an instance I of P in polynomial time such that I' is true if and only if I is true.

In addition to examine the difficulty of solving computational problems, it is of essential interest whether a given problem is solvable at all. A computational problem P is *decidable* if and only if there exists a Turing machine (or simply an algorithm) that solves each instance of P in finite time. Otherwise, i.e. there does not exist any algorithm to solve all instances of P in finite time, we call P *undecidable.* Such undecidable problems actually exist. One famous example is the *halting problem*, i.e. to decide whether a given program eventually halts on a given input, which was proven to be undecidable by Turing [Tur37]. To show that a problem P is undecidable, we can again employ the idea of reduction, namely from some undecidable problem to P.

2.4 Probability theory

The mathematical research field of probability theory investigates the analysis of random phenomena, where first activities can be traced back some hundred years ago. While a plethora of significant notions and results were achieved, we turn our attention to one basic concept that is essential for this thesis, namely the idea of *probability distributions.* Such distributions are commonly used to probabilistically characterize the unknown outcomes of uncertain experiments. One of the simplest such experiments is *coin tossing* where the outcome of one throw is either *heads* or *tails.* Though the resulting side of the coin cannot be predicted with absolute certainty, it is however possible to specify the chance of yielding heads or tails as an outcome. When assuming a “fair” coin then both potential results are equiprobable, i.e. heads occurs with probability 0.5 and with the same probability the outcome is tails. The set of all possible outcomes of an experiment is called the *sample space*. Observe that the sample space for coin tossing consists of the two elements heads and tails. A function $X : \mathcal{S} \to \mathbb{R}$, mapping each element of the sample space $\mathcal{S}$ to a real number, is typically introduced that is called *random variable.* If the image (or range) of X, denoted by $X[\mathcal{S}]$, is countable then X is called *discrete* random variable. In the following, we restrict ourselves to discrete random variables with *finite* images. For the coin tossing experiment, we may define the discrete random variable as follows:

$$X(s) = \begin{cases} 1 & \text{if } s = \text{heads}, \\ 0 & \text{if } s = \text{tails}. \end{cases}$$

Given some discrete random variable X, a *discrete probability distribution* specifies the probabilities of the values of X by means of a *probability mass function* $p_X : X[\mathcal{S}] \to [0, 1]$

such that all these probabilities add up to one, i.e. $\sum_{x \in X[\mathcal{S}]} p_X(x) = 1$. Throughout this thesis, we identify a discrete probability distribution with its probability mass function. Recalling that both outcomes of coin tossing are equiprobable, we are able to state the corresponding discrete probability distribution:

$$p_X(x) = \begin{cases} 0.5 & \text{if } x = 1, \\ 0.5 & \text{if } x = 0. \end{cases}$$

While the formalisms and methods presented in this thesis mainly deal with discrete probability distributions, we occasionally touch upon the more general concept of *continuous probability distributions*. As opposed to the consideration above, the image of a random variable X may also be uncountable. In such cases, X is called *continuous* random variable. While each single outcome of a discrete random variable can be associated with a probability, like for coin tossing, this is not possible for continuous random variables. This convention also matches our intuition: assume that a continuous random variable X talks about the *length of a captured pike*. Then, a length of exactly 100 cm is possible but the probability is equivalent to zero. When however asking for the probability p that the length of the captured pike lies in between 95 cm and 105 cm, then p is potentially greater than zero. Formally, the probability $Pr[x_1 \leq X \leq x_2]$ that a continuous random variable X takes a value within the interval $[x_1, x_2]$ is computed by means of a *probability density function* g_X. More precisely, the above probability is defined by the integral $\int_{x_1}^{x_2} g_X(x)\mathrm{d}x$.

3 Probabilistic Hybrid Systems

This chapter elaborates on the concept of probabilistic hybrid systems. We first present a motivating and illustrative example from the networked automation systems domain in Section 3.1. Thereafter, we give an overview on existing formal models of probabilistic hybrid systems as well as on related approaches to probabilistic model checking in Section 3.2. The formal model of probabilistic hybrid systems being investigated in this thesis is finally introduced in Section 3.3.

3.1 Motivation: A networked automation system

In order to motivate our formal model of a probabilistic hybrid system as well as the probabilistic model checking approaches proposed in this thesis, we start by introducing a realistic industrial application, namely a case study of the networked automation system (NAS) studied in [GF06]. We first give a detailed description of the NAS application, as it was published in [TEF11], from which we then derive the motivation for the formal automata model that is introduced in Section 3.3. Finally, we pose questions about the system behavior that are relevant to the formal analysis of the NAS. These questions are then addressable by the model checking procedures presented in Chapters 5 and 7, while a detailed analysis of the NAS case study is given in Chapter 8.

Description of the NAS. A schematic overview of the *networked automation system* (NAS) studied in [GF06] is depicted in Figure 3.1. As a typical NAS, it involves networked control by programmable logic controllers (PLCs) connected to several sensors and actuators via wired and wireless networks. Its objective is to transport a workpiece from its initial position to the drilling position by means of a transportation unit which controls the speed of the conveyor belt on which the object is transported. The PLC can set the deceleration of the belt via network messages to the transportation unit, but cannot determine the position of the object unless it hits two sensors SA and SB close to the drilling position. The sensors are connected to the IO card of the PLC over the network. When the object reaches sensor SA, the PLC reacts with sending a command to the transportation unit that forces it to decelerate to slow speed. Likewise, the transportation unit is asked to decelerate to standstill when the PLC notices that SB has been reached. The goal is that the object halts close to the drilling position despite the uncontrollable latencies in the communication network. The parameters of the system are taken from [GF06] as far as indicated. Thus, one length unit (lu) is 0.01 mm, and one time step (ts) is 1 ms. The positions of SA and SB are 699 lu and 470 lu, respectively, while the desired drilling position is at 0 lu. The initial speed of the object is 24 lu/ts and the slow speed is 4 lu/ts; the decelerations for the two types of speed changes at SA and SB are 2 and 4 $\mathrm{lu/ts}^2$, respectively. The network routing time is determined stochastically, needing

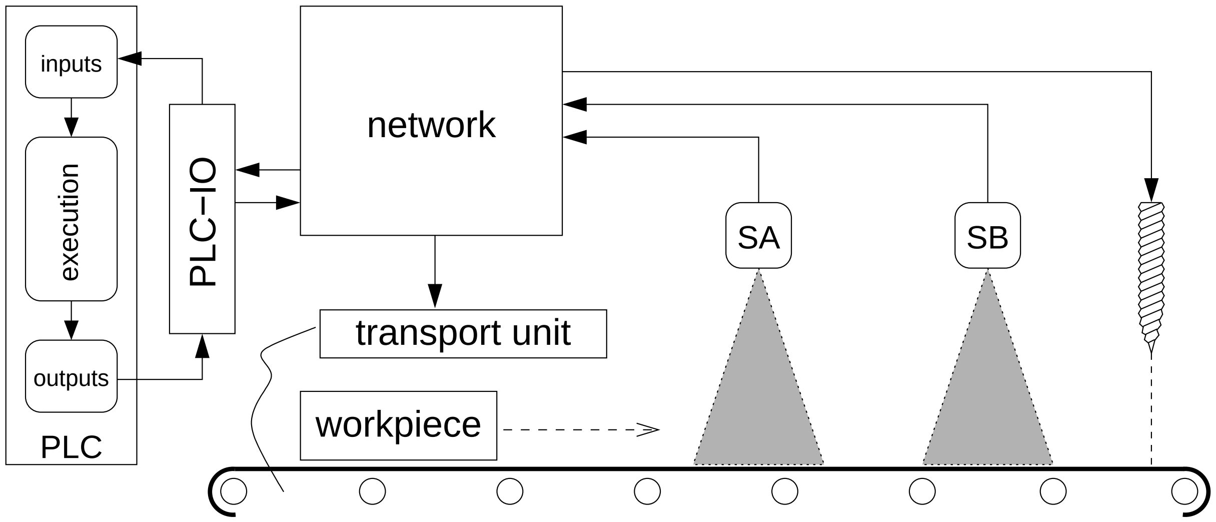


Figure 3.1: A networked automation system from [GF06]. (Source of figure: [TEF11])

1 ts for delivery with probability 0.9 and 2 ts with probability 0.1. The cycle time of the PLC-IO card is 10 ts and of the PLC is 7 ts. The minimum sampling interval is 1 ts. Due to the initial speed of 24 lu/ts, the initial position of the object is thus equally distributed over 24 neighboring values. In our setting, the initial position ranges between 999 and 976 lu.

Motivation of formal model. Real-world dynamical systems, as the NAS above, characteristically evolve over time that is a continuous quantity. Mathematical models of such dynamical systems should therefore take account of an appropriate notion of time. Most common notions are *discrete time*, where the system evolution is represented by means of sampling at discrete points of time, and *continuous time*, where the system behavior is described as a continuous evolution over time. The NAS application above relies on discrete time as it specifies a constant sampling interval. To cope with this NAS and similar systems, we concentrate on a

- discrete-time semantics

for the formal automata model. It is important to remark that discrete time is not necessarily equivalent to *equidistant time*, where system sampling actually happens at equidistant time points. Our model of discrete time permits sampling at arbitrary points of time. Moreover, these sampling points may also be state-dependent. This allows for a *scheduled-event* semantics, akin to [AL94], where the next sampling point is determined by the current system state.

The continuous dynamics of the NAS workpiece underlies the laws of motion with uniform (or constant) acceleration. That is, given the current position s, the current speed v, the constant acceleration a, and a time step of duration Δt, then the position s' and the speed v' after Δt time units are given by the formulae

$$s' = s + v \cdot \Delta t + \frac{1}{2} \cdot a \cdot \Delta t^2 \quad \text{and} \quad v' = v + a \cdot \Delta t \ .$$

In order to describe such dynamics within a formal automaton model, we demand that

- rich arithmetic theories over continuous domains

are supported.

We need to remark that more complex continuous dynamics in fields like nuclear physics or biophysical chemistry are usually modeled by means of ordinary differential equations (ODEs) or even by partial differential equations (PDEs). While some ODEs have so-called closed-form solutions, that are expressible by common arithmetic functions, others have not. It is however always possible to safely approximate ODEs by, for instance, Taylor series. In Chapter 10, we elaborate on a much more expressive automata model that incorporates ODEs and is interpreted over continuous time.

The NAS also includes components like the PLC that execute discrete programs. Moreover, the computed output of the PLC can change the continuous dynamics of the workpiece by setting a new value for the deceleration. Therefore,

- discrete computations and a logic for switching the continuous behavior

should be comprised by the formal automata model.

The behavior of the network depends on random phenomena as its routing time is determined probabilistically. Since all probabilistic events that may occur in the NAS range in a finite sample space, it suffices to require

- discrete probabilistic choices

to be present in the automata model. The latter should furthermore support

- non-deterministic choices

in order to cope with, for instance, input data or open design questions. Though the NAS above does not show any non-determinism, the model could be enhanced, for instance, by a non-deterministic choice between several PLCs with different cycle times in order to find out whether some safety requirements are fulfilled regardless of which PLC is actually employed.

Finally, a formal automaton model should facilitate a convenient way of

- parallel composition

as many real-world systems like the NAS consist of several components. To meet this requirement, we aim at a formalism that allows to model each subsystem itself, and thus renders unnecessary the construction of one automaton for the overall system.

Analysis questions. As mentioned above, the goal of the NAS application is to transport the workpiece close to the drilling position. The main analysis question thus is whether this property actually holds. As usual in scenarios of probabilistic nature, it cannot be assumed that desired properties are definitely satisfied. In such cases, engineers are however interested in whether desired properties hold with very high probability. We therefore investigate questions like

- “*What* is the probability that the workpiece stops close to the drilling position?”

or, phrased as a decision problem,

- “Is the probability that the workpiece stops close to the drilling position *high enough*?”

In Chapter 5, we present a model checking procedure addressing such questions. To be a bit more precise, our model checking approach deals with probabilistic bounded state reachability problems, i.e. it computes the probability of reaching some target states within a bounded number of system steps. In applications where the target states are considered to be bad, i.e. they violate some safety property, this bounded model checking (BMC) procedure is potentially able to falsify safety properties like “the probability of reaching the bad states is at most 1‰”. From the NAS description, it is not hard to see that the workpiece finally stops. Thus, the dynamics of the workpiece is bounded and BMC is able to respond to both questions above.

Industrial applications often call for quantitative measures distinct from classical reachability probabilities to gain a more precise insight into the system behavior. Considering the NAS case study, it would be very beneficial to answer questions like

- “What is the *mean time to stop* of the workpiece?”

and

- “What is the *expected final position* of the workpiece?”

Motivated by these questions, Chapter 7 enhances probabilistic BMC to compute expected values. The enhanced probabilistic BMC procedure is then potentially able to verify safety properties of the shape “the worst-case mean time to failure is at least 20 minutes”. With the same argument as above, i.e. the workpiece finally stops, we are able to answer both questions above.

A detailed analysis of the NAS case study, including the complete formal system model and probabilistic model checking results, is given in Chapter 8.

3.2 Related work: Probabilistic hybrid models and model checking

In this section, we give an overview on existing formal models of probabilistic hybrid systems as well as on related approaches to probabilistic model checking.

3.2.1 Probabilistic hybrid systems

Hybrid discrete-continuous behavior arises when discrete and continuous dynamic processes become connected, as in the case of embedded computers and their physical environment. For an example, recall the NAS application from Section 3.1 where the PLC influences the continuous dynamics of the workpiece. An increasing number of the technical artifacts shaping our ambience are such *hybrid systems* relying on, often invisible, embedded computer systems. Their safety assessment amounts to showing that the joint dynamics of the embedded system and its environment is well-behaved, for instance, that it avoids undesirable states or that it converges to a desirable state, regardless of the actual disturbance. Disturbances may originate from uncontrolled inputs in an open system, like a car driver performing her driving task, as well as from internal sources of the overall technical system, like failing system components, including sensors or even actuators. Gradually advancing the capabilities of addressing such systems, research in hybrid system verification has thus traditionally focused on different classes of system structures and disturbances, ranging from a closed-system view over non-deterministic to probabilistic or stochastic hybrid systems. While the closed-system view necessitates a reasonably exact representation of the rather intricate yet deterministic feedback dynamics of coupled discrete and continuous systems, non-deterministic systems extend this view by unknown inputs of an open system. Probabilistic hybrid systems, finally, allow to capture unpredictable, yet statistically characterizable disturbances.

In the context of hybrid systems augmented with probabilities, a wealth of models has been suggested by several authors. These models particularly vary in the degree to which they support random phenomena. The cornerstones are formed by the following models. In *probabilistic hybrid automata* [Spr01], state changes forced by continuous dynamics may involve discrete random events, akin to Markov decision processes [Bel57], determining both the discrete and the continuous successor state. *Piecewise deterministic Markov processes* [Dav84, Dav93] permit that state changes may happen spontaneously in a manner similar to continuous-time Markov chains, confer, for instance, [Tij03, Chapter 4]. *Stochastic differential equations* [Arn74] are another model where, like in Brownian motion, the random perturbation affects the dynamics continuously. In full generality, *stochastic hybrid system* models can cover all such ingredients [HLS00, PBLB03, BL06, CL07], thereby having a wide range of applications, for instance, air traffic control [PHLS00, GL04], manufacturing systems [CM03], and communication networks [Hes04].

The model investigated in this thesis belongs to the class of probabilistic hybrid automata. In contrast to the other models mentioned above, this system class is very confined in its stochastic behavior as it only admits discrete probabilistic choices within state transitions. Albeit being simple, interesting random phenomena like component failures or message losses are characterizable. Concerning the NAS case study, it is able to express the uncertain latencies of the network as well as the distribution of the initial position of the workpiece.

3.2.2 Probabilistic model checking

The fully automatic verification of whether a given system model meets some specification is referred to as *model checking*. Typically, such specifications describe safety properties like “a fatal system error may never occur” or, in the probabilistic setting, “a fatal system error may only occur with very low probability”. Very expressive specification logics have been developed, among them the very prominent *probabilistic computation tree logic*, or PCTL for short, confer, for instance, [HJ94, BdA95]. While safety properties just talk about (un)reachability of states, logics like PCTL permit the specification of more expressive temporal properties like “each message will be finally delivered with probability 1 and whenever a message is tried to be sent then this message will be delivered in five system steps with very high probability of at least 0.995". Model checking of probabilistic finite-state models like Markov decision processes and continuous-time Markov chains is a well-studied and still very active research area, having originated efficient probabilistic model checking tools like PRISM[1] or MRMC[2]. For a nice survey on probabilistic finite-state model checking we refer the reader to [BK08, Chapter 10].

When turning one's attention to the automatic analysis of richer probabilistic systems with infinite state space like probabilistic hybrid systems, lots of verification approaches "only" aim at safety properties or, dually, unsafe state reachability. On the one hand, this is motivated by the fact that most temporal properties can be reduced to reachability problems due to the very expressive hybrid modeling framework. On the other hand, probabilistic state reachability is a hard and challenging problem. Indeed, this problem is undecidable in general, even for probabilistic hybrid automata. The latter fact immediately follows from general undecidability of the reachability problem for (non-probabilistic) hybrid automata since probabilistic hybrid automata are a superclass of hybrid automata. A very detailed account of the boundary between decidability and undecidability of the reachability problem for hybrid automata is given in [HKPV95]. An important decidable subclass are *timed automata* [AD94] where the continuous behavior is limited in the sense that real-valued variables, so-called clocks, may only progress with rate 1. In [HKPV95], this decidability result was generalized to *initialized rectangular automata* where the continuous evolution of each real-valued variable is governed by arbitrary constant rates from some interval, for instance from $[-3, 5]$, but whenever the continuous activity of a variable changes then the value of that variable must be re-initialized. As observed in [HKPV95], when slightly generalizing the latter automata class, for instance by imposing an order of the derivatives of variables, then the reachability problem becomes undecidable. Another way to obtain decidable subclasses was suggested by Fränzle: in [Frä99], he exploited the notion of *robustness* to show that reachability for robust hybrid automata is decidable.

Though research concerning stochastic hybrid systems is rapidly increasing over the last years, results related to their analysis and verification are still limited to some extent. While for the general class of stochastic hybrid systems such analysis approaches are often based on Monte Carlo simulation [BB04, BKB06], several subclasses of piecewise deterministic Markov processes, of probabilistic hybrid automata, and of stochastic hybrid systems are recently investigated for which reachability probabilities can be approximated [BL03, BC05, AAP$^+$06b, KR06]. With regard to decidable subcases, Sproston could establish a result similar to the non-probabilistic setting. In [Spr00, Spr01], he showed that model checking is decidable for the subclass of *probabilistic initialized rectangular automata* against specifications of a probabilistic temporal logic called *probabilistic branching time logic* (PBTL) that is similar to PCTL. The dynamics of this restricted subclass is confined in the same way as for its non-probabilistic counterpart mentioned above. His model checking procedure relies, first, on a translation to a probabilistic version of timed automata, second, on the construction of a finite state representation of the latter system called the probabilistic region graph and, third, on using established PBTL model checking techniques [BdA95, BK98]. Sproston argues that his approach suffers from two significant drawbacks: 1) the assumption of re-initialization is too restrictive

[1]More information can be found on `http://www.prismmodelchecker.org`.

[2]More information can be found on `http://www.mrmc-tool.org/trac`.

and 2) the translation process introduces too many new variables and thus increases the state space significantly. He therefore proposed a semi-decision procedure for reachability analysis of the more general class of probabilistic rectangular automata, i.e. without the restriction of re-initialization, by means of a forward search through the reachable state space [Spr01].

In the remainder of the section, we first outline the model checking techniques for probabilistic hybrid automata being introduced in this thesis. Thereafter, competitive approaches for the same and closely related system classes are reviewed.

3.2.3 Approach of this thesis

The probabilistic hybrid automata model of this thesis is mainly restricted as follows: first, it is interpreted over discrete time, second, it does not include ordinary differential equations and, third, the non-determinism and stochasticity must be finite. In spite of these limitations, interesting applications can be covered as shown by the NAS case study of Section 3.1. We further remark that these restrictions are not essential for our approach and that Chapter 10 elaborates on a continuous-time model incorporating ordinary differential equations.

The model checking procedures introduced in this thesis belong to the class of depth-bounded state-space exploration methods based on satisfiability solvers, which have originally been suggested for finite-state systems by Groote et al. in [GKvV95] and Biere et al. in [BCCZ99]. Such methods have become popular under the term *bounded model checking* (BMC) now accounting for a major fraction of the industrial applications of formal verification. The idea of BMC is to encode the next-state relation of a system as a propositional formula, to unroll this to some given finite depth k, and to augment it with a corresponding finite unravelling of the tableaux of the negation of a temporal formula, describing some desired system property, in order to obtain a propositional formula which is satisfiable if and only if an error trace of length k exists. Enabled by the impressive gains in performance of Boolean satisfiability (SAT) solvers in recent years, BMC can now be applied to very large finite-state designs. Though originally formulated for discrete transition systems, the concept of BMC also applies to hybrid discrete-continuous systems. The BMC formulae arising from such systems comprise complex Boolean combinations of arithmetic constraints over real-valued variables, thus entailing the need for so-called *satisfiability modulo theories* (SMT) solvers over arithmetic theories to solve them. Such SMT procedures are thus currently in the focus of the SAT-solving community [BBC$^+$05, DdM06, BPT07, FHT$^+$07, EFH08, GGI$^+$10], as is their application to and tailoring for BMC of hybrid systems [ABCS05, ÁBKS05, FH07, HEFT08, ÁSB$^+$11, ERNF11].

In this thesis, we present a technology that saves the virtues of SMT-based BMC, namely the fully symbolic treatment of hybrid state spaces, while advancing the reasoning power to probabilistic models and requirements. In Chapter 4, we therefore extend SMT by alternating quantifiers of the classical *existential* form as well as of *randomized* or, equivalently, *stochastic* type. This leads to the logical framework of *stochastic satisfiability modulo theories* (SSMT) facilitating a symbolic encoding of probabilistic hybrid automata. Chapter 5 gives a detailed account of this symbolic encoding. The idea of the latter is, in a nutshell, to encode the transition effects as an SMT formula, as usual, yet add the branching structure to the encoding by means of quantification, with existential quantification reflecting non-deterministic choices and randomized quantification reflecting probabilistic events. The step-bounded analysis of probabilistic hybrid automata can then be reduced to solving SSMT formulae. Appropriate algorithms to solve SSMT problems as well as several algorithmic optimizations are presented in Chapter 6. This *probabilistic bounded model checking* (PBMC) approach is then able to falsify safety properties of the shape "the worst-case probability of reaching a fatal system error is at most 1‰" whenever a step depth was found for which this property could be refuted. Akin to the non-probabilistic case, SSMT-based PBMC permits an encoding of concurrent probabilistic hybrid automata that is of size linear in the number of parallel components, alleviating the state explosion arising from an explicit construction of the product automaton with respect to the discrete state space, and thus enhancing the scalability of the automated analysis procedures.

Apart from such classical reachability probabilities, several industrial applications frequently call for more expressive quantitative measures like expected values, confer the latter two analysis questions of Section 3.1. Motivated by this fact, Chapter 7 is devoted to a symbolic method for computing *expected values* of discrete-time probabilistic hybrid systems like, for instance, mean time to failure (MTTF). Though the latter method builds upon SSMT-based PBMC, it has fundamentally different properties: instead of targeting at falsification, the resulting procedure turns into a verification approach being able to verify safety requirements of the shape "the MTTF is always at least 20 minutes".

3.2.4 Competitive approaches

Zhang et al. presented an approach to *verification* of safety properties concerning the probability of reaching unsafe states in probabilistic hybrid automata [ZSR$^+$10]. This approach can thus be seen as complementary to PBMC that establishes a falsification procedure for such safety properties. The automata class of [ZSR$^+$10] is however not that restrictive than the one of this thesis as continuous-time semantics and ordinary differential equations are supported. In a bit more detail, the verification procedure works as follows. First, a *non*-probabilistic hybrid automaton is obtained from the given probabilistic hybrid automaton by simply replacing probabilistic choices with non-deterministic ones. The hybrid automaton is then abstracted into a finite-state system using classical methods, for instance [ADI06, RS07]. While the current implementation employs the tool PHAVer [Fre05], any other tool that produces such abstractions is applicable. In the next step, the abstracting finite-state system is decorated with probabilities via techniques known for Markov decision processes [DJJL01, HWZ08], resulting in a probabilistic finite-state automaton. This overall translation ensures the following property: if the abstracting probabilistic finite-state system satisfies some probabilistic safety property then the original probabilistic hybrid automaton does so. To compute the probability p of reaching unsafe states in the probabilistic abstraction, standard methods, here *value iteration* [Bel57], are used. By above property, p is a safe upper bound of the reachability probability for the original infinite-state system. A safety property of the shape "in worst case, unsafe states are reachable with probability at most θ" is then verified whenever $p \leq \theta$ holds. Otherwise, i.e. if $p > \theta$, the abstraction is refined to obtain a potentially more precise upper bound. We remark that how the refinement is realized depends on the abstraction technique. While PHAVer computes polyhedra to cover the continuous state-space per discrete location, the authors currently reduce the maximal widths of these polyhedra to refine the abstraction.

More recently, the expressiveness of the above system model was enhanced considerably in [FHH$^+$11], namely by permitting *continuous* probability distributions in discrete state changes.[3] The resulting system class is called *stochastic hybrid automata.* With regard to the probabilistic reachability analysis of such systems, the verification procedure from [ZSR$^+$10] was adapted to this more general case as follows. In a first step, a given stochastic hybrid automaton is overapproximated by a probabilistic hybrid automaton, as defined in [ZSR$^+$10], by means of abstracting continuous probability distributions by discrete distributions combined with additional uncountable non-determinism. The latter abstraction satisfies the property that if the resulting probabilistic automaton meets some probabilistic safety requirement then the original stochastic automaton does so. In a second step, the overapproximating probabilistic hybrid automaton is model checked using the verification procedure from [ZSR$^+$10].

The recent work described in [HNP$^+$11] targets at the same problem as in [ZSR$^+$10], namely the probabilistic reachability analysis of probabilistic hybrid automata, with the suggested analysis approach also relying on finite-state abstraction of the given infinite-state system. More precisely, two abstraction techniques are elaborated on, with both of them abstracting the given probabilistic hybrid automaton by an n-player stochastic game. Within this stochastic game, the abstraction is represented by an own player. By defining the strategy of the latter player to minimize or maximize the probability of reaching the target states, lower and upper bounds, respectively, on the optimal reachability probability for the original automaton can be obtained from the abstraction. That is to say, the approach of [HNP$^+$11] establishes a verification as well as falsification procedure for probabilistic safety properties. For the sake of completeness, we remark that the authors of [HNP$^+$11] further considered the computation of lower and upper bounds on optimal long-run average rewards for probabilistic hybrid automata as well as the problem of synthesizing an optimal controller for such systems.

In control theory, a model similar to probabilistic hybrid automaton is currently in the focus that is called *discrete-time stochastic hybrid system* (DTSHS) [AAP$^+$06b, AAP$^+$06a, Aba07, APLS08]. Being sampled at discrete time points, this model comprises non-deterministic as well as discrete probabilistic choices of state transitions. The non-determinism is modeled via so-called control inputs. In comparison to probabilistic hybrid automata, the above system class do not exhibit an explicit notion of symbolic transition guards. Transition guards are an essential concept in hybrid system modeling as they offer the possibility to describe computer programs controlling the system. For instance, a heater should be switched off only if the temperature is above 35°. It is however possible to define transition guards implicitly by picking properly the probabilistic transition functions as the latter may also depend on the continuous state, see [APLS08, Section 2]. The

[3]In fact, another contribution of [FHH$^+$11] is the introduction of *uncountable* non-determinism in discrete assignments. In principle, the latter feature however was already present in probabilistic hybrid automata [ZSR$^+$10] and was neglected only for the sake of a simpler presentation, confer footnote 1 on page 199 of [ZSR$^+$10].

model of DTSHS supports a further and more general concept of randomness: at each time step the continuous state may be determined according to a *continuous probability distribution*. DTSHSs are thus able to describe discretized stochastic differential equations incorporating random phenomena like noise in temperature evolution, confer the thermostat example in [AAP$^+$06b] starting on page 54. With regard to system analysis, the above articles concentrate on the control problem of "keeping a system within a safe region for a given time horizon with sufficiently high probability". The notion of safety is thus understood as to find an optimal control policy that maximizes the probability of staying safe or, in other words, that minimizes the probability of reaching unsafe states. This point of view differs from most common model checking approaches, like the one described above or the one of this thesis, where non-determinism typically arises from open design questions or non-expert inputs and is thus considered as uncontrollable. As a consequence, model checking aims at the *worst case* scenario while optimal control approaches address the *best case*. The maximum probability of remaining in the safe region is expressed using optimal cost functions. These functions basically reflect the branching structure of the DTSHS, while branching according to continuous probability distributions is represented by integrals. These cost functions theoretically establish a backward recursive procedure which is also called *dynamic programming* scheme. Intuitively, dynamic programming determines how the optimal probability for some state at time point $k-1$ is computed if the optimal probabilities for all states at time k are known. The reasoning is thus backward in time and stops when time point 0 is reached. As a finite time horizon is considered, i.e. the number n of discrete time steps is finite, the base cases are given at time point n where the probabilities for all states are trivially known, i.e. probability 1 for safe states and probability 0 for unsafe states. Finding analytical solutions to such dynamic programming equations, i.e. closed-form expressions without integral parts, is however hard in general, since such cost functions can be very general and of non-linear shape. In order to tackle this issue, a numerical approximation approach was suggested in [Aba07, Section 2.2.8]. The idea is to construct a finite discretization of the continuous state-space that is called *grid*. Using this finite-state grid, the dynamic programming scheme can be discretized and then solved approximatively. It was shown in [Aba07, Theorem 9] that the approximation error depends on the grid size[4], i.e. the maximal size of a cell of the grid, from which follows that the quality of approximation enhances for smaller grid sizes. It is well-known that finite-state gridding approaches suffer from the so-called "curse of dimensionality", confer, for instance, [Aba07, Section 2.3.4], i.e. the number of cells of the grid is exponential in the number of continuous state components. In order to apply dynamic programming, the grid must be constructed beforehand. This is an inherent difference to SSMT-based PBMC where the state space is encoded fully symbolically as an SSMT formula without an exponential overhead. Complexity issues potentially arise when solving an SSMT problem: in worst case, the full state space must be traversed while the latter fact is tried to mitigate by several algorithmic enhancements, confer Section 6.5. We finally remark that in [Aba07, Section 2.2.4] also the infinite time horizon case was investigated, addressing the question of convergence of the optimal control law to a stationary policy.

Another approach to a very similar problem as above was suggested in [AKLP10,

[4]Note that this term does *not* denote the number of cells of the grid.

AKLP11], where the DTSHS model is autonomous, i.e. without non-deterministic control inputs. Akin to the method above, the probability of remaining in the safe region is also described as a dynamic programming scheme, and the continuous state space is again discretized by a finite-state grid. The difference however lies in the computation of the probability. As distinguished from using the grid to obtain and then numerically solve a discretized dynamic programming scheme, the grid is exploited to derive a discrete-time Markov chain, the latter being model checked using standard techniques. The result is an approximation of the actual probability of staying safe, while the approximated result converges to the exact probability as the maximal size of a cell of the grid tends to zero.

The above approach employs a uniform-partitioning algorithm for the grid construction and thus frequently suffers from scalability issues. In a more recent work, the authors of [SA11] proposed an adaptive procedure for the grid generation that exploits knowledge about the system dynamics. It was shown that this adaptive procedure can lead to grids with a much smaller number of cells, thus mitigating the "curse of dimensionality".

The authors of [AKM11] proposed an approach to model checking autonomous DTSHSs against *linear time* objectives like, for instance, liveness instead of mere safety properties. Such objectives are specified either as a deterministic finite-state automaton (DFA) or as a generalized (non-deterministic) Büchi automaton, the latter covering properties expressible in the linear temporal logic (LTL). The problem of computing the probability that a given autonomous DTSHS satisfies a linear time property specified by a DFA or Büchi automaton is then reduced to the problem of computing reachability probabilities in the product of the DTSHS and of the automaton encoding the property. The latter probabilistic reachability problem is then addressed by a procedure similar to one of [AKLP10, AKLP11].

More recently, Platzer suggested a logic-based approach to safety analysis of stochastic hybrid systems [Pla11]. As a formal model, *stochastic hybrid programs* (SHPs) were introduced. This system class is very expressive on the stochastic side as it comprises stochastic differential equations, discrete probabilistic branching, and random assignments to real-valued variables, while it seems that non-deterministic branching and parallel composition are not expressible. For the specification of system properties, a logic called *stochastic differential dynamic logic* is considered. A proof calculus is then proposed to verify logical properties of SHPs. The latter calculus was presented without an implementation, and the issue of its automatability was not discussed. With regard to the approach of this thesis, the model of probabilistic hybrid automata, on the one hand, is considerably more confined than SHPs, in particular in its stochastic behavior. On the other hand, SSMT-based PBMC establishes a *fully automatic* analysis procedure which is particularly suitable for the analysis of *concurrent* systems, i.e. of systems consisting of several subsystems.

In [WZH07], model checking *probabilistic programs* against specifications of a fragment of PCTL is examined. Probabilistic programs are very similar to discrete-time probabilistic hybrid automata as they support non-deterministic and discrete probabilistic choices as well as arithmetic expressions over unbounded integers and reals to describe the program execution. Due to infinite data domains, the state space of this system class is also infinite. Like the approaches above, the procedure of [WZH07] relies on finite-state abstraction but of a very different nature. Instead of partitioning the infinite state space by geometric objects explicitly, for instance, by polyhedra or grids, the more general concept of *predicates* is used to obtain a symbolic abstraction. A predicate φ over the variables of the given probabilistic program encodes a set of states, namely the states satisfying φ. Such a predicate may describe a set of states that is of much more complex shape than, for instance, a polyhedron as it may encode, for instance, a union of unconnected polyhedra. Given n predicates, the infinite state space can then be discretized into 2^n abstract states by characterizing for each abstract state which of the n predicates are satisfied and which are not. Such predicates are extracted from the probabilistic program as well as from the PCTL property and may also be provided by the user. This *predicate abstraction* resulting in a probabilistic finite-state automaton works as follows. Given predicates $\varphi_1, \ldots, \varphi_n$, an abstract state $s^\sharp$ is represented by a bit vector $(b_1, \ldots, b_n)$, and $s^\sharp$ encodes an original state s whenever for all $1 \leq i \leq n$ it holds that Boolean variable b_i is `true` if and only if s satisfies predicate φ_i. In order to construct the abstracted finite-state model, the authors of [WZH07] proposed an SMT-based approach: to detect all initial abstract states as well as all transitions between abstract states, SMT formulae are generated that link the abstract states and original states to each other. Each solution of such an SMT formula to the Boolean variables then gives an initial state or identifies transitions between abstract states. An appropriate SMT solver is used to enumerate all solutions, i.e. all initial states and all transitions. Observe that in worst case, the number of generated states and transitions is exponential in the number n of predicates. To mitigate this issue, optimization techniques to decrease the number of Boolean variables in these SMT formulae and thus the number of SMT solutions were presented in [WZH07], and it was shown empirically that these enhancements can improve performance. In a final step, the resulting probabilistic finite-state automaton is analyzed using the probabilistic finite-state model checker PRISM. Due to the fact that predicate abstraction preserves safe PCTL properties, i.e. if the abstraction satisfies a PCTL property then the original probabilistic program also does, the original system is verified whenever the abstracted system satisfies the PCTL specification.

Though both, the approach of [WZH07] and SSMT-based PBMC presented in this thesis, target at very similar infinite-state system models and rely on SMT solving, they show however inherent differences. As mentioned above, the technique of [WZH07] constructs a finite-state abstraction of a given infinite-state system and thus aims at verification, while SSMT-based PBMC precisely encodes the step-bounded behavior of the given infinite-state system as an SSMT formula and is designated for falsification. Although both approaches use symbolic encodings of the system behavior, the application of SMT solving is different. In PBMC, the overall probabilistic reachability problem is directly reduced to solving the corresponding SSMT formula, while [WZH07] first extracts the abstracted system from all solutions of the corresponding SMT formulae and then model checks the abstraction by another method. As mentioned earlier, the approach of [WZH07] must enumerate exponentially many SMT solutions in worst case. This issue is clearly also present when solving an SSMT problem. The difference lies in how optimizations apply: in [WZH07], each guarded command, i.e. each probabilistic choice, of the original system is encoded as one SMT formula to detect abstract transitions. This implies that the optimizations mentioned above are only applicable for one probabilistic transition choice. An SSMT formula in PBMC however comprises the whole system behavior (of bounded step-depth), in particular all non-deterministic and probabilistic transition choices, and thus talks about system runs and not only about single system steps. This gives rise to more sophisticated optimizations that potentially exclude several system runs. Algorithmic optimizations of SSMT solving are investigated in Section 6.5. We finally remark two other differences. The expressiveness of arithmetic expressions occurring in probabilistic programs strongly depends on the employed SMT solver. The implementation in [WZH07] is based on the SMT tool Yices [DdM06] and thus restricted to linear arithmetic. SSMT-based PBMC builds on the non-linear arithmetic SMT solver iSAT, confer [FHT$^+$07] and Section 6.3, such that the probabilistic model of this thesis may contain richer arithmetic expressions involving transcendental functions like sin or exp. With regard to parallel composition, the approach of [WZH07] must also resort to flattening the overall system structure leading to an in general exponentially-sized product automaton. As mentioned above, SSMT permits a linearly-sized encoding of parallel systems and thus alleviates the state explosion arising from an explicit construction of the product automaton with respect to the discrete state space.

Aiming at probabilistic (unsafe) state reachability, the verification technique of [WZH07] was enhanced in [HWZ08] as follows: the authors suggested a method to refine the predicate abstraction based on counterexamples and furthermore extended the approach to falsification. As in [WZH07], predicate abstraction is first used to achieve a finite-state system from a given probabilistic program. Using a probabilistic finite-state model checker, the probabilistic reachability problem of the shape “the maximum probability of reaching unsafe states is at most θ” is then model checked on the abstraction. If the latter property holds for the abstraction then the original probabilistic program is *verified*. The reverse direction, however, does not hold in general. In order to close this gap, the approach in [HWZ08] generates a counterexample for the abstracted system. As opposed to the non-probabilistic case, a counterexample to a probabilistic reachability property is not a single system run but comprises several such runs, and can even be a cyclic (discrete-time) Markov chain. Exploiting a technique from [HK07], the Markov chain counterexample is preprocessed thereby obtaining a finite set of finite system runs whose probability mass exceeds safety threshold θ. The next step is to check whether the abstract counterexample can be realized, i.e. whether there is a corresponding probabilistic counterexample in the original program, or not. This requires to decide whether single runs of the abstract counterexample are realizable in the concrete system or spurious. This problem can be solved by checking satisfiability of a corresponding SMT formula. As an abstract state encodes several (potentially infinitely many) concrete states, a further challenge is to identify one concrete initial state from which the system reaches the target states with highest probability. The latter problem is reduced to a weighted MAX-SMT formula, i.e. to determine an assignment of a formula that maximizes the value of a weighted expression. If the counterexample is actually realizable, then the probabilistic reachability property is *falsified*. Otherwise, i.e. the property was neither verified nor falsified, the current abstraction is refined by adding new predicates. This so-called *probabilistic counterexample-guided abstraction refinement* is done by analyzing the non-realizable probabilistic counterexample. With regard to the reasons of non-realizability mentioned above, new predicates should prevent for the same or similar spurious abstract runs and should facilitate a finer abstraction of the initial states. Such predicates are obtained by the logical concept of *Craig interpolation* as, for instance, in [HJMM04]. We remark that Chapter 9 elaborates on the topic of Craig interpolation. The approach of [HWZ08] is implemented in the tool PASS [HHWZ10].

The authors of [ZPC10] proposed a method for the analysis of discrete-time stochastic hybrid systems with respect to bounded temporal properties that is called *statistical model checking*. Their system model is very similar to the one of [Aba07] and also permits continuous state changes according to continuous probability distributions. The statistical model checking procedure, however, does not belong to the class of exhaustive state-space exploration methods but is based on system simulation and on Bayesian statistics. As a consequence, the results obtained from statistical model checking are not guaranteed to be correct. Nevertheless, two reasons are presented to motivate this simulation-based approach: first, results are usually obtained much faster compared to exhaustive search and, second, the probability of returning a wrong result can be made arbitrarily small.

The work presented in [FHW10] discusses probabilistic reachability analysis of a rich probabilistic model that is called *first-order probabilistic timed automaton* (FPTA). Compared to probabilistic hybrid automata, the continuous dynamics of an FPTA is restricted to timed behavior as in timed automata [AD94], i.e. to real-valued clocks with progress of rate 1. While discrete actions in probabilistic hybrid automata are usually described by (non-linear) arithmetic predicates and assignments like "if $\sin(t) < \cos(t) \wedge t > \frac{1}{2}\pi$ then execute $x := x^2 + y$ and $t := 0$", transition guards and assignments in FPTAs comprise *first-order predicates*. This permits to manipulate more sophisticated data structures like lists. For instance, let the symbol cons denote a list constructor. Then, the transition guard $\mathit{list} = \mathrm{cons}(\mathit{elem}, \mathit{list}')$ followed by the assignment $\mathit{list} := \mathit{list}'$ characterizes the removal of the first list element *elem* in the list *list*. The analysis approach of [FHW10] first translates the given FPTA model into a labeled first-order formula over linear arithmetic, where labels and linear arithmetic are used to preserve all probabilistic aspects and to describe the advance of time, respectively. In the next step, the resulting formula is fed to a first-order theorem prover that is employed to enumerate all proofs. These proofs then facilitate the construction of a probabilistic timed automaton (PTA) that is reachability equivalent to the original first-order model. By the latter reduction, the original probabilistic reachability problem for FPTAs is then solved for the simpler PTA model using the PTA model checking tool MCPTA [HH09].

We finally mention the work on probabilistic safety analysis of discrete-time Markov chains (DTMCs) published in [WBB09]. The considered system model is much more restrictive than probabilistic hybrid automata: the state space is finite and the behavior is fully probabilistic, i.e. without non-determinism. The analysis technique is however closely related to the one presented in this thesis. More precisely, the authors of [WBB09] suggested a BMC-based approach to the falsification of safety properties of the shape "the probability of reaching the target states, while passing only states of some specified set, is at most θ".[5] This approach works as follows. In a first step, the given safety property of above form is reduced to state reachability by removing edges from the DTMC. By mapping probabilistic transitions to non-deterministic ones, the step-bounded behavior of the given DTMC as well as the reachability property are then described as a propositional formula as common for BMC. Note that the original transition probability matrix of the DTMC is maintained in order to keep track of the transition probabilities between states.

[5]These safety properties are actually PCTL formulae of the form $Pr_{\leq\theta}(a\,\mathtt{U}\,b)$.

The idea then is to solve the BMC formulae for increasing step depths thereby collecting all system runs that reach the target states until the probability measure of these runs exceeds θ. In a bit more detail, the BMC formula of some initial step depth k is solved by a SAT solver. If the formula is unsatisfiable then there does not exist a system run of length k that reaches the target. Otherwise, the satisfying assignment provided by the SAT solver is used to extract a run of the DTMC that reaches the target. The probability of this run is retrieved by means of the original probability matrix. After adding an additional clause to the BMC formula which excludes the previous solution, the SAT solver is called again to potentially find another run reaching the target states. Whenever the (modified) BMC formula of depth k is decided to be unsatisfiable, the process continues with BMC formula of depth $k + 1$. The overall procedure terminates if all collected system runs carry enough probability mass to falsify the safety property, thereby establishing a probabilistic counterexample. To reduce the number of SAT solver calls and thus to improve efficiency, the authors of [WBB09] proposed some optimizations. The most important one tries to detect loops in runs reaching the target states in order to achieve infinitely many runs from one solver invocation. Besides the fact that SSMT-based PBMC can deal with probabilistic *infinite*-state systems exhibiting *non-determinism*, the main difference to the approach of [WBB09] is that a PBMC formula preserves full information about the branching structure, in particular all probability information, by means of existential and randomized quantifiers. This naturally allows for a distinction between non-deterministic and probabilistic choices as well as for several algorithmic optimizations including optimizations with respect to probabilistic behavior, confer Section 6.5.

More recently, the latter approach was enhanced in [BWB$^+$11] to the more general case of Markov reward models (MRMs), which are DTMCs extended by real-valued *state rewards*. The corresponding safety properties are now of the shape "the probability of reaching the target states with an accumulated reward of at least θ_l and of at most θ_u, while passing only states of some specified set, is at most θ". In order to cope with such constraints on the accumulated reward, the step-bounded behavior of the given MRM (together with the reformulated safety property) is now encoded as an SMT formula over linear real arithmetic instead of a propositional formula. For the falsification of above safety properties, principally the same procedure as in [WBB09] is feasible. The authors of [BWB$^+$11] actually prefer a slightly different approach: since the probability measure of a run can be encoded in above SMT formula, binary search is used to generate runs of higher probabilities first, as this leads to more compact probabilistic counterexamples in general.

3.3 Concurrent discrete-time probabilistic hybrid automata

After having motivated the probabilistic system model of this thesis by means of a practical NAS application in Section 3.1 and after having surveyed related system classes in Section 3.2, we now present in detail the formal model of *concurrent discrete-time probabilistic hybrid automata* as introduced in [TEF11]. We remark that essential parts of this section were published in [TEF11] by the author of this thesis together with his co-authors.

As exemplified by the NAS case study, hybrid systems occurring in practice generally consist of multiple components evolving concurrently, both in the small, where controllers, sensor, actuators form identifiable units being coupled by one or more communication busses, or in the large, where a number of otherwise independent physical processes becomes connected via embedded control, as in a car platooning maneuver. Given the ubiquity of concurrency in such embedded control applications, it makes sense to avoid the detrimental effects of flattening concurrent systems before verification, which have become known as state explosion in the finite-state case, and offer models directly accommodating concurrency instead. In the sequel, we therefore elaborate on such a model, where probabilistic hybrid automata evolve concurrently subject to a synchronous semantics involving global agreement on transitions as in CSP [Hoa85]. Within their evolution, the individual automata

1. non-deterministically select local transitions and synchronously suggest them to the environment,
2. establish consensus on a global transition comprising one selected local transition from each concurrent component by checking mutual consistency between the individual activation conditions of the selected local transitions, releasing the synchronous global transition if and only if the conditions are consistent,
3. after having committed to this global transition, do locally select one of the available probabilistic variants of the corresponding local transition,
4. establish global consensus on execution of the locally selected probabilistic variants by checking mutual consistency of their side effects,
5. in case of consensus, execute the transition concurrently by applying their associated effects on the global state, or else deadlock due to inconsistent assignments in the committed transitions.

The semantics has been defined with the goals of, first, permitting concise models by not imposing overly restrictive rules on use of variables and, second, providing separation between the possibly non-deterministic process of transition selection and the then purely probabilistic process of selection of a transition variant, as in classical, monolithic probabilistic hybrid automata [Spr01, BC05, FHT08]. To achieve the first, both the (then not really) local conditions for transition selection and the side effects can refer to non-local variables in both pre- and post-states, forcing parallel automata to agree on mutually consistent local transitions. The second, which is a necessary prerequisite for avoiding ill-formed probability measures due to interference between *schedulers* (or *policies, adversaries*) resolving non-determinism and the probabilistic choices, is accomplished by first committing a non-deterministic transition selection and then pursuing the probabilistic selection of a variant, yielding a deadlock if the latter experiment yields an outcome which is inconsistent to the earlier selection.

Definition 3.1 (Syntax of a system of concurrent PHAs)
A system of concurrent discrete-time probabilistic hybrid automata $\mathcal{S} = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ *is given by a set of* discrete-time probabilistic hybrid automata (PHAs)*, where each probabilistic hybrid automaton $\mathcal{A}_i$ for $1 \leq i \leq n$ consists of the following:*

- *A finite set $D_i = \{d^i_1, \ldots, d^i_{k_i}\}$ of* discrete variables *spanning the discrete state space (sometimes called the locations) of the hybrid automaton by means of the Cartesian product $\bigtimes_{j=1}^{k_i} \mathrm{dom}(d^i_j)$ of their* finite *domains* $\mathrm{dom}(d^i_j)$*. Without loss of generality, we assume that each* $\mathrm{dom}(d^i_j)$ *is a bounded integer interval. In order to permit non-local referencing of the state variables, we demand that $D_i \cap D_j = \emptyset$ if $i \neq j$, i.e. that the variable names used in different concurrent automata are disjoint.*

- *A finite set $R_i = \{x^i_1, \ldots, x^i_{m_i}\}$ of* continuous state components *controlled by that automaton (yet visible to all others). Each continuous component x^i_j ranges over a bounded interval* $\mathrm{dom}(x^i_j) = [l_{x^i_j}, u_{x^i_j}]$ *within the reals* $\mathbb{R}$*. Again, we demand that $R_i \cap R_j = \emptyset$ if $i \neq j$. Additionally, we require discrete variable names and continuous variable names to be disjoint, i.e. $D_i \cap R_j = \emptyset$ for all i and j.*

- *A predicate $init_i$ in an arithmetic theory $\mathcal{T}$ with free variables in D_i and R_i describing the* initial state *of the automaton. For technical reasons and without loss of generality, we demand that there is exactly one valuation in the state set $States_i = \bigtimes_{j=1}^{k_i} \mathrm{dom}(d^i_j) \times \bigtimes_{j=1}^{m_i} \mathrm{dom}(x^i_j)$ of the automaton which satisfies $init_i$. Note that due to the disjointness of the local variable name spaces, this implies existence of exactly one global initial state $s \in \bigtimes_{i=1}^{n} States_i$ satisfying $\bigwedge_{i=1}^{n} init_i$.*

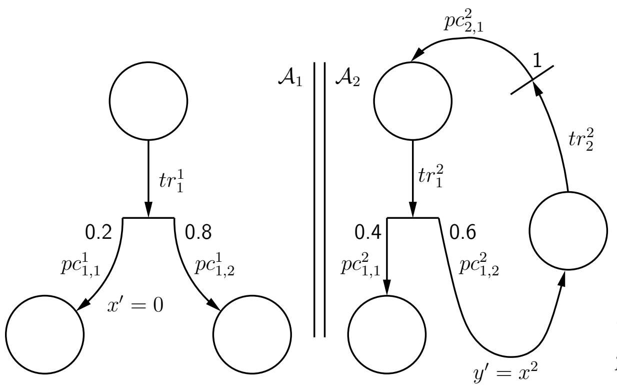
- *A finite family $Tr_i = \{tr^i_1, \ldots, tr^i_{\ell_i}\}$ of* symbolic transitions.

Each symbolic transition tr^i_j comprises the following.

- *A* generalized transition guard $g(tr^i_j)$ *expressing the conditions on local and global variables required for establishing consensus on that transition. As for the description of initial states, $g(tr^i_j)$ is a predicate in the arithmetic theory $\mathcal{T}$ over variables in $D_1, \ldots, D_n$ and $R_1, \ldots, R_n$ as well as primed variants thereof, the latter representing the post-states. A transition guard states the conditions on the discrete as well as the continuous state under which the transition may be taken. Note that the guard predicate can refer to the current states and post-states of all concurrent automata in $\mathcal{S}$. It thus provides an expressive formalism supporting synchronization through global consensus.*

- *A* discrete probability distribution $p(tr^i_j) \in \mathcal{D}(PC_{tr^i_j})$*, where $PC_{tr^i_j}$ is a finite and non-empty set of symbolic transition alternatives and $\mathcal{D}(PC_{tr^i_j})$ denotes the set of discrete probability distributions over $PC_{tr^i_j}$. That is, $p(tr^i_j)$ assigns to transition tr^i_j a distribution over $|PC_{tr^i_j}|$ many transition alternatives. Without loss of generality, we demand that each transition alternative is of positive probability, i.e. for each $pc \in PC_{tr^i_j}$ it holds that $p(tr^i_j)(pc) > 0$.*

- *For each transition alternative $pc \in PC_{tr^i_j}$ of transition tr^i_j an* assignment predicate *$asgn(tr^i_j, pc)$ defining the successor state. As for transition guards, $asgn(tr^i_j, pc)$ is an arithmetic predicate in the arithmetic theory $\mathcal{T}$ over variables in $D_1, \ldots, D_n$ and $R_1, \ldots, R_n$ as well as primed variants thereof, the latter again representing the post-states.*



$$\begin{aligned}
NChoice &= \{(tr_1^1, tr_1^2), (tr_1^1, tr_2^2)\}\\
PChoice((tr_1^1, tr_1^2)) &= \{(pc_{1,1}^1, pc_{1,1}^2), (pc_{1,2}^1, pc_{1,1}^2),\\
&\qquad (pc_{1,1}^1, pc_{1,2}^2), (pc_{1,2}^1, pc_{1,2}^2)\}\\
PChoice((tr_1^1, tr_2^2)) &= \{(pc_{1,1}^1, pc_{2,1}^2), (pc_{1,2}^1, pc_{2,1}^2)\}\\
PChoice &= \ PChoice((tr_1^1, tr_1^2))\\
&\quad \cup PChoice((tr_1^1, tr_2^2))
\end{aligned}$$

$$Assign((tr_1^1, tr_1^2), (pc_{1,1}^1, pc_{1,2}^2)) \equiv x' = 0 \wedge y' = x^2$$

$$p((tr_1^1, tr_1^2), (pc_{1,1}^1, pc_{1,2}^2)) = 0.2 \cdot 0.6 = 0.12$$

Figure 3.2: A parallel composition of probabilistic hybrid automata. Guards are omitted for the sake of clarity. (Source of figure: [TEF11])

Note that the assignment predicate may again refer to the global pre- and post-state, i.e. the current states and the post-states of all concurrent automata in $\mathcal{S}$. This definition enables an automaton to read state variables of other automata, and moreover offers the possibility of non-local writes, entailing agreement in case of multiple concurrent updates to the same variables. Semantically, updates will only be performed in case all concurrent automata agree on them, and the system will become deadlocked in case of inconsistent updates. Furthermore, we require that the concurrent execution of assignments are deterministic with respect to the primed variables, i.e. the concurrent execution of the local transition alternatives of the individual automata uniquely determines the global post-state of the overall system.

The above two requirements imply that each concurrently enabled combination of local transitions may permit at most one successor state for each possible resolution of the local probabilistic choices. This condition necessitates a global view of transitions and their related assignments, which motivates the following definitions, as illustrated in Figure 3.2. To obtain such a global view, let $NChoice = \bigtimes_{i=1}^{n} Tr_i$ *denote the Cartesian product of the local transition sets, thus representing the set of all potentially possible global transitions. As each local transition may have multiple probabilistic variants, the same applies for global transitions. With regard to a single global transition* $(tr^1, \ldots, tr^n) \in NChoice$*, the set of associated probabilistic transition alternatives is* $PChoice((tr^1, \ldots, tr^n)) = \bigtimes_{i=1}^{n} PC_{tr^i}$*, which is the Cartesian product of the local probabilistic transition alternatives available for the individual local transitions* $tr^1, \ldots, tr^n$*. Taking together all the global probabilistic alternatives of all global transitions,* $PChoice = \bigcup_{nc \in NChoice} PChoice(nc)$ *denotes the set of all global probabilistic choices. Given a global non-deterministic transition choice* $tr = (tr^1, \ldots, tr^n) \in NChoice$ *and a corresponding global probabilistic alternative choice* $pc = (pc^1, \ldots, pc^n) \in PChoice(tr)$*, we denote by* $Assign(tr, pc) = \bigwedge_{i=1}^{n} asgn(tr^i, pc^i)$ *the conjunction of the selected local assignment predicates. With these definitions, we can formalize the requirements that each concurrent execution of local transition alternatives be deterministic: We demand that for each global transition* $tr \in NChoice$ *and for each global probabilistic alternative* $pc \in PChoice(tr)$*, the associated global assignment* $Assign(tr, pc)$ *is deterministic or, equivalently, a partial function, i.e. it* *satisfies*

$$Assign(tr, pc) \wedge Assign(tr, pc)[\vec{e}/\vec{d'}, \vec{y}/\vec{x'}] \ \Rightarrow\ \vec{e} = \vec{d'} \wedge \vec{y} = \vec{x'}$$

where $\vec{d'}$ and $\vec{x'}$ denote the vectors of all primed discrete and continuous variables of all automata $\mathcal{A}_1, \ldots, \mathcal{A}_n$, respectively. Intuitively, the current global state and a global assignment uniquely determines the global post-state.

Observe that above definition left blank a concrete description of the arithmetic theory $\mathcal{T}$. In this thesis, we concentrate on the very general theory of non-linear arithmetic over the reals and integers involving transcendental functions like exponential and trigonometric functions. We remark that $\mathcal{T}$-predicates may also comprise logical operators and can thus be a complex-structured non-linear SMT formula as formally introduced in Section 4.3.

Let $States_{\mathcal{S}} = \bigtimes_{i=1}^{n} States_i$ be the global state space of system $\mathcal{S}$. In the sequel, we define the concurrent semantics of the system $\mathcal{S}$. Here, all partners do propose a local transition that is fixed as soon as the partners have reached consensus in the sense of the guards of the involved local transitions being consistent. The latter amounts to checking whether a global post-state exists which together with the current pre-state satisfies the conjunction of the (generalized) local guards. Once the global transition has been negotiated, all partners do randomly select a local transition alternative. Provided that the assignments corresponding to the resulting global probabilistic alternative are consistent, each system enters the unique post-state of $\mathcal{S}$ arising due to determinacy of assignments. In case the selected global system step is impossible due to inconsistency between the selected guards of all $\mathcal{A}_i$ or due to inconsistency of the randomly selected assignments, the overall system $\mathcal{S}$ deadlocks in a distinguished state $\perp$.

Given a selection of transitions and transition alternatives, it immediately follows from Definition 3.1 (determinacy of global assignments) that at most one post-state exists:

Property 3.1 (Uniqueness of post-states)
Let $\mathcal{S}$ be a system of concurrent discrete-time probabilistic hybrid automata. Further, let $s \in States_{\mathcal{S}}$ be a state of $\mathcal{S}$, $tr = (tr^1, \ldots, tr^n) \in NChoice$ be a non-deterministic transition choice, and $pc = (pc^1, \ldots, pc^n) \in PChoice(tr)$ be a probabilistic choice of transition alternatives. We define the predicate $val(z)$ for $z \in States_{\mathcal{S}}$ as a conjunction of equations $\bigwedge_{v \in \bigcup_{i=1}^{n}(D_i \cup R_i)} v = z(v)$, where $z(v)$ is the value of variable v in state z. Then, if

$$val(s) \wedge \bigwedge_{i=1}^{n} \big(g(tr^i) \wedge asgn(tr^i, pc^i)\big)$$

is satisfiable then there exists exactly one state s' such that

$$val(s) \wedge val(s')' \wedge \bigwedge_{i=1}^{n} \big(g(tr^i) \wedge asgn(tr^i, pc^i)\big)$$

is satisfiable, where $val(.)'$ is $val(.)$ with all variable names decorated by primes.

In this case, we denote by $Post(s, tr, pc)$ the unique post-state s'. Otherwise, the system deadlocks and we define $Post(s, tr, pc) = \perp$. For convenience, let be $Post(\perp, tr, pc) = \perp$ for all tr, pc, and let $\perp$ not satisfy any $\mathcal{T}$-predicate.

The next definition explains the executable system behavior:

Definition 3.2 (Semantics of a system of concurrent PHAs)
The semantics of a system $\mathcal{S}$ of concurrent PHAs is defined by runs *of $\mathcal{S}$ that are finite*[6] *alternating sequences of states and transitions, the latter involving both non-deterministic and probabilistic choices. Each run $r = \langle s_0, (tr_1, pc_1), s_1, \ldots, (tr_k, pc_k), s_k\rangle \in (States_{\mathcal{S}} \cup \{\perp\}) \times ((NChoice \times PChoice) \times (States_{\mathcal{S}} \cup \{\perp\}))^*$ of $\mathcal{S}$ meets the following properties:*

1. *$pc_j \in PChoice(tr_j)$ for all $1 \leq j \leq k$.*

2. *$s_{j+1} = Post(s_j, tr_j, pc_j)$ for all $0 \leq j \leq k-1$.*

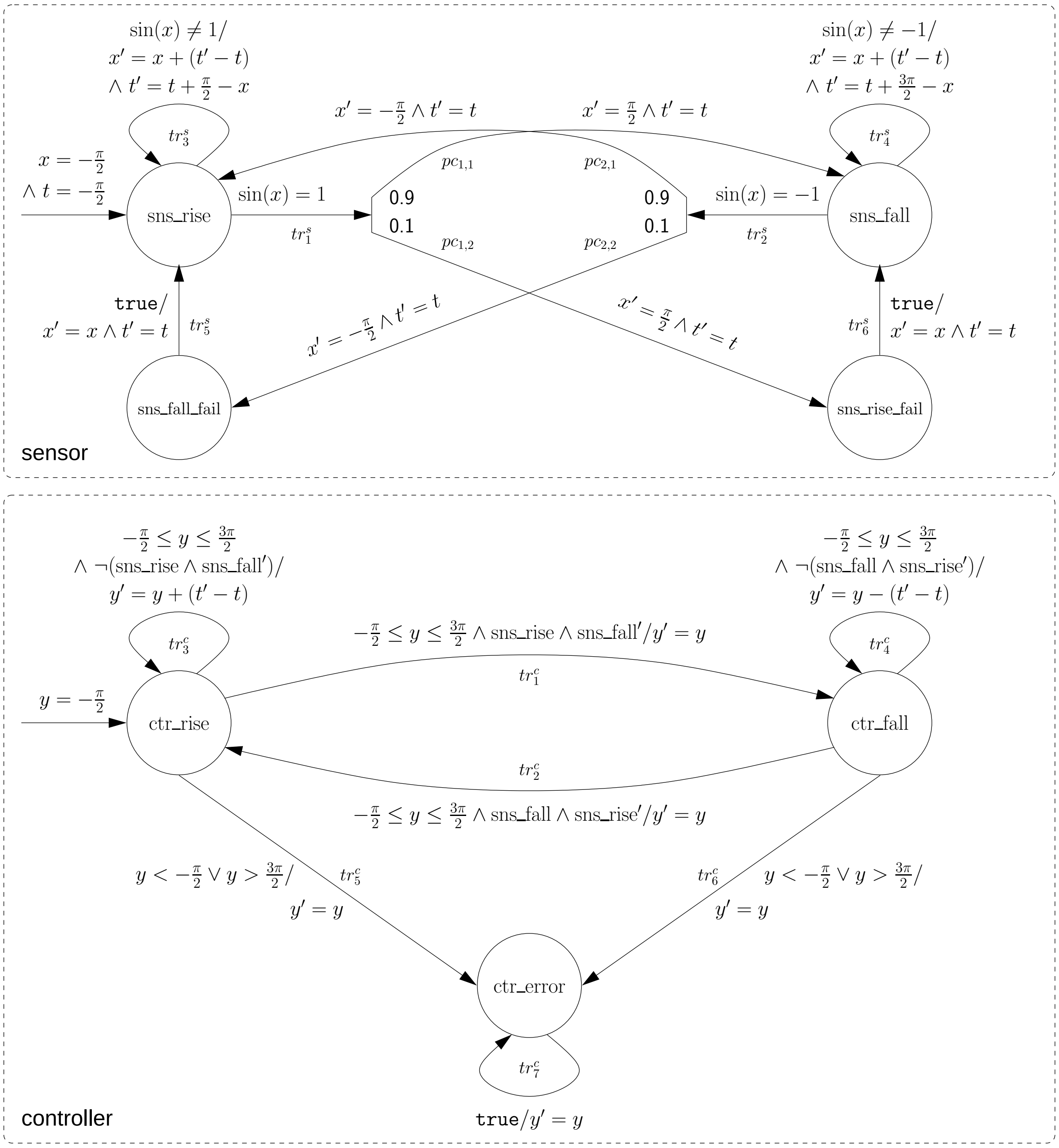
We define $first(r) = s_0$ and $last(r) = s_k$. We say that run r starts in state s_0. *For technical reasons, we do not demand that a run starts in the initial state. We call r* anchored run *whenever s_0 is the (unique) initial state, i.e. s_0 satisfies the initial predicate $\bigwedge_{i=1}^{n} init_i$. The* length of run r, *denoted by $length(r)$, coincides with the number of transition steps involved, i.e. $length(r) = k$. Each subsequence $\langle s_i, (tr_i, pc_i), s_{i+1}\rangle$ of r is referred to as* transition step *or, synonymously,* system step *or* step, *for short.*

Thus, each anchored run starts in the global initial state defined by the initial state predicates of the concurrent components. Upon each transition step, all concurrent automata first select non-deterministically among their transitions and then probabilistically under their variants. The corresponding transition step leads to a unique post-state, if existent, or to deadlock otherwise. The *probability of a transition step* $\langle s, (tr, pc), s'\rangle$ from s to s' under non-deterministic choice $tr = (tr^1, \ldots, tr^n)$ and probabilistic choice $pc = (pc^1, \ldots, pc^n)$ is given by $p(tr, pc) = \prod_{i=1}^{n} p(tr^i)(pc^i)$, confer Figure 3.2. The *probability $p(r)$ of a (finite) run r* is the product of the probabilities of all transition steps of r, with the common convention that the empty product is equal to 1. Hence, for each run r of length 0, i.e. $r = \langle s\rangle$ with $s \in States_{\mathcal{S}} \cup \{\perp\}$, we have $p(r) = 1$. Note that under a given scheduler resolving non-determinism the accumulated probability of all runs that start in the same state and are of the same length is always 1.

Example. Consider the system $\mathcal{S} = \{\textsf{sensor}, \textsf{controller}\}$ depicted in Figure 3.3. For the sake of clarity, we omitted probabilistic transition alternatives whenever just one exists, for instance, in the entire automaton controller. In order to be more intuitive and illustrative, we talk about concrete location names like sns_rise or ctr_rise when explaining the model. Note that the set of all locations, i.e. the discrete state space, is formally given by the valuations of discrete variables, confer Definition 3.1. In our example, we may assume that the discrete state space of each automaton is spanned by one discrete variable whose domain consists of the corresponding locations encoded as integers.

The idea of this very simple model is that sensor shall perform discrete state changes whenever the sine curve (evaluated over time) reaches its extremal values. That is, from sns_rise to sns_fall when hitting the maximum, and vice versa when reaching the minimum value. This switching behavior is synchronized with the controller, i.e. the controller retrieves such state changes of sensor in its guards. The controller regulates the continuous

[6]Considering finite runs of PHAs suffices for the purpose of this thesis, since we investigate bounded reachability.



sensor
sns_rise
sns_fall
sns_fall_fail
sns_rise_fail
$\sin(x) \neq 1/$ $x' = x + (t' - t)$ $\wedge\, t' = t + \frac{\pi}{2} - x$
tr_3^s
$\sin(x) \neq -1/$ $x' = x + (t' - t)$ $\wedge\, t' = t + \frac{3\pi}{2} - x$
tr_4^s
$x = -\frac{\pi}{2}$ $\wedge\, t = -\frac{\pi}{2}$
$x' = -\frac{\pi}{2} \wedge t' = t$
$x' = \frac{\pi}{2} \wedge t' = t$
$\sin(x) = 1$
tr_1^s
$\sin(x) = -1$
tr_2^s
$pc_{1,1}$
$pc_{2,1}$
$pc_{1,2}$
$pc_{2,2}$
0.9
0.1
0.9
0.1
true/ $x' = x \wedge t' = t$
tr_5^s
true/ $x' = x \wedge t' = t$
tr_6^s
$x' = -\frac{\pi}{2} \wedge t' = t$
$x' = \frac{\pi}{2} \wedge t' = t$
controller
ctr_rise
ctr_fall
ctr_error
$-\frac{\pi}{2} \leq y \leq \frac{3\pi}{2}$ $\wedge\, \neg(\text{sns\_rise} \wedge \text{sns\_fall}')/$ $y' = y + (t' - t)$
tr_3^c
$-\frac{\pi}{2} \leq y \leq \frac{3\pi}{2}$ $\wedge\, \neg(\text{sns\_fall} \wedge \text{sns\_rise}')/$ $y' = y - (t' - t)$
tr_4^c
$y = -\frac{\pi}{2}$
$-\frac{\pi}{2} \leq y \leq \frac{3\pi}{2} \wedge \text{sns\_rise} \wedge \text{sns\_fall}'/y' = y$
tr_1^c
tr_2^c
$-\frac{\pi}{2} \leq y \leq \frac{3\pi}{2} \wedge \text{sns\_fall} \wedge \text{sns\_rise}'/y' = y$
$y < -\frac{\pi}{2} \vee y > \frac{3\pi}{2}/$ $y' = y$
tr_5^c
tr_6^c
$y < -\frac{\pi}{2} \vee y > \frac{3\pi}{2}/$ $y' = y$
tr_7^c
true/$y' = y$

Figure 3.3: Graphical representation of the two concurrent probabilistic hybrid automata sensor and controller. (Source of figure: [TEF11])

variable y that is increasing over time in discrete state ctr_rise and decreasing over time in ctr_fall. The global time of the system is modeled by variable t, and the time passage is governed by automaton sensor. A safety requirement of the system, for instance, is that the value of y may never leave the safe region $[-\pi/2, 3\pi/2]$. In case of violation, the controller enters the discrete state ctr_error and remains there forever. It is thus of interest whether the overall system may violate the safety property, and, if so, to quantify the system error.

The probabilistic behavior of $\mathcal{S}$ arises from the fact that the modeled sensor may over-

	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)
sns_rise	$\times$	$\times$	$-$	$-$	$-$	$\times$	$\times$	$-$
sns_fall	$-$	$-$	$-$	$\times$	$\times$	$-$	$-$	$\times$
sns_fall_fail	$-$	$-$	$-$	$-$	$-$	$-$	$-$	$-$
sns_rise_fail	$-$	$-$	$\times$	$-$	$-$	$-$	$-$	$-$
x	-$\frac{\pi}{2}$	$\frac{\pi}{2}$	$\frac{\pi}{2}$	$\frac{\pi}{2}$	$\frac{3\pi}{2}$	-$\frac{\pi}{2}$	$\frac{\pi}{2}$	$\frac{\pi}{2}$
$(\sin(x))$	-1	1	1	1	-1	-1	1	1
t	-$\frac{\pi}{2}$	$\frac{\pi}{2}$	$\frac{\pi}{2}$	$\frac{\pi}{2}$	$\frac{3\pi}{2}$	$\frac{3\pi}{2}$	$\frac{5\pi}{2}$	$\frac{5\pi}{2}$
ctr_rise	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$-$
ctr_fall	$-$	$-$	$-$	$-$	$-$	$-$	$-$	$-$
ctr_error	$-$	$-$	$-$	$-$	$-$	$-$	$-$	$\times$
y	-$\frac{\pi}{2}$	$\frac{\pi}{2}$	$\frac{\pi}{2}$	$\frac{\pi}{2}$	$\frac{3\pi}{2}$	$\frac{3\pi}{2}$	$\frac{5\pi}{2}$	$\frac{5\pi}{2}$

Figure 3.4: Sample anchored run of $\mathcal{S}$. (Source of figure: [TEF11])

look an optimum of the sine curve with some probability, say 0.1. In such cases, the controller may not perform a state change from ctr_rise to ctr_fall or vice versa. As depicted in Figure 3.3, this is modeled by the transition alternatives $pc_{1,2}$ and $pc_{2,2}$ of transitions tr_1^s and tr_2^s, respectively. Each such alternative occurs with probability 0.1 forcing the sensor to visit one of the fail states sns_rise_fail and sns_fall_fail. These fail states are then left immediately, but the effect is that the controller does not detect the discrete state change of sensor as desired.

Figure 3.4 depicts a possible anchored run of $\mathcal{S}$. Initially, it starts in the (unique) initial state (a) $((\text{sns\_rise}, x = -\pi/2, t = -\pi/2), (\text{ctr\_rise}, y = -\pi/2))$. The choice $(tr_3^s, tr_3^c), (-, -)$ (where the probabilistic alternatives are left free due to uniqueness) leads to state (b) $((\text{sns\_rise}, x = \pi/2, t = \pi/2), (\text{ctr\_rise}, y = \pi/2))$. Now, the guard $\sin(x) = 1$ of tr_1^s is true. The only enabled transition of sensor thus is tr_1^s. Assume that the sensor fails now which is modeled by the probabilistic transition alternative $pc_{1,2}$. The controller consistently selects transition tr_3^c to remain in ctr_rise. Under this choice the next system state (c) is $((\text{sns\_rise\_fail}, x = \pi/2, t = \pi/2), (\text{ctr\_rise}, y = \pi/2))$, and immediately thereafter (d) $((\text{sns\_fall}, x = \pi/2, t = \pi/2), (\text{ctr\_rise}, y = \pi/2))$. The sensor now takes transition tr_4^s and the controller tr_3^c, and the system enters state (e) $((\text{sns\_fall}, x = 3\pi/2, t = 3\pi/2), (\text{ctr\_rise}, y = 3\pi/2))$. By the next choice $(tr_2^s, tr_3^c), (pc_{2,1}, -)$, a discrete state change in sensor is performed while setting x to $-\pi/2$, and results in state (f) $((\text{sns\_rise}, x = -\pi/2, t = 3\pi/2), (\text{ctr\_rise}, y = 3\pi/2))$. Then, both automata perform a self loop in their current discrete states yielding (g) $((\text{sns\_rise}, x = \pi/2, t = 5\pi/2)$, $(\text{ctr\_rise}, y = 5\pi/2))$. Now, the value of variable y has left the safety interval $[-\pi/2, 3\pi/2]$, and selecting transition tr_5^c of controller leads to location ctr_error. The sensor selects transition tr_1^s and probabilistically the alternative $pc_{1,1}$. So, the next state (h) is $((\text{sns\_fall}, x = \pi/2, t = 5\pi/2), (\text{ctr\_error}, y = 5\pi/2))$. The length of this run is 7, and its probability is given by the probabilities of its transition steps. There are just three steps with a probability lower than 1, namely these with transition alternatives $pc_{1,2}$, $pc_{2,1}$, and $pc_{1,1}$. Therefore, the probability of this run is $0.1 \cdot 0.9 \cdot 0.9 = 0.081$.

This example shows existence of a single anchored system run violating the safety property with probability 0.081. Assuming that some probabilistic requirement however permits violation with probability at most 0.1, then from above information we cannot conclude whether this requirement is satisfied by the system or not. To obtain a meaningful figure, the full, non-deterministic and probabilistic, system behavior needs to be taken into account. This issue is addressed in Chapter 5 which introduces the problem of probabilistic bounded state reachability and furthermore suggests a symbolic method to solve the problem. Since this symbolic analysis approach is based on a stochastic extension of satisfiability modulo theories called SSMT, we devote the following chapter to a detailed picture of SSMT and its underlying notions.

4 Stochastic Satisfiability Modulo Theories

This chapter introduces the notion of *stochastic satisfiability modulo theories*, SSMT for short, as the underlying formalism for the symbolic analysis of probabilistic hybrid systems being explored in Chapter 5. The idea of SSMT is to combine two well-studied logical concepts, namely first *stochastic Boolean satisfiability* (SSAT) and second *satisfiability modulo theories* (SMT), while both SSAT and SMT are extensions of the well-known *Boolean satisfiability* (SAT) problem. To reasonably approach the definition of SSMT in Section 4.4, we first recall the SAT problem in Section 4.1 and we then devote our attention to SSAT and SMT in Sections 4.2 and 4.3, respectively.

We remark that essential parts of Section 4.2 were published in [TF10] by the author of this thesis together with his co-author.

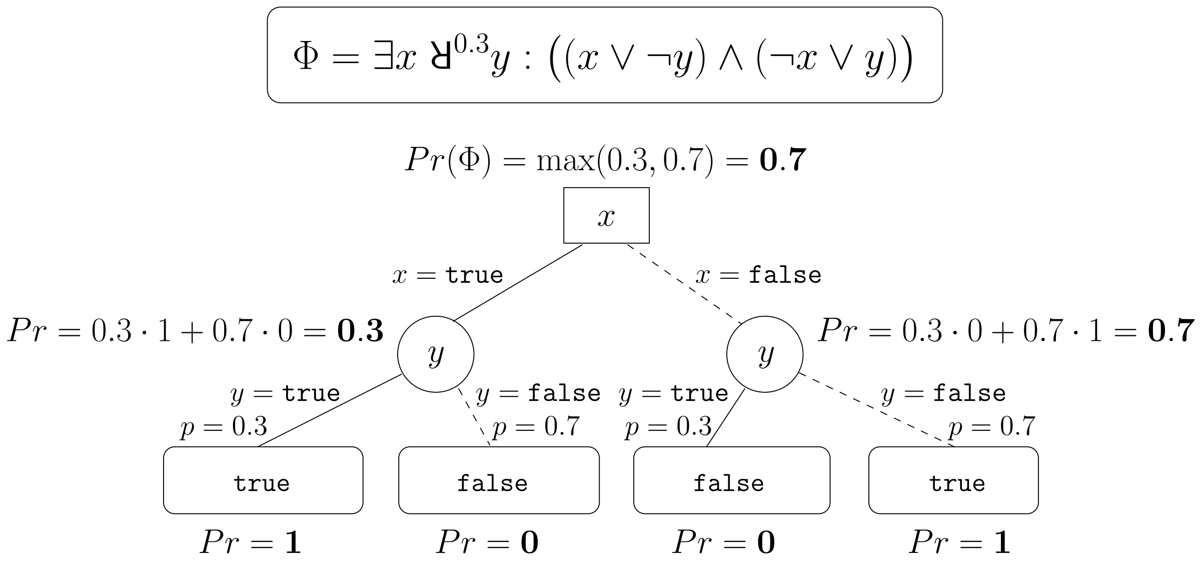
4.1 Boolean satisfiability

Given an arbitrary propositional formula φ, the *Boolean satisfiability problem*, or SAT for short, asks whether a satisfying assignment of φ exists. SAT is one of the most famous and most important decision problems in computer science, in fact from a theoretical as well as practical point of view. SAT was the first problem to be known as NP-complete, i.e. one of the "hardest" problems in NP. This seminal result was proven by Cook in 1971 [Coo71] and facilitated to discover more NP-complete problems by means of reductions from SAT and then from those new NP-complete problems, confer, for instance, [Kar72, GJ90]. The Boolean satisfiability problem remains NP-complete even for formulae in CNF and moreover for formulae in kCNF for $k \geq 3$ [Coo71]. When restricting the formula to be in 2CNF then the resulting problem becomes solvable in linear time [APT79].

Though it is still open whether P = NP holds or not, most experts believe that P $\neq$ NP and thus a polynomial time algorithm for SAT is not expected. As a consequence, hitherto existing SAT algorithms show exponential runtime in worst case. However, there is a lot of work to improve performance of these so-called SAT solvers in practice. The need of practically efficient SAT solvers is motivated by industrial applications of SAT like software and hardware verification. Nowadays, SAT solvers are actually high-performance tools for many real-world problems in formal verification, and are moreover accepted and used by industry. Modern SAT solving techniques are also exploited in algorithms for extensions of SAT, in particular for SSAT, SMT, and SSMT. In Chapter 6, we elaborate on solving technologies for these different problems.

4.2 Stochastic Boolean satisfiability

Papadimitriou [Pap85] has proposed the idea of modeling uncertainty within propositional logic by introducing *randomized* quantification in addition to existential quantification.



$\Phi = \exists x\, \text{Ⴘ}^{0.3} y : \big((x \vee \neg y) \wedge (\neg x \vee y)\big)$
$Pr(\Phi) = \max(0.3, 0.7) = \mathbf{0.7}$
x
$x =$ true
$x =$ false
$Pr = 0.3 \cdot 1 + 0.7 \cdot 0 = \mathbf{0.3}$
$Pr = 0.3 \cdot 0 + 0.7 \cdot 1 = \mathbf{0.7}$
y
y
$y =$ true
$p = 0.3$
$y =$ false
$p = 0.7$
$y =$ true
$p = 0.3$
$y =$ false
$p = 0.7$
true
false
false
true
$Pr = \mathbf{1}$
$Pr = \mathbf{0}$
$Pr = \mathbf{0}$
$Pr = \mathbf{1}$

Figure 4.1: Semantics of an SSAT formula Φ depicted as a tree. (Source of figure: [TF10])

The resultant *stochastic Boolean satisfiability* (SSAT) problems consist of a quantifier prefix followed by a propositional formula. The quantifier prefix is an alternating sequence of existentially quantified variables and variables bound by randomized quantifiers. The meaning of a randomized variable x is that x takes value `true` with a certain probability p and value `false` with the complementary probability $1-p$, while the value of an existential variable can be set arbitrarily. Due to the presence of such probabilistic assignments, the semantics of an SSAT formula Φ no longer is qualitative in the sense that Φ is satisfiable or unsatisfiable as it is for propositional formulae, but rather *quantitative* in the sense that we are interested in the *maximum probability of satisfaction* of Φ. Intuitively, a solution of Φ is a strategy for assigning the existential variables, i.e. a tree of assignments to the existential variables depending on the probabilistically determined values of preceding randomized variables, such that the assignments maximize the probability of satisfying the propositional formula.

The formal definition of the syntax and semantics of SSAT is as follows.

Definition 4.1 (Syntax of SSAT)
A stochastic Boolean satisfiability (SSAT) *formula Φ is of the form $\mathcal{Q} : \varphi$ where*

1. *$\mathcal{Q} = Q_1 x_1 \odot \ldots \odot Q_n x_n$ is a* quantifier prefix *of quantified propositional variables x_i with $1 \leq i \leq n$, where Q_i is either an existential quantifier $\exists$ or a randomized quantifier Ⴘ^{p_i} with a rational constant $0 < p_i < 1$, and*

2. *φ is a propositional formula such that $Var(\varphi) \subseteq \{x_1, \ldots, x_n\}$.*

The quantifier-free propositional formula φ is sometimes called the matrix *of Φ.*

Definition 4.2 (Semantics of SSAT)
The semantics of an SSAT formula Φ is defined by the maximum probability of satisfac-

tion $Pr(\Phi)$ *as follows:*

$$\begin{aligned}
Pr(\varepsilon : \varphi) \quad &= \begin{cases} 0 & \text{if} \quad \varphi \equiv \texttt{false} \ , \\ 1 & \text{if} \quad \varphi \equiv \texttt{true} \ , \end{cases} \\
Pr(\exists x \odot \mathcal{Q} : \varphi) \ &= \max(Pr(\mathcal{Q} : \varphi[\texttt{true}/x]), Pr(\mathcal{Q} : \varphi[\texttt{false}/x])) \ , \\
Pr(\text{Я}^p x \odot \mathcal{Q} : \varphi) &= p \cdot Pr(\mathcal{Q} : \varphi[\texttt{true}/x]) \ + \ (1-p) \cdot Pr(\mathcal{Q} : \varphi[\texttt{false}/x]) \ ,
\end{aligned}$$

where ε denotes the empty and $\mathcal{Q}$ an arbitrary quantifier prefix.

Note that the semantics is well-defined as Φ has no free variables such that all variables have been substituted by the constants `true` and `false` when reaching the quantifier-free base case. For an illustrating example of the SSAT semantics confer Figure 4.1.

Without loss of generality, we may assume that the matrix of an SSAT formula is in CNF. The rationale is that each propositional formula φ can be efficiently translated into an equi-satisfiable formula φ' in CNF using the Tseitin transformation [Tse68], confer Section 2.2. By this transformation, formula φ' may contain fresh auxiliary variables $h_1, \ldots, h_k \notin \mathit{Var}(\varphi)$, which are interpreted as innermost existentially quantified. It then holds that $Pr(\mathcal{Q} : \varphi) = Pr(\mathcal{Q} \odot \exists h_1 \odot \ldots \odot \exists h_k : \varphi')$.

In order to simplify naming, we occasionally use the terms *maximum satisfaction probability*, *probability of satisfaction*, as well as *satisfaction probability* synonymously for *maximum probability of satisfaction* whenever the latter is clear from the context.

For the sake of completeness, we remark that an extension of SSAT that additionally allows *universal* quantifiers $\forall$ has also been considered in the literature. The resulting notion is called *extended* SSAT, or XSSAT for short, confer [LMP01, Maj09]. Semantically, a universal quantifier calls for *minimizing* the satisfaction probability. That is, to define $Pr(\Phi)$ for XSSAT formulae Φ, Definition 4.2 must be extended by rule $Pr(\forall x \odot \mathcal{Q} : \varphi) = \min(Pr(\mathcal{Q} : \varphi[\texttt{true}/x]), Pr(\mathcal{Q} : \varphi[\texttt{false}/x]))$.

Applications and extensions. In recent years, the SSAT framework has attracted interest within the Artificial Intelligence community, as many problems from that area involving uncertainty have concise descriptions as SSAT problems, in particular *probabilistic planning problems* [LMP01, ML98a, ML03, Maj07] and *belief network inference problems* [Rot96]. Inspired by that work, other communities have started to exploit SSAT and closely related formalisms within their domains. The Constraint Programming community has developed the notion of *stochastic constraint satisfaction problem* (SCSP) [Wal02, BS06] to address, among others, multi-objective *decision making under uncertainty* [BS07]. A SCSP is defined by a set of constraints over existential and randomized variables, and extends SSAT in the following sense: first, variables need not range over the Boolean domain but over arbitrary finite domains and, second, a constraint may describe any relation between variables by specifying the allowed tuples of their values. However, SCSP does not add expressive power to the concept of SSAT since each SCSP can be encoded into SSAT. This observation relies on the encoding of (non-stochastic) constraint satisfaction problems into SAT, confer, for instance, [Wal00], and on the fact that an existential or randomized variable over any finite domain with n values can be represented by a binary tree of depth at most $n-1$ and thus by at most $n-1$ existential or randomized propositional variables. In Section 4.4, we enhance the expressiveness of SSAT substantially, namely by integrating the in general undecidable theory of non-linear arithmetic over the reals and integers, while the quantified variables range over finite domains as in SCSP. The resulting logical framework, called *stochastic satisfiability modulo theories*, then opens a new application of stochastic satisfiability in symbolic *probabilistic model checking*, the latter being exposed in Chapter 5.

4.2.1 Computational complexity of SSAT

Given any SSAT formula Φ and any rational constant $0 \leq \theta \leq 1$, the SSAT decision problem (Φ, θ) asks for whether $Pr(\Phi) \geq \theta$ holds. In general, this problem is PSPACE-complete [Pap85, Lit99]. The hardness can be easily shown by a reduction from the *quantified Boolean formula* (QBF) problem: given a QBF instance $\mathcal{Q} : \varphi$, i.e. $\mathcal{Q}$ may contain existential and universal quantifiers, we construct the SSAT formula $\mathcal{Q}' : \varphi$ such that $\mathcal{Q}'$ arises from $\mathcal{Q}$ by replacing all universal quantifiers by randomized ones $Ⅎ^p$ with some rational $0 < p < 1$. Then, $\mathcal{Q} : \varphi$ is `true` if and only if $Pr(\mathcal{Q}' : \varphi) \geq 1$. This reduction shows that QBF can be seen as a special case of SSAT, while both general problems share PSPACE-completeness.

There is some extensive work on the complexity of SSAT and QBF subcases that gives a better insight into the relation of both problems. When restricting a QBF formula to just existential or to just universal variables, this results in the well-known NP-complete SAT problem or in the co-NP-complete tautology (TAUT) problem, respectively. The subclass of SSAT that allows only randomized variables gives the PP-complete ("probabilistic polynomial time") MAJSAT problem. Recall that (co-)NP $\subseteq$ PP $\subseteq$ PSPACE holds. Thus, randomized quantifiers are in some sense computationally harder than just existential or just universal ones. In addition to restricting the quantifier prefix, it is of interest to consider special shapes of the propositional formula. The special cases of SAT, TAUT, QBF, MAJSAT, and SSAT for which the formulae are in 3CNF do not change the complexity results mentioned above. Restricting however a QBF formula to be in 2CNF, the resulting QBF subproblem can be solved in linear time [APT79], and thus also the corresponding subcases of SAT and TAUT. The same restriction of MAJSAT, called MAJ2SAT, however remains PP-complete [GHM05]. This is an interesting result as alternating existential and universal quantifiers seem to be computationally harder than just randomized ones for formulae in kCNF with $k \geq 3$, but computationally weaker for formulae in 2CNF.

In the following, we investigate the complexity of the SSAT subclass for which the propositional formula is in 2CNF. We call this problem S2SAT. As MAJ2SAT is PP-complete, it immediately follows that S2SAT is PP-hard. The precise complexity of S2SAT, however, was open to the best of our knowledge. In [TF10, Section 3], we have shown that S2SAT is as hard as the general SSAT problem, i.e. PSPACE-complete. A summary of the complexity results mentioned above is given in Figure 4.2.

In the rest of this subsection, we prove the new complexity result for SSAT. PSPACE-membership of S2SAT immediately follows from the fact that S2SAT is a subcase of SSAT. We prove PSPACE-hardness by a polynomial-time many-one reduction from the PSPACE-complete decision problem 1-in-3 Q3SAT. A 1-in-3 Q3SAT formula $\mathcal{Q} : \varphi$ is simply a

Formula	SAT	TAUT	MAJSAT	QBF	SSAT
any	NP	co-NP	PP	PSPACE	PSPACE
3CNF	NP	co-NP	PP	PSPACE	PSPACE
2CNF	P	P	PP	P	PSPACE

Figure 4.2: Overview of the computational complexity of SAT, TAUT, MAJSAT, QBF and SSAT as well as of their subcases in which the propositional formulae are in 3CNF and in 2CNF.

Q3SAT formula, i.e. a QBF formula with φ being in 3CNF. While quantifier treatment remains unchanged, satisfaction of φ however differs from the standard definition: φ is *1-in-3 satisfied* under truth assignment τ if and only if each clause $c \in \varphi$ is *1-in-3 satisfied* under τ, i.e. if and only if *exactly one* literal in each c is satisfied under τ. PSPACE-hardness can be shown by reduction from Q3SAT that relies on the reduction from 3SAT to 1-in-3 3SAT by Schaefer [Sch78]. For an arbitrary Q3SAT instance $\mathcal{Q} : \varphi$ with $\varphi = cl_1 \wedge \ldots \wedge cl_m$, we construct the 1-in-3 Q3SAT instance $\mathcal{Q}' : \varphi'$ as follows. For each clause $cl_i = (\ell_1^i \vee \ell_2^i \vee \ell_3^i) \in \varphi$, we introduce five 1-in-3 Q3SAT clauses $\mathit{one\text{-}in\text{-}three}(cl_i) := (\ell_1^i \vee a_i \vee d_i) \wedge (\ell_2^i \vee b_i \vee d_i) \wedge (a_i \vee b_i \vee e_i) \wedge (c_i \vee d_i \vee f_i) \wedge (\ell_3^i \vee c_i \vee \texttt{false})$ with six fresh Boolean variables $a_i, b_i, c_i, d_i, e_i, f_i$ and the constant literal `false` that is never satisfied. It holds that cl_i is satisfied under τ if and only if $\exists a_i, b_i, c_i, d_i, e_i, f_i : \mathit{one\text{-}in\text{-}three}(cl_i)$ is 1-in-3 satisfied under τ. By setting $\mathcal{Q}' := \mathcal{Q} \odot \exists a_1, b_1, \ldots, e_m, f_m$ and $\varphi' := \bigwedge_{i=1}^{m} \mathit{one\text{-}in\text{-}three}(cl_i)$, it follows that $\mathcal{Q} : \varphi$ is true if and only if $\mathcal{Q}' : \varphi'$ is *1-in-3 true*, i.e. true under 1-in-3 satisfaction. Note that $\mathcal{Q}' : \varphi'$ is of size linear in $Q : \varphi$, as $\mathcal{Q}' : \varphi'$ contains $6m$ new variables and $5m$ clauses.

Theorem 4.1
S2SAT *is* PSPACE-*complete.*

Proof. PSPACE-membership is obvious as SSAT lies in PSPACE. We prove PSPACE-hardness by a polynomial-time, actually a linear-time, many-one reduction from 1-in-3 Q3SAT.

Let $\mathcal{Q} : \varphi$ be a 1-in-3 Q3SAT instance. We construct an S2SAT instance (Φ, θ) such that $\mathcal{Q} : \varphi$ is 1-in-3 true if and only if $Pr(\Phi) \geq \theta$. First observe that $\mathcal{Q} : \varphi$ is 1-in-3 true if and only if $Pr(\mathcal{Q}' : \varphi) \geq 1$ under 1-in-3 satisfaction where $\mathcal{Q}'$ arises from $\mathcal{Q}$ by replacing all universal quantifiers by randomized ones $\text{Я}^{0.5}$. Let be $\varphi = \{(\ell_1^1 \vee \ell_2^1 \vee \ell_3^1), \ldots, (\ell_1^m \vee \ell_2^m \vee \ell_3^m)\}$. Now, we introduce $3m$ fresh randomized variables (three randomized variables per clause) all with the same probability $p = 0.9$ resulting in the prefix $\mathcal{Q}'' := \mathcal{Q}' \odot \text{Я}^{0.9} r_1^1, r_2^1, r_3^1, \ldots, r_1^m, r_2^m, r_3^m$ of Φ. The following propositional formula ψ of Φ ensures that *at most* one literal per clause in φ is `true`. To also enforce that *at least* one literal per clause is `true`, the probability threshold θ is set correspondingly by taking account of the probabilities of the randomized variables $\text{Я}^{0.9} r_j^i$ to rule out non-solutions of φ.

$$\psi := \bigwedge_{i=1}^{m} \left(\overbrace{\left(\bigwedge_{j=1}^{3} (\ell_j^i \vee \neg r_j^i) \wedge (neg(\ell_j^i) \vee r_j^i) \right)}^{\text{identify value of literal } \ell_j^i \text{ with variable } r_j^i} \quad \overbrace{\begin{matrix} \wedge(\neg r_1^i \vee \neg r_2^i) \\ \wedge(\neg r_1^i \vee \neg r_3^i) \\ \wedge(\neg r_2^i \vee \neg r_3^i) \end{matrix}}^{\text{at most one } \texttt{true} \text{ literal per clause}} \right)$$

where $neg(\ell)$ returns the opposite literal of ℓ, i.e. it returns x if $\ell = \neg x$, and $\neg x$ otherwise. Note that ψ is in 2CNF and Φ is of size linear in $\mathcal{Q} : \varphi$, since Φ contains $3m$ new variables and $9m$ clauses.

We now show that $Pr(\mathcal{Q}' : \varphi) \geq 1$ under 1-in-3 satisfaction if and only if $Pr(\Phi) \geq 0.009^m$. Let be $\mathcal{Q}' = Q_1 x_1 \ldots Q_n x_n$. Note that under each assignment τ to the variables in $\mathcal{Q}'$ there exists a unique assignment τ' to the randomized variables r_1^i, r_2^i, r_3^i such that the combined assignment τ'' to all variables in $\mathcal{Q}''$ with $\tau''(x_i) = \tau(x_i)$ and $\tau''(r_j^i) = \tau'(r_j^i)$ satisfies $\bigwedge_{j=1}^{3} (\ell_j^i \vee \neg r_j^i) \wedge (neg(\ell_j^i) \vee r_j^i)$ in ψ for each $1 \leq i \leq m$, i.e. at least one of

$$\begin{aligned}&Pr(\text{\rotatebox[origin=c]{180}{R}}^{0.9} r_{k+1}, \ldots, r_{3m} : \psi[\tau(x_1)/x_1] \ldots [\tau(x_n)/x_n][\tau'(r_1)/r_1] \ldots [\tau'(r_{k-1})/r_{k-1}][\mathtt{true}/r_k]) \ , \\ &Pr(\text{\rotatebox[origin=c]{180}{R}}^{0.9} r_{k+1}, \ldots, r_{3m} : \psi[\tau(x_1)/x_1] \ldots [\tau(x_n)/x_n][\tau'(r_1)/r_1] \ldots [\tau'(r_{k-1})/r_{k-1}][\mathtt{false}/r_k])\end{aligned}$$

is 0 for each $1 \leq k \leq 3m$. Due to $(\neg r_1^i \vee \neg r_2^i) \wedge (\neg r_1^i \vee \neg r_3^i) \wedge (\neg r_2^i \vee \neg r_3^i) \in \psi$ and because of setting r_j^i to `true` with probability 0.9 and to `false` with 0.1, for each assignment τ to the variables in $\mathcal{Q}'$ it holds that $Pr(\text{\rotatebox[origin=c]{180}{R}}^{0.9} r_1^1, \ldots, r_3^m : \psi[\tau(x_1)/x_1] \ldots [\tau(x_n)/x_n]) \leq 0.009^m$.

Furthermore, for each assignment τ to the variables in $\mathcal{Q}'$ that 1-in-3 satisfies φ, i.e. $Pr(\varepsilon : \varphi[\tau(x_1)/x_1] \ldots [\tau(x_n)/x_n]) \geq 1$ under 1-in-3 satisfaction, the unique assignment τ' also satisfies $(\neg r_1^i \vee \neg r_2^i) \wedge (\neg r_1^i \vee \neg r_3^i) \wedge (\neg r_2^i \vee \neg r_3^i)$ for each $1 \leq i \leq m$, since each clause in φ has exactly one true literal under τ, and thus τ'' satisfies ψ. Therefore, for each $1 \leq i \leq m$ exactly one variable of r_1^i, r_2^i, r_3^i is set to `true` by τ', from which follows that $Pr(\text{\rotatebox[origin=c]{180}{R}}^{0.9} r_1^1, \ldots, r_3^m : \psi[\tau(x_1)/x_1] \ldots [\tau(x_n)/x_n]) = 0.009^m$. Vice versa, if for some assignment τ to the variables in $\mathcal{Q}'$ it holds that $Pr(\text{\rotatebox[origin=c]{180}{R}}^{0.9} r_1^1, \ldots, r_3^m : \psi[\tau(x_1)/x_1] \ldots [\tau(x_n)/x_n]) = 0.009^m$ then for each $1 \leq i \leq m$ exactly one variable of r_1^i, r_2^i, r_3^i is set to `true` by τ' due to $(\neg r_1^i \vee \neg r_2^i) \wedge (\neg r_1^i \vee \neg r_3^i) \wedge (\neg r_2^i \vee \neg r_3^i)$ and due to $\text{\rotatebox[origin=c]{180}{R}}^{0.9} r_j^i$. From $\bigwedge_{j=1}^{3} (\ell_j^i \vee \neg r_j^i) \wedge (\neg \ell_j^i \vee r_j^i)$, we conclude that each clause in φ has exactly one true literal under τ. Thus, $Pr(\varepsilon : \varphi[\tau(x_1)/x_1] \ldots [\tau(x_n)/x_n]) \geq 1$ under 1-in-3 satisfaction. Summarizing, $Pr(\varepsilon : \varphi[\tau(x_1)/x_1] \ldots [\tau(x_n)/x_n]) \geq 1$ under 1-in-3 satisfaction if and only if $Pr(\text{\rotatebox[origin=c]{180}{R}}^{0.9} r_1^1, \ldots, r_3^m : \psi[\tau(x_1)/x_1] \ldots [\tau(x_n)/x_n]) = 0.009^m$. From this fact and due to $Pr(\text{\rotatebox[origin=c]{180}{R}}^{0.9} r_1^1, \ldots, r_3^m : \psi[\tau(x_1)/x_1] \ldots [\tau(x_n)/x_n]) \leq 0.009^m$ for each τ, it immediately follows by definition that $Pr(\mathcal{Q}' : \varphi) \geq 1$ under 1-in-3 satisfaction if and only if $Pr(\Phi) = 0.009^m$ if and only if $Pr(\Phi) \geq 0.009^m$. To complete the reduction, we choose the rational constant $\theta := 0.009^m$.

The resulting S2SAT instance (Φ, θ) contains $n + 3m$ variables and $9m$ clauses where n is the number of variables and m is the number of clauses in $\mathcal{Q} : \varphi$. The rational constant $\theta = 0.009^m$ can be represented by a decimal fraction of size $\mathcal{O}(m)$. Thus, (Φ, θ) can be constructed in linear time. □

We remark that S2SAT with just *homogeneous* probabilities in randomized quantifiers, i.e. $\text{\rotatebox[origin=c]{180}{R}}^{0.5}$, is of the same complexity while the proof is slightly more complex. In brief, $3m$ more randomized variables are appended to $\mathcal{Q}''$ (on the right) and ψ is extended by $3m$ more clauses, i.e. $\text{\rotatebox[origin=c]{180}{R}}^{0.5} h_1^i, h_2^i, h_3^i$ and $(r_1^i \vee \neg h_1^i) \wedge (r_2^i \vee \neg h_2^i) \wedge (r_3^i \vee \neg h_3^i)$ per clause c_i. Then, if $r_j^i =$ `true` both assignments to h_j^i satisfy $(r_j^i \vee \neg h_j^i)$, and otherwise, i.e. $r_j^i =$ `false`, just $h_j^i =$ `false` does. Thus, for each i one of r_1^i, r_2^i, r_3^i is `true` if and only if the corresponding probability is 0.5^5, and all r_j^i are `false` if and only if the probability is 0.5^6. It remains to set $\theta := 0.5^{5m}$.

4.3 Satisfiability modulo theories

As mentioned in Section 4.1, the Boolean satisfiability problem has many practical applications, particularly in formal verification of software and hardware systems. Although modern SAT solvers are highly efficient tools to solve many industrial problems, system designs and their corresponding verification tasks become more and more intricate and often require logical frameworks that are more expressive than propositional logic. To meet these requirements, the Boolean satisfiability problem was extended by integrating *background theories*. The resulting notion is generally known as *satisfiability modulo theories*, or SMT for short. Some theories of interest are *equality logic with uninterpreted functions*, *arithmetic* like difference logic or linear arithmetic, the theories of *arrays*, *bit vectors*, and *inductive data types*. Being expressive enough to encode the behavior of probabilistic hybrid automata introduced in Section 3.3, we direct our attention in this section to the *theory of non-linear arithmetic* over the reals and integers involving transcendental functions like exponential and trigonometric functions. For more details on SMT for the above mentioned theories, the interested reader is referred to the nice survey [BSST09].

4.3.1 SMT for non-linear arithmetic

An *SMT formula with respect to the theory of non-linear arithmetic over the reals and integers* is an arbitrary quantifier-free Boolean combination of non-linear arithmetic constraints including transcendental functions. An example is given by the formula

$$\psi = \big((\sin(y^2) \le 0.1) \;\Rightarrow\; (x \le 0 \lor z > 3x + \exp(y))\big)$$

where $x \in \mathbb{Z}$, and $y, z \in \mathbb{R}$. In order to obviate the issue with undefined values of partial operations, we demand that all arithmetic operators in arithmetic constraints are total. Practically, this need not be a huge restriction as most common partial arithmetic operators can be expressed by their inverse operation. For instance, the constraint $y = 1/x$ in which the term $1/x$ is undefined for $x = 0$ can be rephrased as $y \cdot x = 1 \land x \neq 0$. We further remark that propositional literals b and $\neg b$ with b being a propositional variable can be encoded as $b' \geq 1$ and $b' \leq 0$, respectively, with b' being an integer variable with domain $\{0, 1\}$.

Semantically, non-linear arithmetic SMT formulae φ are interpreted over *assignments* $\tau \in (\mathit{Var}_{\mathbb{Z}}(\varphi) \to \mathbb{Z}) \times (\mathit{Var}_{\mathbb{R}}(\varphi) \to \mathbb{R})$ to their variables $\mathit{Var}(\varphi) = \mathit{Var}_{\mathbb{Z}}(\varphi) \cup \mathit{Var}_{\mathbb{R}}(\varphi)$, where $\mathit{Var}_{\mathbb{Z}}(\varphi)$ and $\mathit{Var}_{\mathbb{R}}(\varphi)$ denote the set of φ's integer and real-valued variables, respectively. Given such assignment $\tau = (\tau_{\mathbb{Z}}, \tau_{\mathbb{R}})$, we slightly abuse notation and identify $\tau(x) = \tau_{\mathbb{Z}}(x)$ if $x \in \mathit{Var}_{\mathbb{Z}}(\varphi)$ and $\tau(x) = \tau_{\mathbb{R}}(x)$ if $x \in \mathit{Var}_{\mathbb{R}}(\varphi)$. *Satisfaction of arithmetic constraints* under some assignment is with respect to the standard interpretation of the arithmetic operators and the ordering relations over the integers and reals. For instance, both constraints $\sin(y^2) \le 0.1$ and $z > 3x + \exp(y)$ are satisfied under assignment τ with $\tau(x) = 1$, $\tau(y) = 1.9$, and $\tau(z) = 24.3$ because $\sin(1.9^2) \le 0.1$ and $24.3 > 3 + \exp(1.9)$. The constraint $x \le 0$ is clearly not satisfied under above τ. If a constraint c is satisfied under some τ, we also say that c has truth value `true` under τ and otherwise, i.e. c is not satisfied under τ, c has truth value `false` under τ. Please note that this semantics is well-defined since all arithmetic operators in a constraint are total. That is, each arithmetic term evaluates to a defined value in the reals or integers under each assignment and, thus, each arithmetic constraint has a definite truth value under each assignment. *Satisfaction of a non-linear arithmetic SMT formula* under some assignment is then based on the standard interpretation of the logical operators, confer Section 2.2. For instance, the disjunction $(x \leq 0 \vee z > 3x + \exp(y))$ is satisfied under above assignment τ because one constraint, namely $z > 3x + \exp(y)$, is satisfied under τ. As a consequence, the whole formula ψ above is satisfied under τ. If an SMT formula φ is satisfied under an assignment τ, denoted by $\tau \models \varphi$, then τ is called *satisfying assignment* (or *solution* or *model*) of φ. An SMT formula φ is *satisfiable* if and only if there is a satisfying assignment of φ. If no solution of φ exists, φ is *unsatisfiable*.

Undecidability. The satisfiability problem for SMT formulae with respect to the theory of non-linear arithmetic, i.e. the problem of deciding whether a given non-linear arithmetic SMT formula is satisfiable or not, is undecidable in general. This is due to the fact that non-linear *Diophantine equations*, i.e. equations between polynomials in several integer variables, can be encoded in non-linear SMT since the latter supports addition and multiplication over integer variables. The problem of deciding whether a Diophantine equation has an integer solution or not, also known as *Hilbert's Tenth Problem*, was proven to be undecidable by Matiyasevich [Mat70]. We remark that non-linear SMT that allows just real-valued variables is also undecidable since a model of the integer numbers can be filtered out from the reals by exploiting the periodicity of trigonometric functions. Despite the fact of general undecidability, necessarily incomplete algorithms addressing non-linear arithmetic SMT problems were developed. In Section 6.3, we describe such an SMT algorithm, more precisely the so-called iSAT algorithm [FHT$^+$07, Her10], that constitutes the algorithmic basis of the SSMT solver introduced in Section 6.4.

Conjunctive form. With regard to the development of such SMT solving algorithms, it is common to deal with formulae of syntactically restricted shape. Similar to propositional formulae in conjunctive normal form, we rewrite an arbitrary SMT formula as above into a *conjunction* of clauses where *clauses* are *disjunctions* of primitive constraints. A *primitive constraint* is either a *simple bound* consisting of one variable, one relational operator, and one rational constant like $x = 3.1$ or $z < -12.8$, or it is an *arithmetic equation* containing up to three variables and one arithmetic operation like $x = y + z$ or $x = \sin(y)$. SMT formulae of the above shape are called to be in *conjunctive form* or CF for short.

In [Her10, Chapter 5], Herde presented a linear-time procedure to convert an arbitrary non-linear arithmetic SMT formula into an equi-satisfiable formula in CF. This procedure is a generalized version of the Tseitin transformation [Tse68], the latter being applied to obtain propositional formulae in CNF. In brief, the generalized Tseitin transformation for non-linear arithmetic SMT formulae is based on introducing fresh auxiliary variables for the values of arithmetic subexpressions and of logical subformulae. It furthermore supports optimizations like the elimination of common subexpressions and common subformulae through reuse of the auxiliary variables. An important property of this generalized Tseitin transformation is the following: given any SMT formula φ, it computes an SMT formula φ' in CF such that $\varphi \equiv \exists h_1, \ldots, h_n : \varphi'$ where $h_1, \ldots, h_n$ are the introduced auxiliary variables. Considering the SMT formula ψ above, an equi-satisfiable SMT formula in CF, for instance, is

$$\begin{aligned}
&(h_{\sin(y^2)} > 0.1 \ \vee \ x \le 0 \ \vee \ h_{z-3x-\exp(y)} > 0) \\
&\wedge \ (h_{\sin(y^2)} = \sin(h_{y^2})) \ \wedge \ (h_{y^2} = y^2) \ \wedge \ (h_{z-3x-\exp(y)} = z - h_{3x-\exp(y)}) \\
&\wedge \ (h_{3x-\exp(y)} = h_{3x} - h_{\exp(y)}) \ \wedge \ (h_{3x} = 3x) \ \wedge \ (h_{\exp(y)} = \exp(y))
\end{aligned}$$

with the auxiliary variables $h_{\sin(y^2)}, h_{y^2}, h_{z-3x-\exp(y)}, h_{3x-\exp(y)}, h_{\exp(y)} \in \mathbb{R}$, and $h_{3x} \in \mathbb{Z}$.

The formal syntax of an SMT formula in CF with respect to the theory of non-linear arithmetic is specified in the following definition.

Definition 4.3 (Syntax of non-linear arithmetic SMT formulae in CF)
SMT formulae in conjunctive form (CF) with respect to the theory of non-linear arithmetic over the reals and integers *are formed according to the following grammar:*

$$\begin{aligned}
\mathit{smt\_formula} &::= \{\mathit{clause} \wedge\}^* \mathit{clause} \\
\mathit{clause} &::= (\{\mathit{constraint} \vee\}^* \mathit{constraint}) \\
\mathit{constraint} &::= \mathit{bound} \ \mid \ \mathit{equation} \\
\mathit{bound} &::= \mathit{var} \ \mathit{relop} \ \mathit{const} \\
\mathit{equation} &::= \mathit{var} \ = \ \mathit{term} \\
\mathit{term} &::= \mathit{uop} \ \mathit{var} \ \mid \ \mathit{var} \ \mathit{bop} \ \mathit{var} \\
\mathit{relop} &::= \ < \mid \le \mid = \mid \ge \mid > \\
\mathit{uop} &::= - \ \mid \ \sin \ \mid \ \cos \ \mid \ \exp \ \mid \ \mathrm{abs} \ \mid \ \ldots \\
\mathit{bop} &::= + \ \mid \ - \ \mid \ \cdot \ \mid \ \ldots
\end{aligned}$$

where var denotes a real-valued or integer variable, and const ranges over the rational constants.

From the general semantics it follows that an SMT formula φ in CF is satisfied under an assignment τ if and only if at least one constraint is satisfied under τ in each clause of φ.

4.4 Stochastic satisfiability modulo theories

This section presents the logical framework of *stochastic satisfiability modulo theories*, or SSMT for short, that we have first introduced in [FHT08]. Roughly speaking, SSMT combines the concepts of SSAT, described in Section 4.2, and SMT, dealt with in Section 4.3, and thus enhances the reasoning power of SMT to probabilistic logics. In several publications, we have investigated different definitions of SSMT. In very general terms, an SSMT formula Φ can be viewed as an SMT formula φ over some theory $\mathcal{T}$, while φ is preceded by a quantifier prefix $\mathcal{Q}$ comprising some of the variables in $\mathit{Var}(\varphi)$, i.e. $\Phi = \mathcal{Q} : \varphi$.

In the original paper [FHT08], we required that theory $\mathcal{T}$ is *decidable* and that prefix $\mathcal{Q}$ includes only *existential* and *randomized* variables over finite domains. The requirement of decidability of $\mathcal{T}$ was relaxed in [TF08] where we considered *non-linear arithmetic over the reals and integers.* The latter definition of SSMT establishes the fundamental basis of the symbolic analysis procedure for discrete-time probabilistic hybrid systems being investigated in Chapter 5. In addition to existential and randomized quantification, the SSMT version of [TF09, TEF11] also permits *universal* quantifiers in $\mathcal{Q}$ akin to XSSAT. Expressiveness of SSMT has been enhanced considerably in [FTE10a] by two major extensions: first, by reasoning over *ordinary differential equations* (ODEs) as an additional theory and, second, by *existential* quantification over *continuous-domain* variables. The latter version of SSMT then allows for the symbolic analysis of *continuous-time* probabilistic hybrid systems. We elaborate on this issue in Chapter 10. While all of the above papers define the semantics of SSMT by the maximum probability of satisfaction as for SSAT, we have generalized the interpretation of SSMT to *maximum conditional expectation of a designated variable* in [FTE10b]. Being able to deal with such expectations in SSMT, the scope of the probabilistic reachability analysis approach of Chapter 5 has been extended to the computation of *expected values* of probabilistic hybrid systems like, for instance, mean time to failure. A comprehensive elaboration on this topic is given in Chapter 7.

In Subsection 4.4.1, we characterize the notion of SSMT as described in [TF08], i.e. SSMT for the theory of non-linear arithmetic over the reals and integers involving transcendental functions. As mentioned above, this definition of SSMT serves as the fundamental basis of the symbolic approach to bounded reachability analysis of probabilistic hybrid systems being introduced in Chapter 5. In Subsection 4.4.2, we then suggest an extension of SSMT which provides stronger capabilities in problem modeling with a view to improving performance of SSMT algorithms.

4.4.1 Syntax and semantics

In contrast to SSAT, quantified variables in SSMT need not range over the Boolean domain but over arbitrary finite domains as in SCSP. We thus write $Qx \in \mathcal{D}_x$ to denote that variable x over finite domain $\mathcal{D}_x$ is bound by quantifier Q. Without loss of generality, we demand that $\mathcal{D}_x$ is given by a set of integers, since each finite domain can be encoded using the integers. We may moreover assume–again without loss of generality–that $\mathcal{D}_x$ can be represented by an integer interval. The latter can be achieved, for instance, by taking *successive* integers for the encoding of the finite domain $\mathcal{D}_x$. A quantifier Q, associated with variable x, is either *existential*, denoted as $\exists$, or *randomized*, denoted as ꓤ_{d_x} where d_x is a discrete probability distribution over $\mathcal{D}_x$. A variable x is called *existential* or *randomized* variable if x is bound by an existential quantifier, i.e. $\exists x \in \mathcal{D}_x$, or by a randomized quantifier, i.e. $\text{ꓤ}_{d_x} x \in \mathcal{D}_x$, respectively. Similar to SSAT, the value of a randomized variable is determined stochastically according to the corresponding distribution, while the value of an existential variable can be set arbitrarily. We denote a probability distribution d_x by a function $[v_1 \to p_1, \ldots, v_m \to p_m]$ with $\mathcal{D}_x = \{v_1, \ldots, v_m\}$ associating probability $0 < p_i \leq 1$ to value v_i. The mapping $v_i \to p_i$ is understood as p_i is the probability of setting variable x to value v_i. The distribution satisfies $v_i \neq v_j$ for $i \neq j$ and $\sum_{i=1}^{m} p_i = 1$. For instance, $\text{ꓤ}_{[-1 \to 0.2, 0 \to 0.5, 1 \to 0.3]} x \in \{-1, 0, 1\}$ expresses that the variable x is assigned the values -1, 0, and 1 with probabilities 0.2, 0.5, and 0.3, respectively.

The formal definition of the syntax and semantics of SSMT is as follows.

Definition 4.4 (Syntax of SSMT)
A stochastic satisfiability modulo theories (SSMT) *formula* Φ *is of the form* $\mathcal{Q} : \varphi$ *where*

$$\Phi = \exists x \in \{0,1\}\ \text{\reflectbox{R}}_{[0\to 0.3, 1\to 0.6, 2\to 0.1]} y \in \{0,1,2\} : \\ \big((x \geq 1 \vee 2a \cdot \sin(4b) \geq 3) \wedge (y \geq 2 \vee 2a \cdot \sin(4b) < 1) \wedge (x \leq y)\big)$$

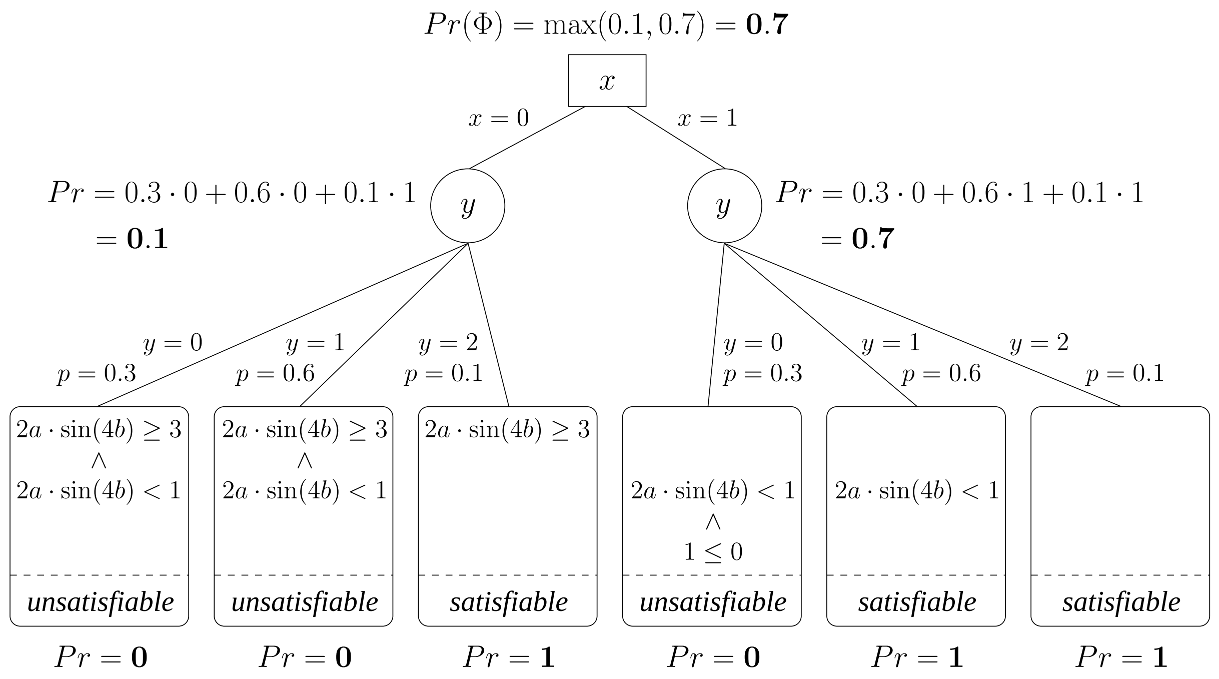


Figure 4.3: Semantics of an SSMT formula Φ depicted as a tree illustrating the recursive descent through the quantifier prefix.

1. *φ is an arbitrary SMT formula with respect to the theory of non-linear arithmetic over the reals and integers, and*
2. *$\mathcal{Q} = Q_1 x_1 \in \mathcal{D}_{x_1} \odot \ldots \odot Q_n x_n \in \mathcal{D}_{x_n}$ is a* quantifier prefix *binding some variables $x_i \in Var(\varphi)$ over finite domains $\mathcal{D}_{x_i}$ by existential and randomized quantifiers Q_i.*

The quantifier-free SMT formula φ is sometimes called the matrix *of Φ.*

Observe that not all variables of matrix φ need to be quantified by prefix $\mathcal{Q}$ and that non-quantified variables may range over continuous domains. These non-quantified variables are interpreted as innermost existentially quantified by Definition 4.5.

Definition 4.5 (Semantics of SSMT)
The semantics of an SSMT formula Φ is given by its maximum probability of satisfaction *$Pr(\Phi)$ defined as follows:*

$$\begin{aligned} Pr(\varepsilon : \varphi) &= \begin{cases} 0 & \textit{if}\ \ \varphi \textit{ is unsatisfiable}, \\ 1 & \textit{if}\ \ \varphi \textit{ is satisfiable}, \end{cases} \\ Pr(\exists x \in \mathcal{D}_x \odot \mathcal{Q} : \varphi) &= \max\nolimits_{v \in \mathcal{D}_x} Pr(\mathcal{Q} : \varphi[v/x]) , \\ Pr(\text{\reflectbox{R}}_{d_x} x \in \mathcal{D}_x \odot \mathcal{Q} : \varphi) &= \textstyle\sum_{v \in \mathcal{D}_x} d_x(v) \cdot Pr(\mathcal{Q} : \varphi[v/x]) , \end{aligned}$$

where ε denotes the empty and $\mathcal{Q}$ an arbitrary quantifier prefix.

Definition 4.5 is an extension of the semantics of SSAT, confer Definition 4.2. While the interpretation of quantifiers remains the same as for SSAT, their treatment is adapted to handle domains with more than two values. That is, the maximum probability of satisfaction $Pr(\Phi)$ of an SSMT formula Φ with a leftmost existential quantifier in the prefix, i.e. $\Phi = \exists x \in \mathcal{D}_x \odot \mathcal{Q} : \varphi$, is defined as the *maximum* of the satisfaction probabilities of all subformulae $\mathcal{Q} : \varphi[v/x]$ that arise by removing the leftmost quantified variable from the prefix and by substituting values $v \in \mathcal{D}_x$ for variable x in the matrix φ. If the leftmost variable is randomized, i.e. $\Phi = \text{Ⅎ}_{d_x} x \in \mathcal{D}_x \odot \mathcal{Q} : \varphi$, then $Pr(\Phi)$ demands to compute the *weighted sum* of the satisfaction probabilities of all subformulae $\mathcal{Q} : \varphi[v/x]$. The base cases of this recursion, that are reached whenever the quantifier prefix becomes empty, yield SMT formulae over the non-quantified variables. This fact differs from SSAT where all variables are quantified and each base case thus gives a formula equivalent to either `true` or `false`. Being conform with the intuition of the maximum probability of satisfaction, we assign satisfaction probability 1 to the remaining quantifier-free SMT formula φ in case φ is satisfiable, and probability 0 otherwise, i.e. if φ is unsatisfiable. Therefore, the non-quantified variables of an SSMT formula can be seen as innermost existentially quantified.

Intuitively, the above recursive process spans a tree in which the inner nodes represent quantified variables, the edges encode assignments to the quantified variables, and the leaves identify quantifier-free SMT formulae. For an example see Figure 4.3.

Without loss of generality, we may assume that the matrix of an SSMT formula is in conjunctive form. The rationale is as follows. Let $\Phi = \mathcal{Q} : \varphi$ be any SSMT formula. Then, we can apply the generalized Tseitin transformation for SMT formulae, confer Subsection 4.3.1, to rewrite matrix φ into an equi-satisfiable SMT formula φ' in CF such that $\varphi \equiv \exists h_1, \ldots, h_m : \varphi'$ with $h_1, \ldots, h_m$ being the introduced auxiliary variables. From Definition 4.5 it then follows that $Pr(\mathcal{Q} : \varphi) = Pr(\mathcal{Q} : \varphi')$.

4.4.2 Extension of SSMT involving dependent probability distributions

With regard to SSMT solving algorithms, which are dealt with in Chapter 6, it is beneficial to reduce the potential search space of SSMT problems in order to improve performance. One direction to attain this objective is to devise powerful algorithmic enhancements and heuristics. Such latter optimizations are investigated in Section 6.5. Another approach being as important is to provide SSMT encodings of the problems to be solved that are eminently suitable for the SSMT procedure employed. As the SSMT solving approach of this thesis, confer Chapter 6, is based on an explicit traversal through the tree spanned by the quantifier prefix, as indicated by Figure 4.3, it seems reasonable to favor SSMT encodings inducing *smaller* quantifier trees.

In what follows, we suggest an extended notion of the SSMT framework that supports such problem encodings reducing the potential search space. For a motivating example, let us assume that we have modeled some problem as an SSMT formula

$$\Phi \quad = \quad Qx \in \mathcal{D}_x \; \text{Ⅎ}_{d_{y_1}} y_1 \in \mathcal{D}_{y_1} \; \text{Ⅎ}_{d_{y_2}} y_2 \in \mathcal{D}_{y_2} : ((x > 0 \;\Rightarrow\; \varphi_1) \;\wedge\; (x \leq 0 \;\Rightarrow\; \varphi_2))$$

where the randomized variables y_1 and y_2 occur only in φ_1 and in φ_2, respectively, i.e.

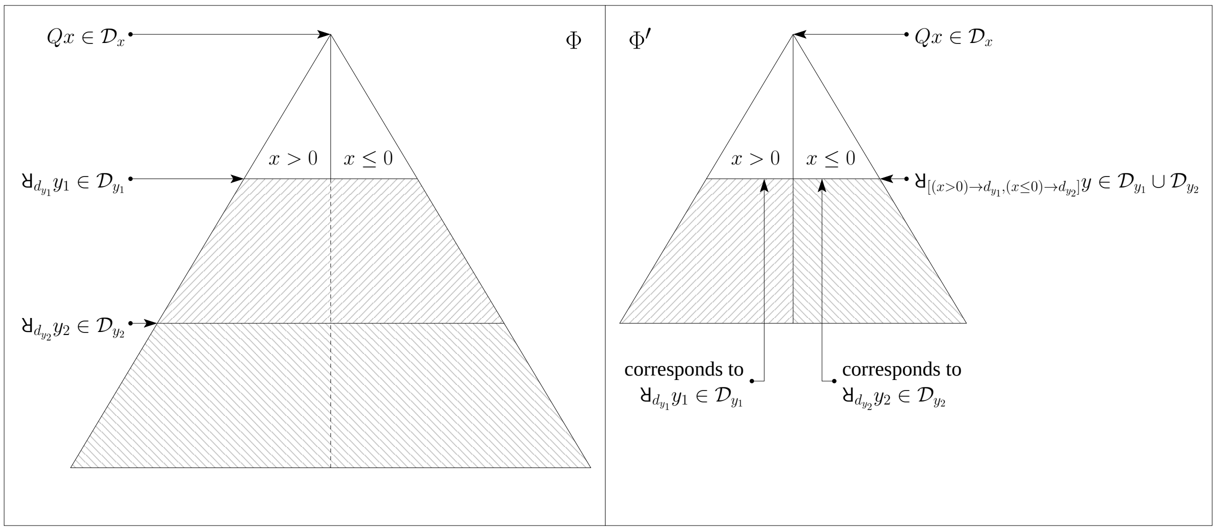

$Qx \in \mathcal{D}_x$
Φ
$x > 0$
$x \leq 0$
$\text{ꓤ}_{d_{y_1}} y_1 \in \mathcal{D}_{y_1}$
$\text{ꓤ}_{d_{y_2}} y_2 \in \mathcal{D}_{y_2}$
Φ'
$Qx \in \mathcal{D}_x$
$x > 0$
$x \leq 0$
$\text{ꓤ}_{[(x>0)\to d_{y_1},(x\leq 0)\to d_{y_2}]} y \in \mathcal{D}_{y_1} \cup \mathcal{D}_{y_2}$
corresponds to $\text{ꓤ}_{d_{y_1}} y_1 \in \mathcal{D}_{y_1}$
corresponds to $\text{ꓤ}_{d_{y_2}} y_2 \in \mathcal{D}_{y_2}$

Figure 4.4: Reduction of the potential search space by means of SSMT involving dependent probability distributions: illustration of the tree spanned by the quantifier prefix of the "classical" SSMT formula Φ (left) and the reduced tree for Φ' using a randomized quantifier with dependent probability distributions (right).

$y_1 \in \mathit{Var}(\varphi_1)$, $y_2 \notin \mathit{Var}(\varphi_1)$ and $y_1 \notin \mathit{Var}(\varphi_2)$, $y_2 \in \mathit{Var}(\varphi_2)$. That is, if variable x takes a value greater 0 then randomized variable y_2 becomes unnecessary since, first, y_2 does not occur in predicate $(x > 0 \Rightarrow \varphi_1)$ and, second, predicate $(x \leq 0 \Rightarrow \varphi_2)$ is trivially satisfied under each value of y_2. The same holds for randomized variable y_1 whenever x carries a value at most 0. Such circumstances may often arise in practice, for instance, in applications where random phenomena are triggered only in certain system states or where the probability distributions vary in different system states.

Taking the above observation into account, we aim at the possibility of "disabling" certain quantified variables. Observe that the latter can be simply achieved for existential variables x, namely by adding a corresponding predicate to the formula that fixes a value for x whenever the truth value of the remaining formula does not depend on x. This treatment is sound since existential variables call for maximizing the satisfaction probability. The same approach however is infeasible for randomized variables in general as this would lead to incorrect probability results, more precisely, to results that are too small. Our solution to this issue is as follows: we enable randomized quantifiers to carry several probability distributions such that exactly one of them will be activated once all preceding quantified variables are assigned. The selection of the distribution then depends on predicates over the preceding quantified variables with the semantic condition that exactly one of these predicates holds under each assignment to the preceding variables.

Before formally introducing the extended notion of SSMT, we exemplify this concept using the example above. In order to "disable" randomized variable y_2 and y_1 if $x > 0$ and $x \leq 0$ holds, respectively, we first merge the quantifiers $\text{ꓤ}_{d_{y_1}} y_1 \in \mathcal{D}_{y_1}$ and $\text{ꓤ}_{d_{y_2}} y_2 \in \mathcal{D}_{y_2}$ to a single one using the idea of *dependent probability distributions*, namely to $\text{ꓤ}_{[(x>0)\to d_{y_1},(x\leq 0)\to d_{y_2}]} y \in \mathcal{D}_{y_1} \cup \mathcal{D}_{y_2}$. The latter expresses that distribution d_{y_1} is selected if $x > 0$ and d_{y_2} otherwise. Second, we replace each occurrence of y_1 in φ_1 and of y_2 in

φ_2 by y resulting in φ_1' and φ_2', respectively. Then, the SSMT formula

$$\Phi' \;=\; Qx \in \mathcal{D}_x \; ᖇ_{[(x>0)\to d_{y_1},(x\leq 0)\to d_{y_2}]} y \in \mathcal{D}_{y_1} \cup \mathcal{D}_{y_2} : ((x > 0 \;\Rightarrow\; \varphi_1') \;\wedge\; (x \leq 0 \;\Rightarrow\; \varphi_2'))$$

characterizes the same problem as Φ does but reduces the potential search space, as illustrated in Figure 4.4.

The formal definition of the syntax and semantics of SSMT involving randomized quantifiers with dependent probability distributions is as follows.

Definition 4.6 (Syntax of SSMT involving dependent distributions)
An SSMT formula involving dependent probability distributions *is an SSMT formula $\mathcal{Q} : \varphi$ where each randomized quantifier in $\mathcal{Q}$ however is of the form $ᖇ_{[c_1\to d_1,\ldots,c_m\to d_m]}$ such that for each $(ᖇ_{[c_1\to d_1,\ldots,c_m\to d_m]} x \in \mathcal{D}_x) \in \mathcal{Q}$ where*

$$\mathcal{Q} \;=\; Q_1 x_1 \in \mathcal{D}_{x_1} \odot \ldots \odot Q_i x_i \in \mathcal{D}_{x_i} \odot ᖇ_{[c_1\to d_1,\ldots,c_m\to d_m]} x \in \mathcal{D}_x \odot \mathcal{Q}' : \varphi$$

the following conditions are satisfied:

1. *each c_j with $j \in \{1,\ldots,m\}$ is a predicate over variables $x_1,\ldots,x_i$,*
2. *for each assignment τ to variables $x_1,\ldots,x_i$, exactly one of the predicates $c_1,\ldots,c_m$ is satisfied, i.e. $\exists j \in \{1,\ldots,m\} : \tau \models c_j$ and $\forall k \neq j : \tau \not\models c_k$, and*
3. *each d_j with $j \in \{1,\ldots,m\}$ is a probability distribution, denoted by a function $[v_1 \to p_1,\ldots,v_t \to p_t]$, with $\{v_1,\ldots,v_t\} \subseteq \mathcal{D}_x$ associating probability $0 < p_k \leq 1$ to value v_k and satisfying $v_k \neq v_{k'}$ for $k \neq k'$ and $\sum_{k=1}^{t} p_k = 1$.*

Observe that a distribution d_j in $ᖇ_{[c_1\to d_1,\ldots,c_m\to d_m]} x \in \mathcal{D}_x$ may be a *partial* function, i.e. $d_j(v)$ is *not* necessarily defined for all values $v \in \mathcal{D}_x$. This is just of technical nature, namely to avoid probabilities 0 in distributions d_j. An alternative definition may enforce that each d_j is *total*, i.e. defined for all values $v \in \mathcal{D}_x$, but should then permit that $d_j(v)$ can be 0 for some $v \in \mathcal{D}_x$.

We remark that a "classical" randomized quantifier $ᖇ_{d_x} x \in \mathcal{D}_x$ can be simply represented by a randomized quantifier involving dependent probability distributions, namely by $ᖇ_{[\texttt{true}\to d_x]} x \in \mathcal{D}_x$. The latter fact becomes clear from the semantics which follows next.

Definition 4.7 (Semantics of SSMT involving dependent distributions)
The semantics of an SSMT formula Φ involving dependent probability distributions is given by its maximum probability of satisfaction $Pr(\Phi)$ *defined as follows:*

$$\begin{aligned}
Pr(\varepsilon : \varphi) &= \begin{cases} 0 & \textit{if} \;\; \varphi \textit{ is unsatisfiable} \;, \\ 1 & \textit{if} \;\; \varphi \textit{ is satisfiable} \;, \end{cases} \\
Pr(\exists x \in \mathcal{D}_x \odot \mathcal{Q} : \varphi) &= \max_{v\in\mathcal{D}_x} Pr(\mathcal{Q}[v/x] : \varphi[v/x]) \;, \\
Pr(ᖇ_{[c_1\to d_1,\ldots,c_m\to d_m]} x \in \mathcal{D}_x \odot \mathcal{Q} : \varphi) &= \textstyle\sum_{(v\to p)\in d_j \textit{ with } c_j\equiv\texttt{true}} p \cdot Pr(\mathcal{Q}[v/x] : \varphi[v/x]) \;,
\end{aligned}$$

where ε denotes the empty and $\mathcal{Q}$ an arbitrary quantifier prefix.

Note that $\mathcal{Q}[v/x]$ substitutes value v for variable x in prefix $\mathcal{Q}$ such that all variables in the predicates $c_1, \ldots, c_m$ have been substituted in a leftmost randomized quantifier $\rotatebox[origin=c]{180}{R}_{[c_1 \to d_1, \ldots, c_m \to d_m]}$.

It is important to remark that SSMT involving dependent probability distributions, as formalized in Definition 4.6, does *not* establish the basic concept of this thesis. The following chapters essentially build upon the notion of SSMT from Definition 4.4. A pragmatic use case of SSMT involving dependent probability distributions however is investigated in Sections 6.6 and 6.7 and furthermore exploited in Chapter 8, the latter dealing with the analysis of the NAS case study introduced in Section 3.1 and depicted in Figure 3.1.

5 SSMT-Based Bounded Reachability Analysis of Probabilistic Hybrid Automata

After having introduced the formal model of concurrent discrete-time probabilistic hybrid automata in Chapter 3 and after having explained the logical framework of SSMT in Chapter 4, this chapter is devoted to the analysis of concurrent PHAs and is mainly based on the work described in [TEF11]. We remark that parts of this chapter were published in [TEF11] by the author of this thesis together with his co-authors.

We start our presentation with the formal definition of the analysis problem, namely probabilistic bounded state reachability, in Section 5.1. As already sketched in Section 3.2, our symbolic analysis approach to probabilistic reachability is based on a translation of the original problem to an SSMT formula. The latter is then solved by an appropriate SSMT algorithm being introduced in Chapter 6. The reduction to SSMT is first illustrated by an introductory example in Section 5.2 and thereafter formally introduced in Section 5.3.

5.1 Probabilistic bounded state reachability

In what follows, let $\mathcal{S} = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ be a system of concurrent discrete-time probabilistic hybrid automata as in Definition 3.1, and *Target* be a predicate, defining the set of *target states* by means of all its models, in the arithmetic theory $\mathcal{T}$ over the discrete and continuous variables in $D_1, \ldots, D_n$ and $R_1, \ldots, R_n$ of all the automata. With respect to system analysis, we are interested in the probability of reaching the target states within a bounded number of transition steps. As usual in models blending non-deterministic and probabilistic choices, like Markov decision processes [Bel57], we assume that the dynamics is controlled by a *decision maker* or *scheduler (policy, adversary)* resolving the non-determinism based on (complete) observation of the current state and history. Based on the system behavior exhibited so far, such a scheduler can decide which global transition should be executed next by system $\mathcal{S}$. We allow a rather general notion and assume that these decisions depend deterministically on the current run prefix, i.e. that the permissible schedulers are history-dependent and deterministic, confer, for instance, [BHKH05].

Definition 5.1 (History-dependent, deterministic scheduler)
Let $\mathcal{R}_\mathcal{S}$ denote the set of all (finite) runs of $\mathcal{S}$. Then, a history-dependent, deterministic scheduler *$\sigma : \mathcal{R}_\mathcal{S} \to NChoice$ for $\mathcal{S}$ maps a run to a global transition choice.*

We call a run $\langle s_0, (tr_1, pc_1), s_1, \ldots, (tr_k, pc_k), s_k\rangle \in \mathcal{R}_\mathcal{S}$ of $\mathcal{S}$ *consistent with scheduler σ* if $\sigma(\langle s_0\rangle) = tr_1$ and for each $1 \leq i < k$ it holds that $\sigma(\langle s_0, \ldots, (tr_i, pc_i), s_i\rangle) = tr_{i+1}$. A run $\langle s_0, (tr_1, pc_1), s_1, \ldots, (tr_k, pc_k), s_k\rangle \in \mathcal{R}_\mathcal{S}$ *hits the target states* if there is at least one state in the run that satisfies predicate *Target*, i.e. $\exists i \in \{0, \dots, k\} : s_i \models \mathit{Target}$. We are now able to define the probabilistic bounded reachability problem with respect to some scheduler:

Definition 5.2 (Scheduler-dependent probabilistic bounded reachability)
Let $k \in \mathbb{N}$ be a step bound and σ be a history-dependent, deterministic scheduler for $\mathcal{S}$. For each state $s \in \mathit{States}_{\mathcal{S}} \cup \{\bot\}$, let $\mathcal{R}^k_{\mathcal{S},\sigma,\mathit{Target}}(s)$ denote the set of all runs $r \in \mathcal{R}_{\mathcal{S}}$ such that

- *r starts in s, i.e. $\mathit{first}(r) = s$,*
- *r is of length k, i.e. $\mathit{length}(r) = k$,*
- *r is consistent with σ, and*
- *r hits the target states.*

Then, the probability of reaching the target states within k steps under scheduler σ *is given by $\mathcal{P}^k_{\mathcal{S},\sigma,\mathit{Target}}(\imath)$ with $\imath \models \bigwedge_{i=1}^{n} \mathit{init}_i$ being the (unique) initial state of $\mathcal{S}$ and $\mathcal{P}^k_{\mathcal{S},\sigma,\mathit{Target}}(s)$ for $s \in \mathit{States}_{\mathcal{S}} \cup \{\bot\}$ being defined as follows:*

$$\mathcal{P}^k_{\mathcal{S},\sigma,\mathit{Target}}(s) = \sum\nolimits_{r \in \mathcal{R}^k_{\mathcal{S},\sigma,\mathit{Target}}(s)} p(r)$$

where $p(r)$ denotes the probability of run r.

The above definition can be characterized recursively as follows:

Lemma 5.1 (Recursive characterization)
Let $k \in \mathbb{N}$ be a step bound, σ be a history-dependent, deterministic scheduler for $\mathcal{S}$, and $\imath$ be the (unique) initial state of $\mathcal{S}$, i.e. $\imath \models \bigwedge_{i=1}^{n} \mathit{init}_i$. Then, it holds that

$$\mathcal{P}^k_{\mathcal{S},\sigma,\mathit{Target}}(\imath) = P^k_{\mathcal{S},\sigma,\mathit{Target}}(\langle \imath \rangle)$$

where $P^k_{\mathcal{S},\sigma,\mathit{Target}}(r)$ with $r \in \mathcal{R}_{\mathcal{S}}$ is defined as follows:

$$P^k_{\mathcal{S},\sigma,\mathit{Target}}(r) = \begin{cases} 1 & \text{if} \quad \mathit{last}(r) \models \mathit{Target}\ , \\ 0 & \text{if} \quad \mathit{last}(r) \not\models \mathit{Target} \text{ and } k = 0\ , \\ \sum_{pc \in \mathit{PChoice}(tr)} p(tr, pc) \cdot P^{k-1}_{\mathcal{S},\sigma,\mathit{Target}}(r \odot \langle (tr, pc), s' \rangle) & \\ \quad\quad\quad \text{if} \quad \mathit{last}(r) \not\models \mathit{Target} \text{ and } k > 0\ , \end{cases}$$

with $tr = \sigma(r)$ being the transition scheduled by σ and $s' = \mathit{Post}(\mathit{last}(r), tr, pc)$ being the corresponding successor state. Recall that $\bot$ does not satisfy any $\mathcal{T}$-predicate, in particular $\bot \not\models \mathit{Target}$.

Proof. By induction over k, we show that for all runs $r \in \mathcal{R}_{\mathcal{S}}$ the following holds:

$$\mathcal{P}^k_{\mathcal{S},\rho_r,\mathit{Target}}(\mathit{last}(r)) = P^k_{\mathcal{S},\sigma,\mathit{Target}}(r)$$

where the history-dependent, deterministic scheduler ρ_r is defined as follows. Let be $r = \langle s_0, (tr_1, pc_1), s_1, \dots, (tr_i, pc_i), s_i \rangle$. For each run $r' \in \mathcal{R}_{\mathcal{S}}$ with $\mathit{first}(r') = \mathit{last}(r) = s_i$, we define $\rho_r(r') := \sigma(\langle s_0, (tr_1, pc_1), s_1, \ldots, (tr_i, pc_i)\rangle \odot r')$. The lemma then follows directly from the special case $r = \langle \imath \rangle$, since here $\rho_r(r') = \sigma(r')$ for all runs $r' \in \mathcal{R}_\mathcal{S}$ starting in the initial state $\imath$.

Observe that the result obviously holds whenever $\mathit{last}(r) \models \mathit{Target}$. In this case, $\mathcal{R}^k_{\mathcal{S},\rho_r,\mathit{Target}}(\mathit{last}(r))$ contains all runs of length k that start in $\mathit{last}(r)$ and are consistent with ρ_r. The accumulated probability $\mathcal{P}^k_{\mathcal{S},\rho_r,\mathit{Target}}(\mathit{last}(r))$ is therefore 1. Immediately by definition, $P^k_{\mathcal{S},\sigma,\mathit{Target}}(r) = 1$. We thus assume in the remaining proof that $\mathit{last}(r) \not\models \mathit{Target}$.

For the base case, let be $k = 0$. As $\mathit{last}(r) \not\models \mathit{Target}$, it follows that $\mathcal{R}^0_{\mathcal{S},\rho_r,\mathit{Target}}(\mathit{last}(r)) = \emptyset$ and thus $\mathcal{P}^0_{\mathcal{S},\rho_r,\mathit{Target}}(\mathit{last}(r)) = 0$, and, immediately by definition, $P^0_{\mathcal{S},\sigma,\mathit{Target}}(r) = 0$.

For the induction step, let be $k \geq 0$. We need to conclude that

$$\mathcal{P}^{k+1}_{\mathcal{S},\rho_r,\mathit{Target}}(\mathit{last}(r)) = P^{k+1}_{\mathcal{S},\sigma,\mathit{Target}}(r)$$

for all runs $r \in \mathcal{R}_\mathcal{S}$ follows from induction hypothesis, i.e. from

$$\mathcal{P}^k_{\mathcal{S},\rho_{r'},\mathit{Target}}(\mathit{last}(r')) = P^k_{\mathcal{S},\sigma,\mathit{Target}}(r')$$

for all runs $r' \in \mathcal{R}_\mathcal{S}$.

In what follows, let be $r \in \mathcal{R}_\mathcal{S}$, $\sigma(r) = tr$ and thus $\rho_r(\langle \mathit{last}(r)\rangle) = tr$, and further

$$r' = \langle s'_0, (tr'_1, pc'_1), s'_1, \ldots, (tr'_{k+1}, pc'_{k+1}), s'_{k+1}\rangle \in \mathcal{R}^{k+1}_{\mathcal{S},\rho_r,\mathit{Target}}(\mathit{last}(r)) \quad .$$

We denote by $pc_1(r')$ the first probabilistic choice in r', i.e. $pc_1(r') = pc'_1$. For each such r' above, it holds that $s'_0 = \mathit{last}(r)$, $tr'_1 = tr$, and $pc_1(r') \in \mathit{PChoice}(tr)$. Using these properties, we conclude

$$\begin{aligned}
\mathcal{P}^{k+1}_{\mathcal{S},\rho_r,\mathit{Target}}(\mathit{last}(r)) &= \sum_{r' \in \mathcal{R}^{k+1}_{\mathcal{S},\rho_r,\mathit{Target}}(s'_0)} p(r') \\
&= \sum_{pc \in \mathit{PChoice}(tr)} \quad \sum_{r' \in \mathcal{R}^{k+1}_{\mathcal{S},\rho_r,\mathit{Target}}(s'_0) \text{ with } pc_1(r') = pc} p(r') \\
&= \sum_{pc \in \mathit{PChoice}(tr)} \left(p(tr, pc) \cdot \sum_{r' \in \mathcal{R}^k_{\mathcal{S},\rho',\mathit{Target}}(\mathit{Post}(s'_0, tr, pc))} p(r') \right)
\end{aligned}$$

with $\rho' = \rho_{r \odot \langle (tr,pc), \mathit{Post}(s'_0,tr,pc)\rangle}$ being given by $\rho'(r'') := \sigma(r \odot \langle (tr, pc)\rangle \odot r'')$ for all $r'' \in \mathcal{R}_\mathcal{S}$ with $\mathit{first}(r'') = \mathit{Post}(s'_0, tr, pc)$. By definition and induction hypothesis:

$$\begin{aligned}
\mathcal{P}^{k+1}_{\mathcal{S},\rho,\mathit{Target}}(\mathit{last}(r)) &= \sum_{pc \in \mathit{PChoice}(tr)} p(tr, pc) \cdot \mathcal{P}^k_{\mathcal{S},\rho',\mathit{Target}}(\mathit{Post}(s'_0, tr, pc)) \\
&= \sum_{pc \in \mathit{PChoice}(tr)} p(tr, pc) \cdot P^k_{\mathcal{S},\sigma,\mathit{Target}}(r \odot \langle (tr, pc), \mathit{Post}(s'_0, tr, pc)\rangle) \\
&= P^{k+1}_{\mathcal{S},\sigma,\mathit{Target}}(r)
\end{aligned}$$

For the last step, which applies the definition of $P^{k+1}_{\mathcal{S},\sigma,\mathit{Target}}(r)$, recall that $\mathit{last}(r) \not\models \mathit{Target}$ and $k + 1 > 0$. This completes the proof. □

In the sequel, we are interested in the *maximum* probability of reaching a given set of target states under an *arbitrary* scheduler within a given number $k \in \mathbb{N}$ of transition steps. Semantically, this is adequate for modeling and analyzing situations where the target states are considered undesirable and a demonic perspective to non-determinism is taken (rendering the scheduler adversarial), or symmetrically to cases where the target states are considered desirable and an angelic perspective (rendering the scheduler cooperative) is taken. In particular, depth-bounded probabilistic reachability in probabilistic hybrid systems is representative for a number of verification problems for embedded systems, for instance

- performing quantitative safety analysis (in the sense of estimating failure probability) of a conflict resolution scheme which is expected to terminate after a finite number of actions whenever triggered, like collision avoidance maneuvers in road traffic,
- performing quantitative safety analysis (in the sense of estimating failure probability) of a finite critical mission, like the descent of an airplane,
- assessing the reliability of a system subject to regular maintenance, where the number of system actions between maintenance is bounded by a constant k, or
- step-bounded region stability[1] of hybrid systems subject to probabilistic disturbances, i.e. determining whether a system will with sufficient probability converge into a target region, which is assumed to be stable, within a given step (and thus, time) bound.

The following definition formalizes this maximum bounded reachability probability as the maximum over arbitrary schedulers of the scheduler-dependent reachability probability.

Definition 5.3 (Probabilistic bounded reachability)
Let $k \in \mathbb{N}$ be a step bound, $\imath \models \bigwedge_{i=1}^{n} init_i$ be the (unique) initial state of $\mathcal{S}$, and Υ be the set of all history-dependent, deterministic schedulers for $\mathcal{S}$. Then, the maximum probability of reaching the target states within k steps *is defined by*

$$\mathcal{P}^k_{\mathcal{S},\,Target}(\imath) = \max_{\sigma \in \Upsilon} \mathcal{P}^k_{\mathcal{S},\sigma,\,Target}(\imath).$$

Similar to Lemma 5.1, we may characterize above notion in a recursive manner.

Lemma 5.2 (Recursive characterization of probabilistic bounded reachability)
Let $k \in \mathbb{N}$ be a step bound and $\imath \models \bigwedge_{i=1}^{n} init_i$ be the (unique) initial state of $\mathcal{S}$. Then, it holds that

$$\mathcal{P}^k_{\mathcal{S},\,Target}(\imath) = P^k_{\mathcal{S},\,Target}(\imath)$$

[1]Note that eventual stability, while frequently considered due to its simpler mathematics, is hardly ever a convincing notion in practice. In most practical applications, bounds on stabilization time are desirable.

where $P^k_{\mathcal{S},\mathit{Target}}(s)$ *with* $s \in \mathit{States}_{\mathcal{S}} \cup \{\bot\}$ *is defined as follows:*

$$P^k_{\mathcal{S},\mathit{Target}}(s) = \begin{cases} 1 & \text{if} \quad s \models \mathit{Target}\ , \\ 0 & \text{if} \quad s \not\models \mathit{Target} \text{ and } k = 0\ , \\ \max\limits_{tr \in \mathit{NChoice}} \sum\limits_{pc \in \mathit{PChoice}(tr)} p(tr, pc) \cdot P^{k-1}_{\mathcal{S},\mathit{Target}}(\mathit{Post}(s, tr, pc)) & \\ & \text{if} \quad s \not\models \mathit{Target} \text{ and } k > 0\ . \end{cases}$$

Proof. By Definition 5.3 and due to Lemma 5.1, it holds that

$$\mathcal{P}^k_{\mathcal{S},\mathit{Target}}(\imath) = \max_{\sigma\in\Upsilon} \mathcal{P}^k_{\mathcal{S},\sigma,\mathit{Target}}(\imath) = \max_{\sigma\in\Upsilon} P^k_{\mathcal{S},\sigma,\mathit{Target}}(\langle\imath\rangle).$$

It therefore suffices to show that

$$\max_{\sigma\in\Upsilon} P^k_{\mathcal{S},\sigma,\mathit{Target}}(r) = P^k_{\mathcal{S},\mathit{Target}}(\mathit{last}(r))$$

is true for each run $r \in \mathcal{R}_{\mathcal{S}}$. The lemma then follows directly from the special case $r = \langle\imath\rangle$.

First observe that the result obviously holds whenever $\mathit{last}(r) \models \mathit{Target}$. In this case, it follows immediately by definitions that $P^k_{\mathcal{S},\sigma,\mathit{Target}}(r) = 1$ for each scheduler σ and $P^k_{\mathcal{S},\mathit{Target}}(\mathit{last}(r)) = 1$. We thus assume in the remaining proof that $\mathit{last}(r) \not\models \mathit{Target}$.

The proof is done via induction over step depth $k \in \mathbb{N}$. The base case is given by $k = 0$. As $\mathit{last}(r) \not\models \mathit{Target}$, clearly by definition: $P^0_{\mathcal{S},\sigma,\mathit{Target}}(r) = 0$ for each scheduler σ and $P^0_{\mathcal{S},\mathit{Target}}(\mathit{last}(r)) = 0$.

For the induction step, we assume that above statement holds for $k \geq 0$. We need to show that

$$\max_{\sigma\in\Upsilon} P^{k+1}_{\mathcal{S},\sigma,\mathit{Target}}(r) = P^{k+1}_{\mathcal{S},\mathit{Target}}(\mathit{last}(r))$$

follows from induction hypothesis for each but fixed run $r \in \mathcal{R}_{\mathcal{S}}$. Since $\mathit{last}(r) \not\models \mathit{Target}$ and $k + 1 > 0$, application of definition yields

$$\max_{\sigma\in\Upsilon} P^{k+1}_{\mathcal{S},\sigma,\mathit{Target}}(r) = \max_{\sigma\in\Upsilon} \left(\sum_{pc \in \mathit{PChoice}(\sigma(r))} p(\sigma(r), pc) \cdot P^k_{\mathcal{S},\sigma,\mathit{Target}}(r \odot \langle(\sigma(r), pc), s'\rangle) \right)$$

with $s' = \mathit{Post}(\mathit{last}(r), \sigma(r), pc)$ being the corresponding successor state. We now prove that

$$\max_{\sigma\in\Upsilon} P^{k+1}_{\mathcal{S},\sigma,\mathit{Target}}(r) = \max_{\sigma\in\Upsilon} \left(\sum_{pc \in \mathit{PChoice}(\sigma(r))} p(\sigma(r), pc) \cdot \max_{\sigma'\in\Upsilon} \left(P^k_{\mathcal{S},\sigma',\mathit{Target}}(r \odot \langle(\sigma(r), pc), s'\rangle) \right) \right)$$

is true. It is not hard to see that the right-hand side is always greater than or equal to the left-hand side. The proof that the right-hand side is also less than or equal to $\max_{\sigma\in\Upsilon} P^{k+1}_{\mathcal{S},\sigma,\mathit{Target}}(r)$ is by contradiction: we first fix two schedulers ρ and ρ' such that

$$\begin{aligned} \max_{\sigma\in\Upsilon} P^{k+1}_{\mathcal{S},\sigma,\mathit{Target}}(r) &= P^{k+1}_{\mathcal{S},\rho,\mathit{Target}}(r) \quad \text{, and} \\ \max_{\sigma'\in\Upsilon} \left(P^k_{\mathcal{S},\sigma',\mathit{Target}}(r \odot \langle(\rho(r), pc), s''\rangle) \right) &= P^k_{\mathcal{S},\rho',\mathit{Target}}(r \odot \langle(\rho(r), pc), s''\rangle) \end{aligned}$$

with $s'' = Post(last(r), \rho(r), pc)$. Then,

$$\max_{\sigma\in\Upsilon} P^{k+1}_{\mathcal{S},\sigma,Target}(r) = \sum_{pc\in PChoice(\rho(r))} p(\rho(r), pc) \cdot P^{k}_{\mathcal{S},\rho,Target}(r \odot \langle(\rho(r), pc), s''\rangle) \enspace .$$

Observe that it suffices to show that

$$P^{k}_{\mathcal{S},\rho',Target}(r \odot \langle(\rho(r), pc), s''\rangle) \;\leq\; P^{k}_{\mathcal{S},\rho,Target}(r \odot \langle(\rho(r), pc), s''\rangle) \enspace .$$

Now assume the converse, i.e. $P^{k}_{\mathcal{S},\rho',Target}(r\odot\langle(\rho(r), pc), s''\rangle) > P^{k}_{\mathcal{S},\rho,Target}(r\odot\langle(\rho(r), pc), s''\rangle)$. Then, we can construct the scheduler ρ'' that works as ρ for the (fixed) run r, i.e. $\rho''(r) = \rho(r)$, and as ρ' for all extensions $r' = r\odot w \in \mathcal{R}_{\mathcal{S}}$ of r, i.e. $\rho''(r') = \rho'(r')$. As a consequence, $P^{k+1}_{\mathcal{S},\rho'',Target}(r) > P^{k+1}_{\mathcal{S},\rho,Target}(r)$. From the fact that $\max_{\sigma\in\Upsilon} P^{k+1}_{\mathcal{S},\sigma,Target}(r) \geq P^{k+1}_{\mathcal{S},\rho'',Target}(r)$ the contradiction follows, namely $\max_{\sigma\in\Upsilon} P^{k+1}_{\mathcal{S},\sigma,Target}(r) > P^{k+1}_{\mathcal{S},\rho,Target}(r)$.

Since $\sigma(r) \in NChoice$ for each scheduler σ, we trivially have

$$\max_{\sigma\in\Upsilon} P^{k+1}_{\mathcal{S},\sigma,Target}(r) \leq \max_{tr\in NChoice}\left(\sum_{pc\in PChoice(tr)} p(tr, pc) \cdot \max_{\sigma'\in\Upsilon}\left(P^{k}_{\mathcal{S},\sigma',Target}(r \odot \langle(tr, pc), s''\rangle)\right)\right)$$

with $s'' = Post(last(r), tr, pc)$. The inequality

$$\max_{\sigma\in\Upsilon} P^{k+1}_{\mathcal{S},\sigma,Target}(r) \geq \max_{tr\in NChoice}\left(\sum_{pc\in PChoice(tr)} p(tr, pc) \cdot \max_{\sigma'\in\Upsilon}\left(P^{k}_{\mathcal{S},\sigma',Target}(r \odot \langle(tr, pc), s''\rangle)\right)\right)$$

also holds, which is again proven by contradiction: as above, we first fix scheduler ρ with $\max_{\sigma\in\Upsilon} P^{k+1}_{\mathcal{S},\sigma,Target}(r) = P^{k+1}_{\mathcal{S},\rho,Target}(r)$. Now assume that there exists a $tr \in NChoice$ for which the value of the right-hand side is strictly greater than $P^{k+1}_{\mathcal{S},\rho,Target}(r)$. Clearly, $\rho(r) \neq tr$. We can construct a scheduler ρ' which is defined as ρ except for $\rho'(r) = tr$. As a consequence, $P^{k+1}_{\mathcal{S},\rho',Target}(r) > P^{k+1}_{\mathcal{S},\rho,Target}(r)$. From $\max_{\sigma\in\Upsilon} P^{k+1}_{\mathcal{S},\sigma,Target}(r) \geq P^{k+1}_{\mathcal{S},\rho',Target}(r)$ the contradiction follows, namely $\max_{\sigma\in\Upsilon} P^{k+1}_{\mathcal{S},\sigma,Target}(r) > P^{k+1}_{\mathcal{S},\rho,Target}(r)$.

Summarizing, we have shown that

$$\max_{\sigma\in\Upsilon} P^{k+1}_{\mathcal{S},\sigma,Target}(r) = \max_{tr\in NChoice}\left(\sum_{pc\in PChoice(tr)} p(tr, pc) \cdot \max_{\sigma'\in\Upsilon}\left(P^{k}_{\mathcal{S},\sigma',Target}(r \odot \langle(tr, pc), s''\rangle)\right)\right) .$$

Application of induction hypothesis yields

$$\max_{\sigma\in\Upsilon} P^{k+1}_{\mathcal{S},\sigma,Target}(r) = \max_{tr\in NChoice}\left(\sum_{pc\in PChoice(tr)} p(tr, pc) \cdot P^{k}_{\mathcal{S},Target}(s'')\right) .$$

Recall that $s'' = Post(last(r), tr, pc)$. Directly by definition, we finally conclude

$$\max_{\sigma\in\Upsilon} P^{k+1}_{\mathcal{S},\sigma,Target}(r) = P^{k+1}_{\mathcal{S},Target}(last(r))$$

which establishes the lemma. □

Lemma 5.2 actually shows that when considering *maximum* step-bounded reachability probabilities in concurrent PHAs then history-dependent schedulers are not more expressive than schedulers that depend only on the current state and step-depth. A similar result was shown in [BHKH05, Theorem 2] for maximum time-bounded reachability probabilities in continuous-time Markov decision processes.

We finally state the decision problem called *probabilistic bounded model checking* that is defined to be the problem of deciding whether the maximum probability of reaching the target states within a given number of steps is below a given threshold:

Definition 5.4 (Probabilistic bounded model checking)
Given a system $\mathcal{S}$ of n concurrent PHAs, a predicate Target defining the target states of $\mathcal{S}$, a step bound $k \in \mathbb{N}$, and a probability threshold $\theta \in [0,1]$, the probabilistic bounded model checking problem (PBMC) with respect to target states *Target*, step bound k, and threshold θ *is to decide whether $\mathcal{P}^k_{\mathcal{S},\mathit{Target}}(\imath) \leq \theta$ or, equivalently,*

$$P^k_{\mathcal{S},\mathit{Target}}(\imath) \leq \theta$$

hold with $\imath \models \bigwedge_{i=1}^{n} \mathit{init}_i$ being the (unique) initial state of $\mathcal{S}$.

After having formally introduced the notion of probabilistic bounded reachability for systems of concurrent PHAs, the remainder of this chapter is devoted to a *symbolic* procedure for solving probabilistic bounded model checking problems. In contrast to explicit-state approaches, which many approaches in the realm of hybrid systems belong to with respect to the discrete state space, confer Section 3.2, the predicative nature of the translation scheme to SSMT *avoids* the explicit construction of the product automaton that grows exponentially in the number of parallel components. This translation to SSMT proceeds in two phases. First, we generate the matrix of the SSMT formula, i.e. the quantifier-free SMT part. This matrix encodes all non-deadlocked, anchored runs of system $\mathcal{S}$ that reach the target states and are of the given length $k \in \mathbb{N}$. The exclusion of deadlocked runs simplifies the matrix and is justified by the fact that such runs have no contribution to the probability of reaching the target states due to $\perp \not\models \mathit{Target}$. Second, we add the quantifier prefix which encodes the non-deterministic and the probabilistic choices of the concurrent automata, whereby non-deterministic choices yield existential quantifiers and probabilistic choices reduce to randomized quantifiers.

Before formally presenting the details of this encoding scheme in Section 5.3, we first introduce the intuition by means of an example in Section 5.2.

5.2 Introductory example of the reduction to SSMT

We illustrate the SSMT encoding of concurrent PHAs by the simple example shown in Figure 5.1. For the sake of simplicity, we just consider a single probabilistic automaton consisting of only one location being described by the discrete variable $d \in [1,1]$, and of one continuous variable x. The *initial state* of this automaton is given by the predicate

$$\mathit{Init}(d,x) = (d = 1 \wedge x = 0).$$

Thus, the (unique) satisfying assignment of $\mathit{Init}(d,x)$ represents the (unique) initial state of the automaton. To perform a transition step, the automaton may non-deterministically

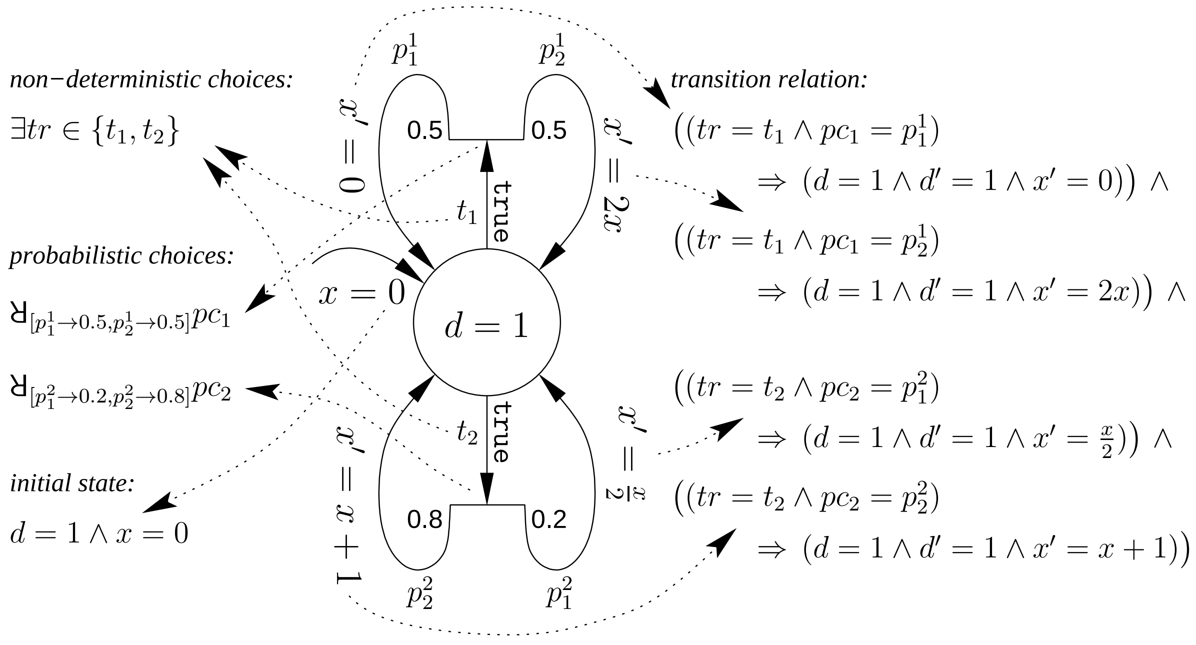

non−deterministic choices:
$\exists tr \in \{t_1, t_2\}$
probabilistic choices:
$\text{Я}_{[p_1^1 \to 0.5, p_2^1 \to 0.5]} pc_1$
$\text{Я}_{[p_1^2 \to 0.2, p_2^2 \to 0.8]} pc_2$
initial state:
$d = 1 \wedge x = 0$
p_1^1
p_2^1
0.5
0.5
$x' = 0$
$x' = 2x$
t_1
true
$x = 0$
$d = 1$
t_2
true
$x' = x + 1$
$x' = \frac{x}{2}$
0.8
0.2
p_2^2
p_1^2
transition relation:
$\big((tr = t_1 \wedge pc_1 = p_1^1)$
$\Rightarrow (d = 1 \wedge d' = 1 \wedge x' = 0)\big) \wedge$
$\big((tr = t_1 \wedge pc_1 = p_2^1)$
$\Rightarrow (d = 1 \wedge d' = 1 \wedge x' = 2x)\big) \wedge$
$\big((tr = t_2 \wedge pc_2 = p_1^2)$
$\Rightarrow (d = 1 \wedge d' = 1 \wedge x' = \frac{x}{2})\big) \wedge$
$\big((tr = t_2 \wedge pc_2 = p_2^2)$
$\Rightarrow (d = 1 \wedge d' = 1 \wedge x' = x + 1)\big)$

Figure 5.1: Example of the SSMT encoding scheme. Note that the domains of the randomized variables pc_1 and pc_2 are omitted for the sake of clarity. (This figure is a slight modification of Figure 5 from [TEF11].)

select either transition t_1 or t_2 since both transition guards `true` are trivially satisfied. As the definition of probabilistic bounded reachability (Definition 5.3) calls for *maximizing* the probability of reaching the target states, we need to select a transition for each step that maximizes the reachability probability according to Lemma 5.2. To do so, we encode the *non-deterministic* selection of transitions by *existential* quantification. In the example, we introduce an existentially quantified variable *tr* with a domain that consists of both transitions t_1 and t_2, i.e.

$$\exists tr \in \{t_1, t_2\}.$$

Transition selection is then followed by a probabilistic choice of transition alternatives. When taking transition t_1, one of alternatives p_1^1 and p_2^1 are executed with equal probability 0.5. In case t_2 was selected, alternative p_1^2 is performed with probability 0.2 and p_2^2 with probability 0.8. This *probabilistic* selection of transition alternatives is mapped to *randomized* quantification. In the example, we introduce two randomized variables pc_1 for the probabilistic choice after transition t_1, and pc_2 for t_2, i.e.

$$\text{Я}_{[p_1^1 \to 0.5, p_2^1 \to 0.5]} pc_1 \in \{p_1^1, p_2^1\} \quad \text{and} \quad \text{Я}_{[p_1^2 \to 0.2, p_2^2 \to 0.8]} pc_2 \in \{p_1^2, p_2^2\} \ .$$

By these quantified variables, we have described the non-deterministic choice of a transition and the probabilistic choice of a transition alternative for one step in the automaton.

In order to symbolically encode all anchored systems runs, we have to symbolically describe all possible transition steps in the automaton, i.e. the relation between the pre- and post-state for all transitions and their transition alternatives. If transition t_1 is selected non-deterministically and transition alternative p_2^1 probabilistically, then the automaton must currently be in location that is described by $d = 1$, re-enters this location, and doubles the value of variable x. This transition step is encoded by the predicate $(tr = t_1 \wedge pc_1 = p_2^1) \Rightarrow (d = 1 \wedge d' = 1 \wedge x' = 2x)$. The primed variables d' and x' represent the values of variables d and x after the system step, respectively. The encodings for the remaining transition steps are shown in Figure 5.1. By conjoining all these encodings by logical conjunction, we obtain the *transition relation* predicate

$$\mathit{Trans}(d, x, tr, pc_1, pc_2, d', x')$$

that describes all possible system steps from some state (d, x) under some non-deterministic choice tr and some probabilistic choices pc_1, pc_2 to state (d', x'). Due to Property 3.1, for fixed (d, x), tr, pc_1, and pc_2 the post-state (d', x') is unique if existent. Otherwise, i.e. (d', x') does not exist, the system deadlocks in the distinguished state $\perp$. In our encoding, we deal with deadlocking as follows: whenever the post-state (d', x') does not exist then the predicate $\mathit{Trans}(d, x, tr, pc_1, pc_2, d', x')$ becomes unsatisfiable, which actually means that all steps leading to $\perp$ are excluded. This treatment is sound since a target state will never be reached once the system has deadlocked.

As the analysis goal is probabilistic bounded state reachability, we furthermore need to take account of the predicate that specifies the target states. Let us be interested in reaching states in which the value of variable x exceeds 100. Then, the *target states* predicate is given by

$$\mathit{Target}(d, x) = (x > 100).$$

We now construct an SMT formula that encodes all anchored system runs of length k that reach the target states. Whenever a run r visits the target states in less than k steps, our encoding ensures that r remains in its current (target) state until step depth k is reached, i.e. target states are sinks of the transition relation. Formally, the SMT formula $\varphi(k)$ is given by

$$\begin{aligned}
& \mathit{Init}(d_0, x_0) \\
\wedge\ & \bigwedge_{j=1}^{k} \left(\begin{array}{l} (\neg \mathit{Target}(d_{j-1}, x_{j-1}) \Rightarrow \mathit{Trans}(d_{j-1}, x_{j-1}, tr_j, pc_{1,j}, pc_{2,j}, d_j, x_j)) \\ \wedge(\mathit{Target}(d_{j-1}, x_{j-1}) \Rightarrow (d_j = d_{j-1} \wedge x_j = x_{j-1})) \end{array} \right) \\
\wedge\ & \mathit{Target}(d_k, x_k)
\end{aligned}$$

where d_j, x_j are copies of the variables d, x encoding the system state after transition step j, and $tr_j, pc_{1,j}, pc_{2,j}$ are copies of tr, pc_1, pc_2 representing the non-deterministic and probabilistic choices of step j.

Taking into account the alternation of non-deterministic selections of transitions and probabilistic choices of transition alternatives for all k transition steps, we add the quantifier prefix

$$\begin{aligned}
& \exists tr_1 \in \{t_1, t_2\}\ \rotatebox[origin=c]{180}{R}_{[p_1^1 \to 0.5, p_2^1 \to 0.5]} pc_{1,1} \in \{p_1^1, p_2^1\}\ \rotatebox[origin=c]{180}{R}_{[p_1^2 \to 0.2, p_2^2 \to 0.8]} pc_{2,1} \in \{p_1^2, p_2^2\} \\
& \ldots \\
& \exists tr_k \in \{t_1, t_2\}\ \rotatebox[origin=c]{180}{R}_{[p_1^1 \to 0.5, p_2^1 \to 0.5]} pc_{1,k} \in \{p_1^1, p_2^1\}\ \rotatebox[origin=c]{180}{R}_{[p_1^2 \to 0.2, p_2^2 \to 0.8]} pc_{2,k} \in \{p_1^2, p_2^2\}
\end{aligned}$$

to the SMT formula $\varphi(k)$ yielding an SSMT formula $\Phi(k)$. Note that the prefix contains k copies of the quantified variables to represent all possible combinations of transitions and transition alternatives for k steps.

By the construction of the overall SSMT formula $\Phi(k)$, it follows an important observation that is formalized in Theorem 5.1 in the next Section 5.3: the maximum probability of satisfaction of $\Phi(k)$ coincides with the maximum probability of reaching the target states within k transition steps, i.e.

$$Pr(\Phi(k)) = P^{k}_{\mathcal{S},\mathit{Target}}(\imath)$$

where $\imath$ is the initial state of the given system $\mathcal{S}$.

To clarify the basic idea of the symbolic SSMT encoding, we have just illustrated the translation scheme for a single probabilistic hybrid automaton. Based on this translation scheme, we can now provide a compact intuition for the SSMT encoding of a *system of concurrent PHAs* before presenting the formalized approach in the next section. When considering a system of concurrently running probabilistic hybrid automata, we separately construct the transition relation predicate $\mathit{Trans}_i(\cdot)$ for each automaton $\mathcal{A}_i$. The conjunction of all these $\mathit{Trans}_i(\cdot)$ then gives the transition relation of the overall system. The latter is then used to obtain SMT formula $\varphi(k)$ as above. For the construction of the quantifier prefix, we need to pay attention to the order of the quantified variables. Before each transition step, all automata non-deterministically select local transitions synchronously. After having established consensus on a global transition, each automaton probabilistically selects one of the available alternatives. In the quantifier prefix, for each unwinding depth, we thus first compile the existential variables of all automata and thereafter the randomized ones. The quantifier prefix for k unwindings of the transition system is then composed by concatenating these quantifier prefixes in the same manner as was presented above for the single automaton.

5.3 Reducing probabilistic bounded reachability to SSMT

After having explained the intuition of encoding probabilistic bounded reachability for concurrent PHAs into the SSMT framework in the previous section, we now introduce the formalized reduction scheme. In what follows, let $\mathcal{S} = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ be a system of concurrent discrete-time probabilistic hybrid automata as in Definition 3.1, and *Target* be a predicate, defining the set of target states by means of all its models, in the arithmetic theory $\mathcal{T}$ over the discrete and continuous variables in $D_1, \ldots, D_n$ and $R_1, \ldots, R_n$ of all the automata. Furthermore, let $k \in \mathbb{N}$ be the bound on the length of the system runs.

The reduction to SSMT proceeds in two phases: we first generate the matrix of the SSMT formula encoding all non-deadlocked, anchored runs of system $\mathcal{S}$ of length k that reach the target states, and we second add the quantifier prefix representing the non-deterministic and the probabilistic choices of the concurrent automata by means of existential and randomized quantifiers.

Phase 1: Constructing the matrix. We start by constructing the matrix, that is denoted by $BMC_{\mathcal{S},\mathit{Target}}(k)$, of the resulting SSMT formula. As said above, each model of $BMC_{\mathcal{S},\mathit{Target}}(k)$ characterizes a non-deadlocked, anchored run of $\mathcal{S}$ of length k that reaches the target states. First of all, we need to declare the variables occurring in $BMC_{\mathcal{S},\mathit{Target}}(k)$

as well as their domains (reduction steps 1–4). We continue by symbolically encoding the global initial state, the transition relation, and the target states of $\mathcal{S}$ (reduction steps 5–9), while reduction step 10 finally states the matrix $BMC_{\mathcal{S},Target}(k)$.

Reduction step 1. For each *discrete variable* $d \in D_i$ of automaton $\mathcal{A}_i$ for $1 \leq i \leq n$, we take $k+1$ integer variables d_j for $0 \leq j \leq k$, each with the integer interval domain $\mathrm{dom}(d)$. An assignment to the variables $d^i_{1,j}, \ldots, d^i_{k_i,j}$ represents the discrete state of automaton $\mathcal{A}_i$ at depth j.

Reduction step 2. For each *continuous state component* $x \in R_i$ of $\mathcal{A}_i$ for $1 \leq i \leq n$, we take $k+1$ real-valued variables x_j for $0 \leq j \leq k$, each with real-valued interval domain $\mathrm{dom}(x)$. The value of x_j encodes the value of x at depth j.

Reduction step 3. For representing the symbolic *transitions* $tr \in Tr_i$ of $\mathcal{A}_i$, for $1 \leq i \leq n$, we take k variables tr^i_j with domain Tr_i, for $1 \leq j \leq k$. We demand here that domain Tr_i is encoded by a set of integers, for instance by the set of the indices $1, \ldots, \ell_i$ of the symbolic transitions, confer Definition 3.1. The value of tr^i_j encodes the transition selection of $\mathcal{A}_i$ at step j.

Reduction step 4. For representing the symbolic *probabilistic transition alternatives* in PC_{tr} for each transition $tr \in Tr_i$ of $\mathcal{A}_i$, for $1 \leq i \leq n$, we take k variables pc^{tr}_j with domain PC_{tr}, for $1 \leq j \leq k$. As for Tr_i, we again assume that PC_{tr} is given as a set of integers. The value of pc^{tr}_j encodes the transition alternative for transition tr of $\mathcal{A}_i$ at step j. It is important to note that the value of such a variable pc^{tr}_j will be irrelevant whenever the associated transition tr is not selected in step j, i.e. in case $tr^i_j \neq tr$.

Reduction step 5. The *initial state* of system $\mathcal{S}$ is encoded by the predicate

$$INIT_{\mathcal{S}}(0) := \bigwedge_{i=0}^{n} init_i[d^i_{1,0}, \ldots, d^i_{k_i,0}, x^i_{1,0}, \ldots, x^i_{m_i,0} / d^i_1, \ldots, d^i_{k_i}, x^i_1, \ldots, x^i_{m_i}]$$

where in $init_i$ each variable v is substituted by its representative v_0 at depth 0.

Reduction step 6. The synchronization conditions of local transitions tr, i.e. validity of the *generalized transition guards* $g(tr)$, for all automata $\mathcal{A}_i$ at step $1 \leq j \leq k$ are enforced through the constraint system

$$\bigwedge_{i=1}^{n} \bigwedge_{tr \in Tr_i} \left(\begin{array}{l} (tr^i_j = tr) \Rightarrow \\ g(tr)[\, d^1_{1,j-1}, d^1_{1,j}, \ldots, d^n_{k_n,j-1}, d^n_{k_n,j}, x^1_{1,j-1}, x^1_{1,j}, \ldots, x^n_{m_n,j-1}, x^n_{m_n,j} / \\ \qquad d^1_1, d'^1_1, \ldots, d^n_{k_n}, d'^n_{k_n}, x^1_1, x'^1_1, \ldots, x^n_{m_n}, x'^n_{m_n}] \end{array} \right)$$

where in transition guard $g(tr)$ each undecorated variable v is substituted by its representative v_{j-1} at depth $j-1$, and each primed variable v' is replaced by v_j for depth j.

Reduction step 7. Likewise, *assignments* $asgn(tr, pc)$ at step $1 \leq j \leq k$ triggered by transitions tr and probabilistic transition alternatives pc are dealt with by

$$\bigwedge_{i=1}^{n} \bigwedge_{tr \in Tr_i} \bigwedge_{pc \in PC_{tr}} \left(\begin{array}{l} (tr^i_j = tr \wedge pc^{tr}_j = pc) \Rightarrow \\ asgn(tr,pc)[d^1_{1,j-1}, d^1_{1,j}, \ldots, d^n_{k_n,j-1}, d^n_{k_n,j}, x^1_{1,j-1}, x^1_{1,j}, \ldots, x^n_{m_n,j-1}, x^n_{m_n,j} / \\ \qquad d^1_1, d'^1_1, \ldots, d^n_{k_n}, d'^n_{k_n}, x^1_1, x'^1_1, \ldots, x^n_{m_n}, x'^n_{m_n}] \end{array} \right)$$

where, as in previous reduction step 6, each undecorated variable v occurring in $asgn(tr, pc)$ is substituted by its representative v_{j-1} at depth $j-1$, and each primed variable v' of $asgn(tr, pc)$ is replaced by v_j for depth j.

Reduction step 8. The conjunction of the formulae of reduction steps 6 and 7 yields the *transition relation* predicate

$$\mathit{TRANS}_{\mathcal{S}}(j-1, j)$$

encoding a transition step from depth $j-1$ to j. Observe that with this predicative encoding, an infeasible choice of transitions and transition alternatives (for instance due to inconsistent assignment predicates) that would lead to the distinguished state $\perp$ in the semantics immediately causes unsatisfiability of the formula $\mathit{TRANS}_{\mathcal{S}}(j-1, j)$. That is, system runs reaching the deadlock state $\perp$ do not satisfy the matrix generated by our translation scheme and are thus excluded. Considering reachability of states satisfying *Target*, this handling is correct as runs entering $\perp$ will never reach any target state and, vice versa, runs reaching *Target* will never become deadlocked since both $\perp$ and target states being sinks. The latter fact, i.e. target states are sinks, is indicated by definition of $P^k_{\mathcal{S},\mathit{Target}}$ in Lemma 5.2 and enforced by formula $\mathit{BMC}_{\mathcal{S},\mathit{Target}}(k)$ in reduction step 10.

Reduction step 9. The next predicate denotes the *target states* for step depth $0 \le j \le k$.

$$\mathit{TARGET}(j) := \mathit{Target}[d^1_{1,j}, \ldots, d^n_{k_n,j}, x^1_{1,j}, \ldots, x^n_{m_n,j} / d^1_1, \ldots, d^n_{k_n}, x^1_1, \ldots, x^n_{m_n}]$$

Predicate $\mathit{TARGET}(j)$ is thus satisfied under an assignment τ if and only if the system state at depth j encoded by τ is a target state.

Reduction step 10. It remains to compile the matrix $\mathit{BMC}_{\mathcal{S},\mathit{Target}}(k)$ of the SSMT formula as follows:

$$\begin{array}{rcl} \mathit{BMC}_{\mathcal{S},\mathit{Target}}(k) & := & \mathit{INIT}_{\mathcal{S}}(0) \\ & \wedge & \bigwedge_{j=1}^{k} \left(\begin{array}{l} (\neg \mathit{TARGET}(j-1) \Rightarrow \mathit{TRANS}_{\mathcal{S}}(j-1, j)) \\ \wedge (\mathit{TARGET}(j-1) \Rightarrow \mathit{SELF\_LOOP}_{\mathcal{S}}(j-1, j)) \end{array} \right) \\ & \wedge & \mathit{TARGET}(k) \end{array}$$

where the predicate

$$\mathit{SELF\_LOOP}_{\mathcal{S}}(j-1, j) := \bigwedge_{i=1}^{n} \left(\begin{array}{rl} & d^i_{1,j} = d^i_{1,j-1} \wedge \ldots \wedge d^i_{k_i,j} = d^i_{k_i,j-1} \\ \wedge & x^i_{1,j} = x^i_{1,j-1} \wedge \ldots \wedge x^i_{m_i,j} = x^i_{m_i,j-1} \end{array} \right)$$

identifies the values of all variables of $\mathcal{S}$ at depth j with the values of the corresponding variables at depth $j-1$. The latter predicate thus encodes a stuttering system step which is independent of the non-deterministic and probabilistic selections of transitions and transition alternatives. Satisfying assignments of the quantifier-free formula $\mathit{BMC}_{\mathcal{S},\mathit{Target}}(k)$ are in one-to-one correspondence to the anchored and non-deadlocked runs of system $\mathcal{S}$ of length k that reach states satisfying the *Target* predicate. Whenever a target state is visited in less than k steps, the system remains in this target state until step depth k is reached due to $\mathit{SELF\_LOOP}_{\mathcal{S}}(j-1, j)$. This treatment ensures that all target states are sinks.

Phase 2: Constructing the prefix. To construct the prefix of the SSMT formula denoted by $PBMC_{\mathcal{S},Target}(k)$, we need to encode the non-deterministic selection of transitions by all concurrent automata followed by the probabilistic choice of transition alternatives. Since we aim at maximizing the probability of reaching the target states, the non-determinism is described by existential quantification in reduction step 11, while the probabilistic choices are mapped to randomized quantifiers in reduction step 12. Combining these quantifiers (reduction step 13) then leads to the final $PBMC_{\mathcal{S},Target}(k)$ formula in reduction step 14.

Reduction step 11. Before step j, $1 \leq j \leq k$, can be executed, each automaton $\mathcal{A}_i$ *non-deterministically* selects a transition. This is encoded by *existential* quantification of the transition variables tr^i_j introduced in reduction step 3.

$$NCHOICE_{\mathcal{S}}(j) := \bigodot_{i=1}^{n} \exists tr^i_j \in Tr_i$$

where $\bigodot$ denotes concatenation, confer Section 2.1.

Reduction step 12. Non-deterministic choice is followed by a *probabilistic* choice of a transition alternative for each automaton $\mathcal{A}_i$ before step j, $1 \leq j \leq k$. This is reflected by *randomized* quantification of the variables introduced by reduction step 4.

$$PCHOICE_{\mathcal{S}}(j) := \bigodot_{i=1}^{n} \bigodot_{tr \in Tr_i} \text{Ǝ}_{d_{tr}} pc^{tr}_j \in PC_{tr}$$

where d_{tr} encodes the discrete probability distribution $p(tr)$ and is syntactically represented as $[v_1 \to p(tr)(v_1), \ldots, v_m \to p(tr)(v_m)]$ with $PC_{tr} = \{v_1, \ldots, v_m\}$.

Reduction step 13. The combined quantifier sequence for a single computation step j, $1 \leq j \leq k$, is given by the existential quantifiers followed by the randomized ones.

$$CHOICE_{\mathcal{S}}(j) := NCHOICE_{\mathcal{S}}(j) \odot PCHOICE_{\mathcal{S}}(j)$$

Reduction step 14. Finally, we construct the SSMT formula $PBMC_{\mathcal{S},Target}(k)$ by concatenating the quantifier prefixes of the different computation steps in their natural sequence, representing the fact that the scheduler may draw decisions for later computation steps based on the outcomes of earlier ones, and by then adding the matrix representing anchored, non-deadlocked runs of the system reaching target states.

$$PBMC_{\mathcal{S},Target}(k) := \left(\bigodot_{j=1}^{k} CHOICE_{\mathcal{S}}(j) \right) : BMC_{\mathcal{S},Target}(k)$$

Given the structural similarity between probabilistic bounded reachability and quantification in SSMT, the above reduction is correct in the following sense.

Theorem 5.1 (Correctness of reduction)
Let be given a system $\mathcal{S}$ of n concurrent PHAs, a predicate Target defining the target states of $\mathcal{S}$, and a step bound $k \in \mathbb{N}$. Then, the maximum probability of reaching states satisfying Target within k steps coincides with the maximum probability of satisfaction of the symbolic encoding $PBMC_{\mathcal{S},Target}(k)$, i.e.

$$P^k_{\mathcal{S},Target}(\imath) = Pr(PBMC_{\mathcal{S},Target}(k))$$

holds with $\imath \models \bigwedge_{i=1}^n init_i$ being the (unique) initial state of $\mathcal{S}$.

Proof. Within this proof, let the predicate

$$val(z) := \bigwedge_{v \in \bigcup_{i=1}^n (D_i \cup R_i)} v = z(v),$$

be a symbolic encoding of a system state $z \in States_{\mathcal{S}}$, where $z(v)$ is the value of variable v in state z. The substitution of all variables in $val(z)$ by their representatives at depth j is abbreviated by

$$val(z,j) := val(z)[d^1_{1,j},\ldots,d^n_{k_n,j},x^1_{1,j},\ldots,x^n_{m_n,j}/d^1_1,\ldots,d^n_{k_n},x^1_1,\ldots,x^n_{m_n}].$$

To prove the theorem, we show that

$$P^k_{\mathcal{S},Target}(z) = Pr(PBMC'_{\mathcal{S},Target}(z,u,u+k)) \tag{5.1}$$

holds for each $k \in \mathbb{N}$, for each state $z \in States_{\mathcal{S}}$, and for each $u \in \mathbb{N}$ where

$$PBMC'_{\mathcal{S},Target}(z,u,u+k) := \left(\bigodot_{j=u+1}^{u+k} CHOICE_{\mathcal{S}}(j)\right) : BMC'_{\mathcal{S},Target}(z,u,u+k)$$

and

$$\begin{array}{rcl} BMC'_{\mathcal{S},Target}(z,u,u+k) & := & val(z,u) \\ & \wedge & \bigwedge_{j=u+1}^{u+k} \left(\begin{array}{l} (\neg TARGET(j-1) \Rightarrow TRANS_{\mathcal{S}}(j-1,j)) \\ \wedge (TARGET(j-1) \Rightarrow SELF\_LOOP_{\mathcal{S}}(j-1,j)) \end{array} \right) \\ & \wedge & TARGET(u+k). \end{array}$$

We remark that the introduction of $PBMC'_{\mathcal{S},Target}(z,u,u+k)$ in this very technical manner is necessary for a sound proof. The intuitive meaning, however, is rather simple: $PBMC'_{\mathcal{S},Target}(z,u,u+k)$ is similar to $PBMC_{\mathcal{S},Target}(k)$ but system runs encoded by $PBMC'_{\mathcal{S},Target}(z,u,u+k)$ start in state z and the representatives of the variables are indexed from u to $u+k$ instead of from 0 to k. This "starting index" u is actually redundant (but eases the proof) in the sense that for each $u,u' \in \mathbb{N}$ it obviously holds that

$$Pr(PBMC'_{\mathcal{S},Target}(z,u,u+k)) = Pr(PBMC'_{\mathcal{S},Target}(z,u',u'+k)) \ .$$

By construction and since initial state $\imath$ is unique, in special case $z = \imath$ and $u = 0$, the SMT formulae $BMC'_{\mathcal{S},Target}(\imath,0,k)$ and $BMC_{\mathcal{S},Target}(k)$ are semantically equivalent, i.e. $\mathit{BMC}'_{\mathcal{S},\mathit{Target}}(\imath,0,k) \equiv \mathit{BMC}_{\mathcal{S},\mathit{Target}}(k)$. As the prefixes of $\mathit{PBMC}'_{\mathcal{S},\mathit{Target}}(\imath,0,k)$ and $\mathit{PBMC}_{\mathcal{S},\mathit{Target}}(k)$ are equal, we thus have

$$\mathit{Pr}(\mathit{PBMC}'_{\mathcal{S},\mathit{Target}}(\imath,0,k)) = \mathit{Pr}(\mathit{PBMC}_{\mathcal{S},\mathit{Target}}(k)) \enspace .$$

In case equation 5.1 holds, we may conclude that $P^k_{\mathcal{S},\mathit{Target}}(\imath) = \mathit{Pr}(\mathit{PBMC}_{\mathcal{S},\mathit{Target}}(k))$ is true from which the theorem follows.

It thus remains to prove equation 5.1 which we do by induction over step depth k. For the base case, let be $k = 0$. By Lemma 5.2,

$$P^0_{\mathcal{S},\mathit{Target}}(z) = \begin{cases} 1 & \text{if } z \models \mathit{Target}, \\ 0 & \text{if } z \not\models \mathit{Target} \end{cases}$$

for each state $z \in \mathit{States}_{\mathcal{S}}$. Furthermore, $\mathit{PBMC}'_{\mathcal{S},\mathit{Target}}(z,u,u) = \mathit{BMC}'_{\mathcal{S},\mathit{Target}}(z,u,u)$ and $\mathit{BMC}'_{\mathcal{S},\mathit{Target}}(z,u,u) = \mathit{val}(z,u) \wedge \mathit{TARGET}(u)$. Observe that $\mathit{BMC}'_{\mathcal{S},\mathit{Target}}(z,u,u)$ is satisfiable if and only if z is a target state, i.e. if and only if $z \models \mathit{Target}$. Since $\mathit{Pr}(\mathit{PBMC}'_{\mathcal{S},\mathit{Target}}(z,u,u)) = \mathit{Pr}(\mathit{BMC}'_{\mathcal{S},\mathit{Target}}(z,u,u))$, it immediately follows from Definition 4.5 that

$$\mathit{Pr}(\mathit{PBMC}'_{\mathcal{S},\mathit{Target}}(z,u,u)) = \begin{cases} 1 & \text{if } z \models \mathit{Target}, \\ 0 & \text{if } z \not\models \mathit{Target} \end{cases}$$

holds for each state $z \in \mathit{States}_{\mathcal{S}}$ and for each $u \in \mathbb{N}$ which establishes the base case, i.e.

$$P^0_{\mathcal{S},\mathit{Target}}(z) = \mathit{Pr}(\mathit{PBMC}'_{\mathcal{S},\mathit{Target}}(z,u,u)) \enspace .$$

For the induction step, let be $k \geq 0$. As induction hypothesis, we assume that equation 5.1 holds for k, i.e. for all $z' \in \mathit{States}_{\mathcal{S}}$ and for all $u' \in \mathbb{N}$

$$P^k_{\mathcal{S},\mathit{Target}}(z') = \mathit{Pr}(\mathit{PBMC}'_{\mathcal{S},\mathit{Target}}(z',u',u'+k))$$

is true. We now show that induction hypothesis implies that equation 5.1 also holds for $k+1$, i.e. for all $z \in \mathit{States}_{\mathcal{S}}$ and for all $u \in \mathbb{N}$

$$P^{k+1}_{\mathcal{S},\mathit{Target}}(z) = \mathit{Pr}(\mathit{PBMC}'_{\mathcal{S},\mathit{Target}}(z,u,u+k+1))$$

is also true.

We first consider the case in which z is a target state, i.e. $z \models \mathit{Target}$. By construction of $\mathit{BMC}'_{\mathcal{S},\mathit{Target}}(z,u,u+k+1)$, for each assignment to the quantified variables in $\bigodot_{j=u+1}^{u+k+1} \mathit{CHOICE}_{\mathcal{S}}(j)$ the remaining SMT formula is then satisfiable. A satisfying assignment is achieved by identifying the values of all state variables at all depths with the value at depth u as given by state z, i.e. for each variable $v \in \bigcup_{i=1}^{n}(D_i \cup R_i)$ we set $v_{u+k+1} := z(v), \ldots, v_u := z(v)$. This assignment clearly satisfies $\mathit{val}(z,u)$, each $\mathit{TARGET}(i)$ for $u \leq i \leq u+k+1$, each $\mathit{SELF\_LOOP}_{\mathcal{S}}(j-1,j)$ for $u+1 \leq j \leq u+k+1$, and thus the whole formula. The rationale is that quantified variables only occur in $\mathit{TRANS}_{\mathcal{S}}(j-1,j)$ and all implications $(\neg \mathit{TARGET}(j-1) \Rightarrow \mathit{TRANS}_{\mathcal{S}}(j-1,j))$ are therefore trivially satisfied. Hence, if $z \models \mathit{Target}$ then we have $\mathit{Pr}(\mathit{PBMC}'_{\mathcal{S},\mathit{Target}}(z,u,u+k+1)) = 1$ and, moreover, $P^{k+1}_{\mathcal{S},\mathit{Target}}(z) = 1$ according to Lemma 5.2, i.e. $P^{k+1}_{\mathcal{S},\mathit{Target}}(z) = \mathit{Pr}(\mathit{PBMC}'_{\mathcal{S},\mathit{Target}}(z,u,u+k+1))$.

In the remainder of this proof, assume that $z \not\models \mathit{Target}$. Let be $Tr_i = \{t^i_1, \ldots, t^i_{\ell_i}\}$ for $1 \leq i \leq n$ be the set of symbolic transitions of automaton $\mathcal{A}_i$. Then, according to reduction steps 11 and 12:

$$\begin{aligned}
\mathit{NCHOICE}_{\mathcal{S}}(u+1) &= \exists tr^1_{u+1} \in Tr_1 \odot \ldots \odot \exists tr^n_{u+1} \in Tr_n \quad , \\
\mathit{PCHOICE}_{\mathcal{S}}(u+1) &= \text{Я}_{d_{t^1_1}} pc^{t^1_1}_{u+1} \in PC_{t^1_1} \odot \ldots \odot \text{Я}_{d_{t^1_{\ell_1}}} pc^{t^1_{\ell_1}}_{u+1} \in PC_{t^1_{\ell_1}} \odot \\
&\qquad \ldots \\
&\qquad \text{Я}_{d_{t^n_1}} pc^{t^n_1}_{u+1} \in PC_{t^n_1} \odot \ldots \odot \text{Я}_{d_{t^n_{\ell_n}}} pc^{t^n_{\ell_n}}_{u+1} \in PC_{t^n_{\ell_n}} \quad .
\end{aligned}$$

By Definition 4.5, it then follows:

$$\begin{aligned}
&Pr(\mathit{PBMC}'_{\mathcal{S},\mathit{Target}}(z, u, u+k+1)) \\
&\quad = \max\nolimits_{t_1 \in Tr_1} \ldots \max\nolimits_{t_n \in Tr_n} \\
(5.2) &\qquad \textstyle\sum_{p^1_1 \in PC_{t^1_1}} d_{t^1_1}(p^1_1) \cdot \Big(\ldots \Big(\sum_{p^1_{\ell_1} \in PC_{t^1_{\ell_1}}} d_{t^1_{\ell_1}}(p^1_{\ell_1}) \cdot \Big(\ldots \\
&\qquad \Big(\textstyle\sum_{p^n_1 \in PC_{t^n_1}} d_{t^n_1}(p^n_1) \cdot \Big(\ldots \Big(\sum_{p^n_{\ell_n} \in PC_{t^n_{\ell_n}}} d_{t^n_{\ell_n}}(p^n_{\ell_n}) \cdot \\
&\qquad\quad Pr\big(\mathit{CHC}^{u+k+1}_{u+2} : \mathit{BMC}'_{\mathcal{S},\mathit{Target}}(z, u, u+k+1)[\vec{t}, \vec{p} / \vec{tr}, \vec{pc}] \big) \ldots \Big) \Big) \ldots \Big) \Big) \ldots \Big)
\end{aligned}$$

where

$$\mathit{CHC}^{u+k+1}_{u+2} := \bigodot_{j=u+2}^{u+k+1} \mathit{CHOICE}_{\mathcal{S}}(j)$$

and $A[\vec{t}, \vec{p} / \vec{tr}, \vec{pc}]$ abbreviates the substitution of all non-deterministic choices t_i and all probabilistic choices p^i_j for the corresponding variables t^i_{u+1} and $pc^{t^i_j}_{u+1}$ in a predicate A, respectively, i.e.

$$A[\vec{t}, \vec{p} / \vec{tr}, \vec{pc}] := A[t_1, \ldots, t_n, p^1_1, \ldots, p^n_{\ell_n} / tr^1_{u+1}, \ldots, tr^n_{u+1}, pc^{t^1_1}_{u+1}, \ldots, pc^{t^n_{\ell_n}}_{u+1}] \quad .$$

Due to reduction step 7, for all assignments to the existential variables $tr^i_{u+1} := t^i_j$ with $t^i_j \in Tr_i$ and $1 \leq i \leq n$ the values of all randomized variables $pc^{t^i_q}_{u+1}$ with $q \neq j$ are irrelevant, i.e. all implication predicates introduced in reduction step 7 involving variables $pc^{t^i_q}_{u+1}$ with $q \neq j$ are trivially satisfied since $tr^i_{u+1} \neq t^i_q$. Since predicates containing randomized variables are only introduced in reduction step 7 as well as due to $\sum_{v \in PC_{t^i_q}} d_{t^i_q}(v) = 1$ and due to common arithmetic laws, above observation gives us the possibility to simplify equation 5.2 by turning all randomized variables $pc^{t^i_q}_{u+1}$ with $q \neq j$ into non-quantified ones:

$$\begin{aligned}
&Pr(\mathit{PBMC}'_{\mathcal{S},\mathit{Target}}(z, u, u+k+1)) \\
(5.3) &\quad = \max\nolimits_{t_1 \in Tr_1} \ldots \max\nolimits_{t_n \in Tr_n} \textstyle\sum_{p_1 \in PC_{t_1}} d_{t_1}(p_1) \cdot \Big(\ldots \Big(\sum_{p_n \in PC_{t_n}} d_{t_n}(p_n) \cdot \\
&\qquad Pr\big(\mathit{CHC}^{u+k+1}_{u+2} : \mathit{BMC}'_{\mathcal{S},\mathit{Target}}(z, u, u+k+1)[\vec{t'}, \vec{p'} / \vec{tr'}, \vec{pc'}] \big) \Big) \ldots \Big)
\end{aligned}$$

with

$$A[\vec{t'},\vec{p'}/\vec{tr'},\vec{pc'}] := A[t_1,\ldots,t_n,p_1,\ldots p_n/tr^1_{u+1},\ldots,tr^n_{u+1},pc^{t_1}_{u+1},\ldots pc^{t_n}_{u+1}] \enspace .$$

The intuition of concluding equation 5.3 is as follows: for each step j the quantifier prefix of SSMT formula $PBMC'_{\mathcal{S},\mathit{Target}}(z,u,u+k+1)$ contains randomized variables pc^{tr}_j for all possible transitions $tr \in Tr_i$ of all automata $\mathcal{A}_i$ to encode the probabilistic transition alternatives when taking transition tr, confer reduction step 12. These variables are always present, even so in all cases where transition tr is not selected in step j. Though in such cases the value of randomized variable pc^{tr}_j is irrelevant (as ensured by reduction step 7), the semantics given in Definition 4.5 needs to consider such variables pc^{tr}_j. This is reflected in equation 5.2. Exploiting the information about irrelevance of variables pc^{tr}_j and hence ignoring all such pc^{tr}_j for step $j = u+1$, we derived the simplified equation 5.3.

Now observe that for each assignment τ to the quantified variables $q_1,\ldots,q_w$ in prefix CHC^{u+k+1}_{u+2} it holds that

$$BMC'_{\mathcal{S},\mathit{Target}}(z,u,u+k+1)[\vec{t'},\vec{p'}/\vec{tr'},\vec{pc'}][\tau(q_1),\ldots,\tau(q_w)/q_1,\ldots,q_w]$$

is satisfiable if and only if

$$BMC'_{\mathcal{S},\mathit{Target}}(z,u,u+k+1)[\vec{t'},\vec{p'}/\vec{tr'},\vec{pc'}][\tau(q_1),\ldots,\tau(q_w)/q_1,\ldots,q_w][z(\vec{v})/\vec{v}_u]$$

is satisfiable. $A[z(\vec{v})/\vec{v}_u]$ means that each representative v_u of the discrete and continuous state variables $v \in \bigcup_{i=1}^n (D_i \cup R_i)$ at depth u in predicate A is replaced by the value $z(v)$ of variable v in state z. Above observation relies on the fact that $BMC'_{\mathcal{S},\mathit{Target}}(z,u,u+k+1)$ comprises the predicate $\mathit{val}(z,u)$ and, by construction, $\mathit{val}(z,u)$ permits exactly one satisfying assignment, namely this given by state z. As z is the same for each τ, we may move substitution in time and obtain:

$$BMC'_{\mathcal{S},\mathit{Target}}(z,u,u+k+1)[\vec{t'},\vec{p'}/\vec{tr'},\vec{pc'}][\tau(q_1),\ldots,\tau(q_w)/q_1,\ldots,q_w][z(\vec{v})/\vec{v}_u]$$

is satisfiable if and only if

$$BMC'_{\mathcal{S},\mathit{Target}}(z,u,u+k+1)[\vec{t'},\vec{p'}/\vec{tr'},\vec{pc'}][z(\vec{v})/\vec{v}_u][\tau(q_1),\ldots,\tau(q_w)/q_1,\ldots,q_w]$$

is satisfiable. Since representatives v_u of state variables $v \in \bigcup_{i=1}^n (D_i \cup R_i)$ are non-quantified, i.e. they do not occur in CHC^{u+k+1}_{u+2}, we may conclude using above facts that:

$$\begin{aligned} & Pr\big(CHC^{u+k+1}_{u+2} : BMC'_{\mathcal{S},\mathit{Target}}(z,u,u+k+1)[\vec{t'},\vec{p'}/\vec{tr'},\vec{pc'}]\big) \\ = \; & Pr\big(CHC^{u+k+1}_{u+2} : BMC'_{\mathcal{S},\mathit{Target}}(z,u,u+k+1)[\vec{t'},\vec{p'}/\vec{tr'},\vec{pc'}][z(\vec{v})/\vec{v}_u]\big) \end{aligned} \tag{5.4}$$

Recall that $z \not\models \mathit{Target}$. Then, $TARGET(u)[z(\vec{v})/\vec{v}_u]$ is equivalent to `false` and predicate $TRANS_{\mathcal{S}}(u,u+1)[\vec{t'},\vec{p'}/\vec{tr'},\vec{pc'}][z(\vec{v})/\vec{v}_u]$ needs to be satisfied in order to satisfy $BMC'_{\mathcal{S},\mathit{Target}}(z,u,u+k+1)[\vec{t'},\vec{p'}/\vec{tr'},\vec{pc'}][z(\vec{v})/\vec{v}_u]$. Due to construction, confer reduction steps 6 and 7, and due to Property 3.1 (uniqueness of post-states), $TRANS_{\mathcal{S}}(u,u+1)[\vec{t'},\vec{p'}/\vec{tr'},\vec{pc'}][z(\vec{v})/\vec{v}_u]$ is satisfied by at most one assignment to the representatives v_{u+1} of state variables $v \in \bigcup_{i=1}^n (D_i \cup R_i)$ for depth $u+1$. Let $tr = (t_1,\ldots,t_n) \in \mathit{NChoice}$ be the combined non-deterministic transition choice and $pc = (p_1,\ldots,p_n) \in \mathit{PChoice}(tr)$ be the combined probabilistic choice with t_i, p_i, $1 \leq i \leq n$, as in equation 5.3. Again due to reduction steps 6 and 7 and due to Property 3.1, we can conclude the following two relations: first, if the (unique) post-state $z' = \mathit{Post}(z, tr, pc)$ of $\mathcal{S}$ exists, i.e. $z' \neq \perp$, then

$$\mathit{val}(z', u+1) \;\equiv\; \mathit{TRANS}_{\mathcal{S}}(u, u+1)[\vec{t'}, \vec{p'}/\vec{tr'}, \vec{pc'}][z(\vec{v})/\vec{v}_u]$$

and, second, if $z' = \perp$ then $\mathit{TRANS}_{\mathcal{S}}(u, u+1)[\vec{t'}, \vec{p'}/\vec{tr'}, \vec{pc'}][z(\vec{v})/\vec{v}_u]$ is unsatisfiable. Thus, if $z' \neq \perp$ then we have

$$\begin{aligned}
& \mathit{BMC}'_{\mathcal{S},\mathit{Target}}(z, u, u+k+1)[\vec{t'}, \vec{p'}/\vec{tr'}, \vec{pc'}][z(\vec{v})/\vec{v}_u] \\
\equiv \quad & \mathit{val}(z', u+1) \\
\wedge \quad & \bigwedge_{j=u+2}^{u+k+1} \begin{pmatrix} (\neg \mathit{TARGET}(j-1) \Rightarrow \mathit{TRANS}_{\mathcal{S}}(j-1, j)) \\ \wedge(\mathit{TARGET}(j-1) \Rightarrow \mathit{SELF\_LOOP}_{\mathcal{S}}(j-1, j)) \end{pmatrix} \\
\wedge \quad & \mathit{TARGET}(u+k+1) \\
= \quad & \mathit{BMC}'_{\mathcal{S},\mathit{Target}}(z', u+1, u+k+1)
\end{aligned}$$

and, using equation 5.4, definition of $\mathit{PBMC}'_{\mathcal{S},\mathit{Target}}(z', u+1, (u+1)+k)$, and induction hypothesis,

$$\begin{aligned}
& \mathit{Pr}\big(\mathit{CHC}_{u+2}^{u+k+1} : \mathit{BMC}'_{\mathcal{S},\mathit{Target}}(z, u, u+k+1)[\vec{t'}, \vec{p'}/\vec{tr'}, \vec{pc'}]\big) \\
= \; & \mathit{Pr}\big(\mathit{CHC}_{u+2}^{u+k+1} : \mathit{BMC}'_{\mathcal{S},\mathit{Target}}(z', u+1, u+k+1)\big) \\
= \; & \mathit{Pr}\big(\mathit{PBMC}'_{\mathcal{S},\mathit{Target}}(z', u+1, (u+1)+k)\big) \\
= \; & \mathit{Pr}\big(\mathit{PBMC}'_{\mathcal{S},\mathit{Target}}(z', u', u'+k)\big) \\
= \; & P^{k}_{\mathcal{S},\mathit{Target}}(z')
\end{aligned} \tag{5.5}$$

with $u' = u+1$. If $z' = \perp$ then unsatisfiability of $\mathit{TRANS}_{\mathcal{S}}(u, u+1)[\vec{t'}, \vec{p'}/\vec{tr'}, \vec{pc'}][z(\vec{v})/\vec{v}_u]$ entails unsatisfiability of $\mathit{BMC}'_{\mathcal{S},\mathit{Target}}(z, u, u+k+1)[\vec{t'}, \vec{p'}/\vec{tr'}, \vec{pc'}][z(\vec{v})/\vec{v}_u]$. By equation 5.4, we thus conclude

$$\mathit{Pr}\big(\mathit{CHC}_{u+2}^{u+k+1} : \mathit{BMC}'_{\mathcal{S},\mathit{Target}}(z, u, u+k+1)[\vec{t'}, \vec{p'}/\vec{tr'}, \vec{pc'}]\big) = 0 \quad .$$

Recall that $\mathit{Post}(\perp, tr', pc') = \perp$ for all $tr' \in \mathit{NChoice}$ and $pc' \in \mathit{PChoice}(tr)$ and that $\perp \not\models \mathit{Target}$. As a consequence, $P^{k'}_{\mathcal{S},\mathit{Target}}(\perp) = 0$ for each $k' \in \mathbb{N}$, confer the definition in Lemma 5.2. By observation above and by equation 5.5, we therefore obtain

$$\mathit{Pr}\big(\mathit{CHC}_{u+2}^{u+k+1} : \mathit{BMC}'_{\mathcal{S},\mathit{Target}}(z, u, u+k+1)[\vec{t'}, \vec{p'}/\vec{tr'}, \vec{pc'}]\big) = P^{k}_{\mathcal{S},\mathit{Target}}(z') \tag{5.6}$$

equally whether $z' \neq \perp$ or $z' = \perp$. Application of equation 5.6 in equation 5.3 leads to

$$\begin{aligned}
& \mathit{Pr}(\mathit{PBMC}'_{\mathcal{S},\mathit{Target}}(z, u, u+k+1)) \\
& \quad = \max_{t_1 \in Tr_1} \ldots \max_{t_n \in Tr_n} \sum_{p_1 \in PC_{t_1}} d_{t_1}(p_1) \cdot \Big(\ldots \Big(\sum_{p_n \in PC_{t_n}} d_{t_n}(p_n) \cdot P^{k}_{\mathcal{S},\mathit{Target}}(z') \Big) \ldots \Big) \quad .
\end{aligned}$$

Using definitions, namely $tr = (t_1, \dots, t_n) \in \mathit{NChoice}$, $pc = (p_1, \dots, p_n) \in \mathit{PChoice}(tr)$, $z' = \mathit{Post}(z, tr, pc)$, $p(tr, pc) = \prod_{i=1}^{n} p(t_i)(p_i)$, and $d_t(v) = p(t)(v)$ for all $t \in \bigcup_{i=1}^{n} Tr_i$ and for all $v \in PC_t$, as well as common arithmetic laws, we can rewrite statement above to

$$\begin{aligned} &Pr(\mathit{PBMC}'_{\mathcal{S},\mathit{Target}}(z, u, u+k+1)) \\ &\quad = \max_{tr \in \mathit{NChoice}} \sum_{pc \in \mathit{PChoice}(tr)} p(tr, pc) \cdot P^{k}_{\mathcal{S},\mathit{Target}}(\mathit{Post}(z, tr, pc)) \quad . \end{aligned}$$

Since $z \not\models \mathit{Target}$ and $k + 1 > 0$, application of definition of $P^{k+1}_{\mathcal{S},\mathit{Target}}(z)$ gives the desired result, i.e.

$$Pr(\mathit{PBMC}'_{\mathcal{S},\mathit{Target}}(z, u, u+k+1)) = P^{k+1}_{\mathcal{S},\mathit{Target}}(z) \quad .$$

This completes the induction step and the theorem follows. □

From Theorem 5.1, it trivially follows that the decision version of probabilistic bounded reachability with respect to probability threshold θ, namely the PBMC problem from Definition 5.4, can be solved by deciding whether the satisfaction probability of the symbolic encoding $\mathit{PBMC}_{\mathcal{S},\mathit{Target}}(k)$ is below θ:

Corollary 5.1 (SSMT-based probabilistic bounded model checking)
Let be given a system $\mathcal{S}$ of n concurrent PHAs, a target states predicate Target, a step bound $k \in \mathbb{N}$, and a probability threshold $\theta \in [0, 1]$. Then,

$$P^{k}_{\mathcal{S},\mathit{Target}}(\imath) \leq \theta \quad \text{if and only if} \quad Pr(\mathit{PBMC}_{\mathcal{S},\mathit{Target}}(k)) \leq \theta$$

with $\imath \models \bigwedge_{i=1}^{n} init_i$ being the (unique) initial state of $\mathcal{S}$.

To complete the symbolic approach to probabilistic bounded reachability, Chapter 6 introduces algorithms to solve SSMT formulae. In addition to a detailed description of underlying concepts and a thorough theoretical investigation, an essential part of Chapter 6 is devoted to algorithmic optimizations with the objective of saving computational effort in practice. One such optimization called *thresholding* is motivated by Corollary 5.1: to solve the PBMC problem with respect to threshold θ, we need not compute the exact satisfaction probability $Pr(\mathit{PBMC}_{\mathcal{S},\mathit{Target}}(k))$ but it is sufficient to decide whether $Pr(\mathit{PBMC}_{\mathcal{S},\mathit{Target}}(k))$ is below or strictly above θ. The latter fact can be exploited in the development of SSMT algorithms. While such SSMT algorithms implement a traversal through the tree given by the quantifier prefix as indicated in Figure 4.3, thresholding enables aggressive pruning rules that save visits to potentially major parts of the search space whenever a threshold for some subtree is already exceeded by processed branches or can no longer be reached by all remaining branches.

6 Algorithms for SSMT Problems

After having introduced the formal model of concurrent probabilistic hybrid automata in Chapter 3 and the logical framework of SSMT in Chapter 4 as well as the reduction from probabilistic bounded reachability problems for PHAs to SSMT in Chapter 5, this chapter finally completes the symbolic reachability analysis approach for concurrent PHAs by elaborating on algorithms to solve SSMT formulae.

The SSMT algorithm presented in this thesis is based on algorithmic concepts to solve SSAT as well as SMT formulae, while algorithms for the latter both build on a decision procedure for the SAT problem. Therefore, we first refer to a procedure for the SAT problem in Section 6.1 which is then followed by explaining its extensions to cope with the more general SSAT and SMT problems in Sections 6.2 and 6.3, respectively. The basic SSMT algorithm finally is elaborated on in Section 6.4, while Section 6.5 proposes algorithmic enhancements in order to improve performance in practice. The theoretical considerations of Sections 6.4 and 6.5 are implemented in the tool SiSAT, described in Section 6.6, and empirically evaluated in Section 6.7.

6.1 Algorithms for SAT

As introduced in Section 4.1, given a propositional formula φ in CNF, the Boolean satisfiability problem (SAT) asks whether a satisfying assignment of φ exists. Though all existing SAT algorithms show exponential runtime in worst case, modern approaches to solve SAT exhibit remarkable performance on practically relevant problems stemming from industrial applications like software and hardware verification. These so-called *SAT solvers* are in the majority of cases based on the *Davis-Putnam-Logemann-Loveland* (DPLL) procedure [DP60, DLL62] enhanced with various algorithmic optimizations.

The DPLL procedure mainly implements a backtracking algorithm that searches for a satisfying assignment of φ by means of manipulating a partial assignment τ to the variables of φ. This process stops either if τ could be extended to a satisfying assignment or if all assignments were probed without finding a satisfying one, the latter proving that φ is unsatisfiable. At the beginning of the search process, the partial assignment τ is empty, i.e. no variable is assigned a truth value. Then, τ is incrementally extended by *decisions* and accompanied *deductions*. In a *decision* step, an unassigned variable $x \in \mathit{Var}(\varphi)$, i.e. $\tau(x)$ is not defined, is selected and then assigned a truth value $t \in \mathbb{B}$, i.e. $\tau(x) := t$. Each decision step is followed by the *deduction phase* involving the search for *unit clauses*, i.e. clauses $c \in \varphi$ that have only one unassigned literal ℓ left while all other literals $\ell' \neq \ell$ in c are `false` under current (partial) assignment τ, i.e. $\tau(\ell') =$ `false`. The unassigned literal ℓ in a unit clause is called *unit literal*. In order to satisfy a unit clause c (and thus prevent the whole formula φ from becoming violated) under an extension of the partial assignment τ, it is clear that we need to satisfy the unit literal $\ell \in c$, i.e. we need to extend τ in such a way that $\tau(\ell) = \texttt{true}$. For instance, let $c = (x \vee \neg y \vee \neg z)$ be a clause that is unit under partial assignment τ with $\tau(x) = \texttt{false}$ and $\tau(y) = \texttt{true}$. To satisfy c, we extend τ by setting $\tau(z) := \texttt{false}$. Then, $\tau(\neg z) = \texttt{true}$ and thus $\tau(c) = \texttt{true}$. Such an extension of τ is called a *deduction*, while the mechanism of deducing unit literals is referred to as *unit propagation*. Please observe that not only decisions but also deductions can trigger further deductions. Whenever no new deduction can be performed and no clause became definitely `false` under τ, i.e. there is no clause $c \in \varphi$ such that $\forall \ell \in c : \tau(\ell) = \texttt{false}$, the procedure continues with a new decision step. A decision step and its accompanied deductions are called *decision level*.

However, deduction may also yield a *conflicting clause* $c' \in \varphi$ which has all its literals assigned `false` under current τ, i.e. $\forall \ell \in c' : \tau(\ell) = \texttt{false}$ and thus $\tau \not\models c'$. Such a situation is called *conflict*. For instance, consider the small example above and assume that there is another clause $c' = (x \vee \neg y \vee z)$. After deducing $\tau(z) := \texttt{false}$, clause c' becomes conflicting as $\tau(x) = \texttt{false}$, $\tau(\neg y) = \texttt{false}$, and $\tau(z) = \texttt{false}$. Since φ is in CNF and $c' \in \varphi$, each (complete) assignment τ' that extends partial assignment τ, i.e. if $\tau(x)$ is defined then $\tau'(x) = \tau(x)$ for each $x \in \mathit{Var}(\varphi)$, does not satisfy φ, i.e. $\tau' \not\models \varphi$. This fact indicates the need for *backtracking*, i.e. to take back some of the latter decision levels. This process then retrieves some previous partial assignment from which the search will be continued. To avoid repeated conflicts due to the same reason, modern SAT algorithms incorporate *conflict-driven clause learning* [MSLM09] to derive a sufficiently general explanation (a combination of variable assignments) for the actual conflict. In our running example, assume that first assignment $\tau(x) = \texttt{false}$ and then $\tau(y) = \texttt{true}$ were made by decision steps. Then, both variable assignments are an explanation for the current conflict. Based on that (ideally minimal) set of assignments that triggered the particular conflict, a *conflict clause cc* is generated and added to the clause set to guide the subsequent search. Such a conflict clause *cc* encodes the negation of the variable assignment leading to the conflict, and ensures to not visit the same conflicting assignment again. In the example, we get $cc = (x \vee \neg y)$ meaning that x must be set to `true` or y to `false` in order to avoid the previous conflicting situtation. Additionally, the conflict clause is also used to compute the backtrack level, i.e. the decision level on which the search will be continued. Most modern SAT solvers implement the *first unique implication point* technique from [ZMMM01]. In the latter, the backtrack level is the oldest decision level on which the current conflict clause *cc* becomes unit. This approach leads to a *non-chronological backtracking* operation, often jumping back more than just one level, thus making conflict-driven clause learning combined with non-chronological backtracking a powerful mechanism to prune large parts of the search space. In our example, we can go back only one level, namely to decision level starting with $\tau(x) = \texttt{false}$, as conflict clause $cc = (x \vee \neg y)$ is unit there but *cc* will be no longer unit when also undoing decision $\tau(x) = \texttt{false}$.

The overall DPLL procedure always terminates, namely if an assignment τ was found that satisfies formula φ or if a conflict cannot be resolved, i.e. there are no decision levels to be taken back. In the latter case, there does not exist any solution of φ, i.e. φ is unsatisfiable.

We finally mention that state-of-the-art SAT solvers exploit a vast number of further algorithmic optimizations that have led to impressive performance gains during the last years. Among others, these are very sophisticated data structures that permit efficient detection of unit clauses based on so-called *two watched literals* as well as powerful *variable and value decision heuristics* [MMZ+01]. For a very detailed account of modern SAT solving techniques as well as applications of SAT, we refer the interested reader to [BHvMW09].

6.2 Algorithms for SSAT

As mentioned in Section 4.2, the general SSAT problem is PSPACE-complete. The plethora of real-world applications like probabilistic planning however calls for practically efficient algorithms. We therefore explain state-of-the-art SSAT algorithms, being based on the DPLL procedure, in Subsection 6.2.1. In contrast to the aforementioned DPLL-style *backtracking* algorithms, a novel approach to solve SSAT problems following the idea of the *resolution* principle is suggested in Subsection 6.2.2. Completing this section, a theoretical comparison between this novel SSAT resolution calculus and the classical DPLL-SSAT procedure as well as potential applications of SSAT resolution are finally presented in Subsection 6.2.3.

6.2.1 DPLL-based SSAT procedure

As pioneered by Littman [Lit99], state-of-the-art SSAT algorithms implement a DPLL-style backtracking search that mimics the semantics of SSAT, thereby explicitly traversing the tree given by the quantifier prefix and recursively computing the individual satisfaction probabilities for each subtree by the scheme illustrated in Figure 4.1. To improve performance in practice, DPLL-SSAT procedures moreover incorporate several algorithmic optimizations where the most prominent ones are *unit propagation*, *purification*, and *thresholding*. Figure 6.1 shows the standard DPLL-SSAT procedure as presented similarly in the recent overview article [Maj09] (in a version without universal quantifiers).

In the following, we explain the DPLL-SSAT algorithm from Figure 6.1. In addition to an SSAT formula $\Phi = \mathcal{Q} : \varphi$ with propositional formula φ being in CNF, the algorithm DPLL-SSAT$(\Phi, \theta_l, \theta_u)$ requires as further inputs two rational values θ_l and θ_u with $\theta_l \leq \theta_u$ that are called *lower threshold* and *upper threshold*, respectively. The idea of these additional parameters is to alleviate workload of the algorithm, and thus to improve performance, whenever precise information about the probability result is not or only partially necessary. More precisely, in case the probability of satisfaction $Pr(\Phi)$ lies in the interval $[\theta_l, \theta_u]$ then the algorithm must return the exact satisfaction probability, i.e. DPLL-SSAT$(\Phi, \theta_l, \theta_u) = Pr(\Phi)$. Otherwise, i.e. $Pr(\Phi) \notin [\theta_l, \theta_u]$, the exact result is not of interest but only some witness value $pr =$ DPLL-SSAT$(\Phi, \theta_l, \theta_u)$ with $pr < \theta_l$ if and only if $Pr(\Phi) < \theta_l$ and with $pr > \theta_u$ if and only if $Pr(\Phi) > \theta_u$. These thresholds are exploited during proof search to boost efficiency by skipping some recursive calls of DPLL-SSAT which is called *thresholding* and described later on. Providing the possibility of thresholding is actually motivated by several industrial applications like the verification of probabilistic safety properties where the problem is to decide whether the probability of reaching unsafe states is below or above some acceptable threshold θ, as in Definition 5.4. In the latter case, the lower threshold and the upper threshold coincide, i.e. $\theta_l = \theta_u = \theta$.

```
DPLL-SSAT( $\mathcal{Q} : \varphi,\ \theta_l,\ \theta_u$ )
  input: SSAT formula $\mathcal{Q} : \varphi$ with $\varphi$ in CNF, rational constants $\theta_l, \theta_u$ with $\theta_l \leq \theta_u$.

  // Base cases
  if $\varphi$ contains a clause equivalent to false then return 0.
  if all clauses in $\varphi$ equivalent to true then return 1.
  // Unit propagation
  if $\varphi$ contains a unit literal $\ell$ with $Var(\ell) = \{x\}$ then
    if $\mathcal{Q} = \mathcal{Q}_1 \exists x \mathcal{Q}_2$ then return DPLL-SSAT( $\mathcal{Q}_1\mathcal{Q}_2 : \varphi[v(\ell)/x],\ \theta_l,\ \theta_u$ ).
    if $\mathcal{Q} = \mathcal{Q}_1 Я^p x \mathcal{Q}_2$ then
      return $p(\ell) \cdot$ DPLL-SSAT( $\mathcal{Q}_1\mathcal{Q}_2 : \varphi[v(\ell)/x],\ \theta_l/p(\ell),\ \theta_u/p(\ell)$ ).
  // Purification
  if $\varphi$ contains a pure literal $\ell$ with $Var(\ell) = \{x\}$ then
    if $\mathcal{Q} = \mathcal{Q}_1 \exists x \mathcal{Q}_2$ then return DPLL-SSAT( $\mathcal{Q}_1\mathcal{Q}_2 : \varphi[v(\ell)/x],\ \theta_l,\ \theta_u$ ).
  // Branching and thresholding
  if $\mathcal{Q} = \exists x \mathcal{Q}'$ then
    $pr_1$ := DPLL-SSAT( $\mathcal{Q}' : \varphi[\texttt{true}/x],\ \theta_l,\ \theta_u$ ).
    if $pr_1 > \theta_u$ then return $pr_1$.
    $pr_2$ := DPLL-SSAT( $\mathcal{Q}' : \varphi[\texttt{false}/x],\ \max(\theta_l, pr_1),\ \theta_u$ ).
    return $\max(pr_1, pr_2)$.
  if $\mathcal{Q} = Я^p x \mathcal{Q}'$ then
    $pr_1$ := DPLL-SSAT( $\mathcal{Q}' : \varphi[\texttt{true}/x],\ (\theta_l - (1-p))/p,\ \theta_u/p$ ).
    if $p \cdot pr_1 + (1-p) < \theta_l$ then return $p \cdot pr_1$.
    if $p \cdot pr_1 > \theta_u$ then return $p \cdot pr_1$.
    $pr_2$ := DPLL-SSAT( $\mathcal{Q}' : \varphi[\texttt{false}/x],\ (\theta_l - p \cdot pr_1)/(1-p),\ (\theta_u - p \cdot pr_1)/(1-p)$ ).
    return $p \cdot pr_1 + (1-p) \cdot pr_2$.
```

Figure 6.1: DPLL-based backtracking algorithm for SSAT as presented similarly in [Maj09]. If literal ℓ is positive then $v(\ell) = \texttt{true}$ and $p(\ell) = p$, and otherwise $v(\ell) = \texttt{false}$ and $p(\ell) = 1 - p$. (This figure is a slight modification of Figure 2 from [TF10].)

The same holds when considering the SSAT decision problem, i.e. to decide whether $Pr(\Phi) \geq \theta$ is true: due to above facts, we conclude that $Pr(\Phi) \geq \theta$ if and only if DPLL-SSAT$(\Phi, \theta, \theta) \geq \theta$. We furthermore remark that the presence of the threshold parameters does not restrict the scope of the SSAT procedure. That is, whenever the exact probability of satisfaction needs to be computed then the lower threshold θ_l should be set to value 0 and the upper threshold θ_u to value 1.

Base cases. Algorithm DPLL-SSAT$(\mathcal{Q} : \varphi, \theta_l, \theta_u)$ first checks whether matrix φ contains a clause $c \in \varphi$ with $c \equiv$ `false`. Such clause c, which corresponds to a *conflicting* clause in the DPLL procedure for SAT, occurs if substitution has replaced the variables of all positive literals in c by `false` and the variables of all negative literals in c by `true` such that all literals in c are equivalent to `false`. In this conflicting situation, DPLL-SSAT correctly returns result 0 since φ is unsatisfiable (as φ is in CNF) and thus $Pr(\mathcal{Q} : \varphi) = 0$

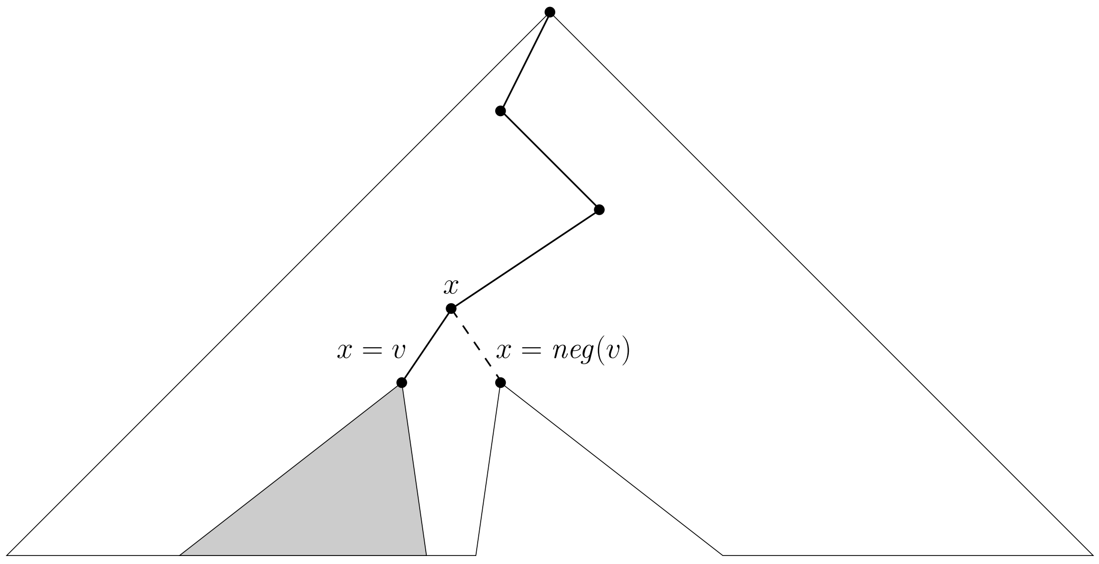


Figure 6.2: Illustration of pruning the search space: whenever DPLL-SSAT detects that branch "$x = neg(v)$" for variable x may be skipped due to some algorithmic optimization like unit propagation, a potentially huge subtree can be cut off. We denote by $neg(v) \in \mathbb{B}$ the opposite of truth value v, i.e. $neg(v) =$ `true` if and only if $v =$ `false`.

according to Definition 4.2.

The next step is to check whether each clause $c \in \varphi$ is equivalent to `true`, i.e. at least one literal ℓ in each clause c is equivalent to `true`, i.e. the variable of ℓ was replaced by `true` if ℓ is positive or by `false` if ℓ is negative. If so, φ is a tautology, i.e. $\models \varphi$, and consequently $Pr(\mathcal{Q} : \varphi) = 1$ according to Definition 4.2. DPLL-SSAT then returns 1.

If both of the above tests have failed then the algorithm tries to prune the search space in the sense of skipping some recursive calls by means of three algorithmic optimizations.

Unit propagation. The first such optimization is called *unit propagation*: as in the deduction phase of the DPLL algorithm for SAT, this enhancement detects *unit clauses* c and immediately propagates their *unit literals* ℓ by substituting the corresponding values $v(\ell)$ for variables x with $Var(\ell) = \{x\}$ such that unit clauses c become `true`, i.e. $v(\ell) =$ `true` for positive literals ℓ and $v(\ell) =$ `false` for negative literals ℓ. Recall that a unit literal ℓ in a unit clause c is the only unassigned literal while all other literals in c are assigned `false`. Observe that unit propagation moves a quantifier in the prefix to the left, i.e. $\mathcal{Q}_1 Q x \mathcal{Q}_2 \rightsquigarrow Q x \mathcal{Q}_1 \mathcal{Q}_2$. This is of course not a valid operation in general. However, this is correct for unit literals $\ell \in \{x, \neg x\}$, i.e. $Pr(\mathcal{Q}_1 Q x \mathcal{Q}_2 : \varphi \wedge \ell) = Pr(Q x \mathcal{Q}_1 \mathcal{Q}_2 : \varphi \wedge \ell)$, as the value of x satisfying ℓ does not depend on $\mathcal{Q}_1$. This fact is evinced by Lemma 6.2 later on. Each application of unit propagation clearly saves one recursive call of DPLL-SSAT, namely this one for the branch where x is set to the opposite of value $v(\ell)$. Intuitively, this optimization cuts off a potentially huge subtree of the overall search tree given by the quantifier prefix, confer Figure 6.2.

Purification. The next algorithmic enhancement, called *purification*, is in some sense similar to unit propagation but it propagates *pure literals*. A literal ℓ occurring in some clause of φ is called *pure* if and only if each clause $c \in \varphi$ that is not equivalent to $\texttt{true}$, i.e. $c \not\equiv \texttt{true}$, does not contain the opposite literal $neg(\ell)$, i.e. $neg(\ell) \notin c$. For an example consider formula $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_4 \vee \texttt{true})$. Literals x_1 and $\neg x_1$ are clearly not pure, while literals $\neg x_2$, x_3, and $\neg x_4$ however are. We remark that literal $\neg x_2$ is pure as clause $(x_2 \vee \neg x_4 \vee \texttt{true})$, containing the opposite literal $x_2 = neg(\neg x_2)$, is equivalent to $\texttt{true}$. With regard to soundness of purification, we argue as follows. Let φ be a propositional formula in CNF and $\ell \in \{x, \neg x\}$ be a pure literal. Let us further denote the set of all clauses in φ that are not equivalent to $\texttt{true}$ by $U(\varphi) := \{c \in \varphi : \varphi \not\equiv \texttt{true}\}$. We define $v(\ell) = \texttt{true}$ and $\bar{v}(\ell) = \texttt{false}$ if ℓ is positive, and $v(\ell) = \texttt{false}$ and $\bar{v}(\ell) = \texttt{true}$ otherwise. It is not hard to see that $U(\varphi[\bar{v}(\ell)/x]) \supseteq U(\varphi[v(\ell)/x])$ and thus $\varphi[\bar{v}(\ell)/x] \Rightarrow \varphi[v(\ell)/x]$ and finally $Pr(\mathcal{Q} : \varphi[\bar{v}(\ell)/x]) \leq Pr(\mathcal{Q} : \varphi[v(\ell)/x])$. Above fact justifies the application of purification for existential variables. Moving the quantifier within the prefix, i.e. $\mathcal{Q}_1 \exists x \mathcal{Q}_2 \rightsquigarrow \exists x \mathcal{Q}_1 \mathcal{Q}_2$, again relies on the observation that the value of x does not depend on $\mathcal{Q}_1$, i.e. for each assignment to the variables in $\mathcal{Q}_1$, x is set to value $v(\ell)$. This fact is also evinced by Lemma 6.2 later on. Purification seems however *not* possible for *randomized* variables as the other branch where x is set to $\bar{v}(\ell)$ may yield some contribution, i.e. potentially, $Pr(\mathcal{Q}_1 \mathcal{Q}_2 : \varphi[\bar{v}(\ell)/x]) > 0$.

A slightly more general definition of pure literals ℓ does *not* demand that ℓ need be present in a clause of φ. Applying the latter definition, literal x_4 would also be pure in above example and clearly each literal ℓ such that neither ℓ nor $neg(\ell)$ occurs in the formula. This more general concept of pure literals does not destroy soundness of purification and is of practical benefit whenever there are some variables x that occur in quantifier prefix $\mathcal{Q}$ but not in matrix φ, i.e. $Qx \in \mathcal{Q}$ but $x \notin Var(\varphi)$. Such cases are actually permitted by Definition 4.1 and potentially occur during solving an SSAT formula, namely when considering all satisfied clauses as "removed" from matrix φ. Observe that DPLL-SSAT prevents an infinite application of purification since variable x must occur in the current quantifier prefix.

Branching and thresholding. In case none of the above checks was successful, the procedure applies *branching* thereby trying to exploit another optimization called *thresholding*. A branching step is similar in nature to a decision step in the DPLL algorithm for SAT. The difference however lies in the selection of the variable to be assigned a value. While in the quantifier-free SAT case the decision variable can be freely chosen, variable selection in DPLL-SSAT is more restrictive and depends on the quantifier prefix. According to the semantics of SSAT formalized in Definition 4.2, the algorithm presented in Figure 6.1 actually selects the leftmost variable in the prefix. We remark that this selection method can be relaxed to some extent: for an SSAT formula $Q_1 x_1 \ldots Q_k x_k \mathcal{Q} : \varphi$ where the quantifiers Q_i for $1 \leq i \leq k$ are the same, i.e. $Q_1 = \ldots = Q_k$, one of variables $x_1, \ldots, x_k$ can be freely selected for branching. The rationale is that quantifiers within a block of same quantifiers may be moved arbitrarily. Formally, $Pr(\mathcal{Q}_1\, Qx\, \mathcal{Q}\, Qy\, \mathcal{Q}_2 : \varphi) = Pr(\mathcal{Q}_1\, Qy\, \mathcal{Q}\, Qx\, \mathcal{Q}_2 : \varphi)$ whenever all variables in $\mathcal{Q}$ are bound by the same quantifier $Q \in \{\exists, \text{Я}^p\}$.[1] The latter proposition follows from Definition 4.2 and common arithmetic laws.

[1]We remark that the probabilities p of randomized quantifiers Я^p need not be homogeneous.

After selection of a variable x for branching, the procedure takes an arbitrary truth value $t \in \mathbb{B}$ and then solves the corresponding subformula $\mathcal{Q}' : \varphi[t/x]$. For presentation reasons only, the algorithm in Figure 6.1 first selects truth value `true`. This restriction could however be simply relaxed to the selection of arbitrary truth values with the consequence of a more technical presentation of the algorithm. Let the result of the first subproblem, i.e. where x is replaced by `true`, be satisfaction probability pr_1. We first assume that the conditions in the subsequent if-statements are not satisfied. Then, the other subproblem, i.e. where `false` is substituted for x, is solved by recursive call of DPLL-SSAT yielding satisfaction probability pr_2. In a final step, DPLL-SSAT returns the corresponding probability of satisfaction pr according to Definition 4.2, i.e. $pr = \max(pr_1, pr_2)$ if $\mathcal{Q} = \exists x \mathcal{Q}'$ and $pr = p \cdot pr_1 + (1-p) \cdot pr_2$ if $\mathcal{Q} = \text{Я}^p x \mathcal{Q}'$.

We now explain the idea of *thresholding*, which is another optimization technique to prune the search space using the threshold parameters θ_l and θ_u. Recall that DPLL-SSAT must return the exact satisfaction probability if the latter lies within the interval $[\theta_l, \theta_u]$. Otherwise, just a witness value pr is required such that $pr < \theta_l$ or $pr > \theta_u$ if and only if the actual satisfaction probability is strictly less than θ_l or strictly greater than θ_u, respectively. As a consequence, the algorithm may skip to solve the second subformula $\mathcal{Q}' : \varphi[\texttt{false}/x]$ whenever the result pr_1 of DPLL-SSAT for the first subproblem $\mathcal{Q}' : \varphi[\texttt{true}/x]$ is large enough in the sense that the final probability result will exceed the upper threshold in any case, i.e. whenever $pr_1 > \theta_u$ in case $\mathcal{Q} = \exists x \mathcal{Q}'$ and $p \cdot pr_1 > \theta_u$ in case $\mathcal{Q} = \text{Я}^p x \mathcal{Q}'$. This pruning technique is justified by a monotonicity argument, i.e. $pr_1 > \theta_u$ implies $\max(pr_1, pr_2) > \theta_u$ and $p \cdot pr_1 > \theta_u$ implies $p \cdot pr_1 + (1-p) \cdot pr_2 > \theta_u$ due to $p, pr_1, pr_2 \in [0,1]$. A similar thresholding rule is applicable if probability result pr_1 for the first branch is too small in the sense that the final probability result will never reach the lower threshold θ_l independent of the actual value of pr_2 for the second branch. The latter rule is clearly infeasible for existential variables since if $pr_2 > \theta_l$ then $\max(pr_1, pr_2) > \theta_l$, i.e. for unknown pr_2 there is no way to conclude that $\max(pr_1, pr_2) < \theta_l$ only from information about pr_1 (even if $pr_1 = 0$). For the randomized case, however, we can reason as follows: given probability pr_1 for the first branch, then the greatest possible overall probability is obtained if $pr_2 = 1$, i.e. $p \cdot pr_1 + (1-p) \cdot pr_2 \leq p \cdot pr_1 + (1-p) \cdot 1$ for each $pr_2 \in [0,1]$. Therefore, if the greatest possible value $p \cdot pr_1 + (1-p)$ is already strictly less than the lower threshold θ_l then there is no chance that the final probability will reach θ_l. This argument justifies to skip the second branch.

It remains to describe how the lower threshold θ_l' and the upper threshold θ_u' are determined for recursive calls of the algorithm, and to show that the use of these thresholds is sound as well as that $\theta_l' \leq \theta_u'$ holds.

Let us first consider the existential case, i.e. $\mathcal{Q} = \exists x \mathcal{Q}'$. In the first recursion, we simply transfer the given thresholds θ_l and θ_u. To recognize soundness of the latter, let $pr' = Pr(\mathcal{Q}' : \varphi[v(\ell)/x])$ be the actual satisfaction probability of the first branch, and $pr'' = Pr(\mathcal{Q}' : \varphi[neg(v(\ell))/x])$ be the actual probability of the potential second branch where $neg(v(\ell))$ is the opposite of truth value $v(\ell)$. The actual final probability of satisfaction then is $pr = \max(pr', pr'')$. For the lower threshold, we first consider the case where $pr' \geq pr''$. Here, $pr = pr'$ and thus $pr < \theta_l$ if and only if $pr' < \theta_l$. We now deal with case $pr' < pr''$ in which we obtain $pr = pr''$. This fact implies that the actual value of pr' is not of interest in order to compute the final result, allowing any lower threshold. Both observations above justify the use of θ_l as lower threshold of first recursion. With regard to upper threshold θ_u, observe that if $pr' > \theta_u$ then $pr > \theta_u$. The latter fact permits the use of upper threshold θ_u for first recursive call of DPLL-SSAT. Clearly by assumption, $\theta_l \leq \theta_u$.

For the second recursion, we potentially strengthen the lower threshold by taking the maximum $\max(\theta_l, pr_1)$ of the current lower threshold θ_l and the probability result pr_1 of the first branch. This potentially enables more aggressive thresholding within the recursive call. The upper threshold is as above, namely θ_u. Observe that second branch is executed only if thresholding has failed before, i.e. we assume $pr_1 \leq \theta_u$ in the following. Let $pr' = Pr(\mathcal{Q}' : \varphi[\mathtt{false}/x])$ be the actual satisfaction probability of the second branch and $pr = \max(pr_1, pr')$ be the actual final probability result. With regard to lower threshold, consider the case $pr_1 \geq pr'$ first. Then, $pr = pr_1$ and thus $pr < \theta_l$ if and only if $pr_1 < \theta_l$, which means that the result pr is independent of the value pr' of second branch. The latter fact allows any lower threshold for second recursion. If $pr_1 < pr'$ then $pr = pr'$, and thus $pr < \theta_l$ if and only if $pr' < \theta_l$ if and only if $pr' < \max(\theta_l, pr_1)$. This establishes the argument for setting the new lower threshold to $\max(\theta_l, pr_1)$. For the upper threshold, we have that $pr > \theta_u$ if and only if $pr' > \theta_u$ as $pr_1 \leq \theta_u$. Note that $\max(\theta_l, pr_1) \leq \theta_u$ since $\theta_l \leq \theta_u$ and $pr_1 \leq \theta_u$.

In the randomized case, i.e. $\mathcal{Q} = \text{Я}^p x \mathcal{Q}'$, modifications of the thresholds are a bit more sophisticated. We first examine the case in which unit propagation applies: here, we definitely know that the branch where x is replaced by value $neg(v(\ell))$ yields probability 0 since unit literal ℓ is then violated, i.e. $Pr(\mathcal{Q}_1\mathcal{Q}_2 : \varphi[neg(v(\ell))/x]) = 0$. Let be $pr' = Pr(\mathcal{Q}_1\mathcal{Q}_2 : \varphi[v(\ell)/x])$. The result returned by DPLL-SSAT then is $p(\ell) \cdot pr'$ with $p(\ell) = p$ if unit literal ℓ is positive and $p(\ell) = 1-p$ if ℓ is negative. Recall that $0 < p < 1$ and thus $p(\ell) > 0$. We then obtain that $p(\ell) \cdot pr' < \theta_l$ if and only if $pr' < \theta_l/p(\ell)$ and $p(\ell) \cdot pr' > \theta_u$ if and only if $pr' > \theta_u/p(\ell)$. This gives us the argument to use $\theta_l/p(\ell)$ as the new lower threshold and $\theta_u/p(\ell)$ as the new upper one. Since $\theta_l \leq \theta_u$ and $p(\ell) > 0$, it follows that $\theta_l/p(\ell) \leq \theta_u/p(\ell)$.

We next consider the first recursive call of DPLL-SSAT in case of branching. The new thresholds are achieved with a similar argument as above. The difference, however, is that we are unaware of the probability of the second branch. Let us denote by $pr' = Pr(\mathcal{Q}' : \varphi[\mathtt{true}/x])$ and by $pr'' = Pr(\mathcal{Q}' : \varphi[\mathtt{false}/x])$ the actual satisfaction probabilities of the first and second branches, respectively. The actual final probability of satisfaction then is $pr = p \cdot pr' + (1-p) \cdot pr''$. Simply by definition, $pr < \theta_l$ if and only if $p \cdot pr' + (1-p) \cdot pr'' < \theta_l$. Due to arithmetic laws and due to $p > 0$, we further conclude that $pr < \theta_l$ if and only if $p \cdot pr' < \theta_l - (1-p) \cdot pr''$ if and only if $pr' < (\theta_l - (1-p) \cdot pr'')/p$. As $pr'' \leq 1$, we finally obtain the rationale for the new lower threshold, namely from $pr' < (\theta_l - (1-p))/p$ it follows that $pr' < (\theta_l - (1-p) \cdot pr'')/p$. That is, we may use $(\theta_l - (1-p))/p$ as the new lower threshold for the first recursion. For the new upper threshold, we reason that $pr > \theta_u$ if and only if $p \cdot pr' + (1-p) \cdot pr'' > \theta_u$ if and only if $pr' > (\theta_u - (1-p) \cdot pr'')/p$. As $pr'' \geq 0$, we clearly have that if $pr' > \theta_u/p$ then $pr' > (\theta_u - (1-p) \cdot pr'')/p$ which justifies new upper threshold θ_u/p for the first recursion of DPLL-SSAT. Since $\theta_l \leq \theta_u$, $(1-p) > 0$, and $p > 0$, we deduce that $(\theta_l - (1-p))/p \leq \theta_u/p$.

We now explain the new thresholds for the second recursion. Observe that the second recursion is executed only if thresholding has failed, confer Figure 6.1. As a consequence, we know that $p \cdot pr_1 + (1-p) \geq \theta_l$ as well as $p \cdot pr_1 \leq \theta_u$, and thus that the result pr_1 of the first recursion lies within the corresponding thresholds, i.e. $pr_1 \in [(\theta_l - (1-p))/p, \theta_u/p]$. From the latter observation, we can conclude that pr_1 is the actual satisfaction probability of the first branch, i.e. $pr_1 = Pr(\mathcal{Q}' : \varphi[\texttt{true}/x])$. Let $pr' = Pr(\mathcal{Q}' : \varphi[\texttt{false}/x])$ be the actual satisfaction probability of the second branch. The actual final probability of satisfaction then is $pr = p \cdot pr_1 + (1-p) \cdot pr'$. Using the result pr_1 of the first branch, we have to weaken the lower threshold for the second branch on the one hand, but we can strengthen the corresponding upper threshold on the other hand. For this purpose, recall that $(1-p) > 0$ and consider the following line of reasoning: $pr < \theta_l$ if and only if $p \cdot pr_1 + (1-p) \cdot pr' < \theta_l$ if and only if $(1-p) \cdot pr' < \theta_l - p \cdot pr_1$ if and only if $pr' < (\theta_l - p \cdot pr_1)/(1-p)$, as well as $pr > \theta_u$ if and only if $p \cdot pr_1 + (1-p) \cdot pr' > \theta_u$ if and only if $(1-p) \cdot pr' > \theta_u - p \cdot pr_1$ if and only if $pr' > (\theta_u - p \cdot pr_1)/(1-p)$. This establishes the lower threshold $(\theta_l - p \cdot pr_1)/(1-p)$ as well as the upper threshold $(\theta_u - p \cdot pr_1)/(1-p)$ for the second recursive call of DPLL-SSAT. From $\theta_l \leq \theta_u$ and $(1-p) > 0$ it follows that $(\theta_l - p \cdot pr_1)/(1-p) \leq (\theta_u - p \cdot pr_1)/(1-p)$.

We finally remark that lower thresholds θ_l may become negative, i.e. $\theta_l < 0$, and that upper thresholds θ_u may exceed 1, i.e. $\theta_u > 1$. This fact, however, does *not* destroy soundness of thresholding as we have *never* stated any assumption on the range of θ_l and θ_u. The only condition is that $\theta_l \leq \theta_u$ holds in each invocation of DPLL-SSAT$(\Phi, \theta_l, \theta_u)$. The latter property is actually preserved for each recursive call DPLL-SSAT$(\Phi', \theta_l', \theta_u')$, i.e. $\theta_l' \leq \theta_u'$, as shown above.

Concluding this subsection about DPLL-based SSAT algorithms, we mention that further algorithmic enhancements of DPLL-SSAT were investigated in the literature, confer [Maj09]. One of them is called *non-chronological backtracking* [Maj04] that works as follows: whenever DPLL-SSAT has reached a base case, then a minimal partial variable assignment is created that serves as an explanation for the current conflict or solution. Based on this explanation, the second branch in DPLL-SSAT can be skipped whenever the value of the current variable x has *no* impact on the current conflict or solution, which is the case if the explanation does not talk about variable x. In [ML98a], a dynamic programming technique called *memoization* was investigated in order to potentially improve performance of DPLL-SSAT by caching the satisfaction probabilities of already solved subformulae. Whenever an already processed subformula Φ needs to be solved again, the corresponding cached satisfaction probability can be retrieved immediately without recomputing $Pr(\Phi)$. Another issue that may lead to enormous performance gains is the development of suitable *branching heuristics*. As mentioned above, the selection of a variable for branching in the SSAT case is more restrictive than in the quantifier-free SAT case, i.e. a decision variable needs to be selected from the leftmost block of same quantifiers in the prefix. Several sophisticated branching heuristics for DPLL-SSAT were investigated in [LMP01, ML03]. In Section 6.5, we elaborate on above-mentioned and further algorithmic optimizations for the more general SSMT case.

6.2.2 Resolution-based SSAT procedure

As reviewed in Subsection 6.2.1, classical SSAT algorithms implement a DPLL-based backtracking procedure thereby explicitly traversing the tree given by the quantifier prefix and recursively computing the individual satisfaction probabilities for each subtree. In this subsection, we propose a novel approach to solve SSAT that is based on the *resolution* principle. Following the idea of resolution for propositional and first-order formulae [Rob65] and for QBF formulae [BKF95], we develop a sound and complete resolution calculus for SSAT and theoretically compare it with the classical DPLL-SSAT approach. The results of this subsection are mainly based on [TF10, TF11, TF12].

We first recall in brief the well-known resolution calculus for propositional formulae φ in CNF, confer [Rob65]. Let $(c_1 \vee x)$ and $(c_2 \vee \neg x)$ be two clauses where c_1 and c_2 are disjunctions of literals and x is a propositional variable. Observe that x and $\neg x$ are complementary literals. The resolution rule then derives from above clauses the new clause $(c_1 \vee c_2)$ which is logically implied by the given clauses, i.e. $(c_1 \vee x) \wedge (c_2 \vee \neg x) \Rightarrow (c_1 \vee c_2)$ is valid. The derived clause $(c_1 \vee c_2)$ is also called *resolvent* of $(c_1 \vee x)$ and $(c_2 \vee \neg x)$. Starting with clauses in φ, resolution successively applies above rule to derive implied clauses. An essential property of resolution for SAT is that the empty clause $\emptyset$ is derivable from the given formula φ if and only if φ is unsatisfiable.

Similar to above scheme for the non-stochastic case, resolution for SSAT formulae $\mathcal{Q} : \varphi$ also derives new clauses $c^{(pl,pu)}$ which are however annotated with a pair of probabilities (pl, pu) where $0 \leq pl \leq pu \leq 1$. More precisely, the resolution calculus derives pairs $c^{(pl,pu)}|\mathcal{Q} : \varphi$ of annotated clauses $c^{(pl,pu)}$ and SSAT formulae $\mathcal{Q} : \varphi$. In contrast to classical resolution, such derived clauses $c^{(pl,pu)}$ need *not* be implications of the given (or rather derived) formula $\mathcal{Q} : \varphi$, but are just entailed with some probability. Loosely speaking, the derivation of a pair $c^{(pl,pu)}|\mathcal{Q} : \varphi$ means that under SSAT formula $\mathcal{Q} : \varphi$, the clause c is violated with a maximum probability at most pu, i.e. the maximum satisfaction probability of $\mathcal{Q} : (\varphi \wedge \neg c)$ is at most pu. More intuitively, the minimum probability that clause c is implied by φ is at least $1 - pu$. The latter fact follows from the observation that $Pr(\mathcal{Q} : \psi) = 1 - Pr(\mathcal{Q}' : \neg\psi)$, where $\mathcal{Q}'$ arises from $\mathcal{Q}$ by replacing existential quantifiers by universal ones, where universal quantifiers call for *minimizing* the satisfaction probability, confer the definition of XSSAT in Section 4.2. We thus have that $Pr(\mathcal{Q}' : (\varphi \Rightarrow c)) = 1 - Pr(\mathcal{Q} : (\varphi \wedge \neg c)) \geq 1 - pu$ holds. Serving the sole purpose of getting an intuition of derived pairs $c^{(pl,pu)}|\mathcal{Q} : \varphi$, the above interpretation is however too imprecise to facilitate a similar estimation for pl. Though the exact meaning of $c^{(pl,pu)}|\mathcal{Q} : \varphi$ is presented in Lemma 6.1, we are a bit more precise at this point. Let x be the rightmost variable in prefix $\mathcal{Q}$ that occurs in clause c. Further, let τ be any partial assignment that falsifies c, i.e. $\tau(c) = \texttt{false}$, and that is defined for x and all the variables to the left of x in $\mathcal{Q}$. Finally, let $\mathcal{Q}'$ be the prefix that arises when removing all variables y from $\mathcal{Q}$ for which $\tau(y)$ is defined, and φ' be the resulting matrix when substituting the values $\tau(y)$ for variables y in φ. Then, the maximum probability of satisfaction of $\mathcal{Q}' : \varphi'$ is at least pl and at most pu. Once a pair $\emptyset^{(pl,pu)}|\mathcal{Q} : \varphi$ comprising the empty clause is derived, it follows that the maximum satisfaction probability of the derived SSAT formula lies in the interval $[pl, pu]$, i.e. $pl \leq Pr(\mathcal{Q} : \varphi) \leq pu$.

Before we formally introduce the resolution calculus for SSAT, we mention the following theoretical observation that sheds some light on the nature of a potential SSAT resolution scheme. Recall that resolution for both SAT and QBF shows polynomial-time solvability on their restrictions to 2CNF as each resolution step yields clauses of size at most two, of which just quadratically many exist. Note that the same property cannot be expected for SSAT resolution due to PSPACE-completeness of S2SAT (Theorem 4.1). Therefore, a sound and complete resolution calculus for SSAT must involve some rule that destroys above property (unless P = PSPACE).

Resolution for SSAT. For presentation reasons, we first introduce some notation.

Definition 6.1
For each quantifier prefix $\mathcal{Q} = Q_1 x_1 \ldots Q_n x_n$, each propositional formula φ with $Var(\varphi) \subseteq \{x_1, \ldots, x_n\}$, and each non-tautological clause c, i.e. $\not\models c$, we define

1. *the quantifier prefix $\mathcal{Q}(\varphi)$ to be shortest prefix of $\mathcal{Q}$ that contains all variables from φ, i.e. $\mathcal{Q}(\varphi) := Q_1 x_1 \ldots Q_i x_i$ where $x_i \in Var(\varphi)$ and for each $j > i : x_j \notin Var(\varphi)$,*
2. *the set $Var(\mathcal{Q}) := \{x_1, \ldots, x_n\}$ of variables quantified by $\mathcal{Q}$, and*
3. *the partial assignment $f\!f_c$ that falsifies c as the mapping $f\!f_c : Var(c) \to \mathbb{B}$ such that for all variables $x \in Var(c)$:*

$$f\!f_c(x) := \begin{cases} \texttt{true} & \textit{if} \quad \neg x \in c \quad , \\ \texttt{false} & \textit{if} \quad x \in c \quad . \end{cases}$$

Observe that above assignment $f\!f_c$ exists and is unique since clause c is non-tautological, and that c evaluates to `false` under assignment $f\!f_c$.

In what follows, let $\mathcal{Q} : \varphi$ be an SSAT formula with matrix φ being in CNF. Without loss of generality, φ contains only non-tautological clauses, i.e. $\forall c \in \varphi : \not\models c$. With regard to the latter, note that tautological clauses c', i.e. $\models c'$, are redundant in the sense that $Pr(\mathcal{Q} : (\varphi \wedge c')) = Pr(\mathcal{Q} : \varphi)$. We furthermore demand that φ does not comprise the constants `true` and `false`. The latter requirement is also without loss of generality since for each propositional formula ψ in CNF a semantically equivalent formula ψ' in CNF can simply be achieved such that ψ' does not contain `true` and `false`, confer the notion of a *cleaning* as defined in Section 2.2.

The resolution calculus for SSAT, which we call *S-resolution*, is defined by the consecutive application of below rules R.1, R.2, R.2s, and R.3 to derive pairs $c^{(pl,pu)} | \mathcal{Q} : \varphi$ where $c^{(pl,pu)}$ with $0 \leq pl \leq pu \leq 1$ is an annotated clause and $\mathcal{Q} : \varphi$ an SSAT formula. An initial pair is given by $\varepsilon | \mathcal{Q} : \varphi$ where ε denotes absence of a clause and $\mathcal{Q} : \varphi$ is any SSAT formula that meets the aforementioned conditions. We explicitly point out that ε must *not* be confused with the empty clause $\emptyset$. We further remark that each of the above rules preserves the given SSAT formula in its derived pair, which in turn means that the information about $\mathcal{Q} : \varphi$ is redundant to some extent. We however opt for the involvement of $\mathcal{Q} : \varphi$ since we enhance S-resolution by some additional rules in Subsection 6.2.3, with some of these rules actually being able to modify the quantifier prefix $\mathcal{Q}$ of the given SSAT formula $\mathcal{Q} : \varphi$.

In case we forbid rule R.2 and thus allow only rules R.1, R.2s, and R.3, we obtain a stronger version that we refer to as *strong S-resolution*. The rationale of the stronger version is that derived clauses $c^{(pl,pu)}$ are then forced to have tight bounds pl and pu, i.e. $pl = pu$, which give the *exact* satisfaction probabilities of the corresponding subformulae, confer Lemma 6.1. Strong S-resolution furthermore establishes the basis of the procedure to compute *generalized Craig interpolants* for SSAT being introduced in Chapter 9.

Given $\varepsilon|\mathcal{Q} : \varphi$, rule R.1 derives a pair $c^{(0,0)}|\mathcal{Q} : \varphi$ where clause $c^{(0,0)}$ is annotated with the smallest possible probability pair $(0,0)$ and c itself is an original clause in φ. Referring to the semantics of SSAT given in Definition 4.2, R.1 corresponds to the quantifier-free base case where φ is equivalent to `false` under any assignment that falsifies c.

$$\text{(R.1)} \qquad \frac{\varepsilon|\mathcal{Q} : \varphi, \\ c \in \varphi}{c^{(0,0)}|\mathcal{Q} : \varphi}$$

Next rule R.2 simply takes any non-tautological disjunction c of literals that talk about variables occurring in prefix $\mathcal{Q}$, and then derives $c^{(0,1)}|\mathcal{Q} : \varphi$, where clause $c^{(0,1)}$ is annotated with the pair consisting of the smallest and highest possible probabilities 0 and 1. Soundness of R.2 follows from the trivial fact that $0 \leq Pr(\Phi) \leq 1$ is always true for each SSAT formula Φ.

$$\text{(R.2)} \qquad \frac{\varepsilon|\mathcal{Q} : \varphi, \\ c \subseteq \{x, \neg x | x \in Var(\mathcal{Q})\},\ \not\models c}{c^{(0,1)}|\mathcal{Q} : \varphi}$$

By strengthening the premise of R.2, we obtain rule R.2s. More precisely, R.2s does not take an arbitrary disjunction c of literals, but c here encodes the opposite of a satisfying (partial) assignment τ of φ. That is, substitution of the values given by τ for the corresponding variables in φ yields a tautology, i.e. $\models \varphi[\tau(x_1)/x_1]\dots[\tau(x_i)/x_i]$. Similar to R.1 and according to Definition 4.2, rule R.2s reflects the quantifier-free base case in which φ is equivalent to `true` under any (complete) assignment τ' that is conform to the partial assignment τ since $\varphi[\tau(x_1)/x_1]\dots[\tau(x_i)/x_i] \equiv$ `true`. This justifies derivation of $c^{(1,1)}|\mathcal{Q} : \varphi$, where clause $c^{(1,1)}$ is annotated with the pair $(1,1)$ of highest possible probabilities.

$$\text{(R.2s)} \qquad \frac{\begin{array}{c}\varepsilon|\mathcal{Q} : \varphi, \\ c \subseteq \{x, \neg x | x \in Var(\mathcal{Q})\},\ \not\models c, \\ \text{for each } \tau : Var(\mathcal{Q}(c)) \to \mathbb{B} \text{ with } \forall x \in Var(c) : \tau(x) = \mathit{ff}_c(x) : \\ \models \varphi[\tau(x_1)/x_1]\dots[\tau(x_i)/x_i] \text{ where } \mathcal{Q}(c) = Q_1 x_1 \dots Q_i x_i\end{array}}{c^{(1,1)}|\mathcal{Q} : \varphi}$$

We remark that finding such a clause c in the premise of R.2s is NP-hard as it is equivalent to finding a satisfying (partial) assignment of a propositional formula in CNF. This strong application condition of R.2s is however justified with regard to a potential integration of S-resolution into DPLL-SSAT solvers, since DPLL-SSAT strongly relies on finding satisfying assignments, confer the base case of DPLL-SSAT in Figure 6.1 where all clauses in φ are equivalent to `true`. Observe that whenever a satisfying (partial) assignment τ' of φ is found by a DPLL-SSAT solver then $\models \varphi[\tau'(y_1)/y_1]\dots[\tau'(y_k)/y_k]$ with $y_1, \dots, y_k \in Var(\varphi)$ being all variables for which $\tau'(y_1), \dots, \tau'(y_k)$ are defined. It is then straightforward to construct from τ' a clause c which meets the requirements of R.2s, namely for each $x \in Var(\mathcal{Q}) : x \in c$ if and only if $\tau'(x) =$ `false`, and $\neg x \in c$ if and only if $\tau'(x) =$ `true`.

As discussed later on, we actually aim at an integration of S-resolution into DPLL-SSAT in order to enhance performance as well as applicability of DPLL-SSAT solvers.

Note that such above rules R.2 as well as R.2s are not present in classical resolution schemes for SAT and QBF. In the stochastic case, we need one of these rules for achieving completeness of S-resolution being evinced by Theorem 6.1. Note that each of both rules impedes a potential polynomial-time solvability for SSAT formulae in 2CNF as sizes of derived clauses are no longer guaranteed to be at most two. Though completeness might be obtained in another way, the choice of rule R.2 or of rule R.2s seems to be justified by Theorem 4.1 which states PSPACE-completeness of S2SAT, indicating that a polynomial time algorithm for S2SAT cannot be expected.

Rule R.3 finally constitutes the actual resolution rule as known from the non-stochastic case. Depending on whether an existential or a randomized variable is resolved upon, both probabilities pl and pu of the probability pair (pl, pu) of the resolvent clause are computed according to the semantics $Pr(\mathcal{Q} : \varphi)$ as given in Definition 4.2.

$$\text{(R.3)}\quad \frac{\begin{array}{c}(c_1 \vee \neg x)^{(pl_1,pu_1)}|\mathcal{Q} : \varphi,\ (c_2 \vee x)^{(pl_2,pu_2)}|\mathcal{Q} : \varphi,\\ Qx \in \mathcal{Q},\ x \notin Var(\mathcal{Q}(c_1 \vee c_2)),\ \not\models (c_1 \vee c_2),\\ (pl,pu) = \begin{cases} (\max(pl_1,pl_2), \max(pu_1,pu_2)) & ; Q = \exists, \\ (p \cdot pl_1 + (1-p) \cdot pl_2, p \cdot pu_1 + (1-p)\cdot pu_2) & ; Q = \text{Я}^p \end{cases}\end{array}}{(c_1 \vee c_2)^{(pl,pu)}|\mathcal{Q} : \varphi}$$

Note that the SSAT formulae in both input pairs $(c_1 \vee \neg x)^{(pl_1,pu_1)}|\mathcal{Q} : \varphi$ and $(c_2 \vee x)^{(pl_2,pu_2)}|\mathcal{Q} : \varphi$ need to be the same and that $\mathcal{Q} : \varphi$ is preserved in the derived pair $(c_1 \vee c_2)^{(pl,pu)}|\mathcal{Q} : \varphi$. Moreover, observe that variable x which is resolved upon does not precede within prefix $\mathcal{Q}$ any of the variables in c_1 and c_2, i.e. x is the *rightmost* variable in both prefixes $\mathcal{Q}(c_1 \vee \neg x)$ and $\mathcal{Q}(c_2 \vee x)$. Though this application condition might seem too restrictive, it is actually not the case with respect to soundness and completeness of S-resolution as subsequently shown by Corollary 6.1 and Theorem 6.1.

For the purpose of comparing the proof complexity of S-resolution and DPLL-SSAT, we actually relax above restriction to some extent in Subsection 6.2.3: in special cases, namely whenever literal $\neg x$ or x is unit or pure under each assignment that falsifies some subset c of c_1 or c_2, respectively, it is allowed to delay resolution upon variable x and to resolve upon variables $y \in c_1 \setminus c$ or $y \in c_2 \setminus c$ first. The latter fact is formalized *implicitly* by the additional rules R.3u and R.3p later on, namely by means of moving Qx to the right within prefix $\mathcal{Q}$, i.e. the quantifier prefix actually changes in the derived pair.

We abbreviate the derivation of a pair $c^{(pl,pu)}|\mathcal{Q} : \varphi$

- from $\varepsilon|\mathcal{Q} : \varphi$ by rule R.1 as $\varepsilon|\mathcal{Q} : \varphi \vdash_{\mathsf{R.1}} c^{(pl,pu)}|\mathcal{Q} : \varphi$,
- from $\varepsilon|\mathcal{Q} : \varphi$ by rule R.2 as $\varepsilon|\mathcal{Q} : \varphi \vdash_{\mathsf{R.2}} c^{(pl,pu)}|\mathcal{Q} : \varphi$,
- from $\varepsilon|\mathcal{Q} : \varphi$ by rule R.2s as $\varepsilon|\mathcal{Q} : \varphi \vdash_{\mathsf{R.2s}} c^{(pl,pu)}|\mathcal{Q} : \varphi$, and
- from $c_1^{(pl_1,pu_1)}|\mathcal{Q} : \varphi$ and $c_2^{(pl_2,pu_2)}|\mathcal{Q} : \varphi$ by R.3 as $(c_1^{(pl_1,pu_1)}|\mathcal{Q} : \varphi, c_2^{(pl_2,pu_2)}|\mathcal{Q} : \varphi) \vdash_{\mathsf{R.3}} c^{(pl,pu)}|\mathcal{Q} : \varphi$.

To show soundness of S-resolution as well as of strong S-resolution, we first state the following lemma that permits a precise interpretation of derived pairs $c^{(pl,pu)}|\mathcal{Q} : \varphi$.

Lemma 6.1 (Interpretation of derived pairs $c^{(pl,pu)}|\mathcal{Q}:\varphi$)
Let pair $c^{(pl,pu)}|\mathcal{Q}:\varphi$ with $\mathcal{Q}=Q_1x_1\dots Q_nx_n$ and $\mathcal{Q}(c)=Q_1x_1\dots Q_ix_i$ be derivable by S-resolution. Then, for each truth assignment $\tau : \mathit{Var}(\mathcal{Q}(c))\to\mathbb{B}$ with $\forall x\in \mathit{Var}(c) : \tau(x)=\mathit{ff}_c(x)$ it holds that

$$pl \;\leq\; Pr(Q_{i+1}x_{i+1}\dots Q_nx_n : \varphi[\tau(x_1)/x_1]\dots[\tau(x_i)/x_i]) \;\leq\; pu \quad ,$$

and, moreover, if $c^{(pl,pu)}|\mathcal{Q}:\varphi$ is derivable by strong S-resolution then $pl=pu$, i.e.

$$pl \;=\; Pr(Q_{i+1}x_{i+1}\dots Q_nx_n : \varphi[\tau(x_1)/x_1]\dots[\tau(x_i)/x_i]) \;=\; pu \quad .$$

Proof. First of all, observe that clause c in each pair $c^{(pl,pu)}|\mathcal{Q}:\varphi$ derivable by S-resolution is non-tautological, i.e. $\not\models c$. The rationale is that each clause in φ is non-tautological by global assumption and, therefore, each rule can only produce non-tautological clauses. As a consequence, each above truth assignment τ is well-defined.

We show the lemma by induction over the application of rules R.1, R.2, R.2s, and R.3. The base cases are given by rules R.1, R.2, and R.2s. For rule R.1, i.e. $\varepsilon|\mathcal{Q}:\varphi \vdash_{\mathsf{R.1}} c^{(0,0)}|\mathcal{Q}:\varphi$, the formula $\varphi[\tau(x_1)/x_1]\dots[\tau(x_i)/x_i]$ is unsatisfiable due to construction of τ that falsifies clause $c\in\varphi$. Thus,

$$0 \;=\; Pr(Q_{i+1}x_{i+1}\dots Q_nx_n : \varphi[\tau(x_1)/x_1]\dots[\tau(x_i)/x_i]) \;=\; 0 \quad .$$

For rule R.2, i.e. $\varepsilon|\mathcal{Q}:\varphi \vdash_{\mathsf{R.2}} c^{(0,1)}|\mathcal{Q}:\varphi$, we trivially obtain that

$$0 \;\leq\; Pr(Q_{i+1}x_{i+1}\dots Q_nx_n : \varphi[\tau(x_1)/x_1]\dots[\tau(x_i)/x_i]) \;\leq\; 1$$

holds according to Definition 4.2. For rule R.2s, i.e. $\varepsilon|\mathcal{Q}:\varphi \vdash_{\mathsf{R.2s}} c^{(1,1)}|\mathcal{Q}:\varphi$, we immediately have that formula $\varphi[\tau(x_1)/x_1]\dots[\tau(x_i)/x_i]$ is a tautology and therefore that

$$1 \;=\; Pr(Q_{i+1}x_{i+1}\dots Q_nx_n : \varphi[\tau(x_1)/x_1]\dots[\tau(x_i)/x_i]) \;=\; 1 \quad .$$

The fact that rule R.2 is not applicable within strong S-resolution establishes the result for the base cases.

Now assume that the respective assumptions hold for the pairs in the premise of R.3. That is, if $((c_1\vee\neg x_j)^{(pl_1,pu_1)}|\mathcal{Q}:\varphi, (c_2\vee x_j)^{(pl_2,pu_2)}|\mathcal{Q}:\varphi) \vdash_{\mathsf{R.3}} c^{(pl,pu)}|\mathcal{Q}:\varphi$ then for each truth assignment $\tau_1 : \mathit{Var}(\mathcal{Q}(c_1\vee\neg x_j))\to\mathbb{B}$ with $\forall x\in \mathit{Var}(c_1\vee\neg x_j) : \tau_1(x)=\mathit{ff}_{(c_1\vee\neg x_j)}(x)$ and for each truth assignment $\tau_2 : \mathit{Var}(\mathcal{Q}(c_2\vee x_j))\to\mathbb{B}$ with $\forall x\in \mathit{Var}(c_2\vee x_j) : \tau_2(x)=\mathit{ff}_{(c_2\vee x_j)}(x)$ it holds that

$$\begin{aligned}
pl_1 &\;\sim\; Pr(Q_{j+1}x_{j+1}\dots Q_nx_n : \varphi[\tau_1(x_1)/x_1]\dots[\tau_1(x_{j-1})/x_{j-1}][\tau_1(x_j)/x_j]) \;\sim\; pu_1 \quad ,\\
pl_2 &\;\sim\; Pr(Q_{j+1}x_{j+1}\dots Q_nx_n : \varphi[\tau_2(x_1)/x_1]\dots[\tau_2(x_{j-1})/x_{j-1}][\tau_2(x_j)/x_j]) \;\sim\; pu_2 \quad ,
\end{aligned}$$

where $\sim$ is $\leq$ for S-resolution and $\sim$ is $=$ for strong S-resolution. Observe that $j\geq i+1$ since $x_j\notin \mathit{Var}(\mathcal{Q}(c_1\vee c_2))$ and $c=(c_1\vee c_2)$, and that each truth assignment τ with $\tau(x)=\tau_1(x)$ if $x\in \mathit{Var}(c_1)$ and $\tau(x)=\tau_2(x)$ if $x\in \mathit{Var}(c_2)$ is well-defined. The latter holds due to the fact that if $x\in \mathit{Var}(c_1)\cap \mathit{Var}(c_2)$ then $\tau_1(x)=\tau_2(x)$ since $\not\models (c_1\vee c_2)$. We furthermore conclude that $\tau_1(x_j)=\texttt{true}$ and $\tau_2(x_j)=\texttt{false}$ since $\tau_1(x_j)=\mathit{ff}_{(c_1\vee\neg x_j)}(x_j)$

and $\tau_2(x_j) = \mathit{ff}_{(c_2 \vee x_j)}(x_j)$ by construction. From Definition 4.2, we may then infer that for each assignment τ as defined above it holds that

$$pl \ \sim \ Pr(Q_j x_j \ Q_{j+1} x_{j+1} \ldots Q_n x_n : \varphi[\tau(x_1)/x_1] \ldots [\tau(x_{j-1})/x_{j-1}]) \ \sim \ pu \ .$$

The lemma directly follows in case $j = i+1$. For $j > i+1$, note that variables $x_{i+1}, \ldots, x_{j-1}$ do not occur in clause $c = (c_1 \vee c_2)$. Hence, for $k = j-1$ down to $i+1$ we successively conclude that

$$pl \ \sim \ Pr(Q_{k+1} x_{k+1} \ldots Q_n x_n : \varphi[\tau(x_1)/x_1] \ldots [\tau(x_{k-1})/x_{k-1}][\mathtt{true}/x_k]) \ \sim \ pu \ ,$$
$$pl \ \sim \ Pr(Q_{k+1} x_{k+1} \ldots Q_n x_n : \varphi[\tau(x_1)/x_1] \ldots [\tau(x_{k-1})/x_{k-1}][\mathtt{false}/x_k]) \ \sim \ pu \ .$$

From case $k = i+1$, we finally achieve

$$pl \ \sim \ Pr(Q_{i+1} x_{i+1} \ldots Q_n x_n : \varphi[\tau(x_1)/x_1] \ldots [\tau(x_i)/x_i]) \ \sim \ pu$$

and the lemma follows. □

As a direct consequence of Lemma 6.1, namely for special case $c^{(pl,pu)}|\mathcal{Q} : \varphi = \emptyset^{(pl,pu)}|\mathcal{Q} : \varphi$ (where τ is the well-defined empty function), we obtain that S-resolution and strong S-resolution are sound in the following sense.

Corollary 6.1 (Soundness of S-resolution)
If the pair $\emptyset^{(pl,pu)}|\mathcal{Q} : \varphi$ is derivable by S-resolution or by strong S-resolution then $pl \leq Pr(\mathcal{Q} : \varphi) \leq pu$ or $pl = Pr(\mathcal{Q} : \varphi) = pu$, respectively.

In case we explicitly refer to *strong* S-resolution then the clauses in all derived pairs $c^{(pl,pu)}|\mathcal{Q} : \varphi$ carry tight bounds, i.e. $pl = pu$, according to Lemma 6.1. For the sake of simplicity, we then write

- $\varepsilon|\mathcal{Q} : \varphi \vdash_{\mathsf{R.1}} c^p|\mathcal{Q} : \varphi$ for $\varepsilon|\mathcal{Q} : \varphi \vdash_{\mathsf{R.1}} c^{(pl,pu)}|\mathcal{Q} : \varphi$,
- $\varepsilon|\mathcal{Q} : \varphi \vdash_{\mathsf{R.2s}} c^p|\mathcal{Q} : \varphi$ for $\varepsilon|\mathcal{Q} : \varphi \vdash_{\mathsf{R.2s}} c^{(pl,pu)}|\mathcal{Q} : \varphi$, and
- $(c_1^{p_1}|\mathcal{Q} : \varphi, c_2^{p_2}|\mathcal{Q} : \varphi) \vdash_{\mathsf{R.3}} c^p|\mathcal{Q} : \varphi$ for $(c_1^{(pl_1,pu_1)}|\mathcal{Q} : \varphi, c_2^{(pl_2,pu_2)}|\mathcal{Q} : \varphi) \vdash_{\mathsf{R.3}} c^{(pl,pu)}|\mathcal{Q} : \varphi$

where $p = pl = pu$, $p_1 = pl_1 = pu_1$, and $p_2 = pl_2 = pu_2$. To further simplify notation, whenever SSAT formula $\mathcal{Q} : \varphi$ is clear from the context we omit $\mathcal{Q} : \varphi$, i.e. S-resolution rules are then denoted by

- $\vdash_{\mathsf{R.1}} c^{(pl,pu)}$,
- $\vdash_{\mathsf{R.2}} c^{(pl,pu)}$,
- $\vdash_{\mathsf{R.2s}} c^{(pl,pu)}$, and
- $(c_1^{(pl_1,pu_1)}, c_2^{(pl_2,pu_2)}) \vdash_{\mathsf{R.3}} c^{(pl,pu)}$.

We use above simplifications in the remainder of this subsection as well as in Chapter 9.

Theorem 6.1 shows completeness of strong S-resolution from which completeness of S-resolution immediately follows as the latter allows one more rule.

Theorem 6.1 (Completeness of (strong) S-resolution)
If $Pr(\mathcal{Q} : \varphi) = p$ for some SSAT formula $\mathcal{Q} : \varphi$ with φ being in CNF then the pair $\emptyset^p|\mathcal{Q} : \varphi$, is derivable by strong S-resolution.

Proof. We prove the theorem by induction over the number of quantifiers in the quantifier prefix $\mathcal{Q}$. For the base case $\mathcal{Q} = \varepsilon$, we distinguish two cases. First, $\varphi \equiv \texttt{false}$. Then, φ must contain the empty clause, i.e. $\emptyset \in \varphi$. As a consequence, $p = 0$ and the empty clause $\emptyset^0$ is derivable by rule $\mathsf{R.1}$, i.e. $\varepsilon|\mathcal{Q} : \varphi \vdash_{\mathsf{R.1}} \emptyset^0|\mathcal{Q} : \varphi$. Second, $\varphi \equiv \texttt{true}$. Then, φ does not contain any clause. Clearly, $p = 1$ and the empty clause $\emptyset^1$ is derivable by rule $\mathsf{R.2s}$, i.e. $\varepsilon|\mathcal{Q} : \varphi \vdash_{\mathsf{R.2s}} \emptyset^1|\mathcal{Q} : \varphi$. For the latter step, note that $\not\models \emptyset$.

In the induction step, we show that $\emptyset^p|Qx\,\mathcal{Q} : \varphi$ is derivable such that $p = Pr(Qx\,\mathcal{Q} : \varphi)$ by means of induction hypothesis that $\emptyset^{p_1}|\mathcal{Q} : \varphi[\texttt{true}/x]$ and $\emptyset^{p_2}|\mathcal{Q} : \varphi[\texttt{false}/x]$ are derivable with $p_1 = Pr(\mathcal{Q} : \varphi[\texttt{true}/x])$ and $p_2 = Pr(\mathcal{Q} : \varphi[\texttt{false}/x])$. Without loss of generality, we demand that the formulae $\varphi[\texttt{true}/x]$ and $\varphi[\texttt{false}/x]$ are syntactically represented by their *cleanings* as defined in Section 2.2, i.e. by the formulae that arise when removing the constants `true` and `false`. Applying the same strong S-resolution sequence deriving $\emptyset^{p_1}|\mathcal{Q} : \varphi[\texttt{true}/x]$ on $Qx\,\mathcal{Q} : \varphi$ yields

$$\emptyset^{p_1}|Qx\,\mathcal{Q} : \varphi \quad \text{or} \quad (\neg x)^{p_1}|Qx\,\mathcal{Q} : \varphi \quad .$$

With regard to the latter observation, we remark that all clauses $c \in \varphi$ containing positive literal x "disappeared" in the cleaning of $\varphi[\texttt{true}/x]$ as $\texttt{true} \in c[\texttt{true}/x]$. Analogously,

$$\emptyset^{p_2}|Qx\,\mathcal{Q} : \varphi \quad \text{or} \quad (x)^{p_2}|Qx\,\mathcal{Q} : \varphi$$

is derivable. If $\emptyset^{p_1}|Qx\,\mathcal{Q} : \varphi$ or $\emptyset^{p_2}|Qx\,\mathcal{Q} : \varphi$ is derivable then $p = p_1$ or $p = p_2$, respectively, by Corollary 6.1. Note that if both $\emptyset^{p_1}|Qx\,\mathcal{Q} : \varphi$ and $\emptyset^{p_2}|Qx\,\mathcal{Q} : \varphi$ are derivable then $p_1 = p_2$. Otherwise, i.e. only $(\neg x)^{p_1}|Qx\,\mathcal{Q} : \varphi$ and $(x)^{p_2}|Qx\,\mathcal{Q} : \varphi$ are derivable, we apply

$$((\neg x)^{p_1}|Qx\,\mathcal{Q} : \varphi, (x)^{p_2}|Qx\,\mathcal{Q} : \varphi) \vdash_{\mathsf{R.3}} \emptyset^p|Qx\,\mathcal{Q} : \varphi$$

to obtain the desired result. □

We remark that, first, S-resolution as defined above is a generalization of the SSAT resolution calculus presented in [TF10] where derived clauses only provide upper probability bounds, and that, second, strong S-resolution coincides with the resolution scheme proposed in [TF11, TF12].

Termination. With regard to *termination*, we remark that, given any SSAT formula $\mathcal{Q} : \varphi$ with φ being in CNF, it is easy to devise a strategy of rule applications such that (strong) S-resolution derives the pair $\emptyset^{(pl,pu)}|\mathcal{Q} : \varphi$ comprising the empty clause and satisfying $pl = pu = Pr(\mathcal{Q} : \varphi)$ after *finitely* many steps. The rationale is as follows. Recall that $Var(\mathcal{Q})$ is finite and therefore the number of clauses in φ as well as the number of literals in each clause are finite. Then, the application conditions of each rule are decidable. Furthermore, rules $\mathsf{R.1}$, $\mathsf{R.2}$, and $\mathsf{R.2s}$ are only able to derive finitely many different pairs $c^{(pl,pu)}|\mathcal{Q} : \varphi$ for the same initial pair $\varepsilon|\mathcal{Q} : \varphi$. Finally, rule $\mathsf{R.3}$ can produce from finitely many pairs $c^{(pl,pu)}|\mathcal{Q} : \varphi$ only finitely many different pairs as well. To obtain a termination strategy for S-resolution, it hence suffices to fix the initial pair $\varepsilon|\mathcal{Q} : \varphi$ and to ensure that each pair $c^{(pl,pu)}|\mathcal{Q} : \varphi$ is derived at most once.

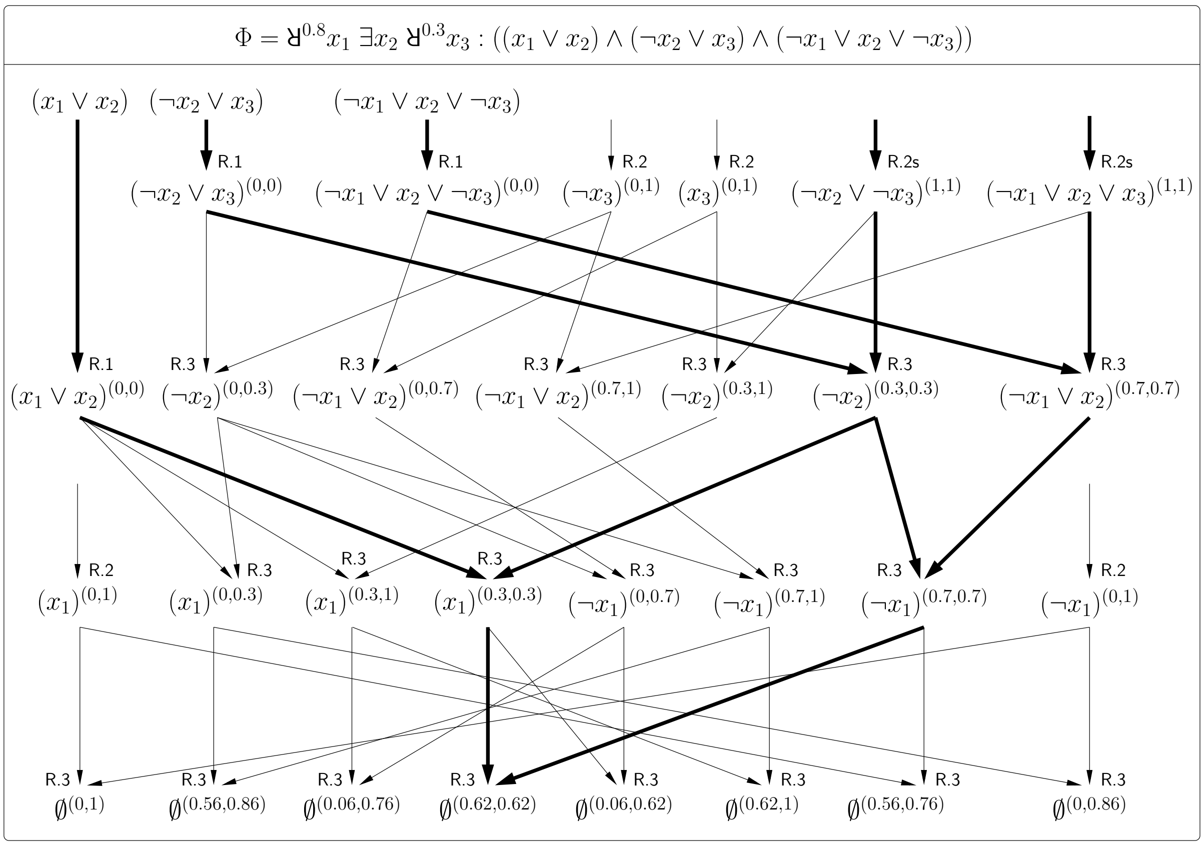


Figure 6.3: Different derivations of pairs $\emptyset^{(pl,pu)}|\Phi$ comprising the empty clause by S-resolution. Arrows denote applications of the specified resolution rules.

Example of S-resolution. Consider the SSAT formula

$$\Phi = Я^{0.8}x_1\ \exists x_2\ Я^{0.3}x_3 : ((x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)) \ .$$

Figure 6.3 shows different derivations of pairs $\emptyset^{(pl,pu)}|\Phi$ comprising the empty clause by S-resolution. Due to Corollary 6.1, each $\emptyset^{(pl,pu)}|\Phi$ proves a lower as well as an upper bound on the satisfaction probability of Φ, i.e. $pl \leq Pr(\Phi) \leq pu$. As the empty clause in $\emptyset^{(0.62,0.62)}|\Phi$ carries tight bounds, we deduce that $Pr(\Phi) = 0.62$. Observe that generation of $\emptyset^{(0.62,0.62)}|\Phi$ involves rules R.1, R.2s, and R.3 only, confer bold arrows in Figure 6.3. Concerning steps $\vdash_{\mathsf{R.2s}} (\neg x_2 \vee \neg x_3)^{(1,1)}$ and $\vdash_{\mathsf{R.2s}} (\neg x_1 \vee x_2 \vee x_3)^{(1,1)}$, note that for (partial) assignments τ_1 with $\tau_1(x_2) = \texttt{true}$, $\tau_1(x_3) = \texttt{true}$, and τ_2 with $\tau_2(x_1) = \texttt{true}$, $\tau_2(x_2) = \texttt{false}$, $\tau_2(x_3) = \texttt{false}$ it holds that $\models \varphi[\tau_1(x_2)/x_2][\tau_1(x_3)/x_3]$ and $\models \varphi[\tau_2(x_1)/x_1][\tau_2(x_2)/x_2][\tau_2(x_3)/x_3]$, respectively. That is, pair $\emptyset^{(0.62,0.62)}|\Phi$ is derivable by strong S-resolution. Such a derivation of $\emptyset^{(p,p)}|\Phi$ with $p = Pr(\Phi)$ by (strong) S-resolution always exists due to Theorem 6.1.

6.2.3 Theoretical comparison between S-resolution and DPLL-SSAT

To shed a bit more light on the computational behavior of S-resolution and its relation to classical SSAT methods, we theoretically compare S-resolution with the standard SSAT procedure DPLL-SSAT from Figure 6.1. As both approaches are sound and complete, we are interested in their *proof complexity*, i.e. the size of their proofs. For this purpose, we consider the following generalized version of the SSAT decision problem: for a given SSAT formula Φ with its matrix being in CNF, a lower threshold θ_l, and an upper threshold θ_u with $\theta_l \leq \theta_u$, compute $Pr(\Phi)$ if $Pr(\Phi) \in [\theta_l, \theta_u]$ or decide whether $Pr(\Phi) < \theta_l$ or $Pr(\Phi) > \theta_u$ holds otherwise, i.e. if $Pr(\Phi) \notin [\theta_l, \theta_u]$. We abbreviate an instance of above problem as $(\Phi, \theta_l, \theta_u)$. Observe that the original SSAT decision problem (Φ, θ), i.e. to decide whether $Pr(\Phi) \geq \theta$, as introduced in Subsection 4.2.1 is a special case of $(\Phi, \theta_l, \theta_u)$ in the sense that (Φ, θ) is true if and only if the result of (Φ, θ, θ) is θ if $Pr(\Phi) \in [\theta, \theta]$ or that $Pr(\Phi) > \theta$ holds if $Pr(\Phi) \notin [\theta, \theta]$.

A *proof* for an instance $(\Phi, \theta_l, \theta_u)$ is given by a terminating execution of a sound and complete method for the generalized SSAT decision problem. Observe that invocation of DPLL-SSAT$(\Phi, \theta_l, \theta_u)$ precisely solves above problem. As a consequence, each execution of DPLL-SSAT$(\Phi, \theta_l, \theta_u)$ is a proof for $(\Phi, \theta_l, \theta_u)$. With regard to S-resolution, a proof for $(\Phi, \theta_l, \theta_u)$ is a sequence of rule applications that derives a pair $\emptyset^{(pl,pu)}|\Phi$ comprising the empty clause with $pl = pu = Pr(\Phi)$ if $Pr(\Phi) \in [\theta_l, \theta_u]$, $pu < \theta_l$ if $Pr(\Phi) < \theta_l$, and $pl > \theta_u$ if $Pr(\Phi) > \theta_u$. By soundness of S-resolution, the reverse directions are also given in the following sense: if $pl = pu \in [\theta_l, \theta_u]$ then $pl = pu = Pr(\Phi) \in [\theta_l, \theta_u]$, if $pu < \theta_l$ then $Pr(\Phi) \leq pu < \theta_l$, and if $pl > \theta_u$ then $Pr(\Phi) \geq pl > \theta_u$. We define the *size of a proof* as the number of “essential” proof steps, i.e. the number of recursions for DPLL-SSAT and the number of rule applications for S-resolution.

We remark that both DPLL-SSAT as well as S-resolution may produce proofs of *exponential* size in *worst case*: for the DPLL-SSAT case, we point to the family of SSAT formulae given in Proposition 6.1. For S-resolution, consider the very simple family of SSAT instances $(\Phi_{n \in \mathbb{N}_{>0}}, \theta_l, \theta_u)$ with

$$\Phi_{n \in \mathbb{N}_{>0}} = \rotatebox[origin=c]{180}{R}^{0.5} x_1 \ldots \rotatebox[origin=c]{180}{R}^{0.5} x_n : ((x_1) \wedge \ldots \wedge (x_n))$$

and some $\theta_l \leq \theta_u$ where $\mathbb{N}_{>0} := \mathbb{N} \setminus \{0\}$. It is not hard to see that $Pr(\Phi_{n \in \mathbb{N}_{>0}}) = 0.5^n$. The shortest S-resolution proof clearly is of linear size and involves

$$\varepsilon|\Phi_{n \in \mathbb{N}_{>0}} \vdash_{\mathsf{R.2s}} (\neg x_1 \vee \ldots \vee \neg x_n)^{(1,1)}|\Phi_{n \in \mathbb{N}_{>0}}$$

followed by n applications of rule R.3, namely for $i = n$ down to 1:

$$\left(\begin{array}{l} (\neg x_1 \vee \ldots \vee \neg x_i)^{(0.5^{n-i}, 0.5^{n-i})}|\Phi_{n \in \mathbb{N}_{>0}}, \\ (x_i)^{(0,0)}|\Phi_{n \in \mathbb{N}_{>0}} \end{array} \right) \vdash_{\mathsf{R.3}} (\neg x_1 \vee \ldots \vee \neg x_{i-1})^{(0.5^{n-i+1}, 0.5^{n-i+1})}|\Phi_{n \in \mathbb{N}_{>0}}$$

which finally yields $\emptyset^{(0.5^n, 0.5^n)}|\Phi_{n \in \mathbb{N}_{>0}}$. Note that each pair $(x_i)^{(0,0)}|\Phi_{n \in \mathbb{N}_{>0}}$ is derivable by one application of rule R.1. Some S-resolution proofs for $(\Phi_{n \in \mathbb{N}_{>0}}, \theta_l, \theta_u)$, however, may also comprise “needless” resolution steps like

$$\varepsilon|\Phi_{n \in \mathbb{N}_{>0}} \vdash_{\mathsf{R.2}} (\neg x_{k_1} \vee \ldots \vee \neg x_{k_j})^{(0,1)}|\Phi_{n \in \mathbb{N}_{>0}}$$

with $1 \leq k_1 < \ldots < k_j \leq n$. The number of different pairs derived by such latter steps is given by the number of all combinations of i (different) literals chosen from the set

$\{\neg x_1, \ldots, \neg x_n\}$ for all $i \in \{1, \ldots, n\}$, which is $\sum_{i=1}^{n} \binom{n}{i} = 2^n - 1$. This shows that S-resolution proofs may be of exponential size in worst case. Our particular interest thus is to investigate *shortest* proofs.

In what follows, we show that S-resolution is capable of producing proofs for above SSAT problem that are *never longer* and sometimes even *significantly shorter* than the shortest proofs generated by DPLL-SSAT. More precisely, we establish the following results.

1. For any, in particular the shortest, DPLL-SSAT$(\Phi, \theta_l, \theta_u)$ proof of length k it is possible to devise a strategy of rule applications such that the resulting S-resolution proof is also of size k.
2. There are infinitely many SSAT formulae Φ such that all and particularly the shortest DPLL-SSAT$(\Phi, \theta_l, \theta_u)$ proofs (with $\theta_u \geq 0$) are of exponential size while the shortest S-resolution proofs for the same instances $(\Phi, \theta_l, \theta_u)$ are of constant size.

Concerning item 1, we devise a strategy of rule applications that is based on the corresponding DPLL-SSAT proof in order to produce an S-resolution proof of the desired size. A bit more precisely, such a strategy is determined by an integration of S-resolution into the DPLL-SSAT algorithm. For that purpose and to improve clarity of the technical presentation, we introduce an *enhanced* version of S-resolution which directly permits to describe all algorithmic optimizations of DPLL-SSAT. Item 1 is finally demonstrated by Corollary 6.2. With regard to item 2, Proposition 6.1 states explicitly an infinite family of SSAT instances for which the shortest S-resolution proofs are of constant size while DPLL-SSAT needs exponentially many computation steps even in best case.

We first show that item 1 holds. Before addressing the underlying integration of S-resolution into DPLL-SSAT, we present the enhanced version of S-resolution: *enhanced S-resolution* extends S-resolution, given by rules R.1, R.2, R.2s, and R.3, by four additional rules R.1g, R.3t, R.3u, and R.3p.

Enhanced S-resolution. Rule R.1g is a generalization of rule R.1 in the sense that clause c' in the derived pair $(c')^{(0,0)}|\mathcal{Q} : \varphi$ does not necessarily coincide with an original clause $c \in \varphi$ but c' is a potentially proper superset of c. Obviously, whenever $\varepsilon|\mathcal{Q} : \varphi \vdash_{\mathsf{R.1}} c^{(pl,pu)}|\mathcal{Q} : \varphi$ then also $\varepsilon|\mathcal{Q} : \varphi \vdash_{\mathsf{R.1g}} c^{(pl,pu)}|\mathcal{Q} : \varphi$. One may also think of R.1g as a special case of R.2 where clause c in the derived pair $c^{(pl,pu)}|\mathcal{Q} : \varphi$ must be a superset of some original clause in φ. Clearly, $\varepsilon|\mathcal{Q} : \varphi \vdash_{\mathsf{R.1g}} c^{(0,0)}|\mathcal{Q} : \varphi$ implies $\varepsilon|\mathcal{Q} : \varphi \vdash_{\mathsf{R.2}} c^{(0,1)}|\mathcal{Q} : \varphi$. To some extent, it is unreasonable to derive a clause $(c')^{(0,0)}$ by R.1g although it is possible to derive a shorter clause $c^{(0,0)}$ with $c \subset c'$ by R.1, which gives a clear argument against R.1g. The only reason of providing R.1g is to simplify some technicalities in the proof of Lemma 6.4 that states correctness of the integration of S-resolution into DPLL-SSAT.

$$\text{(R.1g)} \qquad \frac{\begin{array}{c}\varepsilon|\mathcal{Q} : \varphi, \\ c \in \varphi,\ c \subseteq c' \subseteq \{x, \neg x | x \in \mathit{Var}(\mathcal{Q})\},\ \not\models c'\end{array}}{(c')^{(0,0)}|\mathcal{Q} : \varphi}$$

The three rules R.3t, R.3u, and R.3p can be considered as special cases of rule R.3 in the sense that only one input pair $(c \vee \ell)^{(pl,pu)}|\mathcal{Q} : \varphi$ needs to be explicitly derived by

S-resolution. Derivability of the second input pair $(c' \vee neg(\ell))^{(pl',pu')}|\mathcal{Q} : \varphi$ is then ensured by the application conditions of the rules. Recall that $neg(\ell)$ returns the opposite literal of ℓ. Rules R.3t, R.3u, and R.3p are devised to instantaneously reflect the algorithmic optimizations *thresholding*, *unit propagation*, and *purification*, respectively, which are implemented in DPLL-SSAT.

The next rule R.3t characterizes thresholding within S-resolution. Let $(c \vee \ell)^{(pl,pu)}|\mathcal{Q} : \varphi$ be the input pair and let us assume that the probability pair (pl, pu) is “good” enough to obtain a sufficient probability result (pl', pu') for the derived pair $c^{(pl',pu')}|\mathcal{Q} : \varphi$ independent of any other input pair. Concerning the integration of S-resolution into DPLL-SSAT, the latter assessment is performed by means of the application conditions of thresholding, confer Figure 6.1. Due to rule R.2, we may produce the pair $(neg(\ell))^{(0,1)}|\mathcal{Q} : \varphi$ where clause $(neg(\ell))^{(0,1)}$ carries the most conservative lower and upper probability bounds 0 and 1. By step $((c \vee \ell)^{(pl,pu)}|\mathcal{Q} : \varphi, (neg(\ell))^{(0,1)})|\mathcal{Q} : \varphi \vdash_{\mathsf{R.3}} c^{(pl',pu')}|\mathcal{Q} : \varphi$, we achieve the derived pair involving a conservative but–as assumed above–sufficient probability pair (pl', pu').

$$\text{(R.3t)} \quad \frac{\begin{array}{c}(c \vee \ell)^{(pl,pu)}|\mathcal{Q} : \varphi \\ \ell \in \{x, \neg x\},\ Qx \in \mathcal{Q},\ x \notin Var(\mathcal{Q}(c)), \\ (pl', pu') = \begin{cases} (pl, 1) & ; Q = \exists, \\ (p \cdot pl, p \cdot pu + (1-p)) & ; Q = \text{Я}^p, \ell = \neg x, \\ ((1-p) \cdot pl, (1-p) \cdot pu + p) & ; Q = \text{Я}^p, \ell = x \end{cases}\end{array}}{c^{(pl',pu')}|\mathcal{Q} : \varphi}$$

To be prepared for the integration of enhanced S-resolution into DPLL-SSAT, we additionally need to cope with the operation of *moving quantifiers* within the quantifier prefix as it is done within DPLL-SSAT for unit propagation and purification, confer Figure 6.1 and the associated explanations in Subsection 6.2.1. The remaining two rules R.3u and R.3p describing unit propagation and purification, respectively, therefore take care of abovementioned issue.

To cope with unit propagation, rule R.3u takes some derived pair $(c \vee \ell)^{(pl,pu)}|\mathcal{Q} : \varphi$ with $\mathcal{Q} = \mathcal{Q}'\, Qx\, \mathcal{Q}_1\, \mathcal{Q}_2$ and $\ell \in \{x, \neg x\}$ as an input and then checks whether some clause $(c' \vee neg(\ell))$ is present in φ that is unit under partial assignment ff_c, i.e. $c' \subseteq c$. Then, $neg(\ell)$ must be the unit literal in unit clause $(c' \vee neg(\ell))$ since $\mathit{ff}_c(x)$ is not defined due to $x \notin Var(c)$. In terms of DPLL-SSAT, unit literal $neg(\ell)$ is deduced in order to satisfy the formula. The probability result of the corresponding SSAT subformula is taken into account by the input pair $(c \vee \ell)^{(pl,pu)}|\mathcal{Q} : \varphi$. Due to presence of unit clause $(c' \vee neg(\ell)) \in \varphi$, application of rule R.1 gives $(c' \vee neg(\ell))^{(0,0)}|\mathcal{Q} : \varphi$, and thus the result of the opposite branch, i.e. where ℓ holds, is clearly zero. Application of R.3 then yields $c^{(pl',pu')}|\mathcal{Q} : \varphi$. We finally give an intuition why also pair $c^{(pl',pu')}|\mathcal{Q}'\, \mathcal{Q}_1\, Qx\, \mathcal{Q}_2 : \varphi$ is derivable. Let φ' be the cleaning of the formula that arises from φ by substituting the values $\tau(y)$ for variables y in φ, where τ is any partial assignment that falsifies c, i.e. $\tau(c) = \texttt{false}$. Consequently, clause $(neg(\ell))$ occurs in φ', and hence $Pr(Qx\, \mathcal{Q}_1\, \mathcal{Q}_2 : \varphi') = Pr(\mathcal{Q}_1\, Qx\, \mathcal{Q}_2 : \varphi')$. The latter is formally shown by Lemma 6.2.

$$\text{(R.3u)}\quad \frac{\begin{array}{c}(c \vee \ell)^{(pl,pu)} | \mathcal{Q} : \varphi, \\ \ell \in \{x, \neg x\},\ \mathcal{Q} = \mathcal{Q}'\, Qx\, \mathcal{Q}_1\, \mathcal{Q}_2,\ x \notin \mathit{Var}(\mathcal{Q}(c)), \\ \exists (c' \vee \mathit{neg}(\ell)) \in \varphi : c' \subseteq c, \\ (pl', pu') = \begin{cases} (pl, pu) & ; Q = \exists, \\ (p \cdot pl, p \cdot pu) & ; Q = \text{Я}^p, \ell = \neg x, \\ ((1-p) \cdot pl, (1-p) \cdot pu) & ; Q = \text{Я}^p, \ell = x \end{cases}\end{array}}{c^{(pl',pu')} | \mathcal{Q}'\, \mathcal{Q}_1\, Qx\, \mathcal{Q}_2 : \varphi}$$

Purification is described by rule R.3p. This rule is similar in nature to R.3u but instead of searching for unit literals, it checks whether literal $\mathit{neg}(\ell)$ is pure under partial assignment ff_c. We remark that we use the slightly more general definition of pure literals here, i.e. where pure literals need *not* be present in φ, confer Subsection 6.2.1. It is therefore checked whether each clause $c' \in \varphi$ which is not equivalent to `true` under ff_c, i.e. $\not\models (c \vee c')$, does not comprise the opposite literal ℓ, i.e. $\ell \notin c'$. In terms of DPLL-SSAT, literal $\mathit{neg}(\ell)$ is propagated while the opposite branch, i.e. where ℓ holds, is skipped as the probability of the latter cannot be greater. This justifies application of rule R.3p in order to derive pair $c^{(pl,pu)} | \mathcal{Q} : \varphi$ from input pair $(c \vee \ell)^{(pl,pu)} | \mathcal{Q} : \varphi$. The operation of moving the corresponding existential quantifier within the prefix in order to also derive $c^{(pl,pu)} | \mathcal{Q}'\, \mathcal{Q}_1\, \exists x\, \mathcal{Q}_2 : \varphi$ again relies on the observation that $Pr(\exists x\, \mathcal{Q}_1\, \mathcal{Q}_2 : \varphi') = Pr(\mathcal{Q}_1\, \exists x\, \mathcal{Q}_2 : \varphi')$ as already considered above for R.3u. The latter is also evinced by Lemma 6.2.

$$\text{(R.3p)}\quad \frac{\begin{array}{c}(c \vee \ell)^{(pl,pu)} | \mathcal{Q} : \varphi, \\ \ell \in \{x, \neg x\},\ \mathcal{Q} = \mathcal{Q}'\, \exists x\, \mathcal{Q}_1\, \mathcal{Q}_2,\ x \notin \mathit{Var}(\mathcal{Q}(c)), \\ \forall c' \in \varphi \text{ with } \not\models (c \vee c') : \ell \notin c'\end{array}}{c^{(pl,pu)} | \mathcal{Q}'\, \mathcal{Q}_1\, \exists x\, \mathcal{Q}_2 : \varphi}$$

As for S-resolution, $\varepsilon | \mathcal{Q} : \varphi \vdash_{\mathsf{R.1g}} c^{(pl,pu)} | \mathcal{Q} : \varphi$ abbreviates the derivation of $c^{(pl,pu)} | \mathcal{Q} : \varphi$ from $\varepsilon | \mathcal{Q} : \varphi$ by rule R.1g. Likewise, $c^{(pl,pu)} | \mathcal{Q} : \varphi \vdash_{\mathsf{R.3t}} (c')^{(pl',pu')} | \mathcal{Q} : \varphi$ denotes derivation of $(c')^{(pl',pu')} | \mathcal{Q} : \varphi$ from $c^{(pl,pu)} | \mathcal{Q} : \varphi$ by R.3t. The same meaning applies to $c^{(pl,pu)} | \mathcal{Q} : \varphi \vdash_{\mathsf{R.3u}} (c')^{(pl',pu')} | \mathcal{Q}' : \varphi$ and $c^{(pl,pu)} | \mathcal{Q} : \varphi \vdash_{\mathsf{R.3p}} (c')^{(pl',pu')} | \mathcal{Q}' : \varphi$. Note that the latter rules R.3u and R.3p potentially modify the given quantifier prefix $\mathcal{Q}$ to $\mathcal{Q}'$.

Observe that enhanced S-resolution remains *complete* in the sense of Theorem 6.1 since (strong) S-resolution is a special case of enhanced S-resolution. With regard to *termination*, a strategy of rule applications such that enhanced S-resolution derives the pair $\emptyset^{(pl,pu)} | \mathcal{Q} : \varphi$ with $pl = pu = Pr(\mathcal{Q} : \varphi)$ after *finitely* many steps can be formulated by simply taking an appropriate strategy for S-resolution. The latter exists and can be devised as shown in Subsection 6.2.2. To prove *soundness* of enhanced S-resolution in the sense of Corollary 6.1, Lemma 6.3 interprets pairs $c^{(pl,pu)} | \mathcal{Q} : \varphi$ derivable by enhanced S-resolution in the same way as Lemma 6.1 does for S-resolution. Directly from special case $c^{(pl,pu)} | \mathcal{Q} : \varphi = \emptyset^{(pl,pu)} | \mathcal{Q} : \varphi$, soundness of enhanced S-resolution then follows. Since the proof of Lemma 6.3 deals with the issue of moving quantifiers (stemming from rules R.3u and R.3p) that itself provokes a rather technical proof, we constitute the latter in an own lemma before.

Lemma 6.2 (Moving quantifiers within quantifier prefix)
Let $Qx\, \mathcal{Q}_1\, \mathcal{Q}_2 : \varphi$ be an SSAT formula with matrix φ in CNF. If

1. one of the clauses (x) and $(\neg x)$ occurs in φ, or

2. $Q = \exists$ and

a) each non-tautological clause $c \in \varphi$ does not contain literal x, i.e. $x \notin c$, or

b) each non-tautological clause $c \in \varphi$ does not contain literal $\neg x$, i.e. $\neg x \notin c$

then

$$Pr(Qx\,\mathcal{Q}_1\,\mathcal{Q}_2 : \varphi) = Pr(\mathcal{Q}_1\,Qx\,\mathcal{Q}_2 : \varphi) \quad .$$

Proof. We prove the lemma by induction over the number of quantifiers in $\mathcal{Q}_1$. The result for both items is obvious in the base case, i.e. if $\mathcal{Q}_1 = \varepsilon$. Now assume that the statement is true for arbitrary $\mathcal{Q}_1$. We show that $Pr(Qx\,Q'y\,\mathcal{Q}_1\,\mathcal{Q}_2 : \varphi) = Pr(Q'y\,\mathcal{Q}_1\,Qx\,\mathcal{Q}_2 : \varphi)$ then follows.

With regard to item 1, we assume that $(x) \in \varphi$. The proof for $(\neg x) \in \varphi$ works analogously. Then, $\varphi[\mathtt{false}/x]$ is unsatisfiable. We need to distinguish four cases.

First, $Q = \text{\rotatebox[origin=c]{180}{R}}^p$ and $Q' = \text{\rotatebox[origin=c]{180}{R}}^{p'}$:

$$\begin{align*}
Pr(\text{\rotatebox[origin=c]{180}{R}}^p x\,\text{\rotatebox[origin=c]{180}{R}}^{p'} y\,\mathcal{Q}_1\,\mathcal{Q}_2 : \varphi) &= p \cdot Pr(\text{\rotatebox[origin=c]{180}{R}}^{p'} y\,\mathcal{Q}_1\,\mathcal{Q}_2 : \varphi[\mathtt{true}/x]) \tag{6.6}\\
&= p \cdot (p' \cdot Pr(\mathcal{Q}_1\,\mathcal{Q}_2 : \varphi[\mathtt{true}/x][\mathtt{true}/y])\\
&\quad + (1-p') \cdot Pr(\mathcal{Q}_1\,\mathcal{Q}_2 : \varphi[\mathtt{true}/x][\mathtt{false}/y]))\\
&= p' \cdot p \cdot Pr(\mathcal{Q}_1\,\mathcal{Q}_2 : \varphi[\mathtt{true}/x][\mathtt{true}/y])\\
&\quad + (1-p') \cdot p \cdot Pr(\mathcal{Q}_1\,\mathcal{Q}_2 : \varphi[\mathtt{true}/x][\mathtt{false}/y])\\
&= p' \cdot Pr(\text{\rotatebox[origin=c]{180}{R}}^p x\,\mathcal{Q}_1\,\mathcal{Q}_2 : \varphi[\mathtt{true}/y]) \tag{6.7}\\
&\quad + (1-p') \cdot Pr(\text{\rotatebox[origin=c]{180}{R}}^p x\,\mathcal{Q}_1\,\mathcal{Q}_2 : \varphi[\mathtt{false}/y])\\
&= p' \cdot Pr(\mathcal{Q}_1\,\text{\rotatebox[origin=c]{180}{R}}^p x\,\mathcal{Q}_2 : \varphi[\mathtt{true}/y]) \tag{6.8}\\
&\quad + (1-p') \cdot Pr(\mathcal{Q}_1\,\text{\rotatebox[origin=c]{180}{R}}^p x\,\mathcal{Q}_2 : \varphi[\mathtt{false}/y])\\
&= Pr(\text{\rotatebox[origin=c]{180}{R}}^{p'} y\,\mathcal{Q}_1\,\text{\rotatebox[origin=c]{180}{R}}^p x\,\mathcal{Q}_2 : \varphi) \quad .
\end{align*}$$

Concerning equations 6.6 and 6.7, recall that formula $\varphi[\mathtt{false}/x]$ is unsatisfiable and therefore both $\varphi[\mathtt{false}/x][\mathtt{true}/y]$ and $\varphi[\mathtt{false}/x][\mathtt{false}/y]$ are unsatisfiable. Consequently, $Pr(\text{\rotatebox[origin=c]{180}{R}}^{p'} y\,\mathcal{Q}_1\,\mathcal{Q}_2 : \varphi[\mathtt{false}/x]) = 0$, $Pr(\mathcal{Q}_1\,\mathcal{Q}_2 : \varphi[\mathtt{false}/x][\mathtt{true}/y]) = 0$, and $Pr(\mathcal{Q}_1\,\mathcal{Q}_2 : \varphi[\mathtt{false}/x][\mathtt{false}/y]) = 0$ by Definition 4.2. We further remark that equation 6.8 exploits induction hypothesis which is applicable since $(x) \in \varphi[\mathtt{true}/y]$ and $(x) \in \varphi[\mathtt{false}/y]$.

Second, $Q = \text{\rotatebox[origin=c]{180}{R}}^p$ and $Q' = \exists$:

$$\begin{align*}
Pr(\text{\rotatebox[origin=c]{180}{R}}^p x\,\exists y\,\mathcal{Q}_1\,\mathcal{Q}_2 : \varphi) &= p \cdot Pr(\exists y\,\mathcal{Q}_1\,\mathcal{Q}_2 : \varphi[\mathtt{true}/x])\\
&= p \cdot \max(Pr(\mathcal{Q}_1\,\mathcal{Q}_2 : \varphi[\mathtt{true}/x][\mathtt{true}/y]),\\
&\quad Pr(\mathcal{Q}_1\,\mathcal{Q}_2 : \varphi[\mathtt{true}/x][\mathtt{false}/y]))\\
&= \max(p \cdot Pr(\mathcal{Q}_1\,\mathcal{Q}_2 : \varphi[\mathtt{true}/x][\mathtt{true}/y]), \tag{6.9}\\
&\quad p \cdot Pr(\mathcal{Q}_1\,\mathcal{Q}_2 : \varphi[\mathtt{true}/x][\mathtt{false}/y]))\\
&= \max(Pr(\text{\rotatebox[origin=c]{180}{R}}^p x\,\mathcal{Q}_1\,\mathcal{Q}_2 : \varphi[\mathtt{true}/y]),\\
&\quad Pr(\text{\rotatebox[origin=c]{180}{R}}^p x\,\mathcal{Q}_1\,\mathcal{Q}_2 : \varphi[\mathtt{false}/y]))
\end{align*}$$

$$\begin{aligned}
&= \max(Pr(\mathcal{Q}_1 \rotatebox[origin=c]{180}{R}^{p} x\, \mathcal{Q}_2 : \varphi[\texttt{true}/y]),\\
&\quad\; Pr(\mathcal{Q}_1 \rotatebox[origin=c]{180}{R}^{p} x\, \mathcal{Q}_2 : \varphi[\texttt{false}/y]))\\
&= Pr(\exists y\, \mathcal{Q}_1 \rotatebox[origin=c]{180}{R}^{p} x\, \mathcal{Q}_2 : \varphi) \enspace .
\end{aligned}$$

Equation 6.9 is true as $p > 0$.

Third, $Q = \exists$ and $Q' = \rotatebox[origin=c]{180}{R}^{p'}$:

$$\begin{aligned}
Pr(\exists x \rotatebox[origin=c]{180}{R}^{p'} y\, \mathcal{Q}_1\, \mathcal{Q}_2 : \varphi) &= Pr(\rotatebox[origin=c]{180}{R}^{p'} y\, \mathcal{Q}_1\, \mathcal{Q}_2 : \varphi[\texttt{true}/x])\\
&= p' \cdot Pr(\mathcal{Q}_1\, \mathcal{Q}_2 : \varphi[\texttt{true}/x][\texttt{true}/y])\\
&\quad + (1-p') \cdot Pr(\mathcal{Q}_1\, \mathcal{Q}_2 : \varphi[\texttt{true}/x][\texttt{false}/y])\\
&= p' \cdot Pr(\exists x\, \mathcal{Q}_1\, \mathcal{Q}_2 : \varphi[\texttt{true}/y])\\
&\quad + (1-p') \cdot Pr(\exists x\, \mathcal{Q}_1\, \mathcal{Q}_2 : \varphi[\texttt{false}/y])\\
&= p' \cdot Pr(\mathcal{Q}_1 \exists x\, \mathcal{Q}_2 : \varphi[\texttt{true}/y])\\
&\quad + (1-p') \cdot Pr(\mathcal{Q}_1 \exists x\, \mathcal{Q}_2 : \varphi[\texttt{false}/y])\\
&= Pr(\rotatebox[origin=c]{180}{R}^{p'} y\, \mathcal{Q}_1 \exists x\, \mathcal{Q}_2 : \varphi) \enspace .
\end{aligned}$$

Fourth, $Q = \exists$ and $Q' = \exists$:

$$\begin{aligned}
Pr(\exists x \exists y\, \mathcal{Q}_1\, \mathcal{Q}_2 : \varphi) &= Pr(\exists y\, \mathcal{Q}_1\, \mathcal{Q}_2 : \varphi[\texttt{true}/x])\\
&= \max(Pr(\mathcal{Q}_1\, \mathcal{Q}_2 : \varphi[\texttt{true}/x][\texttt{true}/y]),\\
&\quad\; Pr(\mathcal{Q}_1\, \mathcal{Q}_2 : \varphi[\texttt{true}/x][\texttt{false}/y]))\\
&= \max(Pr(\exists x\, \mathcal{Q}_1\, \mathcal{Q}_2 : \varphi[\texttt{true}/y]),\\
&\quad\; Pr(\exists x\, \mathcal{Q}_1\, \mathcal{Q}_2 : \varphi[\texttt{false}/y]))\\
&= \max(Pr(\mathcal{Q}_1 \exists x\, \mathcal{Q}_2 : \varphi[\texttt{true}/y]),\\
&\quad\; Pr(\mathcal{Q}_1 \exists x\, \mathcal{Q}_2 : \varphi[\texttt{false}/y]))\\
&= Pr(\exists y\, \mathcal{Q}_1 \exists x\, \mathcal{Q}_2 : \varphi) \enspace .
\end{aligned}$$

With regard to item 2, we prove the statement for subitem 2a. The proof for subitem 2b works analogously. From the fact that $\forall c \in \varphi : x \notin c$, we deduce that $\varphi[\texttt{true}/x] \Rightarrow \varphi[\texttt{false}/x]$. Hence, $Pr(Q'y\, \mathcal{Q}_1\, \mathcal{Q}_2 : \varphi[\texttt{true}/x]) \leq Pr(Q'y\, \mathcal{Q}_1\, \mathcal{Q}_2 : \varphi[\texttt{false}/x])$ and

$$Pr(\exists x\, Q'y\, \mathcal{Q}_1\, \mathcal{Q}_2 : \varphi) = Pr(Q'y\, \mathcal{Q}_1\, \mathcal{Q}_2 : \varphi[\texttt{false}/x]) \enspace . \tag{6.10}$$

Due to $\varphi[\texttt{true}/x] \Rightarrow \varphi[\texttt{false}/x]$, we conclude that both

$$\begin{aligned}
\varphi[\texttt{true}/x][\texttt{true}/y] &\Rightarrow \varphi[\texttt{false}/x][\texttt{true}/y] \;\text{ and}\\
\varphi[\texttt{true}/x][\texttt{false}/y] &\Rightarrow \varphi[\texttt{false}/x][\texttt{false}/y]
\end{aligned}$$

hold which in turn gives

$$\begin{aligned}
Pr(\exists x\, \mathcal{Q}_1\, \mathcal{Q}_2 : \varphi[\texttt{true}/y]) &= Pr(\mathcal{Q}_1\, \mathcal{Q}_2 : \varphi[\texttt{false}/x][\texttt{true}/y]) \enspace ,\\
Pr(\exists x\, \mathcal{Q}_1\, \mathcal{Q}_2 : \varphi[\texttt{false}/y]) &= Pr(\mathcal{Q}_1\, \mathcal{Q}_2 : \varphi[\texttt{false}/x][\texttt{false}/y]) \enspace .
\end{aligned} \tag{6.11}$$

We distinguish two cases.

First, $Q' = \text{Я}^p$:

$$\begin{aligned}
(6.12)\qquad Pr(\exists x\, \text{Я}^p y\, \mathcal{Q}_1\, \mathcal{Q}_2 : \varphi) &= Pr(\text{Я}^p y\, \mathcal{Q}_1\, \mathcal{Q}_2 : \varphi[\texttt{false}/x]) \\
&= p \cdot Pr(\mathcal{Q}_1\, \mathcal{Q}_2 : \varphi[\texttt{false}/x][\texttt{true}/y]) \\
&\quad + (1-p) \cdot Pr(\mathcal{Q}_1\, \mathcal{Q}_2 : \varphi[\texttt{false}/x][\texttt{false}/y]) \\
(6.13)\qquad &= p \cdot Pr(\exists x\, \mathcal{Q}_1\, \mathcal{Q}_2 : \varphi[\texttt{true}/y]) \\
&\quad + (1-p) \cdot Pr(\exists x\, \mathcal{Q}_1\, \mathcal{Q}_2 : \varphi[\texttt{false}/y]) \\
(6.14)\qquad &= p \cdot Pr(\mathcal{Q}_1\, \exists x\, \mathcal{Q}_2 : \varphi[\texttt{true}/y]) \\
&\quad + (1-p) \cdot Pr(\mathcal{Q}_1\, \exists x\, \mathcal{Q}_2 : \varphi[\texttt{false}/y]) \\
&= Pr(\text{Я}^p y\, Q_1\, \exists x\, \mathcal{Q}_2 : \varphi) \enspace .
\end{aligned}$$

Correctness of equations 6.12 and 6.13 follows from equations 6.10 and 6.11, respectively. Equation 6.14 exploits induction hypothesis. Regarding the latter, observe that whenever $\forall c \in \varphi : x \notin c$ then also $\forall c \in \varphi[\texttt{true}/y] : x \notin c$ and $\forall c \in \varphi[\texttt{false}/y] : x \notin c$.

Second, $Q' = \exists$:

$$\begin{aligned}
Pr(\exists x\, \exists y\, \mathcal{Q}_1\, \mathcal{Q}_2 : \varphi) &= Pr(\exists y\, \mathcal{Q}_1\, \mathcal{Q}_2 : \varphi[\texttt{false}/x]) \\
&= \max(Pr(\mathcal{Q}_1\, \mathcal{Q}_2 : \varphi[\texttt{false}/x][\texttt{true}/y]), \\
&\quad Pr(\mathcal{Q}_1\, \mathcal{Q}_2 : \varphi[\texttt{false}/x][\texttt{false}/y])) \\
&= \max(Pr(\exists x\, \mathcal{Q}_1\, \mathcal{Q}_2 : \varphi[\texttt{true}/y]), \\
&\quad Pr(\exists x\, \mathcal{Q}_1\, \mathcal{Q}_2 : \varphi[\texttt{false}/y])) \\
&= \max(Pr(\mathcal{Q}_1\, \exists x\, \mathcal{Q}_2 : \varphi[\texttt{true}/y]), \\
&\quad Pr(\mathcal{Q}_1\, \exists x\, \mathcal{Q}_2 : \varphi[\texttt{false}/y])) \\
&= Pr(\exists y\, Q_1\, \exists x\, \mathcal{Q}_2 : \varphi) \enspace .
\end{aligned}$$

Hence, the result follows. □

Having proven Lemma 6.2 tackling the issue of moving quantifiers, which arises when applying unit propagation and purification, we are now prepared to interpret pairs $c^{(pl,pu)}|\mathcal{Q} : \varphi$ derivable by enhanced S-resolution. This interpretation is formalized in the following Lemma 6.3, namely in the same way as for the case of S-resolution in Lemma 6.1.

Lemma 6.3 (Pairs $c^{(pl,pu)}|\mathcal{Q} : \varphi$ derivable by enhanced S-resolution)
Let pair $c^{(pl,pu)}|\mathcal{Q} : \varphi$ with $\mathcal{Q} = Q_1x_1 \ldots Q_nx_n$ and $\mathcal{Q}(c) = Q_1x_1 \ldots Q_ix_i$ be derivable by enhanced S-resolution. Then, for each truth assignment $\tau : Var(\mathcal{Q}(c)) \to \mathbb{B}$ with $\forall x \in Var(c) : \tau(x) = \mathit{ff}_c(x)$ it holds that

$$pl \;\le\; Pr(Q_{i+1}x_{i+1} \ldots Q_nx_n : \varphi[\tau(x_1)/x_1] \ldots [\tau(x_i)/x_i]) \;\le\; pu \enspace .$$

Proof. First of all, observe that clause c in each pair $c^{(pl,pu)}|\mathcal{Q} : \varphi$ derivable by enhanced S-resolution is non-tautological, i.e. $\not\models c$. The rationale is that each clause in φ is non-tautological by global assumption and, therefore, each rule can only produce non-tautological clauses. As a consequence, each above truth assignment τ is well-defined.

We show the lemma by induction over the application of rules R.1, R.1g, R.2, R.2s, R.3, R.3t, R.3u, and R.3p. The induction proof of this lemma works in the very same way as the induction proof of Lemma 6.1.

The base cases are given by rules R.1, R.1g, R.2, and R.2s. As shown in the proof of Lemma 6.1, the statement holds for R.1, R.2, and R.2s. For rule R.1g, i.e.

$$\varepsilon|\mathcal{Q} : \varphi \vdash_{\mathsf{R.1g}} c^{(0,0)}|\mathcal{Q} : \varphi \quad ,$$

observe that there exists some clause $c' \in \varphi$ such that $c' \subseteq c$. By construction of τ that falsifies clause $c' \in \varphi$, it then follows that the formula $\varphi[\tau(x_1)/x_1]\ldots[\tau(x_i)/x_i]$ is unsatisfiable. Thus,

$$0 \;=\; Pr(Q_{i+1}x_{i+1}\ldots Q_n x_n : \varphi[\tau(x_1)/x_1]\ldots[\tau(x_i)/x_i]) \;=\; 0 \quad .$$

For the induction step, assume that the respective assumptions hold for the clauses in the premise of rules R.3, R.3t, R.3u, and R.3p. The proof of Lemma 6.1 already shows the statement for R.3. In the remainder of this proof, we use the following definition: for a truth value $v \in \mathbb{B}$, let $neg(v)$ be denote the opposite truth value of v, i.e. $neg(v) = \mathtt{true}$ if and only if $v = \mathtt{false}$.

With regard to rule R.3t, i.e.

$$(c \vee \ell)^{(pl',pu')}|\mathcal{Q} : \varphi \vdash_{\mathsf{R.3t}} c^{(pl,pu)}|\mathcal{Q} : \varphi \quad ,$$

induction hypothesis gives the following: for each truth assignment $\tau : Var(\mathcal{Q}(c \vee \ell)) \to \mathbb{B}$ with $\forall x \in Var(c \vee \ell) : \tau(x) = ff_{(c\vee\ell)}(x)$ it holds that

$$pl' \;\leq\; Pr(Q_{j+1}x_{j+1}\ldots Q_n x_n : \varphi[\tau(x_1)/x_1]\ldots[\tau(x_{j-1})/x_{j-1}][\tau(x_j)/x_j]) \;\leq\; pu' \quad .$$

where $\ell \in \{x_j, \neg x_j\}$. Clearly, $j \geq i+1$ as $x \notin Var(\mathcal{Q}(c))$. Using the trivial observation that

$$0 \;\leq\; Pr(Q_{j+1}x_{j+1}\ldots Q_n x_n : \varphi[\tau(x_1)/x_1]\ldots[\tau(x_{j-1})/x_{j-1}][neg(\tau(x_j))/x_j]) \;\leq\; 1 \quad ,$$

we obtain

$$pl \;\leq\; Pr(Q_j x_j\, Q_{j+1}x_{j+1}\ldots Q_n x_n : \varphi[\tau(x_1)/x_1]\ldots[\tau(x_{j-1})/x_{j-1}]) \;\leq\; pu \quad .$$

The statement for R.3t directly follows in case $j = i+1$. For $j > i+1$, note that variables $x_{i+1}, \ldots, x_{j-1}$ do not occur in clause c. Hence, for $k = j-1$ down to $i+1$ we successively conclude that

$$\begin{aligned} pl \;&\leq\; Pr(Q_{k+1}x_{k+1}\ldots Q_n x_n : \varphi[\tau(x_1)/x_1]\ldots[\tau(x_{k-1})/x_{k-1}][\mathtt{true}/x_k]) \;\leq\; pu \quad , \\ pl \;&\leq\; Pr(Q_{k+1}x_{k+1}\ldots Q_n x_n : \varphi[\tau(x_1)/x_1]\ldots[\tau(x_{k-1})/x_{k-1}][\mathtt{false}/x_k]) \;\leq\; pu \quad . \end{aligned}$$

From case $k = i+1$, we finally achieve the statement for R.3t, i.e.

$$pl \;\leq\; Pr(Q_{i+1}x_{i+1}\ldots Q_n x_n : \varphi[\tau(x_1)/x_1]\ldots[\tau(x_i)/x_i]) \;\leq\; pu \quad .$$

We now consider rule R.3u, i.e.

$$(c \vee \ell)^{(pl',pu')}|\mathcal{Q}' : \varphi \vdash_{\mathsf{R.3u}} c^{(pl,pu)}|\mathcal{Q} : \varphi$$

with $\mathcal{Q}' = \mathcal{Q}''\, Qx\ \mathcal{Q}_1\, \mathcal{Q}_2$ and $\mathcal{Q} = \mathcal{Q}''\, \mathcal{Q}_1\, Qx\, \mathcal{Q}_2$. By induction hypothesis, for each truth assignment $\tau : \mathit{Var}(\mathcal{Q}'(c \vee \ell)) \to \mathbb{B}$ with $\forall y \in \mathit{Var}(c \vee \ell) : \tau(y) = \mathit{ff}_{(c\vee\ell)}(y)$ it holds that

$$pl' \quad\leq\quad Pr(\mathcal{Q}_1\, \mathcal{Q}_2 : \varphi[\tau(x_1)/x_1] \ldots [\tau(x_j)/x_j][\tau(x)/x]) \quad\leq\quad pu'$$

where $\ell \in \{x, \neg x\}$ and $\mathcal{Q}'' = Q_1 x_1 \ldots Q_j x_j$. Observe that $j \geq i$ as $x \notin \mathit{Var}(\mathcal{Q}(c))$ and thus $\mathit{Var}(\mathcal{Q}(c)) \subseteq \mathit{Var}(\mathcal{Q}'')$. Due to application condition of R.3u, there is some clause $(c' \vee \mathit{neg}(\ell)) \in \varphi$ such that $c' \subseteq c$. As a consequence, for each truth assignment $\tau' : \mathit{Var}(\mathcal{Q}'(c' \vee \mathit{neg}(\ell))) \to \mathbb{B}$ with $\forall y \in \mathit{Var}(c' \vee \mathit{neg}(\ell)) : \tau'(y) = \mathit{ff}_{(c'\vee \mathit{neg}(\ell))}(y)$ it holds that

$$0 \quad=\quad Pr(\mathcal{Q}_1\, \mathcal{Q}_2 : \varphi[\tau'(x_1)/x_1] \ldots [\tau'(x_j)/x_j][\tau'(x)/x]) \quad=\quad 0 \quad .$$

Note that $\tau(y) = \tau'(y)$ for all $y \in \mathit{Var}(c')$ as $c' \subseteq c$, and that $\tau(x) = \mathit{neg}(\tau'(x))$. Thus,

$$pl \quad\leq\quad Pr(Qx\, \mathcal{Q}_1\, \mathcal{Q}_2 : \varphi[\tau(x_1)/x_1] \ldots [\tau(x_j)/x_j]) \quad\leq\quad pu \quad .$$

Let φ' be the semantically equivalent cleaning of $\varphi[\tau(x_1)/x_1] \ldots [\tau(x_j)/x_j]$ as defined in Section 2.2, i.e. the formula that arises when removing the constants `true` and `false`. Obviously,

$$Pr(Qx\, \mathcal{Q}_1\, \mathcal{Q}_2 : \varphi') = Pr(Qx\, \mathcal{Q}_1\, \mathcal{Q}_2 : \varphi[\tau(x_1)/x_1] \ldots [\tau(x_j)/x_j]) \quad .$$

Observe that $(\mathit{neg}(\ell)) \in \varphi'$ since $(c' \vee \mathit{neg}(\ell)) \in \varphi$ and $(c' \vee \mathit{neg}(\ell))[\tau(x_1)/x_1] \ldots [\tau(x_j)/x_j] \equiv (\mathit{neg}(\ell))$. Instantaneously by Lemma 6.2, item 1,

$$Pr(Qx\, \mathcal{Q}_1\, \mathcal{Q}_2 : \varphi') = Pr(\mathcal{Q}_1\, Qx\, \mathcal{Q}_2 : \varphi') \quad .$$

It clearly follows that

$$Pr(Qx\, \mathcal{Q}_1\, \mathcal{Q}_2 : \varphi[\tau(x_1)/x_1] \ldots [\tau(x_j)/x_j]) = Pr(\mathcal{Q}_1\, Qx\, \mathcal{Q}_2 : \varphi[\tau(x_1)/x_1] \ldots [\tau(x_j)/x_j])$$

and, therefore,

$$pl \quad\leq\quad Pr(\mathcal{Q}_1\, Qx\, \mathcal{Q}_2 : \varphi[\tau(x_1)/x_1] \ldots [\tau(x_j)/x_j]) \quad\leq\quad pu$$

hold. We have that $\mathcal{Q}_1\, Qx\, \mathcal{Q}_2 = Q_{j+1} x_{j+1} \ldots Q_n x_n$ due to $\mathcal{Q} = \mathcal{Q}''\, \mathcal{Q}_1\, Qx\, \mathcal{Q}_2$, $\mathcal{Q} = Q_1 x_1 \ldots Q_n x_n$, and $\mathcal{Q}'' = Q_1 x_1 \ldots Q_j x_j$. Recall that $j \geq i$. Whenever $j = i$ then the statement for rule R.3u follows immediately. In case $j > i$, note that variables $x_{i+1}, \ldots, x_j$ do not occur in clause c. Hence, for $k = j$ down to $i + 1$ we successively conclude that

$$\begin{aligned} pl \quad&\leq\quad Pr(Q_{k+1} x_{k+1} \ldots Q_n x_n : \varphi[\tau(x_1)/x_1] \ldots [\tau(x_{k-1})/x_{k-1}][\mathtt{true}/x_k]) \quad\leq\quad pu \quad , \\ pl \quad&\leq\quad Pr(Q_{k+1} x_{k+1} \ldots Q_n x_n : \varphi[\tau(x_1)/x_1] \ldots [\tau(x_{k-1})/x_{k-1}][\mathtt{false}/x_k]) \quad\leq\quad pu \quad . \end{aligned}$$

From case $k = i + 1$, we finally achieve the statement for R.3u, i.e.

$$pl \quad\leq\quad Pr(Q_{i+1} x_{i+1} \ldots Q_n x_n : \varphi[\tau(x_1)/x_1] \ldots [\tau(x_i)/x_i]) \quad\leq\quad pu \quad .$$

We finally consider rule R.3p, i.e.

$$(c \vee \ell)^{(pl,pu)} | \mathcal{Q}' : \varphi \vdash_{\mathsf{R.3p}} c^{(pl,pu)} | \mathcal{Q} : \varphi$$

with $\mathcal{Q}' = \mathcal{Q}''\,\exists x\ \mathcal{Q}_1\,\mathcal{Q}_2$ and $\mathcal{Q} = \mathcal{Q}''\,\mathcal{Q}_1\,\exists x\,\mathcal{Q}_2$. By induction hypothesis, for each truth assignment $\tau : \mathit{Var}(\mathcal{Q}'(c \vee \ell)) \to \mathbb{B}$ with $\forall y \in \mathit{Var}(c \vee \ell) : \tau(y) = \mathit{ff}_{(c\vee\ell)}(y)$ it holds that

$$pl \;\leq\; Pr(\mathcal{Q}_1\,\mathcal{Q}_2 : \varphi[\tau(x_1)/x_1]\dots[\tau(x_j)/x_j][\tau(x)/x]) \;\leq\; pu \tag{6.15}$$

where $\ell \in \{x, \neg x\}$ and $\mathcal{Q}'' = Q_1x_1\dots Q_jx_j$. Observe that $j \geq i$ as $x \notin \mathit{Var}(\mathcal{Q}(c))$ and thus $\mathit{Var}(\mathcal{Q}(c)) \subseteq \mathit{Var}(\mathcal{Q}'')$. Due to application condition of R.3p, each clause $c' \in \varphi$ with $\not\models (c \vee c')$ does not contain literal ℓ, i.e. $\ell \notin c'$. Observe that $\not\models (c \vee c')$ is equivalent to $\not\models c'[\mathit{ff}_c(y_1)/y_1]\dots[\mathit{ff}_c(y_m)/y_m]$ with $\mathit{Var}(c) = \{y_1,\dots,y_m\}$ since $\not\models c$, and that $\tau(\mathit{neg}(\ell)) = \texttt{true}$ as $\tau(\ell) = \mathit{ff}_{(c\vee\ell)}(\ell) = \texttt{false}$. As a consequence,

$$\varphi[\tau(x_1)/x_1]\dots[\tau(x_j)/x_j][\mathit{neg}(\tau(x))/x] \;\Rightarrow\; \varphi[\tau(x_1)/x_1]\dots[\tau(x_j)/x_j][\tau(x)/x]$$

is true which in turn entails

$$\begin{aligned} &Pr(\mathcal{Q}_1\,\mathcal{Q}_2 : \varphi[\tau(x_1)/x_1]\dots[\tau(x_j)/x_j][\mathit{neg}(\tau(x))/x]) \\ \leq\;\; &Pr(\mathcal{Q}_1\,\mathcal{Q}_2 : \varphi[\tau(x_1)/x_1]\dots[\tau(x_j)/x_j][\tau(x)/x]) \;\; . \end{aligned} \tag{6.16}$$

Inequalities 6.15 and 6.16 give

$$pl \;\leq\; Pr(\exists x\,\mathcal{Q}_1\,\mathcal{Q}_2 : \varphi[\tau(x_1)/x_1]\dots[\tau(x_j)/x_j]) \;\leq\; pu \;\; .$$

Let φ' be the semantically equivalent cleaning of $\varphi[\tau(x_1)/x_1]\dots[\tau(x_j)/x_j]$ as defined in Section 2.2, i.e. the formula that arises when removing the constants `true` and `false`. Obviously,

$$Pr(\exists x\,\mathcal{Q}_1\,\mathcal{Q}_2 : \varphi') = Pr(\exists x\,\mathcal{Q}_1\,\mathcal{Q}_2 : \varphi[\tau(x_1)/x_1]\dots[\tau(x_j)/x_j]) \;\; .$$

As observed above, all clauses $c' \in \varphi$ with $\not\models c'[\mathit{ff}_c(y_1)/y_1]\dots[\mathit{ff}_c(y_m)/y_m]$ where $\mathit{Var}(c) = \{y_1,\dots,y_m\}$ do not contain literal ℓ. Then, the same holds for all clauses $c' \in \varphi$ with $\not\models c'[\tau(x_1)/x_1]\dots[\tau(x_j)/x_j]$. Consequently, each non-tautological clause $c' \in \varphi'$ does not contain literal ℓ, i.e. $\ell \notin c'$. Instantaneously by Lemma 6.2, item 2,

$$Pr(\exists x\,\mathcal{Q}_1\,\mathcal{Q}_2 : \varphi') = Pr(\mathcal{Q}_1\,\exists x\,\mathcal{Q}_2 : \varphi') \;\; .$$

We may thus conclude that

$$pl \;\leq\; Pr(\mathcal{Q}_1\,\exists x\,\mathcal{Q}_2 : \varphi[\tau(x_1)/x_1]\dots[\tau(x_j)/x_j]) \;\leq\; pu$$

hold. With the same reasoning as for rule R.3u, we finally achieve the statement for R.3p, i.e.

$$pl \;\leq\; Pr(Q_{i+1}x_{i+1}\dots Q_nx_n : \varphi[\tau(x_1)/x_1]\dots[\tau(x_i)/x_i]) \;\leq\; pu \;\; .$$

This completes the proof and the lemma follows. □

Integration of enhanced S-resolution into DPLL-SSAT. After having introduced enhanced S-resolution as well as proven its soundness, we now present the integration of enhanced S-resolution into the classical DPLL-SSAT algorithm from Figure 6.1. For that purpose, we extend the DPLL-SSAT procedure and thereby develop the algorithm DPLL-SSAT-R being introduced below. In addition to the inputs of

$$\text{DPLL-SSAT}(\,\mathcal{Q}:\varphi,\ \theta_l,\ \theta_u\,)\ ,$$

the algorithm

$$\text{DPLL-SSAT-R}(\,\mathcal{Q}:\varphi,\ \theta_l,\ \theta_u,\ \mathcal{Q}',\ \tau,\ \psi\,)$$

takes a quantifier prefix $\mathcal{Q}'$, an assignment $\tau : \mathit{Var}(\mathcal{Q}') :\to \mathbb{B}$, and a propositional formula ψ as *three further inputs* where

- prefix $\mathcal{Q}'$ reflects in chronological order the sequence of all quantified variables having been substituted already,
- partial assignment τ keeps track of all those substitutions, and
- propositional formula ψ simply preserves the original matrix, i.e. when substituting the values $\tau(y)$ for variables y in ψ we achieve the current matrix φ.

Observe that above specifications imply that $\mathcal{Q}'\mathcal{Q}:\psi$ is the original SSAT formula. The *result* of a call DPLL-SSAT-R($\mathcal{Q}:\varphi$, θ_l, θ_u, $\mathcal{Q}'$, τ, ψ) now is a pair

$$(\,pr,\ \ c^{(pl,pu)}\mid \mathcal{Q}'\mathcal{Q}:\psi\,)$$

consisting of

- a probability pr and
- another pair $c^{(pl,pu)} \mid \mathcal{Q}'\mathcal{Q}:\psi$ comprising
  - an annotated clause $c^{(pl,pu)}$ and
  - the original SSAT formula $\mathcal{Q}'\mathcal{Q}:\psi$.

Apart from the additional inputs and the extended output, DPLL-SSAT-R works as DPLL-SSAT, i.e. the modifications do not have an impact on the computation of the probability pr. More precisely,

$$pr = \text{DPLL-SSAT}(\mathcal{Q}:\varphi,\,\theta_l,\,\theta_u)\ \ .$$

We prove later on that

- the pair $c^{(pl,pu)} \mid \mathcal{Q}'\mathcal{Q}:\psi$ is derivable by enhanced S-resolution and satisfies the following:
  - $pl = pu = pr$ if $pr \in [\theta_l, \theta_u]$,
  - $pu < \theta_l$ if $pr < \theta_l$, and
  - $pl > \theta_u$ if $pr > \theta_u$.

Before presenting DPLL-SSAT-R in full detail, we need to introduce some notation. For a partial assignment τ that is not defined on variable x, i.e. $\tau(x)$ is not defined, we denote by $\tau \oplus [x \to v]$ with $v \in \mathbb{B}$ the extended partial assignment τ' with $\tau'(x) = v$ and for all $y \neq x : \tau'(y) = \tau(y)$ if $\tau(y)$ is defined and $\tau'(y)$ is not defined if $\tau(y)$ is not defined. Recall that $neg(\ell)$ returns the opposite literal of ℓ. As in Figure 6.1, $v(\ell) = \texttt{true}$ for positive literals ℓ, and $v(\ell) = \texttt{false}$ for negative ℓ. For $\text{Ꟊ}^p x$, we further have $p(\ell) = p$ if $\ell = x$, and $p(\ell) = 1 - p$ if $\ell = \neg x$. For each propositional formula ψ and for each partial assignment τ to the variables $Var(\psi)$, we define ψ_τ as the formula that arises from ψ by substituting the values $\tau(y)$ for variables y in ψ, i.e.

$$\psi_\tau := \psi[\tau(y_1)/y_1] \dots [\tau(y_m)/y_m]$$

where $\tau(y_i)$ is defined if and only if $1 \leq i \leq m$.

In what follows, we give a detailed account of algorithm DPLL-SSAT-R. For the sake of a convenient and reasonable presentation, we intersperse the formal description with explanatory comments on the corresponding parts of the algorithm.

The inputs of DPLL-SSAT-R were already specified above.

```
1  DPLL-SSAT-R( $\mathcal{Q} : \varphi,\ \theta_l,\ \theta_u,\ \mathcal{Q}',\ \tau,\ \psi$ )
2    input: SSAT formula $\mathcal{Q} : \varphi$ with $\varphi$ in CNF, rational constants $\theta_l, \theta_u$ with $\theta_l \leq \theta_u$,
3           quantifier prefix $\mathcal{Q}'$, assignment $\tau : Var(\mathcal{Q}') \to \mathbb{B}$, propositional formula $\psi$
4           in CNF with $\psi_\tau = \varphi$.
```

In the base cases, the partial assignment τ is used to construct a clause c that encodes the negation of τ, i.e. $\tau(c) = \texttt{false}$. In case φ contains a clause equivalent to $\texttt{false}$, the original matrix ψ comprises some clause c' that is falsified under τ, i.e. $\tau(c') = \texttt{false}$, since $\psi_\tau = \varphi$. Moreover $c' \subseteq c$, as c contains all variables y for which $\tau(y)$ is defined. This justifies to apply

$$\varepsilon | \mathcal{Q}'\mathcal{Q} : \psi \ \vdash_{\mathsf{R.1g}} \ c^{(0,0)} | \mathcal{Q}'\mathcal{Q} : \psi \quad .$$

In the other base case, where all clauses in φ are equivalent to $\texttt{true}$, we know that $\models \varphi$ and $\models \psi_\tau$. We may therefore perform

$$\varepsilon | \mathcal{Q}'\mathcal{Q} : \psi \ \vdash_{\mathsf{R.2s}} \ c^{(1,1)} | \mathcal{Q}'\mathcal{Q} : \psi \quad .$$

```
5    // Base cases
6    if $\varphi$ contains a clause equivalent to false then
7      $c := \{\neg x : \tau(x) = \texttt{true}\} \cup \{x : \tau(x) = \texttt{false}\}$.
8      return (0, $c^{(0,0)} | \mathcal{Q}'\mathcal{Q} : \psi$).
9    if all clauses in $\varphi$ equivalent to true then
10     $c := \{\neg x : \tau(x) = \texttt{true}\} \cup \{x : \tau(x) = \texttt{false}\}$.
11     return (1, $c^{(1,1)} | \mathcal{Q}'\mathcal{Q} : \psi$).
```

In case unit propagation applies, DPLL-SSAT just needs to explore the SSAT subproblem where unit literal ℓ is equivalent to $\texttt{true}$ after substitution, confer Subsection 6.2.1. The corresponding recursion of DPLL-SSAT-R then returns the probability pr as well as the pair $c^{(pl,pu)}|\Phi$. We show later on, namely in Lemma 6.4, that the pair $c^{(pl,pu)}|\Phi$, where

$\Phi = \mathcal{Q}' Q x \mathcal{Q}_1 \mathcal{Q}_2 : \psi$ with $Q \in \{\exists, Ⅎ^p\}$, is derivable by S-resolution and that clause c contains the opposite of unit literal ℓ, i.e. $neg(\ell) \in c$. We may thus apply

$$c^{(pl,pu)} | \mathcal{Q}' \, Qx \, \mathcal{Q}_1 \, \mathcal{Q}_2 : \psi \;\vdash_{\mathsf{R.3u}}\; (c \setminus \{neg(\ell)\})^{(pl',pu')} | \mathcal{Q}' \, \mathcal{Q}_1 \, Qx \, \mathcal{Q}_2 : \psi \quad .$$

Let $(pr', (c')^{(pl',pu')} | \mathcal{Q}'\mathcal{Q} : \psi)$ be the pair to be returned. Lemma 6.4 further shows that $pl' = pu' = pr'$ if $pr' \in [\theta_l, \theta_u]$, $pu' < \theta_l$ if $pr' < \theta_l$, and $pl' > \theta_u$ if $pr' > \theta_u$.

```
// Unit propagation
if $\varphi$ contains a unit literal $\ell$ with $Var(\ell) = \{x\}$ then
  $\tau' := \tau \oplus [x \to v(\ell)]$.
  if $\mathcal{Q} = \mathcal{Q}_1 \exists x \mathcal{Q}_2$ then
    $(pr,\ c^{(pl,pu)} | \Phi) :=$
              DPLL-SSAT-R( $\mathcal{Q}_1\mathcal{Q}_2 : \varphi[v(\ell)/x],\ \theta_l,\ \theta_u,\ \mathcal{Q}' \exists x,\ \tau',\ \psi$ ).
    $c' := c \setminus \{neg(\ell)\}$.
    return $(pr,\ (c')^{(pl,pu)} | \mathcal{Q}'\mathcal{Q} : \psi)$.
  if $\mathcal{Q} = \mathcal{Q}_1 Ⅎ^p x \mathcal{Q}_2$ then
    $\theta_l' := \theta_l / p(\ell)$.
    $\theta_u' := \theta_u / p(\ell)$.
    $(pr,\ c^{(pl,pu)} | \Phi) :=$
              DPLL-SSAT-R( $\mathcal{Q}_1\mathcal{Q}_2 : \varphi[v(\ell)/x],\ \theta_l',\ \theta_u',\ \mathcal{Q}' Ⅎ^p x,\ \tau',\ \psi$ ).
    $pr' := p(\ell) \cdot pr$.
    $pl' := p(\ell) \cdot pl$.
    $pu' := p(\ell) \cdot pu$.
    $c' := c \setminus \{neg(\ell)\}$.
    return $(pr', (c')^{(pl',pu')} | \mathcal{Q}'\mathcal{Q} : \psi)$.
```

The case of purification is similar to unit propagation. Here, we apply

$$c^{(pl,pu)} | \mathcal{Q}' \, \exists x \, \mathcal{Q}_1 \, \mathcal{Q}_2 : \psi \;\vdash_{\mathsf{R.3p}}\; (c \setminus \{neg(\ell)\})^{(pl,pu)} | \mathcal{Q}' \, \mathcal{Q}_1 \, \exists x \, \mathcal{Q}_2 : \psi$$

where $neg(\ell) \in c$ is the opposite of pure literal ℓ.

```
// Purification
if $\varphi$ contains a pure literal $\ell$ with $Var(\ell) = \{x\}$ then
  $\tau' := \tau \oplus [x \to v(\ell)]$.
  if $\mathcal{Q} = \mathcal{Q}_1 \exists x \mathcal{Q}_2$ then
    $(pr,\ c^{(pl,pu)} | \Phi) :=$
              DPLL-SSAT-R( $\mathcal{Q}_1\mathcal{Q}_2 : \varphi[v(\ell)/x],\ \theta_l,\ \theta_u, \mathcal{Q}' \exists x,\ \tau',\ \psi$).
    $c' := c \setminus \{neg(\ell)\}$.
    return $(pr, (c')^{(pl,pu)} | \mathcal{Q}'\mathcal{Q} : \psi)$.
```

We now consider the optimization of thresholding. The result of the first branch, where `true` is substituted for x, is $(pr_1,\ c_1^{(pl_1,pu_1)} | \mathcal{Q}'\mathcal{Q} : \psi)$. Concerning the existential case, i.e. $\mathcal{Q} = \exists x \mathcal{Q}''$, if the probability result pr_1 already exceeds the upper threshold, i.e. $pr_1 > \theta_u$, then also pl_1 does, i.e. $pl_1 > \theta_u$, and application of

$$c_1^{(pl_1,pu_1)} | \mathcal{Q}'\mathcal{Q} : \psi \;\vdash_{\mathsf{R.3t}}\; (c_1 \setminus \{\neg x\})^{(pl_1,1)} | \mathcal{Q}'\mathcal{Q} : \psi$$

is justified.

In the randomized case, i.e. $\mathcal{Q} = \text{Я}^p x \mathcal{Q}''$, $pr'_1 = p \cdot pr_1$ is the weighted probability of the first branch, and $pl'_1 = p \cdot pl_1$ and $pu'_1 = p \cdot pu_1 + (1-p)$ are the updated probabilities for the derived pair. If value pr'_1 is too small in order to reach the lower threshold even if the probability of the second branch is 1, i.e. if $pr'_1 + (1-p) < \theta_l$, or if pr'_1 already exceeds θ_u then also $pu'_1 < \theta_l$ or $pl'_1 > \theta_u$, respectively. Thus,

$$c_1^{(pl_1,pu_1)} | \mathcal{Q}'\mathcal{Q} : \psi \ \vdash_{\mathsf{R.3t}} \ (c_1 \setminus \{\neg x\})^{(pl'_1,pu'_1)} | \mathcal{Q}'\mathcal{Q} : \psi$$

is feasible.

Whenever thresholding is not possible, we clearly utilize resolution rule R.3, i.e.

$$(c_1^{(pl_1,pu_1)} | \mathcal{Q}'\mathcal{Q} : \psi,\ c_2^{(pl_2,pu_2)} | \mathcal{Q}'\mathcal{Q} : \psi) \vdash_{\mathsf{R.3}} ((c_1 \setminus \{\neg x\}) \ \cup \ (c_2 \setminus \{x\}))^{(pl',pu')} | \mathcal{Q}'\mathcal{Q} : \psi \ \ .$$

```
// Branching and thresholding
if $\mathcal{Q} = \exists x \mathcal{Q}''$ then
  $\tau_1 := \tau \oplus [x \to \texttt{true}]$.
  $(pr_1,\ c_1^{(pl_1,pu_1)} | \Phi) :=$
              DPLL-SSAT-R( $\mathcal{Q}'' : \varphi[\texttt{true}/x],\ \theta_l,\ \theta_u,\ \mathcal{Q}' \exists x,\ \tau_1,\ \psi$ ).
  $c'_1 := c_1 \setminus \{\neg x\}$.
  if $pr_1 > \theta_u$ then return $(pr_1,\ (c'_1)^{(pl_1,1)} | \mathcal{Q}'\mathcal{Q} : \psi)$.
  $\tau_2 := \tau \oplus [x \to \texttt{false}]$.
  $\theta'_l := \max(\theta_l, pr_1)$.
  $(pr_2,\ c_2^{(pl_2,pu_2)} | \Phi) :=$
              DPLL-SSAT-R( $\mathcal{Q}'' : \varphi[\texttt{false}/x],\ \theta'_l,\ \theta_u,\ \mathcal{Q}' \exists x,\ \tau_2,\ \psi$ ).
  $pr' := \max(pr_1, pr_2)$.
  $pl' := \max(pl_1, pl_2)$.
  $pu' := \max(pu_1, pu_2)$.
  $c'_2 := c_2 \setminus \{x\}$.
  return $(pr',\ (c'_1 \cup c'_2)^{(pl',pu')} | \mathcal{Q}'\mathcal{Q} : \psi)$.
if $\mathcal{Q} = \text{Я}^p x \mathcal{Q}''$ then
  $\tau_1 := \tau \oplus [x \to \texttt{true}]$.
  $\theta'_l := (\theta_l - (1-p))/p$.
  $\theta'_u := \theta_u/p$.
  $(pr_1,\ c_1^{(pl_1,pu_1)} | \Phi) :=$
              DPLL-SSAT-R( $\mathcal{Q}'' : \varphi[\texttt{true}/x],\ \theta'_l,\ \theta'_u,\ \mathcal{Q}' \text{Я}^p x,\ \tau_1,\ \psi$ ).
  $pr'_1 := p \cdot pr_1$.
  $pl'_1 := p \cdot pl_1$.
  $pu'_1 := p \cdot pu_1 + (1-p)$.
  $c'_1 := c_1 \setminus \{\neg x\}$.
  if $pr'_1 + (1-p) < \theta_l$ then return $(pr'_1,\ (c'_1)^{(pl'_1,pu'_1)} | \mathcal{Q}'\mathcal{Q} : \psi)$.
  if $pr'_1 > \theta_u$ then return $(pr'_1,\ (c'_1)^{(pl'_1,pu'_1)} | \mathcal{Q}'\mathcal{Q} : \psi)$.
  $\tau_2 := \tau \oplus [x \to \texttt{false}]$.
  $\theta''_l := (\theta_l - pr'_1)/(1-p)$.
  $\theta''_u := (\theta_u - pr'_1)/(1-p)$.
```

```
    $(pr_2,\ c_2^{(pl_2,pu_2)}|\Phi) :=$
              DPLL-SSAT-R( $\mathcal{Q}'' : \varphi[\texttt{false}/x],\ \theta_l'',\ \theta_u'',\ \mathcal{Q}'\,\text{Я}^p x,\ \tau_2,\ \psi$).
    $pr' := p \cdot pr_1 + (1-p) \cdot pr_2$.
    $pl' := p \cdot pl_1 + (1-p) \cdot pl_2$.
    $pu' := p \cdot pu_1 + (1-p) \cdot pu_2$.
    $c_2' := c_2 \setminus \{x\}$.
    **return** $(pr',\ (c_1' \cup c_2')^{(pl',pu')}|\mathcal{Q}'\mathcal{Q} : \psi)$.
```

The next lemma formalizes above observations, thereby establishing the fact that the algorithm DPLL-SSAT-R produces an enhanced S-resolution proof for each instance ($\mathcal{Q} : \varphi, \theta_l, \theta_u$) of the generalized SSAT decision problem as defined at the beginning of this subsection.

Lemma 6.4 (Correctness of integration)
Let be given an SSAT formula $\mathcal{Q}'\mathcal{Q} : \psi$, two rational constants θ_l and θ_u with $\theta_l \leq \theta_u$, and an assignment $\tau : Var(\mathcal{Q}') :\to \mathbb{B}$. Let further be $\varphi = \psi_\tau$ and

$$(\,pr,\ \ c^{(pl,pu)} \mid \Phi\,) \ = \ \text{DPLL-SSAT-R}(\,\mathcal{Q} : \varphi,\ \theta_l,\ \theta_u,\ \mathcal{Q}',\ \tau,\ \psi\,)\ .$$

It then holds that

1. *$\Phi = \mathcal{Q}'\mathcal{Q} : \psi$,*
2. *$\tau(c) = \texttt{false}$ and $Var(c) = Var(\mathcal{Q}')$,*
3. *the pair $c^{(pl,pu)} \mid \Phi$ is derivable by enhanced S-resolution, and*
4. *$pl = pu = pr$ if $pr \in [\theta_l, \theta_u]$, $pu < \theta_l$ if $pr < \theta_l$, and $pl > \theta_u$ if $pr > \theta_u$.*

Proof. We prove the lemma by induction over recursive calls of DPLL-SSAT-R. First of all, observe that item 1 is trivially true as $\Phi = \mathcal{Q}'\mathcal{Q} : \psi$ in each returned result.

There are two base cases. First, there is a clause $c' \in \varphi = \psi_\tau$ such that $c' \equiv \texttt{false}$. Then, there is a corresponding clause $c'' \in \psi$ such that $c''[\tau(y_1)/y_1]\ldots[\tau(y_m)/y_m] = c'$ where $\tau(y_i)$ is defined if and only if $1 \leq i \leq m$. Clearly, $\tau(c'') = \texttt{false}$. By construction of clause c, $\tau(c) = \texttt{false}$ and $Var(c) = Var(\mathcal{Q}')$. Thus, item 2 holds. From the latter facts, it follows that $c'' \subseteq c$. Items 3 and 4 are true since

$$\varepsilon|\mathcal{Q}'\mathcal{Q} : \psi \ \vdash_{\mathsf{R.1g}}\ c^{(0,0)}|\mathcal{Q}'\mathcal{Q} : \psi$$

is applicable and $pl = pu = pr = 0$, respectively. In the second base case, all clauses in φ are equivalent to $\texttt{true}$. With regard to item 2, we again conclude by construction that $\tau(c) = \texttt{false}$ and $Var(c) = Var(\mathcal{Q}')$. Item 3 holds since $\models \psi_\tau$ and therefore

$$\varepsilon|\mathcal{Q}'\mathcal{Q} : \psi \ \vdash_{\mathsf{R.2s}}\ c^{(1,1)}|\mathcal{Q}'\mathcal{Q} : \psi$$

is feasible. The fact that $pl = pu = pr = 1$ shows item 4.

In the induction step, we show that the statement of the lemma for

$$(\,pr',\ \ (c')^{(pl',pu')} \mid \mathcal{Q}'\mathcal{Q} : \psi\,) \ = \ \text{DPLL-SSAT-R}(\,\mathcal{Q} : \varphi,\ \theta_l,\ \theta_u,\ \mathcal{Q}',\ \tau,\ \psi\,)$$

follows from induction hypothesis. The latter assumes that the lemma holds for each call

$$(\,pr,\;\; c^{(pl,pu)} \mid \mathcal{Q}'Qx\mathcal{Q}_1\mathcal{Q}_2 : \psi\,) \;=\; \text{DPLL-SSAT-R}(\,\mathcal{Q}_1\mathcal{Q}_2 : \varphi',\; \theta_l',\; \theta_u',\; \mathcal{Q}'Qx,\; \tau',\; \psi\,)$$

such that $\mathcal{Q} = \mathcal{Q}_1\,Qx\,\mathcal{Q}_2$ with $Q \in \{\exists, \text{Я}^p\}$ and potentially empty $\mathcal{Q}_1$, $\varphi' = \varphi[v/x]$ with $v \in \mathbb{B}$, $\theta_l' \le \theta_u'$, and $\tau' = \tau \oplus [x \to v]$. Clearly, $\psi_{\tau'} = \varphi'$. Observe that each recursive call within DPLL-SSAT-R($\mathcal{Q} : \varphi,\ \theta_l,\ \theta_u,\ \mathcal{Q}',\ \tau,\ \psi$) satisfies above conditions on the inputs.

We consider unit propagation first. As $\varphi = \psi_\tau$ contains a unit clause with unit literal $\ell \in \{x, \neg x\}$, we conclude that there is a clause $(c'' \vee \ell) \in \psi$ with $\tau(c'') = \texttt{false}$. Since $\tau'(\ell) = \texttt{true}$, we deduce that $neg(\ell) \in c$ according to induction hypothesis, item 2. Thus, $c = (c' \vee neg(\ell))$ and item 2 holds for c' since $Var(c') = Var(c \setminus \{neg(\ell)\}) = Var(\mathcal{Q}'Qx) \setminus \{x\} = Var(\mathcal{Q}')$ and $\tau(c') = \tau'(c \setminus \{neg(\ell)\}) = \texttt{false}$. It clearly follows that $c'' \subseteq c'$ as $\tau(c'') = \texttt{false}$, $\tau(c') = \texttt{false}$ and $Var(c') = Var(\mathcal{Q}')$. The latter allows us to perform

$$c^{(pl,pu)} | \mathcal{Q}'\,Qx\,\mathcal{Q}_1\,\mathcal{Q}_2 : \psi \;\vdash_{\mathsf{R.3u}}\; (c')^{(pl',pu')} | \mathcal{Q}'\,\mathcal{Q} : \psi$$

that proves item 3. With regard to item 4, we conclude from definitions of pr', pl', pu', θ_l' and θ_u' by using common arithmetic laws the following: $pr' \in [\theta_l, \theta_u]$ if and only if $pr \in [\theta_l', \theta_u']$, $pr' < \theta_l$ if and only if $pr < \theta_l'$, and $pr' > \theta_u$ if and only if $pr > \theta_u'$. In case $Q = \exists$, the latter is clear since $\theta_l' = \theta_l$, $\theta_u' = \theta_u$, $pr' = pr$, $pl' = pl$, and $pu' = pu$. If $Q = \text{Я}^p$ then $\theta_l' = \theta_l/p(\ell)$, $\theta_u' = \theta_u/p(\ell)$, $pr' = p(\ell) \cdot pr$, $pl' = p(\ell) \cdot pl$, and $pu' = p(\ell) \cdot pu$ which also justifies above observation. Therefore, if $pr' \in [\theta_l, \theta_u]$ then $pr \in [\theta_l', \theta_u']$ and thus $pl = pu = pr$ due to induction hypothesis. By construction, $pl' = pu' = pr'$. Likewise, if $pr' < \theta_l$ then $pu' < \theta_l$ since $pu < \theta_l'$, and if $pr' > \theta_u$ then $pl' > \theta_u$ since $pl > \theta_u'$.

Let us consider purification next, i.e. $\varphi = \psi_\tau$ contains a pure literal $\ell \in \{x, \neg x\}$. Since $\tau'(\ell) = \texttt{true}$, we know that $neg(\ell) \in c$ according to induction hypothesis, item 2. Thus, $c = (c' \vee neg(\ell))$ and item 2 holds for c' since $Var(c') = Var(c \setminus \{neg(\ell)\}) = Var(\mathcal{Q}'Qx) \setminus \{x\} = Var(\mathcal{Q}')$ and $\tau(c') = \tau'(c \setminus \{neg(\ell)\}) = \texttt{false}$. As ℓ is pure, each clause $c''[\tau(y_1)/y_1]\ldots[\tau(y_m)/y_m] \in \psi_\tau$ with $\not\models c''[\tau(y_1)/y_1]\ldots[\tau(y_m)/y_m]$ where $\tau(y_i)$ is defined if and only if $1 \le i \le m$ does not contain literal $neg(\ell)$. As a consequence, for each clause $c'' \in \psi$ with $\not\models (c' \vee c'')$ it holds that $neg(\ell) \notin c''$. Note that $\not\models (c' \vee c'')$ is equivalent to $\not\models c''[\tau(y_1)/y_1]\ldots[\tau(y_m)/y_m]$ since $c'[\tau(y_1)/y_1]\ldots[\tau(y_m)/y_m] \equiv \texttt{false}$. This gives us the argument to apply

$$c^{(pl,pu)} | \mathcal{Q}'\,\exists x\,\mathcal{Q}_1\,\mathcal{Q}_2 : \psi \;\vdash_{\mathsf{R.3p}}\; (c')^{(pl',pu')} | \mathcal{Q}'\,\mathcal{Q} : \psi$$

that proves item 3. As $pr' = pr$, $pl' = pl$, $pu' = pu$, $\theta_l' = \theta_l$, and $\theta_u' = \theta_u$, item 4 follows instantaneously from induction hypothesis.

We finally elaborate on branching and thresholding. For the sake of clarity, we state explicitly the induction hypothesis, i.e. the statement of the lemma holds for

$$(\,pr_1,\;\; c_1^{(pl_1,pu_1)} \mid \mathcal{Q}'Qx\mathcal{Q}'' : \psi\,) \;=\; \text{DPLL-SSAT-R}(\,\mathcal{Q}'' : \varphi[\texttt{true}/x],\; \theta_l',\; \theta_u',\; \mathcal{Q}'Qx,\; \tau_1,\; \psi\,)$$

as well as for

$$(\,pr_2,\;\; c_2^{(pl_2,pu_2)} \mid \mathcal{Q}'Qx\mathcal{Q}'' : \psi\,) \;=\; \text{DPLL-SSAT-R}(\,\mathcal{Q}'' : \varphi[\texttt{false}/x],\; \theta_l'',\; \theta_u'',\; \mathcal{Q}'Qx,\; \tau_2,\; \psi\,)$$

where

- $\mathcal{Q} = Qx\mathcal{Q}''$ with $Q \in \{\exists, \text{Я}^p\}$,
- $\tau_1 = \tau \oplus [x \to \texttt{true}]$, $\tau_2 = \tau \oplus [x \to \texttt{false}]$,
- if $Q = \exists$ then $\theta'_l = \theta_l$, $\theta'_u = \theta_u$, $\theta''_l = \max(\theta_l, pr_1)$, $\theta''_u = \theta_u$, and
- if $Q = \text{Я}^p$ then $\theta'_l = (\theta_l - (1-p))/p$, $\theta'_u = \theta_u/p$, $\theta''_l = (\theta_l - p \cdot pr_1)/(1-p)$, $\theta''_u = (\theta_u - p \cdot pr_1)/(1-p)$.

Since $\tau_1(x) = \texttt{true}$ and $\tau_2(x) = \texttt{false}$, we know that $\neg x \in c_1$ and $x \in c_2$, respectively, according to induction hypothesis, item 2. Let us define $c'_1 := c_1 \setminus \{\neg x\}$ and $c'_2 := c_2 \setminus \{x\}$. Due to above facts, it is clear that $c_1 = (c'_1 \vee \neg x)$ and $c_2 = (c'_2 \vee x)$. Observe that $\tau_1(c'_1) = \tau(c'_1) = \texttt{false}$ and $\tau_2(c'_2) = \tau(c'_2) = \texttt{false}$ as well as that $Var(c'_1) = Var(c_1 \setminus \{\neg x\}) = Var(\mathcal{Q}'Qx) \setminus \{x\} = Var(\mathcal{Q}')$ and $Var(c'_2) = Var(c_2 \setminus \{x\}) = Var(\mathcal{Q}'Qx) \setminus \{x\} = Var(\mathcal{Q}')$. This implies that $c'_1 = c'_2$. Above reasoning shows that item 2 holds for c'_1 and for $c'_1 \cup c'_2$. Let be

- $pr'_1 := pr_1$, $pl'_1 := pl_1$, $pu'_1 := 1$ if $Q = \exists$, and
- $pr'_1 := p \cdot pr_1$, $pl'_1 := p \cdot pl_1$, $pu'_1 := p \cdot pu_1 + (1-p)$ if $Q = \text{Я}^p$.

We first consider the case where thresholding applies, i.e. if $pr'_1 > \theta_u$ or if $pr'_1 + (1-p) < \theta_l$ and $Q = \text{Я}^p$, then DPLL-SSAT-R returns $(pr'_1,\ (c'_1)^{(pl'_1, pu'_1)} | \mathcal{Q}'\mathcal{Q} : \psi)$. The valid execution of

$$c_1^{(pl_1,pu_1)} | \mathcal{Q}'Qx\mathcal{Q}'' : \psi \ \vdash_{\mathsf{R.3t}} \ (c_1 \setminus \{\neg x\})^{(pl'_1, pu'_1)} | \mathcal{Q}'\mathcal{Q} : \psi$$

proves item 3. With respect to item 4, observe that $pr'_1 \notin [\theta_l, \theta_u]$ by assumption. This trivially implies that if $pr'_1 \in [\theta_l, \theta_u]$ then $pl'_1 = pu'_1 = pr'_1$. By above definitions and by using common arithmetic laws, we have that $pr'_1 > \theta_u$ if and only if $pr_1 > \theta'_u$. If $pr'_1 < \theta_l$ then $pr'_1 + (1-p) < \theta_l$ and $Q = \text{Я}^p$ by assumption above and due to $\theta_l \leq \theta_u$. Thus, $pr'_1 + (1-p) < \theta_l$ if and only if $pr_1 < \theta'_l$ as $Q = \text{Я}^p$. By induction hypothesis, item 4, if $pr_1 > \theta'_u$ then $pl_1 > \theta'_u$ and if $pr_1 < \theta'_l$ then $pu_1 < \theta'_l$. To show item 4, we finally reason as follows: $pl_1 > \theta'_u$ if and only if $pl'_1 > \theta_u$, and $pu_1 < \theta'_l$ if and only if $pu'_1 < \theta_l$ since $Q = \text{Я}^p$.

Otherwise, i.e. if, let be

- $pr' := \max(pr_1, pr_2)$, $pl' := \max(pl_1, pl_2)$, $pu' := \max(pu_1, pu_2)$ if $Q = \exists$, and
- $pr' := p \cdot pr_1 + (1-p) \cdot pr_2$, $pl' := p \cdot pl_1 + (1-p) \cdot pl_2$, $pu' := p \cdot pu_1 + (1-p) \cdot pu_2$ if $Q = \text{Я}^p$.

DPLL-SSAT-R then returns $(pr',\ (c'_1 \cup c'_2)^{(pl', pu')} | \mathcal{Q}'\mathcal{Q} : \psi)$. Recall that $c_1 = (c'_1 \vee \neg x)$ and $c_2 = (c'_2 \vee x)$ as well as that $c'_1 = c'_2$. The latter ensures applicability of

$$(c_1^{(pl_1,pu_1)} | \mathcal{Q}'\mathcal{Q} : \psi,\ c_2^{(pl_2,pu_2)} | \mathcal{Q}'\mathcal{Q} : \psi) \vdash_{\mathsf{R.3}} (c'_1 \cup c'_2)^{(pl',pu')} | \mathcal{Q}'\mathcal{Q} : \psi \ .$$

which proves item 3. We finally need to show item 4. Let first be $Q = \exists$. Then, $pr' = \max(pr_1, pr_2)$. Let be $pr' \in [\theta_l, \theta_u]$. If $pr' = pr_1$ then $pr_1 \in [\theta'_l, \theta'_u]$. By induction hypothesis, $pl_1 = pu_1 = pr_1$. Since $pr_2 \leq pr_1$, we have that $pl_2 \leq pu_2 \leq pl_1 = pu_1$. The latter clearly holds if $pr_2 \in [\theta''_l, \theta''_u]$ since then $pl_2 = pu_2 = pr_2 \leq pr_1$ by induction hypothesis. If $pr_2 < \theta''_l$ then $pu_2 < \theta''_l$ by induction hypothesis and therefore $pl_2 \leq$

$pu_2 < \theta_l'' = \max(\theta_l, pr_1) = \max(\theta_l', pr_1) \le pr_1$. By application condition of R.3, $pl' = pl_1$ and $pu' = pu_1$, and thus $pl' = pu' = pr'$. Otherwise, i.e. if $pr' = pr_2$, we have that $pr_2 \in [\theta_l'', \theta_u'']$. Induction hypothesis gives $pl_2 = pu_2 = pr_2$. Due to the fact that $pr_1 \le pr_2$, we infer that $pl_1 \le pu_1 \le pl_2 = pu_2$. As above, if $pr_1 \in [\theta_l', \theta_u']$ then $pl_1 = pu_1 = pr_1 \le pr_2$ by induction hypothesis, and if $pr_1 < \theta_l'$ then $pu_1 < \theta_l'$ by induction hypothesis and therefore $pl_1 \le pu_1 < \theta_l' = \theta_l \le pr_2$. Rule R.3 ensures $pl' = pl_2$ and $pu' = pu_2$, and thus $pl' = pu' = pr'$. We next consider case $pr' < \theta_l$. Obviously, $pr_1 < \theta_l'$ and $pr_2 < \theta_l''$. By induction hypothesis, $pu_1 < \theta_l'$ and $pu_2 < \theta_l''$. Note that $\theta_l' = \theta_l$ by definition and that $\theta_l'' = \max(\theta_l, pr_1) = \theta_l$ as $pr_1 < \theta_l' = \theta_l$. Therefore, $pu_1 < \theta_l$ and $pu_2 < \theta_l$ and thus $pu' < \theta_l$ according to R.3. If $pr' > \theta_u$ then $pr_2 > \theta_u'' = \theta_u$ since thresholding has failed, i.e. $pr_1 \le \theta_u$. It follows that $pl_2 > \theta_u$ by induction hypothesis, and that $pl' = \max(pl_1, pl_2) > \theta_u$ due to R.3.

Let second be $Q = \text{Я}^p$. Then, $pr' = p \cdot pr_1 + (1-p) \cdot pr_2$. As thresholding has failed, we know that $p \cdot pr_1 + (1-p) \ge \theta_l$ and that $p \cdot pr_1 \le \theta_u$, i.e. $pr_1 \in [\theta_l', \theta_u']$. Induction hypothesis thus yields $pl_1 = pu_1 = pr_1$. If $pr' \in [\theta_l, \theta_u]$ then $pr_2 \ge (\theta_l - p \cdot pr_1)/(1-p) = \theta_l''$ and $pr_2 \le (\theta_u - p \cdot pr_1)/(1-p) = \theta_u''$. By induction hypothesis, $pl_2 = pu_2 = pr_2$. It follows that $pl' = pu' = pr'$ according to R.3. If $pr' < \theta_l$ then $pr_2 < (\theta_l - p \cdot pr_1)/(1-p) = \theta_l''$. Induction hypothesis states that $pu_2 < \theta_l'' = (\theta_l - p \cdot pr_1)/(1-p)$, and thus $p \cdot pr_1 + (1-p) \cdot pu_2 = pu' < \theta_l$ as $pr_1 = pu_1$. If $pr' > \theta_u$ then $pr_2 > (\theta_u - p \cdot pr_1)/(1-p) = \theta_u''$. Then, $pl_2 > \theta_u'' = (\theta_u - p \cdot pr_1)/(1-p)$ by induction hypothesis. Finally, $p \cdot pr_1 + (1-p) \cdot pl_2 = pl' > \theta_u$ as $pr_1 = pl_1$.

This completes the proof and the lemma follows. $\square$

The next result finally shows that item 1 from the beginning of this subsection holds. Loosely speaking, enhanced S-resolution is capable of producing proofs for the generalized SSAT decision problem that are *never longer* than the shortest proofs generated by DPLL-SSAT.

Corollary 6.2 (Shortest S-resolution proofs never longer than DPLL-SSAT **proofs)** *Let $(\Phi, \theta_l, \theta_u)$ be an instance of the generalized SSAT decision problem and let some corresponding* DPLL-SSAT$(\Phi, \theta_l, \theta_u)$ *proof be of length k. Then, it is always feasible to construct an enhanced S-resolution proof for $(\Phi, \theta_l, \theta_u)$ that is of the same size k.*

Proof. Let be $\Phi = \mathcal{Q} : \varphi$. We denote the empty function by $\tau_\emptyset$ and the empty quantifier prefix by ε. Let further be

$$(\, pr,\ \ c^{(pl,pu)} \mid \Phi \,) \ \ := \ \ \text{DPLL-SSAT-R}(\, \Phi,\ \theta_l,\ \theta_u,\ \varepsilon,\ \tau_\emptyset,\ \varphi \,) \ .$$

First of all, it is not hard to see that

$$pr \ \ = \ \ \text{DPLL-SSAT}(\, \Phi,\ \theta_l,\ \theta_u \,)$$

and that the number of recursions of DPLL-SSAT-R$(\Phi, \theta_l, \theta_u, \varepsilon, \tau_\emptyset, \varphi)$ is the same as of DPLL-SSAT$(\Phi, \theta_l, \theta_u)$, namely k. The latter is true since the additional inputs and the extended output of DPLL-SSAT-R do *not* have an effect on the computation of the probabilities pr and on the common inputs Φ, θ_l, and θ_u.

By Lemma 6.4, we have that pair $c^{(pl,pu)} | \Phi$ is derivable by enhanced S-resolution and that $c = \emptyset$ as $\mathit{Var}(c) = \mathit{Var}(\varepsilon) = \emptyset$. From soundness of DPLL-SSAT, it follows that

$pr = Pr(\Phi)$ if $Pr(\Phi) \in [\theta_l, \theta_u]$, $pr < \theta_l$ if $Pr(\Phi) < \theta_l$, and $pr > \theta_u$ if $Pr(\Phi) > \theta_u$. Using above facts as well as Lemma 6.4, we furthermore conclude that

- if $Pr(\Phi) \in [\theta_l, \theta_u]$ then $pr = Pr(\Phi) \in [\theta_l, \theta_u]$ and then $pl = pu = pr = Pr(\Phi)$,
- if $Pr(\Phi) < \theta_l$ then $pr < \theta_l$ and then $pu < \theta_l$, and
- if $Pr(\Phi) > \theta_u$ then $pr > \theta_u$ and then $pl > \theta_u$.

Summarizing, DPLL-SSAT-R$(\Phi, \theta_l, \theta_u, \varepsilon, \tau_\emptyset, \varphi)$ produces an enhanced S-resolution proof for $(\Phi, \theta_l, \theta_u)$. We finally observe that in each recursive call of DPLL-SSAT-R exactly one new pair $(c')^{(pl',pu')}|\Phi'$, namely the one returned in the result of the call, is derived by some rule of enhanced S-resolution. Thus, the enhanced S-resolution proof mentioned above is of size k. This proves the claim. □

We remark that Corollary 6.2 improves a previous result on the proof complexity of S-resolution and DPLL-SSAT that was published in [TF10]. The latter article has shown that if $Pr(\Phi) < \theta_l$ then from each DPLL-SSAT$(\Phi, \theta_l, \theta_u)$ proof of size k, it is feasible to construct an S-resolution proof with a quadratic overhead, i.e. of size $\mathcal{O}(k^2)$, confer [TF10, Proposition 2]. This restriction, i.e. $Pr(\Phi) < \theta_l$, was imposed as [TF10] considers a version of S-resolution where derived clauses c^{pu} provide upper probability bounds pu only. By extending S-resolution such that derived clauses $c^{(pl,pu)}$ carry lower probability bounds pl in addition to upper ones pu as it was done in this section, Corollary 6.2 can state a more general result by not imposing $Pr(\Phi) < \theta_l$ and by improving the S-resolution proof size to k.

Proposition 6.1 now deals with item 2 from the beginning of this subsection, claiming that S-resolution proofs are sometimes *significantly shorter* than the shortest proofs generated by DPLL-SSAT. For that purpose, Proposition 6.1 states explicitly an infinite family of SSAT instances for which the shortest S-resolution proofs are of constant size while DPLL-SSAT needs exponentially many computation steps even in best case.

Proposition 6.1 (S-resolution proofs can be much shorter than DPLL-SSAT proofs) *Let*

$$\Phi_n = \mathcal{Q}_n \; \mathcal{Q}'_n \; \mathcal{Q}''_n : \varphi_n$$

with $n \in \mathbb{N}_{>0}$ be an infinite family of SSAT formulae such that

$$\begin{array}{rcllcll} \mathcal{Q}_n & = & \exists x_{1,1} & \exists x_{1,2} & \ldots & \exists x_{1,n-1} & \exists x_{1,n} & , \\ \mathcal{Q}'_n & = & Q_{2,1} x_{2,1} & Q_{2,2} x_{2,2} & \ldots & Q_{2,n-1} x_{2,n-1} & Q_{2,n} x_{2,n} & , \\ \mathcal{Q}''_n & = & Q_{3,1} x_{3,1} & Q_{3,2} x_{3,2} & \ldots & Q_{3,n-1} x_{3,n-1} & Q_{3,n} x_{3,n} & \end{array}$$

where $Q_{j,i} \in \{\exists, \text{Я}^{p_{j,i}}\}$ with $0 < p_{j,i} < 1$ for $2 \le j \le 3$ and for $1 \le i \le n$, as well as

$$\varphi_n = \bigwedge_{i=1}^{n} all\_comb(x_{1,i}, x_{2,i}, x_{3,i})$$

where

$$all\_comb(x,y,z) = \left(\begin{array}{ccccccccccccccc} & (x & \vee & y & \vee & z) & \wedge & (\neg x & \vee & y & \vee & z) \\ \wedge & (x & \vee & y & \vee & \neg z) & \wedge & (\neg x & \vee & y & \vee & \neg z) \\ \wedge & (x & \vee & \neg y & \vee & z) & \wedge & (\neg x & \vee & \neg y & \vee & z) \\ \wedge & (x & \vee & \neg y & \vee & \neg z) & \wedge & (\neg x & \vee & \neg y & \vee & \neg z) \end{array} \right)$$

is a propositional formula in 3CNF that consists of all non-tautological clauses with the three variables x, y, z. Then, for each instance $(\Phi_n, \theta_l, \theta_u)$ which is non-trivial in the sense that $\theta_u \geq 0$,

1. *there is an enhanced S-resolution proof for $(\Phi_n, \theta_l, \theta_u)$ of size (at most) 15, i.e. of constant size, and*

2. *each* DPLL-SSAT$(\Phi_n, \theta_l, \theta_u)$ *proof is of size at least 2^n, i.e. of size exponential in n.*

Proof. First of all, observe that $\mathit{all\_comb}(x_{1,i}, x_{2,i}, x_{3,i})$ is unsatisfiable for each i. Since $n > 0$, it follows that φ_n is unsatisfiable and thus $Pr(\Phi_n) = 0$.

To show item 1, we present an S-resolution proof with 15 rule applications. As $n > 0$, $\mathit{all\_comb}(x_{1,1}, x_{2,1}, x_{3,1}) \in \varphi_n$. This allows the following rule applications:

$$\begin{aligned}
&1)\ \ \varepsilon|\Phi_n \vdash_{\mathsf{R.1}} (x_{1,1} \vee x_{2,1} \vee x_{3,1})^{(0,0)}|\Phi_n \ \ ,\\
&2)\ \ \varepsilon|\Phi_n \vdash_{\mathsf{R.1}} (x_{1,1} \vee x_{2,1} \vee \neg x_{3,1})^{(0,0)}|\Phi_n \ \ ,\\
&\qquad\qquad\qquad\qquad \dots\\
&8)\ \ \varepsilon|\Phi_n \vdash_{\mathsf{R.1}} (\neg x_{1,1} \vee \neg x_{2,1} \vee \neg x_{3,1})^{(0,0)}|\Phi_n \ \ .
\end{aligned}$$

We proceed with

$$\begin{aligned}
&9)\ ((x_{1,1} \vee x_{2,1} \vee x_{3,1})^{(0,0)}, (x_{1,1} \vee x_{2,1} \vee \neg x_{3,1})^{(0,0)})|\Phi_n \vdash_{\mathsf{R.3}} (x_{1,1} \vee x_{2,1})^{(0,0)}|\Phi_n \ ,\\
&10)\ ((x_{1,1} \vee \neg x_{2,1} \vee x_{3,1})^{(0,0)}, (x_{1,1} \vee \neg x_{2,1} \vee \neg x_{3,1})^{(0,0)})|\Phi_n \vdash_{\mathsf{R.3}} (x_{1,1} \vee \neg x_{2,1})^{(0,0)}|\Phi_n \ ,\\
&11)\ ((\neg x_{1,1} \vee x_{2,1} \vee x_{3,1})^{(0,0)}, (\neg x_{1,1} \vee x_{2,1} \vee \neg x_{3,1})^{(0,0)})|\Phi_n \vdash_{\mathsf{R.3}} (\neg x_{1,1} \vee x_{2,1})^{(0,0)}|\Phi_n \ ,\\
&12)\ ((\neg x_{1,1} \vee \neg x_{2,1} \vee x_{3,1})^{(0,0)}, (\neg x_{1,1} \vee \neg x_{2,1} \vee \neg x_{3,1})^{(0,0)})|\Phi_n \vdash_{\mathsf{R.3}} (\neg x_{1,1} \vee \neg x_{2,1})^{(0,0)}|\Phi_n \ .
\end{aligned}$$

Then,

$$\begin{aligned}
&13)\ ((x_{1,1} \vee x_{2,1})^{(0,0)}, (x_{1,1} \vee \neg x_{2,1})^{(0,0)})|\Phi_n \vdash_{\mathsf{R.3}} (x_{1,1})^{(0,0)}|\Phi_n \ ,\\
&14)\ ((\neg x_{1,1} \vee x_{2,1})^{(0,0)}, (\neg x_{1,1} \vee \neg x_{2,1})^{(0,0)})|\Phi_n \vdash_{\mathsf{R.3}} (\neg x_{1,1})^{(0,0)}|\Phi_n \ ,
\end{aligned}$$

and finally

$$15)\ ((x_{1,1})^{(0,0)}, (\neg x_{1,1})^{(0,0)})|\Phi_n \vdash_{\mathsf{R.3}} \emptyset^{(0,0)}|\Phi_n \ .$$

For item 2, observe that DPLL-SSAT first assigns successively truth values to all variables $x_{1,1}, x_{1,2}, \dots, x_{1,n-1}, x_{1,n}$. During this process, construction of φ_n ensures that, first, the base cases (where φ_n becomes `true` or `false`) are *not* reached and that, second, *no* literal in φ_n becomes unit or pure. Therafter, branching for variable $x_{2,1}$ is executed, i.e. some truth value v is subsituted for $x_{2,1}$. (Actually, $v =$ `true` but this does not make any difference.) As $n > 0$, the current formula contains exactly two unit clauses, namely $(x_{3,1})$ and $(\neg x_{3,1})$ which arise from the corresponding clauses in $\mathit{all\_comb}(x_{1,1}, x_{2,1}, x_{3,1})$ depending on which values are assigned to $x_{1,1}$ and $x_{2,1}$. With regard to the latter, note that for each of the four possible assignments to $x_{1,1}$ and $x_{2,1}$, the unit clauses $(x_{3,1})$ and $(\neg x_{3,1})$ exist due to construction of $\mathit{all\_comb}(x_{1,1}, x_{2,1}, x_{3,1})$. Then, unit propagation is performed for one of the unit literals $x_{3,1}$ and $\neg x_{3,1}$. Irrespective of which unit literal was chosen, one of the clauses $(x_{3,1})$ and $(\neg x_{3,1})$ becomes equivalent to `false` after substitution. As a consequence, DPLL-SSAT has reached the base case where φ_n evaluates to `false` under current partial assignment. Therefore, the branch where $x_{2,1}$ is set to v yields probability $pr_1 = 0$. As $\theta_u \geq 0$, thresholding is only applicable if $Q_{2,1} = \reflectbox{R}^{p_{2,1}}$ and $1 - p_{2,1} < \theta_l$. In the latter case the returned probability is 0. Otherwise, i.e. thresholding fails, DPLL-SSAT also investigates the other branch for $x_{2,1}$, i.e. where the opposite value $neg(v)$ is substituted for $x_{2,1}$. After substitution, unit clauses $(x_{3,1})$ and $(\neg x_{3,1})$ recur, again leading to base case "`false`". In this case, the returned probability result is also 0. Thus, DPLL-SSAT needs to try the other branch for $x_{1,n}$ in any case since $x_{1,n}$ is an existential variable and $\theta_u \geq 0$. Then, branching is performed for $x_{2,1}$ again. We now state two simple but important facts. First, base case "`false`" is revisited if and only if all variables $x_{1,1}, x_{1,2}, \ldots, x_{1,n-1}, x_{1,n}$ as well as variable $x_{2,1}$ are assigned truth values by means of substitution. Second, thresholding will never apply for variables $x_{1,1}, x_{1,2}, \ldots, x_{1,n-1}, x_{1,n}$ since these variables are existentially quantified, the probability results are always 0, and $\theta_u \geq 0$. The latter implies that each of the opposite branches for variables $x_{1,1}, x_{1,2}, \ldots, x_{1,n-1}, x_{1,n}$ is actually explored. We therefore conclude that DPLL-SSAT needs to traverse all 2^n assignments to the variables $x_{1,1}, x_{1,2}, \ldots, x_{1,n-1}, x_{1,n}$ in order to solve $(\Phi_n, \theta_l, \theta_u)$. That is, the number of recursive calls and thus the size of the DPLL-SSAT$(\Phi_n, \theta_l, \theta_u)$ proof is at least 2^n. □

We would like to clarify that Proposition 6.1 along with Corollary 6.2 should not be misconceived in the sense that enhanced S-resolution implements a stand-alone algorithm that always outperforms DPLL-SSAT. Proposition 6.1 gives only evidence about the shortest S-resolution proofs for a particular (yet infinite) family of SSAT formulae. Moreover, it is not specified how a strategy to produce shortest proofs can be devised. Both Proposition 6.1 and Corollary 6.2 should serve the sole purpose of indicating the potential of S-resolution. It is furthermore important to remark that Proposition 6.1 refers to the classical DPLL-SSAT procedure which does not take into account the additional algorithmic enhancements mentioned at the end of Subsection 6.2.1. Without going into detail, the size of the DPLL-SSAT proof for instance $(\Phi_n, \theta_l, \theta_u)$ from Proposition 6.1 can be significantly reduced to $\mathcal{O}(n)$ by exploiting the idea of *non-chronological backtracking* [Maj04].

We briefly remark that *explanations* for conflicts and solutions that are used for non-chronological backtracking, confer Subsection 6.2.1, are closely related to clauses derived by S-resolution. It is moreover feasible to enhance DPLL-SSAT-R such that a non-chronological backtracking operation can be realized by means of derived clauses. To this end, clauses c in the base cases of DPLL-SSAT-R, i.e. in lines 7 and 10, should be *minimized* by removing "unnecessary" literals as long as the resulting clauses c' are still derivable by rules R.1g (or rather R.1) and R.2s, respectively. Then, a further technique to skip solving the second subproblems within branching, i.e. to skip recursive calls in lines 48 and 70, becomes available that corresponds to non-chronological backtracking: due to clause minimization in the base cases, literal $\neg x$ need not be present in clause c_1, i.e. it potentially holds that $c'_1 = c_1 \setminus \{\neg x\} = c_1$, confer lines 41, 43, 58, and 63. We now suppose that $\neg x \notin c_1$ is actually true. Let be $\mathcal{Q}' = Q_1 x_1 \ldots Q_j x_j$ and $\mathcal{Q}'(c_1) = Q_1 x_1 \ldots Q_i x_i$. Obviously, $i \leq j$. According to Lemma 6.3,

$$pl_1 \quad \leq \quad Pr(Q_{i+1} x_{i+1} \ldots Q_j x_j \, Qx \, \mathcal{Q}'' : \psi[\tau(x_1)/x_1] \ldots [\tau(x_i)/x_i]) \quad \leq \quad pu_1$$

with $\tau : \mathit{Var}(\mathcal{Q}'(c_1)) \to \mathbb{B}$ such that $\forall y \in \mathit{Var}(c_1) : \tau(y) = \mathit{ff}_c(y)$. That is, exactly the same probability bounds pl_1 and pu_1 are valid for the second branch of x. This allows to skip the second branch and to immediately return $(pr_1,\ c_1^{(pl_1,pu_1)}|\mathcal{Q}'\mathcal{Q} : \psi)$. Moreover, the same applies successively for all variables x_j down to x_{i+1}, establishing a non-chronological backtracking operation.

The quintessence of this subsection is that S-resolution should *not* be considered as a *competitive* approach to solve SSAT problems in practice *but* as *complementary* to DPLL-SSAT as both, S-resolution and DPLL-SSAT, are capable of "cross-fertilizing" each other:

- on the one hand, DPLL-SSAT or rather DPLL-SSAT-R is most likely the best choice to achieve a general and, at the same time, practically reasonable strategy for S-resolution and,
- on the other hand, S-resolution opens a variety of new ideas to enhance performance as well as applicability of DPLL-SSAT solvers, some of them are touched upon hereafter.

Applications of S-resolution. One promising application of S-resolution is to develop a *generalized clause learning scheme* for DPLL-SSAT that is motivated by the success of *conflict-driven clause learning* for DPLL, confer Section 6.1 and [MSLM09]. The DPLL-SSAT-R procedure indicates the construction of clauses $c^{(pl,pu)}$ derivable by (enhanced) S-resolution. As mentioned before, these clauses can be exploited for *non-chronological backtracking*. Recall that unit propagation within the DPLL procedure also works for *conflict clauses* as these clauses are implications of the given formula. On the contrary, clauses $c^{(pl,pu)}$ derived by S-resolution are *not* necessarily semantic consequences but just entailed with some probability. That is, unit propagation is in general not applicable for derived clauses $c^{(pl,pu)}$. Notwithstanding, it would be worthwhile and promising to conceive of a similar scheme for the probabilistic case, i.e. some kind of *generalized unit propagation*. We illustrate our idea to the latter issue by means of an example: suppose that we have derived the clause $(\neg x_3 \vee x_{21} \vee \neg x_{85})^{(0.5,0.6)}$ at some point of the DPLL-SSAT-R search. Let us further assume that the current subproblem is $Q_{22}x_{22} \ldots Q_{85}x_{85}\mathcal{Q} : \varphi$ and that `true` was substituted for x_3 and `false` for x_{21} such that above clause becomes "unit". Due to derived clause $(\neg x_3 \vee x_{21} \vee \neg x_{85})^{(0.5,0.6)}$ and according to Lemma 6.3, we know that whenever we assign *arbitrary* values to variables $x_{22}, \ldots, x_{84}$ and `true` to x_{85} then the satisfaction probability of the resulting SSAT formula lies within $[0.5, 0.6]$. It is therefore unreasonable to assign `true` to x_{85} later on as the corresponding satisfaction probability is already known. That is, we set x_{85} to `false` in any case. Though we are already sure of which value we assign to x_{85}, we *cannot* do this, in general, before the variables $x_{22}, \ldots, x_{84}$ are assigned. The rationale is that such a latter operation would involve modification of the quantifier prefix from $Q_{22}x_{22} \ldots Q_{85}x_{85}\mathcal{Q}$ to $Q_{85}x_{85}\, Q_{22}x_{22} \ldots Q_{84}x_{84}\mathcal{Q}$ which is not correct in general. This dilemma is dissatisfactory in a sense. However, a simple observation may be helpful in several cases. Suppose that there is some variable x_i in between $Q_{22}x_{22}$ and $Q_{85}x_{85}$, i.e. $22 \leq i \leq 85$, such that all variables $x_i, \ldots, x_{85}$ are bound by the same quantifier Q, i.e. $Q = Q_i = Q_{i+1} = \ldots = Q_{85}$. Since quantifiers within a block of same quantifiers may be moved arbitrarily, we are allowed to modify the current formula

$$Q_{22}x_{22} \ \ldots \ Q_{i-1}x_{i-1} \ Qx_i \ \ldots \ Qx_{84} \ Qx_{85} \ \mathcal{Q} : \varphi$$

to

$$Q_{22}x_{22} \ \ldots \ Q_{i-1}x_{i-1} \ Qx_{85} \ Qx_i \ \ldots \ Qx_{84} \ \mathcal{Q} : \varphi$$

while preserving the semantics, i.e.

$$\begin{aligned} & Pr(Q_{22}x_{22} \ \ldots \ Q_{i-1}x_{i-1} \ Qx_i \ \ldots \ Qx_{84} \ Qx_{85} \ \mathcal{Q} : \varphi) \\ = \ & Pr(Q_{22}x_{22} \ \ldots \ Q_{i-1}x_{i-1} \ Qx_{85} \ Qx_i \ \ldots \ Qx_{84} \ \mathcal{Q} : \varphi) \ . \end{aligned}$$

This facilitates to assign `true` to x_{85} potentially much earlier, namely immediately after $x_{22}, \ldots, x_{i-1}$ are assigned. Observe that such variable x_i always exists as in "worst case" $x_i = x_{85}$ may be taken, and that i should be chosen as small as possible for efficiency reasons. We further remark that this *generalized unit propagation* actually generalizes unit propagation employed in the DPLL procedure as the variables of propositional formulae can be considered as existentially quantified, i.e. $x_i = x_{22}$ would hold.

While a realization of generalized unit propagation as sketched above potentially improves performance of DPLL-SSAT solvers in practice, the following ideas enhance their applicability:

- Akin to *proofs of unsatisfiability* produced by resolution for propositional formulae, confer Subsection 6.2.2, S-resolution provides a theoretical framework to generate *proofs of satisfiability with insufficient probability.* In the propositional case, such unsatisfiability proofs are exploited to validate SAT solvers [ZM03b, GN03]. The rationale is that modern SAT solvers are very complex pieces of software and thus prone to errors, as they incorporate several sophisticated algorithmic optimizations based on very complex data structures, confer Section 6.1 and, for a very detailed account, [BHvMW09]. On the contrary, resolution establishes a much simpler calculus that consists of only one resultion rule in the propositional case. That is, cross-checking the result of a SAT solver by means of validating a resolution proof increases the confidence of the result significantly. We remark that reliability of computational results is vitally important and, moreover, compulsory in the verification of safety-critical systems.

  Above motivation clearly carries over to the DPLL-SSAT case. Though enhanced S-resolution comprises some more rules than "common" resolution, it provides a yet simple enough calculus such that cross-checking the results of DPLL-SSAT solvers still leads to more confidence.

- In addition to checking the validity of unsatisfiability results of SAT solvers, the extraction of *small or even minimal unsatisfiable subformulae* is another application of unsatisfiability proofs [ZM03b, GN03, ZM03a, LMS04]. Such small unsatisfiable cores can potentially help to analyze why some system does not satisfy a desired property, for instance, when considering some planning or scheduling problems. Given an unsatisfiable propositional formula φ in CNF, a small unsatisfiable subformula $\varphi' \subseteq \varphi$ is obtained from a resolution proof for φ by collecting all original clauses $c \in \varphi$ that actually occur in the resolution proof. To potentially achieve smaller unsatisfiable subformulae, this process is iterated on the correspondingly previous subformula until a fixed point is reached as, for instance, in [ZM03b].

In the SSAT case, S-resolution offers the possibility to generate small or even minimal SSAT subformulae $\mathcal{Q} : \varphi'$ having the *same satisfaction probabilities* as the original SSAT formula $\mathcal{Q} : \varphi$, i.e. $Pr(\mathcal{Q} : \varphi') = Pr(\mathcal{Q} : \varphi)$ with $\varphi' \subseteq \varphi$. Moreover, S-resolution can be exploited for a more general setting by taking the idea of *thresholds* into account, confer the generalized SSAT decision problem given by instances $(\mathcal{Q} : \varphi, \theta_l, \theta_u)$: whenever $Pr(\mathcal{Q} : \varphi) < \theta_l$ or $Pr(\mathcal{Q} : \varphi) > \theta_u$ then we do not demand that the small SSAT subformula $\mathcal{Q} : \varphi'$ must have the same satisfaction probability as $\mathcal{Q} : \varphi$ but rather $Pr(\mathcal{Q} : \varphi') < \theta_l$ or $Pr(\mathcal{Q} : \varphi') > \theta_u$, respectively. The latter relaxation may lead to much smaller subformulae $\varphi' \subseteq \varphi$.

- One of the most interesting and, to some extent, most challenging topics is the potential application of S-resolution for computing *stochastic variants of Craig interpolants*. In Chapter 9, we elaborate on such a technique and illustrate the potential use of "stochastic" Craig interpolation in SSAT-based bounded model checking of probabilistic (finite-state) systems like Markov decision processes, turning the *falsification* procedure of Chapter 5 into a *verification* approach for probabilistic safety properties.

6.3 Algorithms for SMT

This section elaborates on solving procedures for SMT formulae. As explained in Section 4.3, the concept of SMT extends Boolean satisfiability with respect to background theories like *equality logic with uninterpreted functions*, *arithmetic* like difference logic or linear arithmetic, the theories of *arrays*, *bit vectors*, and *inductive data types*. In recent years, the development of highly efficient SMT solvers, supporting above and several other theories as well as combinations of different theories, has evolved to a very active research area. With regard to the analysis of discrete-time probabilistic hybrid automata, being the main target of this thesis, we direct our attention to the *theory of non-linear arithmetic* over the reals and integers involving transcendental functions like exponential and trigonometric functions. For an overview of SMT algorithms for above mentioned theories, we refer to the survey [BSST09].

Efficient algorithms addressing quantifier-free *Boolean combinations of non-linear arithmetic constraints* including transcendental functions are rather scarce. In [FHR+06, FHT+07], we have presented an algorithmic approach to above problem that is called the iSAT algorithm. To cope with the Boolean structure of the problem as well as with the non-linear arithmetic part, iSAT basically integrates the DPLL procedure, confer Section 6.1, with techniques from *interval analysis* [Moo66, Moo79, Moo80], the latter establishing a very general method to deal with non-linearities, in particular transcendental functions. There are two implementations of the iSAT algorithm, namely the HySAT-II[2] tool and its successor iSAT[3], the latter sharing the same name with the un-

[2]More information can be found on `http://hysat.informatik.uni-oldenburg.de`.

[3]More information can be found on `http://isat.gforge.avacs.org`.

derlying algorithm. Both tools are especially designed for *bounded model checking* of hybrid systems [HEFT08].

Another approach to the same problem has been proposed by Bauer, Pister, and Tautschnig [BPT07]. The latter authors developed the tool ABsolver[4], which permits the integration of various subordinate solvers for the Boolean, linear, and non-linear parts of the input formula. ABsolver itself coordinates the overall solving process and delegates the currently active constraint sets to the corresponding subordinate solvers. To address non-linear constraints, ABsolver uses the numerical optimization tool Ipopt[5]. Consequently, it may produce incorrect results due to the local nature of the solver and due to rounding errors. In [FHT$^+$07, Subsection 5.3], we have shown that iSAT consistently outperforms ABsolver, usually by orders of magnitude when formulae with non-trivial Boolean structure are involved.

More recently, a further approach to solve non-linear arithmetic SMT formulae has been proposed by the authors of [GGI$^+$10]. Their algorithm pursues the idea of combining DPLL and interval analysis by means of additionally integrating a special and more efficient subordinate solver for *linear* arithmetic. The experimental results given in [GGI$^+$10] show that this more sophisticated approach can lead to significant performance gains, in particular on non-linear SMT problems involving huge *linear* parts. We remark that the authors of [ABDK11] enhanced the iSAT tool with equivalent reasoning capability by also integrating a subordinate solver for *linear* arithmetic.

The SSMT algorithm that is introduced in Section 6.4 extends the iSAT procedure by adding an additional layer for quantifier treatment. For this reason, we examine the iSAT approach in more detail. Subsection 6.3.1 first gives an intuitive insight into iSAT, which is then followed by a more formal presentation of the algorithmic details in Subsection 6.3.2.

6.3.1 Intuitive description of iSAT

The iSAT algorithm [FHR$^+$06, FHT$^+$07] has been designed to address the satisfiability problem of SMT formulae with respect to the theory of non-linear arithmetic over the reals and integers. As an input, iSAT requires an SMT formula φ such that

1. φ is in *conjunctive form* (CF), confer Definition 4.3, and
2. the domains $\mathrm{dom}(x)$ of all variables $x \in \mathit{Var}(\varphi)$ are given by *bounded intervals*.

As already explained in Subsection 4.3.1 and as observed in [Her10, Chapter 5], above requirement 1 is without loss of generality since it is always possible to efficiently rewrite an arbitrary non-linear arithmetic SMT formula into an equi-satisfiable SMT formula in CF by means of a generalized version of the Tseitin transformation [Tse68]. We further remark that the aforementioned tools HySAT-II and iSAT accept arbitrary SMT formulae and convert the inputs into CF automatically.

Above requirement 2, however, actually poses a restriction as each variable of the input formula has to range in a bounded interval. Note that the latter is also required for each auxiliary variable introduced by the Tseitin transformation process. From a practical

[4]More information can be found on `http://absolver.sourceforge.net`.
[5]More information can be found on `https://projects.coin-or.org/Ipopt`.

perspective, this prerequisite seems not to be too restrictive as variables encoding physical quantities like *temperature*, *velocity*, or *volume* are naturally bounded in their values. In cases where such an estimation should not be feasible for any reason, the lower and upper interval borders can be chosen arbitrarily small and arbitrarily large, respectively.

The iSAT algorithm is a generalization of the DPLL procedure [DP60, DLL62], confer Section 6.1, that further incorporates *interval constraint propagation* (ICP), confer [BG06] for a nice survey, to reason about the non-linear arithmetic part in addition to *unit propagation* handling the Boolean structure. Algorithmically, iSAT works very similar to DPLL: to solve the problem, the algorithm manipulates *assignments* to the variables by alternating *deduction* and *decision* phases, interspersed with *backtracking* whenever a *conflict* was detected. Due to the internal use of ICP, iSAT deals with *interval assignments* σ, i.e. σ maps variables to intervals, instead of real-valued and integer assignments.

During the *deduction* phase, iSAT searches for *unit clauses* in which all but one constraint are inconsistent under the current interval assignment σ. A constraint c is *inconsistent* under an interval assignment σ if and only if each (real-valued and integer) assignment τ that is contained in σ does *not* satisfy c. Using interval arithmetic, checking inconsistency of a constraint c can be done efficiently. However, it is important to remark that due to numerical issues when dealing with non-rational functions like sin or exp, only soundness of the inconsistency check can be guaranteed, i.e. whenever a constraint c is said to be inconsistent then c is actually inconsistent. In other words, the check may overlook some inconsistencies. The remaining constraint c in a unit clause cl, i.e. c was not detected to be inconsistent, is called *unit*. In order to retain a chance for satisfaction of cl and thus of the whole formula, unit constraints need to be satisfied. This process is akin to *unit propagation* in DPLL SAT solving, and we occasionally also refer to this term in the iSAT context. The difference to the DPLL SAT case lies in *how* new values or rather new intervals are deduced for the involved variables. While a unit literal x or $\neg x$ in the propositional case immediately entails a value for variable x, namely `true` or `false`, respectively, this is not necessarily feasible for a constraint c, unless c is already a simple (interval) bound, i.e. c is, for instance, $x \geq -2.01$ or $x < 8.3$. In order to derive new intervals, ICP is employed on unit constraints c within the deduction phase. For instance, from constraint $y = 2 \cdot x$ and from the interval assignment σ given by $\sigma(x) = [-1, 4]$ and $\sigma(y) = [-100, 100]$, we may conclude that y can only be in the interval $[-2, 8]$, leading to a narrowing of the interval assignment σ. ICP is then repeatedly applied on all unit constraints until no new intervals can be obtained. Observe that ICP can also trigger unit propagation yielding new unit constraints. We remark that ICP can cause very long and, at least theoretically, infinite deduction chains. With regard to the latter, consider the constraint $x = \frac{1}{2} \cdot x$ and the initial interval $[0, 1]$ for x. Application of ICP then yields the infinite interval contraction $[0, 1] \rightsquigarrow [0, \frac{1}{2}] \rightsquigarrow [0, \frac{1}{4}] \rightsquigarrow [0, \frac{1}{8}] \rightsquigarrow \ldots$. To mitigate this problem and to achieve termination, ICP is stopped whenever the *progress* of newly deduced interval bounds becomes negligible. More precisely, each new interval bound is rejected if the difference to the old bound is below some *progress parameter* δ. For instance, let $x \geq 0.999$ be the new bound and $x \geq 1$ be the old one. The difference clearly is $|1 - 0.999| = 0.001$. Whenever $0.001 < \delta$, for instance if $\delta = 0.05$, then the new bound $x \geq 0.999$ is dismissed. For implementation details concerning the deduction phase and in particular for insights into the internal data structures, the interested reader is referred to [THF$^+$07].

In case the deduction phase leads to a *conflict*, i.e. all constraints of some clause of the SMT formula are inconsistent under the current interval assignment, a conflict resolution procedure is called which analyzes the reason for the conflict in a very similar manner as in the DPLL algorithm. If the conflict can be resolved then a *conflict clause* is built from the reason of the conflict and added to the formula in order to prevent the solver from revisiting the same or a similar conflict again. To retrieve a consistent solver state from which the proof search will be continued, conflict resolution involves *non-chronological backtracking* that is undoing some of the decisions and their accompanying deductions that have been performed so far. If the conflict cannot be resolved then it follows that the given formula is *unsatisfiable* and the algorithm stops. More details about this *conflict analysis* can be found in [FHT$^+$07, THFA08, Her10].

In case the deduction phase yields a *solution*, i.e. at least one constraint in each clause is satisfied by every point in the current interval assignment, the given formula is *satisfiable* and the algorithm stops. It is important to remark that equations like $x = y \cdot z$ can only be satisfied by point intervals in general. However, reaching such point intervals by ICP cannot be guaranteed for continuous domains and thus is usually infeasible. One option to mitigate this problem is to stop the search whenever all intervals have a width smaller than a certain threshold ε, the so-called *minimum splitting width*. Then, the current interval assignment is referred to as an *approximate* solution. Though there is no mathematical certainty about such latter result in general, i.e. the problem may be nevertheless unsatisfiable, such approximate solutions are often sufficient from an engineering perspective, in particular, if ε is chosen "small" enough.

Having completed the deduction phase and neither found a conflict nor an (approximate) solution, iSAT performs a *decision* step by *splitting* the current interval of some variable. That is, some variable x whose interval width is still greater than or equal to the minimum splitting width ε is selected arbitrarily or according to some decision heuristics. The current interval $\sigma(x) = [l_x, u_x]$ of x is then split into two parts $[l_x, s]$ and $(s, u_x]$ using, for instance, the midpoint s of the interval. It is furthermore decided which of the two parts is to be investigated first. The search is then resumed by selecting and asserting one of the interval bounds $x \leq s$ or $x > s$, which potentially triggers new deductions as described above.

Though iSAT is *not* a complete algorithm, i.e. it cannot decide satisfiability of all non-linear arithmetic SMT formulae, it always *terminates* and meets the following notion of *soundness*, confer [FHT$^+$07]: if iSAT claims that the given SMT formula φ is satisfiable or unsatisfiable then φ is actually satisfiable or unsatisfiable, respectively. In all other cases, i.e. in which iSAT is not able to decide the SMT problem, an approximate solution of user-defined volume is returned. Although the latter represents an inconclusive result, such approximate solutions are expected to be sufficient and helpful in most practical applications.

To mitigate the issue of *approximate* solutions, iSAT is capable of performing a subsequent satisfiability check that is called *strong satisfaction check*, confer [FHT$^+$07, Subsection 4.5] as well as [Ked08, EKK$^+$11]. The idea is to exploit the internal structure of the SMT formula φ as well as the approximate solution σ in order to prove satisfiability of φ. A bit more precisely, from each clause in φ one constraint is selected that is still consistent under the approximate solution σ. It is then tried to construct a variable dependency graph involving all *equations* among above constraints such that each variable is defined by at most one equation. An interval assignment ρ is then computed as follows. The intervals $\rho(x)$ for all variables x that are *not* defined by any equation in the dependency graph are given by the corresponding intervals $\sigma(x)$ of the approximate solution σ. The variable dependency graph is then exploited to safely propagate these intervals through the graph such that intervals $\rho(y)$ for all variables y are finally computed. The construction of ρ certifies that ρ contains a (real-valued) solution τ of the conjunction of the above equations. Moreover, if all inequalities among the constraints selected above are definitely satisfied under ρ, i.e. each point in ρ satisfies the inequalities, and if the intervals $\rho(y)$ for all variables $y \in \mathit{Var}(\varphi)$ are subsets of the corresponding initial domains, i.e. $\rho(y) \subseteq \mathrm{dom}(y)$, then the interval assignment ρ comprises a *solution* τ of the SMT formula φ that lies within the domains of the variables in $\mathit{Var}(\varphi)$.

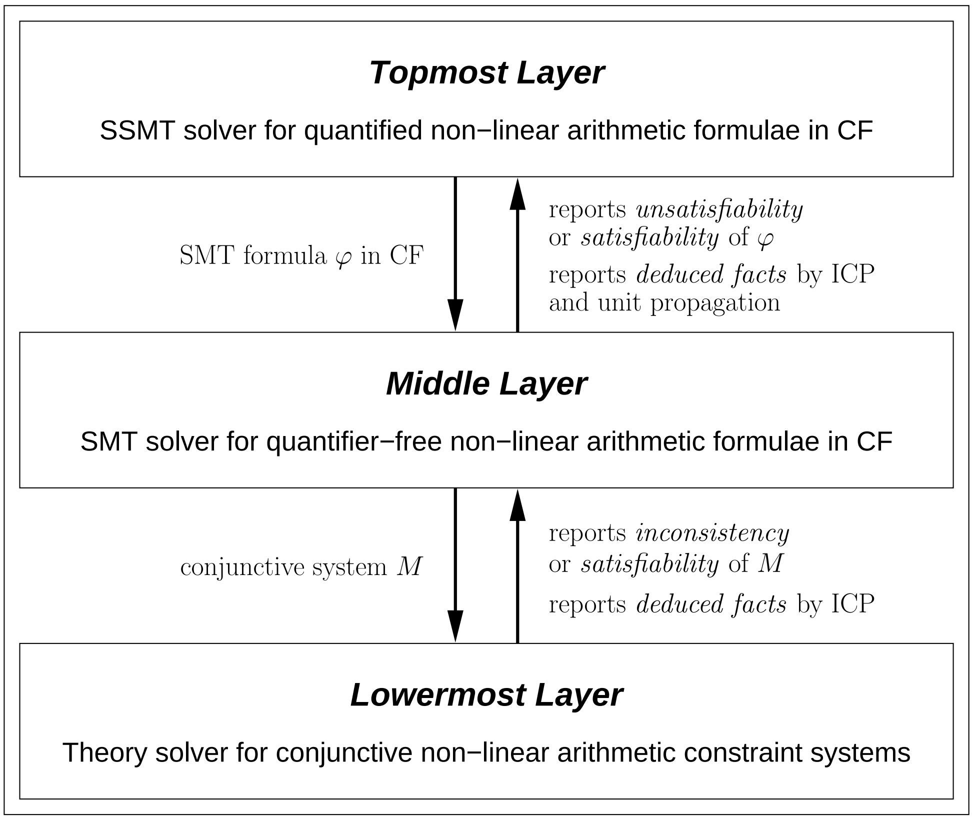
For instance, let

$$\varphi \;=\; (x > 4.1 \vee y \leq -3.78 \vee b \geq 1) \;\wedge\; (b \leq 0 \vee x = y + z) \;\wedge\; (b \leq 0 \vee x = \sin(z))$$

be an SMT formula over real-valued variables $x, y, z \in [-100, 100]$ and integer variable $b \in [-100, 100]$. Assume that iSAT returns the approximate solution σ with $\sigma(x) = [-0.05, 0.05]$, $\sigma(y) = [-3.2, -3.1]$, $\sigma(z) = [3.1, 3.2]$, and $\sigma(b) = [1, 1]$. According to σ, we select from each clause of φ one constraint, namely $b \geq 1$ from the first, $x = y + z$ from the second, and $x = \sin(z)$ from the third clause. As variable x is defined by two equations, we need to redirect one of the corresponding constraints such that some other variable occurs on the left-hand side. We choose equation $x = y + z$ and redirect it to $y = x - z$. In the resulting variable dependency graph, each variable is defined by at most one equation. The next step is to compute the interval assignment ρ. As variables z and b are not defined by any equation in the graph, we set $\rho(z) = \sigma(z) = [3.1, 3.2]$ and $\rho(b) = \sigma(b) = [1, 1]$. Taking equation $x = \sin(z)$, we safely compute the interval $\rho(x) = [-0.06, 0.05]$ for variable x. The latter step ensures that for each value $v_z \in \rho(z)$ there is some value $v_x \in \rho(x)$ such that $v_x = \sin(v_z)$. Using equation $y = x - z$ as well as the intervals for x and z, we can now determine $\rho(y) = [-3.26, -3.05]$ which guarantees that for all $v_x \in \rho(x)$ and for all $v_z \in \rho(z)$ there exists some $v_y \in \rho(y)$ such that $v_y = v_x - v_z$. By the construction of ρ, it immediately follows that ρ contains a real-valued assignment that satisfies both equations $x = y + z$ and $x = \sin(z)$. It also holds that each value $v_b \in \rho(b)$ satisfies inequality $v_b \geq 1$. Furthermore, the intervals $\rho(x)$, $\rho(y)$, $\rho(z)$, and $\rho(b)$ are subsets of the initial domains of x, y, z, and b, respectively. As a consequence, the interval assignment ρ comprises a solution of φ which certifies that φ is satisfiable.

A successful strong satisfaction check thus generates a *graph-based proof of satisfiability* for the given SMT formula. We remark that iSAT is also capable of providing *proofs of unsatisfiability* [KTBF09, KBTF11] which are based on *resolution* and which are similar in nature to resolution proofs for propositional formulae, confer Subsection 6.2.2.

We finally mention that the iSAT algorithm has been extended in several directions, confer [EKKT08, EKKT09] for an overview. To improve the solver's performance, some *parallelization* schemes were considered in [KSÁ$^+$09, KÁS$^+$10]. Expressiveness of input formulae has been enhanced considerably in [EFH08, EFH09] by incorporating *ordinary*



Topmost Layer
SSMT solver for quantified non−linear arithmetic formulae in CF
SMT formula φ in CF
reports *unsatisfiability* or *satisfiability* of φ
reports *deduced facts* by ICP and unit propagation
Middle Layer
SMT solver for quantifier−free non−linear arithmetic formulae in CF
conjunctive system M
reports *inconsistency* or *satisfiability* of M
reports *deduced facts* by ICP
Lowermost Layer
Theory solver for conjunctive non−linear arithmetic constraint systems

Figure 6.4: Interaction between the different layers of the overall SSMT algorithm. Please note that the theory solver is based on interval arithmetic and thus is *incomplete*. That is, reporting inconsistency/unsatisfiability or satisfiability of M as well as of φ cannot be ensured in general.

differential equations (ODEs) as an additional theory. Employing the version of iSAT supporting ODEs as the underlying core engine within the SSMT algorithm of Section 6.4 would then allow for the symbolic analysis of *continuous-time* probabilistic hybrid systems. We elaborate on this issue in Chapter 10. Another direction extends the scope of iSAT within model checking. While the primary use of iSAT lies in *bounded* model checking, the work described in [KB11a, KB11b] proposes an approach to *unbounded* or *full* model checking. The latter is based on overapproximating the reachable state set by means of *Craig interpolants* which are constructed from the aforementioned resolution proofs of unsatisfiability [KTBF09, KBTF11]. The notion of Craig interpolation as well as a generalization of this concept suitable for the SSAT framework and its potential applications to probabilistic *full* model checking are considered in Chapter 9.

6.3.2 Formal description of iSAT

After having explained the intuition of the iSAT procedure in the previous subsection, we now present the algorithm in a formal way. The content of this subsection is mainly based on Section 5 of the article [FTE10a] by Fränzle, Teige, and Eggers, but occasionally differs in detail, for instance, the theory of ODEs is not taken into account. The algorithmic ingredients of iSAT are presented in a *rule*-based fashion and are grouped into two *layers*.

- The first layer constitutes the *theory solver TS* responsible for reasoning about *conjunctive systems of non-linear arithmetic constraints over bounded reals and integers*, while
- the second layer describes the *SMT solver* iSAT for *non-linear arithmetic SMT problems* employing the theory solver *TS*.

The theory solver *TS* is interval-based, i.e. *TS* uses interval calculations and interval constraint propagation [BG06] as safe, yet incomplete reasoning mechanisms.

We anticipate that the SSMT algorithm of Section 6.4 is built upon above layers and actually establishes an own layer. That is, the *lowermost* layer represents the *theory* solver, the *middle* layer the *SMT* solver, and the *topmost* layer the *SSMT* solver. The interaction between these layers is sketched in Figure 6.4.

Theory layer for non-linear arithmetic constraints. We start our exposition with a description of the lowermost layer, the *theory layer*, which deals with conjunctive arithmetic constraint systems. As reasoning mechanisms for such non-linear arithmetic constraint systems over the reals and integers, we employ safe *interval analysis* for approximating real-valued satisfaction and augment it with *interval constraint propagation* (ICP) as a powerful deduction method. Due to the use of interval-based reasoning, we formally define the concept of an *interval assignment* σ as a pair of two functions $(\sigma_{\mathbb{R}}, \sigma_{\mathbb{Z}})$ such that $\sigma_{\mathbb{R}} : V_{\mathbb{R}} \to \mathbb{I}_{\mathbb{R}}$ and $\sigma_{\mathbb{Z}} : V_{\mathbb{Z}} \to \mathbb{I}_{\mathbb{Z}}$ with $V_{\mathbb{R}}$ and $V_{\mathbb{Z}}$ being some sets of real-valued and integer variables, respectively, and $\mathbb{I}_{\mathbb{R}}$ and $\mathbb{I}_{\mathbb{Z}}$ being the sets of all intervals over the real numbers $\mathbb{R}$ and the integers $\mathbb{Z}$, respectively. Intuitively, $\sigma_{\mathbb{R}}$ maps real-valued variables to real-valued intervals and $\sigma_{\mathbb{Z}}$ maps integer variables to integer intervals. To ease notation, we simply write $\sigma(x)$ for $\sigma_{\mathbb{R}}(x)$ if $x \in V_{\mathbb{R}}$ and for $\sigma_{\mathbb{Z}}(x)$ if $x \in V_{\mathbb{Z}}$.

Interval analysis [Moo66] enables to evaluate the *interval consistency* of a set M of non-linear arithmetic constraints involving functions like sin and exp. Interval consistency is a necessary yet not sufficient condition for real-valued satisfiability of M. Thus, refutation by interval consistency is always correct. There are several definitions of interval consistency in the literature, confer [BG06] for an overview. They mainly differ in the strength of their consistency notions and in the computational effort to decide consistency. Our consistency concept is *hull consistency*, confer [BMH94, BG06] for details, which is easy to decide and which gives good results in practice. Let be given an equation $x = y \circ z$ according to Definition 4.3, where x, y, z are variables and $\circ$ is a binary arithmetic operator, and an interval assignment σ that maps above variables to *non-empty* intervals, i.e. $\sigma(x) \neq \emptyset, \sigma(y) \neq \emptyset, \sigma(z) \neq \emptyset$. Then, the equation $x = y \circ z$ is *hull consistent under* σ [BMH94] if and only if

- $\sigma(x) = \mathit{hull}(\sigma(x) \cap \{v_x : \exists v_y \in \sigma(y) \exists v_z \in \sigma(z) : v_x = v_y \circ v_z\})$,
- $\sigma(y) = \mathit{hull}(\sigma(y) \cap \{v_y : \exists v_x \in \sigma(x) \exists v_z \in \sigma(z) : v_x = v_y \circ v_z\})$, and
- $\sigma(z) = \mathit{hull}(\sigma(z) \cap \{v_z : \exists v_x \in \sigma(x) \exists v_y \in \sigma(y) : v_x = v_y \circ v_z\})$,

where $\mathit{hull}(A)$ for some set $A \subseteq \mathbb{R}$ (or $A \subseteq \mathbb{Z}$), called the *interval hull* of A, is the smallest interval containing the set A. With regard to implementation using machine data types like *floating-point numbers*, $\mathit{hull}(A)$ for some set $A \subseteq \mathbb{R}$ is usually defined as the smallest interval containing A such that the interval borders are representable in the corresponding data type [BMH94]. Hull consistency is analogously defined for equations $x = \circ y$ involving unary arithmetic operators. As an example, consider the equation $x = y + z$ and the interval assignment σ with $\sigma(x) = [0,3]$, $\sigma(y) = [0,1]$, and $\sigma(z) = [0,1]$. Then, $x = y + z$ is *not* hull consistent under σ since

$$\begin{aligned} \sigma(x) = [0,3] \;\; &\neq \;\; \mathit{hull}(\sigma(x) \cap \{v_x : \exists v_y \in \sigma(y) \exists v_z \in \sigma(z) : v_x = v_y + v_z\}) \\ &\;\; = \mathit{hull}([0,3] \cap [0,2]) \;=\; [0,2] \;\; . \end{aligned}$$

Equation $x = y + z$ is however hull consistent under σ' with $\sigma'(x) = [0,2]$, $\sigma'(y) = [0,1]$, and $\sigma'(z) = [0,1]$ since

$$\begin{aligned} \sigma'(x) &= \mathit{hull}([0,2] \cap [0,2]) &&= [0,2], \\ \sigma'(y) &= \mathit{hull}([0,1] \cap [-1,2]) &&= [0,1], \text{ and} \\ \sigma'(z) &= \mathit{hull}([0,1] \cap [-1,2]) &&= [0,1] \;\; . \end{aligned}$$

We also define hull consistency for simple bounds. Let $x \sim r$ be a bound according to Definition 4.3, where x is a variable, $\sim \in \{<, \leq, =, \geq, >\}$, and r is a rational constant, and let σ be an interval assignment with $\sigma(x) \neq \emptyset$. Then, the bound $x \sim r$ is *hull consistent under* σ if and only if $\sigma(x) \subseteq \{v \in \mathbb{R} : v \sim r\}$. For instance, $x < 3$ is hull consistent under σ with $\sigma(x) = [-1.1, 2.4]$ as well as under σ' with $\sigma'(x) = [-1.1, 3)$ but not hull consistent under σ'' with $\sigma''(x) = [-1.1, 3]$.

The notion of hull consistency can be lifted to conjunctive systems of constraints M requiring that each constraint $c \in M$ is hull consistent under σ. We denote the fact that *M is hull consistent under σ* by $\sigma \models_{\mathtt{hc}} M$. For instance, $M = \langle x = y, x = y^2, y > 0, y < 1 \rangle$ is hull consistent under σ with $\sigma(x) = \sigma(y) = (0,1)$ as all constraints in M are hull consistent under σ. This example further shows that hull consistency is *not* a sufficient condition for real-valued satisfiability since the conjunctive constraint system M is unsatisfiable. To achieve hull consistency of a conjunctive constraint system, interval constraint propagation (ICP) is exploited. As already shown on a simple example in Subsection 6.3.1, ICP can cause very long and, at least theoretically, infinite deduction chains. We remark that practical implementations of ICP commonly employ floating-point data types such that infinite deductions are not possible due to finiteness of floating-point numbers. Nevertheless, to avoid very long and time-consuming deduction sequences, ICP is usually stopped whenever the *progress* of newly deduced interval bounds becomes negligible, confer *progress parameter* δ explained in Subsection 6.3.1. Due to the latter fact, *hull consistency* is generally not achieved with full rigor in practice but just up to some desired, potentially high accuracy.

As shown by the examples above, if some constraint c is *not* hull consistent under some interval assignment σ then there potentially exists some *refinement* σ' of σ, i.e. $\sigma'(x) \subseteq \sigma(x)$ for each variable x, such that c is hull consistent under σ'. That is, the property of not being hull consistent under σ does *not* entail *inconsistency* of a constraint under σ in general. We therefore define the notion of interval inconsistency of a constraint. We say that an equation $x = y \circ z$, where x, y, z are variables and $\circ$ is a binary arithmetic operator, is *inconsistent under an interval assignment* σ if and only if

- $\mathit{hull}(\sigma(x) \cap \{v_x : \exists v_y \in \sigma(y) \exists v_z \in \sigma(z) : v_x = v_y \circ v_z\}) \;=\; \emptyset$.

Inconsistency is analogously defined for equations $x = \circ y$ involving unary arithmetic operators. A simple bound $x \sim r$, where x is a variable, $\sim \in \{<, \leq, =, \geq, >\}$, and r is a rational constant, is *inconsistent under an interval assignment* σ if and only if $\sigma(x) \cap \{v \in \mathbb{R} : v \sim r\} = \emptyset$. Observe that above definition implies the following: whenever the interval $\sigma(x)$ of any variable x of a constraint c is empty, i.e. $\sigma(x) = \emptyset$, then c is trivially inconsistent under σ. We denote the fact that some constraint c is inconsistent under σ by $\sigma \sharp c$ as well as that a conjunctive system of constraints M contains some constraint $c \in M$ with $\sigma \sharp c$ by $\sigma \sharp M$. Observe that if some constraint c is inconsistent under some σ then the interval assignment σ does not contain any (real-valued) solution of c. For instance, $x = 2 \cdot y$ is inconsistent under σ with $\sigma(x) = [-2, -1]$ and $\sigma(y) = [1, 2]$ since all values of $2 \cdot y$ for $y \in [1, 2]$ lie within the interval $[2, 4]$.

Interval constraint propagation (ICP) [BMH94, Ben96, BG06] complements interval analysis as a deduction mechanism pruning off non-solutions by narrowing the intervals while trying to achieve hull consistency. Given a constraint c and an interval assignment σ, ICP potentially computes a refinement σ' of σ, i.e. $\sigma'(x) \subseteq \sigma(x)$ for each variable x, such that σ' contains all solutions of c in σ. Implementations of interval arithmetic support common functions like $+, -, \cdot$ as well as transcendental functions like sin and cos [Neu90, HJvE01]. It is important to remark that interval calculations are guaranteed to be *safe*: for instance, if using floating-point data types then intervals are always rounded outwards such that the results remain correct also under rounding errors.

As an example, assume the arithmetic constraint $z = x + y$ over real-valued variables x, y, z and the interval assignment σ with $\sigma(x) = [1, 4]$, $\sigma(y) = [2, 3]$, and $\sigma(z) = [0, 4.5]$ are given. Each solved form of the constraint, i.e. $x = z - y$, $y = z - x$, and $z = x + y$, allows contraction of the interval for the variable on the left-hand side. For $x = z - y$, we subtract the interval $[2, 3]$ for y from the interval $[0, 4.5]$ for z, concluding that x can only be in $[-3, 2.5]$. Intersecting this interval with the original interval $[1, 4]$, we know that x can only be in $[1, 2.5]$. Proceeding in a similar way for $y = z - x$ does not change the interval for y. Finally, from $z = x + y$ we conclude that z can only be in $[3, 4.5]$. Observe that constraint $z = x + y$ is hull consistent under the narrowed interval assignment σ' with $\sigma'(x) = [1, 2.5]$, $\sigma'(y) = [2, 3]$, and $\sigma'(z) = [3, 4.5]$. The treatment of integer variables is as above but followed by clipping the integer intervals accordingly. For instance, if variables x, y, and z are integer variables then the narrowed interval assignment σ' is given by $\sigma'(x) = [1, 2]$, $\sigma'(y) = [2, 3]$, and $\sigma'(z) = [3, 4]$.

We formally denote the deduction of a new interval border ib from a constraint c and from an interval assignment σ with $\sigma(y) \neq \emptyset$ for each variable $y \in \mathit{Var}(c)$ by

$$(c, \sigma) \longrightarrow_{\mathit{ICP}} \mathit{ib}$$

where $\mathit{ib} = (x \sim r)$ with $x \in \mathit{Var}(c)$ being a variable, $\sim \in \{<, \leq, \geq, >\}$, and r being a rational constant. Semantically, above ICP step ensures that for each (real-valued) assignment $\tau \in \sigma$, i.e. $\tau(y) \in \sigma(y)$ for each variable y for which $\sigma(y)$ is defined, it holds that $\tau \models c \Rightarrow \mathit{ib}$.

In what follows, we formalize above observations. For a given set of real-valued and integer variables V, let M be a conjunctive system of non-linear arithmetic constraints over variables V representing the *system state* of the theory solver TS. We require that each constraint $c \in M$ is in accordance with Definition 4.3, i.e. c is a bound or an equation. In order to avoid redundancies and to ease the technical presentation, the system state does *not* comprise the current interval assignment *explicitly.* In fact, the current interval assignment σ is encoded in M itself. We therefore demand that M contains at least one lower and at least one upper interval bound for each variable $x \in V$, i.e. $(x \sim_1 l_x) \in M$ with $\sim_1 \in \{\geq, >\}$ and $(x \sim_2 u_x) \in M$ with $\sim_2 \in \{\leq, <\}$. It is then straightforward to retrieve from M the current interval assignment σ_M: let M_b be the set of all bounds $(x \sim r) \in M$ where x is a variable, $\sim \in \{<, \leq, =, \geq, >\}$, and r is a rational constant. Then, σ_M is defined as the largest (with respect to interval size) interval assignment that satisfies all bounds in M_b, i.e. for each (real-valued) assignment $\tau \in \sigma_M : \tau \models M_b$. For instance, let be $M = \langle x \geq 2.1, x < 7, x \leq 333.4, y > -499.3, y \leq 0.9, x = y^2, y > -4.06\rangle$. Then, $M_b = \langle x \geq 2.1, x < 7, x \leq 333.4, y > -499.3, y \leq 0.9, y > -4.06\rangle$ and, thus, $\sigma_M(x) = [2.1, 7)$ and $\sigma_M(y) = (-4.06, 0.9]$. As we see later on, M is extended and reduced by the SMT and SSMT layers during the proof search, which in turn means that also σ_M frequently changes. With regard to implementation, we remark that it is not necessary to re-extract σ_M each such time. In fact, data structures like stacks are exploited to update σ_M efficiently.

The first rule TS.1 checks inconsistency of the current solver state M. In case M is inconsistent under the current interval assignment σ_M, i.e. there is some $c \in M$ with $\sigma_M \sharp\, c$, then TS enters the distinguished state `incons` exhibiting inconsistency of M.

(TS.1)
$$\frac{\sigma_M \sharp\, M}{M \longrightarrow_{\mathit{TS}} \texttt{incons}}$$

Deduction of a *tighter* interval border ib from some constraint in M and from the current interval assignment σ_M by means of ICP is addressed by rule TS.2. Such a step yields a narrowing of the interval assignment from σ_M to $\sigma_{M \odot \langle \mathit{ib}\rangle}$. We require that the current interval assignment σ_M maps each variable occurring in M to a non-empty interval.

(TS.2)
$$\frac{\forall x \in \mathit{Var}(M) : \sigma_M(x) \neq \emptyset, \quad c \in M, \quad (c, \sigma_M) \longrightarrow_{\mathit{ICP}} \mathit{ib}}{M \longrightarrow_{\mathit{TS}} M \odot \langle \mathit{ib}\rangle}$$

Given some conjunctive constraint system M, multiple application of above rule TS.2 either leads to an inconsistent solver state M, i.e. $\sigma_M \sharp\, M$, or converges[6] against a hull consistent constraint system M, i.e. $\sigma_M \models_{\texttt{hc}} M$.

As mentioned earlier, hull consistency of M is a necessary yet not sufficient condition for real-valued satisfiability of M. Pragmatically, a hull consistent interval assignment of sufficiently small volume, confer *minimum splitting width* ε explained in Subsection 6.3.1, can be considered as an *approximate* solution which is an acceptable approach from an engineering perspective. In some cases, however, we are able to mitigate the issue of *approximate* solutions, namely by the subsequent *strong satisfaction check*, confer Subsection 6.3.1 and [FHT$^+$07, Ked08, EKK$^+$11].

Recall that the strong satisfaction check for an SMT formula φ and an approximate solution σ works as follows. In the first step, from each clause of φ at least one constraint is

[6]Recall that reaching hull consistency within finitely many ICP steps cannot be ensured in general.

selected that is not inconsistent under σ. In the second step, it is tried to determine an interval assignment ρ that contains a solution of the conjunction of above constraints within the initial domains of the variables. The computation of ρ relies on constructing a variable dependency graph as well as on the approximate solution σ. While responsibility for the first step of the strong satisfaction check is with the SMT layer, the second step is handled by the theory solver *TS*. For this purpose, we formally denote the call of the second step of the strong satisfaction check on some conjunctive non-linear arithmetic constraint system M and on some interval assignment σ by $\mathit{strong\_sat\_check}(M, \sigma)$. The latter call returns `success` *only if* M is satisfiable, i.e. if $\mathit{strong\_sat\_check}(M, \sigma) = \texttt{success}$ then M is satisfiable. We assume here that system M already comprises the initial domains of the involved variables such that each solution of M lies within the domains of the variables. The incorporation of the strong satisfaction check into the theory solver *TS* is formalized by the following rule TS.3 where the distinguished state `sat` exhibits satisfiability of M.

We remark that the system state of *TS* was slightly extended to (M, σ) since (the second step of) the strong satisfaction check has to be aware of the approximate solution σ which is provided by the SMT layer. It is of course possible to also use the extended state (M, σ) in rules TS.1 and TS.2, namely by just ignoring σ. However, the latter treatment would be meaningless and is thus not realized.

$$\text{(TS.3)} \qquad \frac{\mathit{strong\_sat\_check}(M, \sigma) = \texttt{success}}{(M, \sigma) \longrightarrow_{TS} \texttt{sat}}$$

SMT layer for non-linear arithmetic SMT formulae. In what follows, we present the rule-based description of the SMT layer that formalizes the iSAT algorithm. In addition to the conjunctive constraint system M that establishes the solver state of the theory solver *TS*, the state of the SMT solver further incorporates a non-linear arithmetic SMT formula φ. That is, the solver state of the SMT layer *SMT* is given by a pair (M, φ). As already mentioned in Subsection 6.3.1, we require that φ is in *conjunctive form* (CF, confer Definition 4.3) and that the domains $\mathrm{dom}(x)$ of all variables $x \in \mathit{Var}(\varphi)$ are given by *bounded intervals*. As motivated within the description of the theory solver *TS*, the system state does *not* comprise the current interval assignment σ *explicitly* but σ is encoded by M and can be simply retrieved from M, namely by constructing the interval assignment σ_M. We therefore need to ensure that σ_M maps each variable $x \in \mathit{Var}(\varphi)$ to a bounded interval for each solver state (M, φ), i.e. M contains at least one lower and at least one upper interval bound for each $x \in \mathit{Var}(\varphi)$. The latter is achieved by enforcing that M comprises at least all the interval bounds specified by the domains $\mathrm{dom}(x)$ of the variables $x \in \mathit{Var}(\varphi)$. Formally, these domains are encoded symbolically by the conjunctive system

$$\begin{aligned} M_{\mathrm{dom}} \;:=\; & \quad \{x > \inf(\mathrm{dom}(x)) : x \in \mathit{Var}(\varphi), \inf(\mathrm{dom}(x)) \notin \mathrm{dom}(x)\} \\ & \cup \;\{x \geq \inf(\mathrm{dom}(x)) : x \in \mathit{Var}(\varphi), \inf(\mathrm{dom}(x)) \in \mathrm{dom}(x)\} \\ & \cup \;\{x < \sup(\mathrm{dom}(x)) : x \in \mathit{Var}(\varphi), \sup(\mathrm{dom}(x)) \notin \mathrm{dom}(x)\} \\ & \cup \;\{x \leq \sup(\mathrm{dom}(x)) : x \in \mathit{Var}(\varphi), \sup(\mathrm{dom}(x)) \in \mathrm{dom}(x)\} \quad . \end{aligned} \tag{6.17}$$

The SMT layer ensures that system M_{dom} is part of M for each state (M, φ). This property holds, in particular, for the *initial SMT solver state* given by $(M_{\mathrm{dom}}, \varphi)$ with φ being the given SMT formula to be solved.

The application of the SMT layer rules below characterizes a backtracking procedure by means of manipulating the solver state (M, φ), where the conjunctive system M consists of constraints which are asserted during the proof search. The goal is to find some M such that the corresponding interval assignment σ_M is a solution or an approximate solution of φ, or to prove the absence of such a σ_M showing unsatisfiability of φ. During this process, M is enlarged by adding *unit* constraints from φ as well as tighter interval bounds obtained by ICP or *interval splitting*. If it turns out at some point that M is inconsistent under the current interval assignment σ_M then some of the constraints in M are removed in reverse chronological order in order to regain a consistent solver state. The latter realizes the operation of *backtracking*. To keep track of the chronological order of added constraints, the conjunctive system M is represented as a *sequence* of constraints rather than just a set. To simply detect backtrack points in this data structure, the sequence M is interspersed with a special marker symbol $|$. Though the formula part φ also is manipulated, namely by means of adding implied *conflict clauses*, a special data structure for φ is not required. As usual, we assume φ to be a set of clauses.

Rules SMT.1 and SMT.2 reflect the *deduction* phase of the iSAT algorithm. The first rule SMT.1 applies *unit propagation*, i.e. it detects unit clauses cl and adds unit constraints c to the conjunctive constraint system M. Recall that unit constraints need to be satisfied in order to retain a chance for satisfaction of SMT formula φ under some *refinement* σ'_M of the current interval assignment σ_M, i.e. $\sigma'_M(x) \subseteq \sigma_M(x)$ for each $x \in \mathit{Var}(\varphi)$. We remark that rule SMT.1 is slightly more general than just unit propagation: in addition to unit constraints c, this rule potentially adds already inconsistent constraints c to M, namely if *all* constraints of clause cl are inconsistent under σ_M. This treatment is of technical nature, namely to detect inconsistent states (M, φ) by means of M only. The latter situation, i.e. inconsistency of M, is covered by rules SMT.4 and SMT.5. With regard to termination, observe that at most one inconsistent constraint per inconsistent clause can be added to M.

$$\text{(SMT.1)} \qquad \frac{\begin{array}{c} cl \in \varphi, \quad c \in cl, \quad \forall c' \in cl : c' \notin M, \\ \forall c' \in cl \text{ such that } c' \neq c : M \odot \langle c' \rangle \longrightarrow_{TS} \texttt{incons} \end{array}}{(M, \varphi) \longrightarrow_{SMT} (M \odot \langle c \rangle, \varphi)}$$

Deduction of tighter interval borders ib by means of ICP is done by rule SMT.2. For termination reasons, we enforce that deduced interval borders yield a certain progress which is specified by the *progress parameter* $\delta > 0$. Further observe that whenever ICP yields an inconsistent state $(M \odot \langle ib \rangle, \varphi)$ then the interval $\sigma_{M \odot \langle ib \rangle}(x)$ of variable x has become empty and rule SMT.2 is no longer applicable on $(M \odot \langle ib \rangle, \varphi)$ due to rule TS.2. Inconsistency of M is then treated by rules SMT.4 and SMT.5.

$$\text{(SMT.2)} \qquad \frac{\begin{array}{c} M \longrightarrow_{TS} M \odot \langle ib \rangle, \quad ib = (x \sim r), \\ \sup(\sigma_M(x)) - r \geq \delta \text{ if } \sim \in \{<, \leq\}, \quad r - \inf(\sigma_M(x)) \geq \delta \text{ if } \sim \in \{\geq, >\} \end{array}}{(M, \varphi) \longrightarrow_{SMT} (M \odot \langle ib \rangle, \varphi)}$$

where $x \in \mathit{Var}(\varphi)$, $\sim \in \{<, \leq, \geq, >\}$, and r is a rational constant (representable by the applied machine data type).

The next rule SMT.3 formalizes a *decision* step in the iSAT procedure, namely by *splitting* the current interval $\sigma_M(x)$ of some variable x at some value $r \in \sigma_M(x)$. A common splitting strategy is *bisection*, i.e. r is the midpoint of $\sigma_M(x)$. The resulting interval bound $x \sim r$ is then added to M together with a preceding special marker symbol $|$ indicating this decision step. The relation $\sim \in \{<, \leq, \geq, >\}$ specifies which of the both subintervals is investigated first. To avoid an infinite sequence of splitting steps, we introduce the parameter $\varepsilon > 0$, called *minimum splitting width*, to ensure that only intervals of width at most ε are split. With regard to termination, we furthermore have to make sure that the decided bound $x \sim r$ yields enough progress according to progress parameter δ. As rule SMT.4 potentially deduces the negation of the decided bound $x \sim r$, enough progress is also required for $\neg(x \sim r)$. For a reasonable setting, we assume that $\varepsilon \geq 2\delta$ as this allows to perform a splitting step whenever the width of $\sigma_M(x)$ is greater than or equal to ε.

$$\text{(SMT.3)} \quad \frac{\begin{array}{c} x \in \mathit{Var}(\varphi), \quad \sim \in \{<, \leq, \geq, >\}, \quad r \in \sigma_M(x) \neq \emptyset, \\ \sup(\sigma_M(x)) - \inf(\sigma_M(x)) \geq \varepsilon, \quad \sup(\sigma_M(x)) - r \geq \delta, \quad r - \inf(\sigma_M(x)) \geq \delta \end{array}}{(M, \varphi) \longrightarrow_{SMT} (M \odot \langle |, x \sim r \rangle, \varphi)}$$

If the current solver state (M, φ) is inconsistent, i.e. $M \longrightarrow_{TS}$ `incons`, then the SMT layer has detected a so-called *conflict* meaning that the current interval assignment σ_M cannot contain any solution of φ. In such a situation, the conflict is analyzed. If M comprises at least one special marker symbol $|$, i.e. some decisions are involved in M, then the conflict can be resolved as follows. At first, a small (or even minimal) subsystem of interval bounds $b_1, \ldots, b_k$ from M is determined, building the *reason* for the current conflict, such that the conjunction of these bounds together with the SMT formula φ is unsatisfiable, i.e. $\varphi \wedge b_1 \wedge \ldots \wedge b_k \equiv$ `false`. Observe that such a reason encodes an interval assignment ρ that is potentially more general than σ_M, i.e. $\forall x \in \mathit{Var}(\varphi) : \rho(x) \supseteq \sigma_M(x)$. In order to prevent the SMT layer from unnecessarily probing ρ or a refinement of ρ in future search, ρ is excluded by adding a so-called *conflict clause* to the formula. Such a conflict clause can be easily constructed by means of the *disjunction of the negated interval bounds* $(\neg b_1 \vee \ldots \vee \neg b_k)$. This new clause forbids any refinement of ρ in which a solution cannot exist by forcing that at least one of the bounds $b_1, \ldots, b_k$ may not hold. This is referred to as *conflict-driven clause learning.* Note that a conflict clause is always implied by the SMT formula φ, i.e. $\models \varphi \Rightarrow (\neg b_1 \vee \ldots \vee \neg b_k)$, since $\varphi \wedge b_1 \wedge \ldots \wedge b_k$ is unsatisfiable. Hence, $\varphi \equiv \varphi \wedge (\neg b_1 \vee \ldots \vee \neg b_k)$. We remark that there are very efficient techniques for the generation of such conflict clauses for the propositional SAT case, confer [MSLM09], like the *first unique implication point* (1UIP) technique from [ZMMM01]. Both tools HySAT-II and iSAT employ a conflict resolution strategy very similar to the 1UIP scheme. For more details, the interested reader is referred to [FHT$^+$07, THFA08, Her10]. Conflict-driven clause learning usually appears in combination with *non-chronological backtracking* where the conflict clause *cc* is additionally used for backtracking. Due to construction of *cc* by the 1UIP technique, it is ensured that there is some decision level on which *cc* becomes unit. From this decision level the search will be continued by applying unit propagation for *cc*, i.e. by deduction of the interval bound $\neg b_k$. This approach leads to a *non-chronological backtracking* operation, often jumping back more than just one level.

It is important to mention that the bounds b_i for $1 \leq i \leq k-1$ can be of the form $x = r$. The negation $\neg b_i$ is then encoded by the disjunction of two bounds, i.e. $\neg(x = r) \rightsquigarrow (x < r \vee x > r)$. The bound b_k must however be of the form $x \sim r$ with $\sim \in \{<, \leq, \geq, >\}$, so that its negation $\neg b_k$, which will be added to M, represents an interval border. We remark that such a bound b_k can always be found in M'': at least the leftmost bound in M'', i.e. the decided bound added by rule SMT.3, is of the desired shape. To achieve termination, we again need to ensure that the deduced bound $\neg b_k$ yields enough progress according to progress parameter δ. At least the decided bound added by rule SMT.3 satisfies this condition. The above facts are formalized by the following rule SMT.4.

$$\text{(SMT.4)}\quad \frac{\begin{array}{c} M \longrightarrow_{TS} \texttt{incons},\quad M = M' \odot \langle | \rangle \odot M'',\quad b_1, \ldots, b_{k-1} \in M',\quad b_k \in M'', \\ b_1, \ldots, b_k \text{ are bounds},\quad \varphi \wedge b_1 \wedge \ldots \wedge b_k \equiv \texttt{false},\quad b_k = (x \sim r), \\ \sup(\sigma_{M'}(x)) - r \geq \delta \text{ if } \not\sim \in \{<, \leq\},\quad r - \inf(\sigma_{M'}(x)) \geq \delta \text{ if } \not\sim \in \{\geq, >\} \end{array}}{(M, \varphi) \longrightarrow_{SMT} (M' \odot \langle \neg b_k \rangle, \varphi \wedge (\neg b_1 \vee \ldots \vee \neg b_k))}$$

where $x \in \mathit{Var}(\varphi)$, $\sim \in \{<, \leq, \geq, >\}$, r is a rational constant, and $\not\sim$ gives the opposite relation of $\sim$, i.e. $\not\sim$ is $<$ if and only if $\sim$ is $\geq$ and so forth.

If the current solver state (M, φ) is inconsistent, meaning that the current interval assignment σ_M cannot contain any solution of φ, and if the conjunctive system M does *not* comprise the special marker symbol $|$ then the SMT formula φ is unsatisfiable within the domains of its variables, i.e. $\varphi \wedge M_{\text{dom}}$ is unsatisfiable where M_{dom} encodes the domains of the variables symbolically as defined in equation 6.17. The rationale is that all constraints in M, and thus σ_M itself, are implied by φ and M_{dom}, i.e. $\models \varphi \wedge M_{\text{dom}} \Rightarrow M$, and that M is inconsistent under σ_M entailing that M is unsatisfiable. Let ψ be the original SMT formula. As only implied clauses were added to ψ, namely by rule SMT.4, it clearly holds that $\psi \Rightarrow \varphi$ and, thus, also ψ is unsatisfiable within the domains $\text{dom}(x)$ of its variables $x \in \mathit{Var}(\psi)$. This situation is reflected by the next rule SMT.5 where the distinguished SMT layer state `unsat` exhibits unsatisfiability of the given SMT problem.

$$\text{(SMT.5)}\quad \frac{M \longrightarrow_{TS} \texttt{incons},\quad | \notin M}{(M, \varphi) \longrightarrow_{SMT} \texttt{unsat}}$$

In order to potentially prove satisfiability of the given SMT formula, the next rule SMT.6 integrates the *strong satisfaction check* into the SMT layer. That is, from each clause of φ at least one constraint is selected which is not inconsistent under the current interval assignment σ_M. The latter process results in M_3. Note that such M_3 must exist whenever none of the rules SMT.1 to SMT.5 is applicable. The subsystem M_1, containing all constraints from M which were added before the first decision step by SMT.3, includes the symbolic encoding of the initial domains of the variables given by M_{dom}. Therefore, if the strong satisfaction check, as realized by theory layer rule TS.3, succeeds on the conjunctive system $M_1 \odot M_3$ and on interval assignment σ_M then the SMT formula φ is satisfiable within the domains of its variables, i.e. $\varphi \wedge M_{\text{dom}}$ is satisfiable. Note that φ may contain implied conflict clauses cc. Since $\varphi \equiv \varphi \wedge cc$ with $\varphi \Rightarrow cc$, also the original SMT formula is satisfiable within the domains of its variables. Above observations are formally described by rule SMT.6 where the distinguished SMT layer state `sat` displays satisfiability of the given SMT problem. Observe that rule SMT.6 is not restricted to the case in which σ_M is an approximate solution but can be applied for arbitrary but consistent SMT solver states.

$$M = \begin{cases} M_1 \odot \langle | \rangle \odot M_2 & ; \ | \in M, \\ M_1 & ; \ | \notin M \end{cases}, \quad | \notin M_1,$$

$$(\mathsf{SMT.6}) \quad \frac{\exists M_3\ \forall cl \in \varphi\ \exists c \in cl \text{ with } c \in M_3, \neg(\sigma_M \sharp c) : (M_1 \odot M_3, \sigma_M) \longrightarrow_{TS} \mathtt{sat}}{(M, \varphi) \longrightarrow_{SMT} \mathtt{sat}}$$

Proposition 6.2 states facts about soundness and termination of the iSAT algorithm formalized by the SMT layer above, i.e. by rules SMT.1 to SMT.6. For the corresponding proofs, we refer the reader to [FHT+07, Her10]. As usual, we denote by $\longrightarrow^*_{SMT}$ the reflexive transitive closure of the relation $\longrightarrow_{SMT}$.

Proposition 6.2 (Soundness and termination of the SMT layer)
Let be given a non-linear arithmetic SMT formula φ over the reals and integers in CF as in Definition 4.3 and let the domains $\mathrm{dom}(x)$ of all variables $x \in Var(\varphi)$ be given by bounded intervals. Let further M_{dom} be a conjunctive system that encodes above domains as defined in equation 6.17 and let $(M_{\mathrm{dom}}, \varphi)$ be the initial state of the SMT layer. We require that the minimum splitting width ε and the progress parameter δ are both strictly greater than zero and that $\varepsilon \geq 2\delta$. Then,

1. *the SMT layer always terminates, i.e. there does not exist an infinite sequence $(M_{\mathrm{dom}}, \varphi) \longrightarrow_{SMT} \ldots \longrightarrow_{SMT} S \longrightarrow_{SMT} \ldots$ of SMT rule applications, and*
2. *for each final state F, i.e. $(M_{\mathrm{dom}}, \varphi) \longrightarrow^*_{SMT} F$ and there does not exist a state F' such that $F \longrightarrow_{SMT} F'$, it holds that*
   a) *if $F = \mathtt{unsat}$ then $\varphi \wedge M_{\mathrm{dom}}$ is unsatisfiable,*
   b) *if $F = \mathtt{sat}$ then $\varphi \wedge M_{\mathrm{dom}}$ is satisfiable, and*
   c) *otherwise, i.e. $F = (M, \varphi')$, each clause in φ contains at least one constraint that is not inconsistent under interval assignment σ_M, and the interval of each variable $x \in Var(\varphi)$ has a width below ε, i.e. $\sup(\sigma_M(x)) - \inf(\sigma_M(x)) < \varepsilon$.*

In order that Proposition 6.2 holds for an implementation of the SMT layer using machine data types, we need to ensure that constant r in rule SMT.3 is actually computer-representable. This can be achieved, for instance, by choosing the progress parameter δ at least as large as the maximal quantization step of the applied numerical data type as well as by setting the minimum splitting width ε at least three times larger than δ.

Observe that above case 2c reflects the result of an *approximate solution* where the concept of *not* being *inconsistent* is used rather than the stronger notion of being *hull consistent.*[7] This is due to two reasons. First, hull consistency cannot be achieved with full rigor in general, as already mentioned, but just up to some desired, potentially high accuracy specified by the progress parameter δ. Second, it cannot be guaranteed that M actually comprises constraints from all clauses, i.e. it might be the case that there is some clause $cl \in \varphi$ such that all constraints $c \in cl$ do not occur in M. The latter situation happens if cl contains at least two constraints $c_1, c_2 \notin M$ that are not inconsistent under σ_M, which means that rule SMT.1 was not allowed to add c_1 or c_2 to M. Assume that

[7]Note that hull consistency implies the property of being not inconsistent.

clause cl consists of equations only, then no other rule is able to insert constraints from cl to M. To avoid the latter issue, one could enhance the decision rule SMT.3 by allowing to *decide constraints from clauses* in addition to split intervals. Another option might be to simply *restrict the syntactical shape of clauses* without loss of generality as in [Her10], where a clause consists of either several simple interval bounds or exactly one arithmetic constraint.

Concluding this section, we finally mention a simple but important fact with regard to the integration of iSAT into the SSMT algorithm being introduced in Section 6.4. During the SSMT proof search, several SMT problems need to be solved, as indicated in Figure 4.3. Due to the rather tight integration, these problems are encoded by means of (initial) SMT solver states $(M_{\text{dom}} \odot M', \varphi)$ with M' being conjunctive systems of arithmetic constraints. Such systems M' arise during the SSMT proof search when instantiating quantified variables with values and when deducing new facts using unit propagation and ICP. The following remark adapts Proposition 6.2 to the latter case.

Remark 6.1
Let φ be a non-linear arithmetic SMT formula in CF and M_{dom} be a conjunctive system encoding the domains of all variables in $Var(\varphi)$ as in equation 6.17. Let further M' be a conjunctive system of primitive constraints, i.e. of bounds and equations, confer Definition 4.3, not containing the special marker symbol $|$, i.e. $| \notin M'$, such that

- *each equation $e \in M'$ can be deduced from $M_{\text{dom}} \odot M'_b$ and φ by means of the unit propagation rule* SMT.1*, i.e.*

$$(M_{\text{dom}} \odot M'_b, \varphi) \ \longrightarrow_{SMT} \ (M_{\text{dom}} \odot M'_b \odot \langle e \rangle, \varphi)$$

  where the conjunctive system M'_b consists of all bounds from M'.

Then, Proposition 6.2 also holds when taking $(M_{\text{dom}} \odot M', \varphi)$ as the initial state, and when slightly changing the statements of items 2a and 2b to the facts that $\varphi \wedge M_{\text{dom}} \odot M'$ is unsatisfiable and $\varphi \wedge M_{\text{dom}} \odot M'$ is satisfiable, respectively.

The condition on M' in Remark 6.1 is necessary in order to ensure that conflict-driven clause learning (rule SMT.4) is applicable, namely that conflict clauses consist of bounds only. We see in the next section that aforementioned condition is satisfied whenever the subordinate SMT solver is invoked by the SSMT algorithm.

6.4 Algorithms for SSMT

This section finally presents an algorithm to solve SSMT problems which we call the SiSAT algorithm. Basically, the proposed procedure adds an additional layer to the SMT solver iSAT in order to cope with existential and randomized quantifiers. More precisely, this topmost layer traverses the Cartesian product of the domains of the quantified variables and thereby computes the satisfaction probabilities for the individual quantifiers in accordance with Definition 4.5 and as indicated in Figure 4.3. Upon each complete assignment to the quantified variables, iSAT is called to solve the corresponding quantifier-free

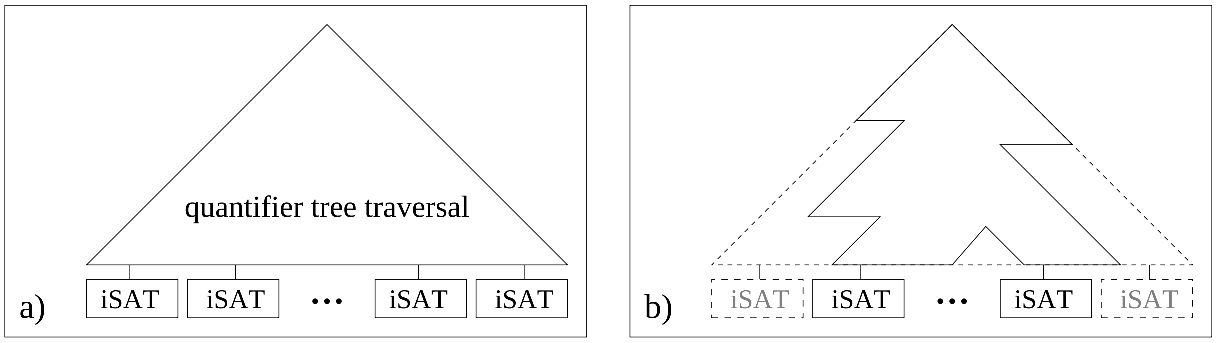


Figure 6.5: Illustration of the SiSAT algorithm: a) brute-force approach where the whole exponentially-sized quantifier tree is traversed and b) enhanced approach that employs several algorithmic optimizations to prune the quantifier tree.

subproblem, as shown in Figure 6.5 a. However, such naive approach is far from scalable as it has to solve one SMT problem per element of the Cartesian product of the quantifier ranges, which is exponential in the number of quantified variables. To overcome this problem, SiSAT employs aggressive pruning rules that save visits to major parts of the quantifier ranges based on semantic inferences, as illustrated in Figure 6.5 b. Such pruning exploits, on the one hand, deduction mechanisms inherited from iSAT like unit propagation (rule SMT.1) and ICP (rule SMT.2) in combination with conflict-driven clause learning (rule SMT.4), as conflict clauses record inconsistent value assignments which need not be probed again when traversing the set of possible assignments to quantified variables. On the other hand, the algorithmic optimizations used in SSAT solvers like thresholding and purification, confer Subsection 6.2.1, can be adapted to the more general SSMT case.

For the sake of clarity, the basic SiSAT algorithm as introduced next only incorporates the deduction techniques from iSAT as well as the concept of thresholding. Several other algorithmic enhancements are investigated in Section 6.5.

6.4.1 SSMT layer for non-linear arithmetic SSMT formulae

The SSMT solver is defined by the layer *SSMT* which rests upon the SMT layer, while the latter in turn is based on the theory layer *TS*. The solver state of this topmost layer is given by a tuple $(M, \mathcal{Q} : \varphi, \theta_l, \theta_u)$ where M is a conjunctive constraint system, $\mathcal{Q} : \varphi$ is an SSMT formula, and θ_l, θ_u with $\theta_l \leq \theta_u$ are rational constants that are called *lower threshold* and *upper threshold*, respectively, and are used for the algorithmic optimization of *thresholding*. With regard to the underlying iSAT algorithm, we demand that φ is in conjunctive form[8] and that the domains $\mathrm{dom}(x)$ of all variables $x \in \mathit{Var}(\varphi)$ are given by bounded intervals. Note that the domains $\mathcal{D}_x$ of quantified variables $x \in \mathit{Var}(\mathcal{Q})$, where $\mathit{Var}(\mathcal{Q})$ gives the set of all quantified variables occurring in prefix $\mathcal{Q}$ as in Definition 6.1 for SSAT formulae, can be represented by integer intervals, confer Subsection 4.4.1. As for the SMT layer and the theory layer, the current interval assignment is not explicitly given by the system state but encoded within M, and can be simply retrieved from M

[8]As observed at the end of Subsection 4.4.1, this restriction is without loss of generality.

by constructing the interval assignment σ_M. The SSMT layer therefore ensures that the conjunctive system M_{dom}, encoding the domains of all variables symbolically as defined in equation 6.17, is part of M for each solver state $(M, \mathcal{Q} : \varphi, \theta_l, \theta_u)$. This property holds, in particular, for the *initial SSMT solver state* given by $(M_{\text{dom}}, \mathcal{Q} : \varphi, \theta_l, \theta_u)$ with $\mathcal{Q} : \varphi$ being the given SSMT formula to be solved and with θ_l, θ_u being the given lower and upper thresholds.

The SSMT layer is specified by the SSMT rules below that define the relation $\longrightarrow_{SSMT}$. Starting with the initial state $(M_{\text{dom}}, \mathcal{Q} : \varphi, \theta_l, \theta_u)$, the SSMT layer applies these rules and terminates in a final state F, i.e. $(M_{\text{dom}}, \mathcal{Q} : \varphi, \theta_l, \theta_u) \longrightarrow^*_{SSMT} F$ where $\longrightarrow^*_{SSMT}$ denotes the reflexive transitive closure of $\longrightarrow_{SSMT}$ as usual. A final state F is a distinguished state of the shape (pr, φ') where $pr \in [0, 1]$ is the probability result and $\varphi' \supseteq \varphi$ is an SMT formula in CF. The rationale of returning an SMT formula φ' is on account of conflict-driven clause learning, i.e. learnt conflict clauses are contained in φ'. As the premises of the SSMT rules involve recursive calls, conflict clauses will thus be available by means of the results returned by the recursions. Recall that conflict clauses are always implied by the original SMT formula φ. Hence, $\varphi \equiv \varphi'$.

Before explaining the meaning of the probability result pr, we first have to address the *incompleteness* issue of iSAT, more precisely, how SiSAT deals with *approximate* solutions obtained by iSAT. Recall that there is no mathematical certainty about such latter results, i.e. the SMT formula may be unsatisfiable or satisfiable, though such approximate solutions σ are often reasonable and helpful in many practical applications, in particular if the intervals $\sigma(x)$ for all variables x are sufficiently small. With regard to the inconclusive nature of approximate solutions, we devise two possible strategies in order to cope with such undecided cases within the SSMT layer: first, we assume that an approximate solution means that the corresponding SMT subproblem is *unsatisfiable*, or, second, an approximate solution is considered as a definite solution, i.e. the corresponding SMT subproblem is regarded as *satisfiable*. To distinguish both strategies, we introduce the parameter $\tilde{p} \in \{0, 1\}$ which is set to 0 or to 1 in order to enable the first strategy (assuming unsatisfiability) or the second strategy (assuming satisfiability), respectively. This parameter $\tilde{p}$ serves as the return value if an approximate solution was found (rule SSMT.9), and is furthermore used to compute the probability result if thresholding applies (rules SSMT.3 and SSMT.4).

The probability result pr is interpreted with respect to the thresholds θ_l and θ_u, where the overall idea is the same as in the DPLL-SSAT case, confer Subsection 6.2.1. Namely, if the probability of satisfaction $Pr(\mathcal{Q} : \varphi)$ lies in the interval $[\theta_l, \theta_u]$ then we aim at computing the exact satisfaction probability, i.e. $pr = Pr(\mathcal{Q} : \varphi)$. Otherwise, i.e. $Pr(\mathcal{Q} : \varphi) \notin [\theta_l, \theta_u]$, the exact probability of satisfaction is not of interest but only some witness value pr is desired such that $pr < \theta_l$ if and only if $Pr(\mathcal{Q} : \varphi) < \theta_l$ and $pr > \theta_u$ if and only if $Pr(\mathcal{Q} : \varphi) > \theta_u$. In contrast to the DPLL-SSAT case, where above interpretation is fully applicable, we have to deal with *undecided* subcases stemming from approximate solutions in the SiSAT case. That is, SiSAT is in general not capable of computing the exact satisfaction probability of every non-linear arithmetic SSMT formula $\mathcal{Q} : \varphi$.

However, using above strategies indicated by parameter $\tilde{p}$, it turns out that we can deal with safe *under*- as well as *overapproximations* of the exact satisfaction probability $Pr(\mathcal{Q} : \varphi)$, namely if $\tilde{p} = 0$ and $\tilde{p} = 1$, respectively. As being demonstrated by Theorem 6.2 later on, we can establish the following interpretation of result pr:

- If $\tilde{p} = 0$, i.e. undecided SMT subproblems are assumed to be unsatisfiable, then pr is an underapproximation of the actual satisfaction probability, i.e. $Pr(\mathcal{Q} : \varphi) \geq pr$. Consequently, whenever $pr > \theta_u$ then $Pr(\mathcal{Q} : \varphi) > \theta_u$.
- If $\tilde{p} = 1$, i.e. undecided SMT subproblems are assumed to be satisfiable, then pr is an overapproximation of the actual satisfaction probability, i.e. $Pr(\mathcal{Q} : \varphi) \leq pr$. Consequently, whenever $pr < \theta_l$ then $Pr(\mathcal{Q} : \varphi) < \theta_l$.

The presence of the thresholds θ_l and θ_u is motivated, for instance, in the verification of probabilistic safety properties where the problem is to decide whether the probability of exhibiting unsafe system behavior is below or above some acceptable threshold, confer Definition 5.4. Akin to the DPLL-SSAT procedure from Figure 6.1, the thresholds θ_l and θ_u are exploited for reducing the computational effort of SSMT algorithms in practice, namely by potentially pruning the search tree, as indicated in Figure 6.5 b, using the idea of *thresholding*. It is important to remark that thresholding can be "disabled" by setting $\theta_l := 0$ and $\theta_u := 1$.

The SSMT layer is defined by the following rules SSMT.1 to SSMT.10. The treatment of the quantifier prefix incorporating the algorithmic concept of thresholding is described by rules SSMT.1 to SSMT.5, while rules SSMT.6 to SSMT.10 integrate the SMT layer into the SSMT proof search.

Branching and thresholding for quantified variables within the SSMT layer slightly generalize the corresponding approach applied in the DPLL-SSAT procedure, confer Figure 6.1. Recall that, as opposed to the SSAT case, quantified variables in SSMT are not restricted to the Boolean domain but may range over arbitrary (but finite) domains. Rules SSMT.1 and SSMT.2 characterize *branching* for existential and randomized variables, respectively, that is in accordance with the formal semantics of SSMT from Definition 4.5. That is, using the leftmost variable x from the quantifier prefix, the current SSMT problem Φ is split into smaller subproblems as follows. First, some value v from domain $\mathcal{D}_x$ is selected and the corresponding subproblem for branch "$x = v$" is solved. All remaining branches "$x = v'$" with $v' \in \mathcal{D}_x \setminus \{v\}$ are then addressed by a recursive call in which value v is removed from domain $\mathcal{D}_x$. Finally, both partial results are combined to obtain the probability result for Φ.

Observe that SSMT.1 and SSMT.2 are only applicable if the domains of all quantified variables are non-empty. The contrary is treated by rule SSMT.5 later on. Furthermore note that SSMT.1 and SSMT.2 require that thresholding fails. More precisely, if x is existential then the probability result pr of the first branch need be smaller than or equal to the upper threshold θ_u. We furthermore require that pr may not be the highest possible probability 1 since otherwise no other branch could yield a higher probability. If x is randomized then the weighted probability $p_v \cdot pr$ may be at most θ_u as well as the maximum possible satisfaction probability in consideration of all remaining branches, i.e. $p_v \cdot pr + p_{remain}$, must be greater than or equal to the lower threshold θ_l. We remark that a similar requirement like in the existential case, namely $p_v \cdot pr < 1$, does not make much sense since $p_v \cdot pr = 1$ can only be true if $p_v = 1$. The latter implies that domain $\mathcal{D}_x = \{v\}$ is a singleton and thus the second recursion immediately reaches a base case as $\mathcal{D}'_x = \emptyset$, which is handled by rule SSMT.5. The issue of determining the lower and upper thresholds for the recursive calls is examined later on, namely after introduction of rule SSMT.4.

$$\text{(SSMT.1)} \qquad \frac{\begin{array}{c} v \in \mathcal{D}_x,\ \forall (Qy \in \mathcal{D}_y) \in \mathcal{Q} : \mathcal{D}_y \neq \emptyset, \\ (M \odot \langle x = v \rangle, \mathcal{Q} : \varphi, \theta_l, \theta_u) \longrightarrow^*_{SSMT} (pr, \varphi'),\ pr \leq \theta_u,\ pr < 1 \\ \mathcal{D}'_x = \mathcal{D}_x \setminus \{v\},\ \theta'_l = \max(pr, \theta_l), \\ (M, \exists x \in \mathcal{D}'_x \odot \mathcal{Q} : \varphi', \theta'_l, \theta_u) \longrightarrow^*_{SSMT} (pr', \varphi'') \end{array}}{(M, \exists x \in \mathcal{D}_x \odot \mathcal{Q} : \varphi, \theta_l, \theta_u) \longrightarrow_{SSMT} (\max(pr, pr'), \varphi'')}$$

$$\text{(SSMT.2)} \qquad \frac{\begin{array}{c} v \in \mathcal{D}_x,\ \forall (Qy \in \mathcal{D}_y) \in \mathcal{Q} : \mathcal{D}_y \neq \emptyset, \\ p_v = d_x(v),\ p_{remain} = \sum_{v' \in \mathcal{D}_x, v' \neq v} d_x(v'), \\ \theta'_l = (\theta_l - p_{remain})/p_v,\ \theta'_u = \theta_u / p_v, \\ (M \odot \langle x = v \rangle, \mathcal{Q} : \varphi, \theta'_l, \theta'_u) \longrightarrow^*_{SSMT} (pr, \varphi'), \\ p_v \cdot pr + p_{remain} \geq \theta_l,\ p_v \cdot pr \leq \theta_u, \\ \mathcal{D}'_x = \mathcal{D}_x \setminus \{v\},\ \theta''_l = \theta_l - p_v \cdot pr,\ \theta''_u = \theta_u - p_v \cdot pr, \\ (M, \text{Я}_{d_x} x \in \mathcal{D}'_x \odot \mathcal{Q} : \varphi', \theta''_l, \theta''_u) \longrightarrow^*_{SSMT} (pr', \varphi'') \end{array}}{(M, \text{Я}_{d_x} x \in \mathcal{D}_x \odot \mathcal{Q} : \varphi, \theta_l, \theta_u) \longrightarrow_{SSMT} (p_v \cdot pr + pr', \varphi'')}$$

Recall that the exact probability result is not of interest whenever the satisfaction probability lies outside the interval $[\theta_l, \theta_u]$. This fact motivates the concept of *thresholding* which is now taken into account by rules SSMT.3 and SSMT.4. More precisely, if it turns out that the (weighted) probability of the first branch already exceeds the upper threshold, i.e. $pr > \theta_u$ in the existential case and $p_v \cdot pr > \theta_u$ in the randomized case, then we skip investigation of all remaining branches "$x = v'$" with $v' \in \mathcal{D}_x \setminus \{v\}$, as we have already found out that the satisfaction probability of the corresponding subproblem is greater than θ_u. We furthermore discard all remaining branches in the existential case if pr is the highest possible probability, i.e. if $pr = 1$, since then no result of any other branch can be greater than pr. Thresholding is also applicable whenever it turns out that the maximum possible probability mass of all remaining branches is *not* large enough to reach the lower threshold. The latter rule can be realized for randomized variables, namely by checking whether $p_v \cdot pr + p_{remain} < \theta_l$ is true. A similar rule for the existential case is however infeasible since we cannot derive in general any non-trivial upper bound on the maximum of the unknown probability results of the remaining branches. Whenever thresholding succeeds, a potentially very large part of the search space is skipped as indicated in Figure 6.5 b, which often leads to enormous performance gains in practice. In Section 6.7, we present empirical results showing the benefit of thresholding and several other algorithmic optimizations, the latter being introduced in Section 6.5.

Particular attention must be devoted to the returned probability result. Recall that if $\tilde{p} = 0$ or $\tilde{p} = 1$ then SiSAT aims at approximating the satisfaction probability safely from below or from above, respectively. We consider rule SSMT.3 first. If $\tilde{p} = 0$ then $pr \geq 0$ is a lower bound of the satisfaction probability of branch "$x = v$". Since variable x is existential, the return value $\max(pr, \tilde{p}) = \max(pr, 0) = pr$ is a safe lower probability bound of the whole SSMT subproblem. Similarly, if $\tilde{p} = 1$ then $pr \leq 1$ is an upper bound of the satisfaction probability of branch "$x = v$". Since variable x is existential, the return value $\max(pr, \tilde{p}) = \max(pr, 1) = 1$ is a safe upper probability bound of the whole SSMT subproblem. One might wonder why the trivial upper bound 1 is used here. The rationale is that the satisfaction probability of some of the remaining branches could be strictly greater than pr. In such cases, pr is *no* upper probability bound of the whole SSMT subformula. It is however clear that the upper bound for the whole subproblem, i.e. when considering all of the remaining branches, must be greater than θ_u and, thus, cannot be strictly smaller than $\theta_l \leq \theta_u$. This gives us the motivation to apply SSMT.3 even if $\tilde{p} = 1$. We now investigate the probability result returned by rule SSMT.4. If $\tilde{p} = 0$ then pr is a lower bound of the satisfaction probability of branch "$x = v$". Since variable x is randomized, the return value $p_v \cdot pr + p_{remain} \cdot \tilde{p} = p_v \cdot pr + p_{remain} \cdot 0 = p_v \cdot pr$, i.e. the weighted result for branch "$x = v$", is a safe lower probability bound of the whole SSMT subproblem. If $\tilde{p} = 1$ then pr is an upper bound of the satisfaction probability of branch "$x = v$". Since variable x is randomized, the return value $p_v \cdot pr + p_{remain} \cdot \tilde{p} = p_v \cdot pr + p_{remain} \cdot 1 = p_v \cdot pr + p_{remain}$, i.e. the sum of the weighted result for branch "$x = v$" and the maximum possible probability of all remaining branches, is a safe upper probability bound of the whole SSMT subproblem.

As for SSMT.1 and SSMT.2 above, rules SSMT.3 and SSMT.4 are only applicable if the domains of all quantified variables are non-empty.

(SSMT.3)
$$\frac{\begin{array}{c} v \in \mathcal{D}_x,\ \forall (Qy \in \mathcal{D}_y) \in \mathcal{Q} : \mathcal{D}_y \neq \emptyset, \\ (M \odot \langle x = v \rangle, \mathcal{Q} : \varphi, \theta_l, \theta_u) \longrightarrow^*_{SSMT} (pr, \varphi'),\ pr > \theta_u \text{ or } pr = 1 \end{array}}{(M, \exists x \in \mathcal{D}_x \odot \mathcal{Q} : \varphi, \theta_l, \theta_u) \longrightarrow_{SSMT} (\max(pr, \tilde{p}), \varphi')}$$

(SSMT.4)
$$\frac{\begin{array}{c} v \in \mathcal{D}_x,\ \forall (Qy \in \mathcal{D}_y) \in \mathcal{Q} : \mathcal{D}_y \neq \emptyset, \\ p_v = d_x(v),\ p_{remain} = \sum_{v' \in \mathcal{D}_x, v' \neq v} d_x(v'), \\ \theta'_l = (\theta_l - p_{remain})/p_v,\ \theta'_u = \theta_u/p_v, \\ (M \odot \langle x = v \rangle, \mathcal{Q} : \varphi, \theta'_l, \theta'_u) \longrightarrow^*_{SSMT} (pr, \varphi'), \\ p_v \cdot pr + p_{remain} < \theta_l \text{ or } p_v \cdot pr > \theta_u \end{array}}{(M, Ⅎ_{d_x} x \in \mathcal{D}_x \odot \mathcal{Q} : \varphi, \theta_l, \theta_u) \longrightarrow_{SSMT} (p_v \cdot pr + p_{remain} \cdot \tilde{p}, \varphi')}$$

With respect to soundness of thresholding, the lower threshold θ'_l and the upper threshold θ'_u for the first recursive call $(M \odot \langle x = v \rangle, \mathcal{Q} : \varphi, \theta'_l, \theta'_u)$ need to be set correctly. For this purpose, we require the following. Above call returns a value $pr < \theta'_l$ if and only if $pr < \theta_l$ whenever variable x is existential and $p_v \cdot pr + p_{remain} < \theta_l$ whenever x is randomized. A value $pr > \theta'_u$ is given back if and only if $pr > \theta_u$ whenever x is existential and $p_v \cdot pr > \theta_u$ whenever x is randomized. As $p_v > 0$, this justifies the use of thresholds θ'_l and θ'_u with

$$\theta'_l = \begin{cases} \theta_l & \text{if } x \text{ is existential,} \\ (\theta_l - p_{remain})/p_v & \text{if } x \text{ is randomized,} \end{cases}$$

$$\theta'_u = \begin{cases} \theta_u & \text{if } x \text{ is existential,} \\ \theta_u/p_v & \text{if } x \text{ is randomized} \end{cases}$$

for the first recursion within the premises of above rules SSMT.1 to SSMT.4. Observe that the new thresholds θ'_l and θ'_u satisfy $\theta'_l \leq \theta'_u$, i.e. $\theta_l \leq \theta_u$ for the existential case, and $(\theta_l - p_{remain})/p_v \leq \theta_u/p_v$ since $p_{remain} \geq 0$ for the randomized case.

The thresholds for the second recursion in rules SSMT.1 and SSMT.2, covering all remaining branches "$x = v'$" for $v' \in \mathcal{D}_x \setminus \{v\}$, are determined as follows. In the existential case, the upper threshold remains untouched while the lower threshold can be increased to the probability result *pr* of the first branch whenever *pr* is greater than the current lower threshold θ_l. Otherwise, i.e. $pr \leq \theta_l$, lower threshold θ_l must be used. Since $pr \leq \theta_u$, it holds that $\max(pr, \theta_l) \leq \theta_u$. The rationale is due to the semantics of existential quantifiers that aim at *maximizing* the probability of satisfaction. That is, all results pr' of the remaining subproblems that are smaller than *pr* do not have any impact on the actual probability result. In the randomized case, we subtract from both thresholds θ_l and θ_u the probability result $p_v \cdot pr$ obtained from the first branch. The rationale is that the problem of deciding whether $\theta_l \leq p_v \cdot pr + pr' \leq \theta_u$ is equivalent to checking whether $\theta_l - p_v \cdot pr \leq pr' \leq \theta_u - p_v \cdot pr$, where pr' is the satisfaction probability of the remaining subproblem $\text{Я}_{d_x} x \in \mathcal{D}'_x \odot \mathcal{Q} : (\varphi' \wedge M)$. Clearly, $\theta_l - p_v \cdot pr \leq \theta_u - p_v \cdot pr$.

As in the DPLL-SSAT case, lower thresholds θ_l may become negative, i.e. $\theta_l < 0$, and upper thresholds θ_u may exceed 1, i.e. $\theta_u > 1$. This fact, however, does *not* destroy soundness of thresholding as we have *never* stated any assumption on the range of θ_l and θ_u. The only condition is that $\theta_l \leq \theta_u$ holds in each SSMT solver state $(M, \mathcal{Q} : \varphi, \theta_l, \theta_u)$, which is actually preserved for the recursive calls as shown above.

The next rule SSMT.5 establishes the stopping criterion for the second recursion within above rules SSMT.1 and SSMT.2, namely in case the domain of the leftmost quantified variable x in the prefix is empty. The probability result then is 0, which is correct as 0 is the neutral element of the maximum operation for non-negative reals (applied if x is existential) and of addition (applied if x is randomized). One might wonder why an own rule is introduced to stop the recursion and why the latter is not integrated, for instance, into rules SSMT.3 and SSMT.4. Moreover, rule SSMT.5 is not restricted to the leftmost element in prefix $\mathcal{Q}$ but is applicable if any domain $\mathcal{D}_x$ within $\mathcal{Q}$ is empty. The presence of this, allegedly too general, rule is motivated by the subsequent rule SSMT.6, which employs reasoning mechanisms to prune the search space including the domains of quantified variables. If application of SSMT.6 yields an empty domain for some quantified variable $x \in \mathit{Var}(\mathcal{Q})$ then this indicates that formula $\varphi \wedge M$ cannot be satisfied, i.e. $\varphi \wedge M$ is unsatisfiable, which in turn means that $Pr(\mathcal{Q} : (\varphi \wedge M)) = 0$. The latter fact follows from Definition 4.5.

$$\text{(SSMT.5)} \qquad \frac{\exists (Qx \in \mathcal{D}_x) \in \mathcal{Q} : \mathcal{D}_x = \emptyset}{(M, \mathcal{Q} : \varphi, \theta_l, \theta_u) \longrightarrow_{SSMT} (0, \varphi)}$$

All of the aforementioned SSMT rules are designed to deal with the quantifier prefix. In the following, we now embed the SMT layer into the SSMT algorithm. The next rule SSMT.6 exploits the deduction mechanisms employed in iSAT, more precisely unit propagation and ICP, in order to cut off potentially huge parts of the quantifier tree as depicted in Figure 6.5 b. That is, if the SMT solver is able to deduce new facts M' from M and φ by rules SMT.1 (unit propagation) and SMT.2 (ICP) then we lift these deductions to the SSMT layer by means of rule SSMT.6 and thereby try to prune the domains of quantified variables.

$$\text{(SSMT.6)} \qquad \frac{(M, \varphi) \longrightarrow^*_{SMT} (M \odot M', \varphi),\ \ \lightning \notin M',\ \ M' \neq \varepsilon,\ \ \mathcal{Q}' = \mathit{prune}(\mathcal{Q}, \sigma_{M \odot M'})}{(M, \mathcal{Q} : \varphi, \theta_l, \theta_u) \longrightarrow_{SSMT} (M \odot M', \mathcal{Q}' : \varphi, \theta_l, \theta_u)}$$

where $prune(\mathcal{Q}, \sigma_{M \odot M'})$ potentially excludes values from the domains $\mathcal{D}_x$ of quantified variables $x \in Var(\mathcal{Q})$ that do not lie within the current intervals $\sigma_{M \odot M'}(x)$ and thus cannot be part of a solution. More formally, for prefix $\mathcal{Q} = Q_1 x_1 \in \mathcal{D}_{x_1} \odot \ldots \odot Q_n x_n \in \mathcal{D}_{x_n}$, we define

$$prune(\mathcal{Q}, \sigma_{M \odot M'}) \;:=\; Q_1 x_1 \in (\mathcal{D}_{x_1} \cap \sigma_{M \odot M'}(x_1)) \odot \ldots \odot Q_n x_n \in (\mathcal{D}_{x_n} \cap \sigma_{M \odot M'}(x_n)) \quad .$$

Observe that M' within above rule SSMT.6 may contain *equations*, confer Definition 4.3. Such equations $e \in M'$ must have been added by the unit propagation rule SMT.1, i.e. by $(M \odot M'', \varphi) \longrightarrow_{SMT} (M \odot M'' \odot \langle e \rangle, \varphi)$ for some subsystem $M'' \subset M'$. As application of SMT.1 does *not* rely on equations from $M \odot M''$ directly but just on bounds from $M \odot M''$ (realized by interval assignment $\sigma_{M \odot M''}$ in rule TS.1), the condition in Remark 6.1 is preserved by rule SSMT.6, i.e. Remark 6.1 applies for SMT solver state $(M \odot M', \varphi)$.

Whenever the quantifier prefix is empty, the SMT solver iSAT is called to solve the remaining (quantifier-free) SMT problem $\varphi \wedge M$. As explained above, the condition in Remark 6.1 holds for the SMT solver state (M, φ). On account of Proposition 6.2 or rather of Remark 6.1, iSAT therefore terminates in one of the distinguished states `unsat` and `sat` or in some state (M', φ') such that $\sigma_{M'}$ is an approximate solution.

The base cases `unsat` and `sat` are treated by rules SSMT.7 and SSMT.8, respectively, returning the probability results 0 and 1 according to Definition 4.5. In addition, the SMT formula φ' of the immediate predecessor state (M', φ') is given back. Recall that $\varphi' \supseteq \varphi$ includes all implied conflict clauses that were learnt during SMT proof search. These conflict clauses, recording inconsistent value assignments, thus are available within the SSMT layer. Then, unit propagation via SSMT.6 is also applicable to conflict clauses potentially pruning the quantifier tree.

(SSMT.7)
$$\frac{(M, \varphi) \longrightarrow^*_{SMT} (M', \varphi') \longrightarrow_{SMT} \texttt{unsat}}{(M, \varepsilon : \varphi, \theta_l, \theta_u) \longrightarrow_{SSMT} (0, \varphi')}$$

(SSMT.8)
$$\frac{(M, \varphi) \longrightarrow^*_{SMT} (M', \varphi') \longrightarrow_{SMT} \texttt{sat}}{(M, \varepsilon : \varphi, \theta_l, \theta_u) \longrightarrow_{SSMT} (1, \varphi')}$$

The result of an *approximate solution* is handled by rule SSMT.9. While the return values for cases `unsat` and `sat`, namely 0 and 1, respectively, are determined by Definition 4.5, it is not clear which probability should be given back in case an approximate solution was found. As already mentioned, we can choose between two possible strategies to cope with such undecided cases: undecided SMT subproblems are assumed either to be unsatisfiable leading to an underapproximation of the actual satisfaction probability, which is indicated by $\tilde{p} = 0$, or to be satisfiable resulting in an overapproximation, which is employed if $\tilde{p} = 1$. According to the latter, the probability result for this base case is directly given by the value of $\tilde{p}$. As for above rules SSMT.7 and SSMT.8, SMT formula φ', containing all conflict clauses learnt, is also returned.

(SSMT.9)
$$\frac{(M, \varphi) \longrightarrow^*_{SMT} (M', \varphi'),\;\; \nexists S : (M', \varphi') \longrightarrow_{SMT} S}{(M, \varepsilon : \varphi, \theta_l, \theta_u) \longrightarrow_{SSMT} (\tilde{p}, \varphi')}$$

For the sake of efficiency, we introduce a further rule SSMT.10 that is similar to SSMT.7. However, SSMT.10 does *not* demand that the quantifier prefix $\mathcal{Q}$ is empty such that the probability result is potentially returned much earlier, thus improving performance of the SSMT procedure. More precisely, whenever the current SMT problem $\varphi \wedge M$ becomes unsatisfiable then it immediately follows from Definition 4.5 that $Pr(\mathcal{Q} : (\varphi \wedge M)) = 0$ even though the quantifier prefix $\mathcal{Q}$ is non-empty. This fact thus justifies return value 0.

(SSMT.10) $$\frac{(M, \varphi) \longrightarrow_{SMT} \texttt{unsat}}{(M, \mathcal{Q} : \varphi, \theta_l, \theta_u) \longrightarrow_{SSMT} (0, \varphi)}$$

A similar rule for result `sat` is not feasible in general as satisfiability of $\varphi \wedge M$ does not imply that $Pr(\mathcal{Q} : (\varphi \wedge M)) = 1$ holds for non-empty $\mathcal{Q}$. We remark that an algorithmic optimization loosely referring to this issue is investigated in Section 6.5. This technique called *solution-directed backjumping* is based on detecting quantified variables which have no impact on the current (approximate) solution, and on computing the satisfaction probabilities for these variables immediately without probing their alternative values.

6.4.2 Soundness and termination of the SSMT layer

In what follows, we prove *soundness* and *termination* of the SSMT layer. For this purpose, let $(M, \mathcal{Q} : \varphi, \theta_l, \theta_u)$ be a (valid) state of the SSMT layer, i.e. M is a conjunctive constraint system, $\mathcal{Q} : \varphi$ is an SSMT formula where matrix φ is in CF as in Definition 4.3, and θ_l and θ_u with $\theta_l \leq \theta_u$ are two rational constants (the lower threshold and upper threshold, respectively). Observe that the special marker symbol $|$ is *never* added to M by any rule of the SSMT layer, i.e. for each above state it holds that $| \notin M$. We further assume that the embedded SMT layer state (M, φ) satisfies the condition in Remark 6.1. We mention that an initial state of the SSMT layer, i.e. where $M = M_{\text{dom}}$, confer equation 6.17, is compliant with the above requirements.

Lemma 6.5 first shows that modification of the SSMT formula by rule SSMT.6 preserves the maximum probability of satisfaction.

Lemma 6.5 (Modification of SSMT formula)
For each sequence

$$(M, \mathcal{Q} : \varphi, \theta_l, \theta_u) \quad \longrightarrow^*_{SSMT} \quad (M', \mathcal{Q}' : \varphi', \theta'_l, \theta'_u)$$

of SSMT rule applications, it holds that $\varphi = \varphi'$, $\theta_l = \theta'_l$, $\theta_u = \theta'_u$, *and*

$$Pr(\mathcal{Q} : (\varphi \wedge M)) \quad = \quad Pr(\mathcal{Q}' : (\varphi' \wedge M')) \quad .$$

Proof. We show the lemma by induction over the number k of rule applications. For base case where $k = 0$, i.e. no rule was applied, the result follows obviously since $M = M'$, $\mathcal{Q} = \mathcal{Q}'$, $\varphi = \varphi'$, $\theta_l = \theta'_l$, and $\theta_u = \theta'_u$. For the induction step, we assume that the lemma holds for $k \geq 0$, i.e. for

$$(M, \mathcal{Q} : \varphi, \theta_l, \theta_u) \quad \longrightarrow^*_{SSMT} \quad (M'', \mathcal{Q}'' : \varphi, \theta_l, \theta_u)$$

such than
$$(M'', \mathcal{Q}'' : \varphi, \theta_l, \theta_u) \quad \longrightarrow_{SSMT} \quad (M', \mathcal{Q}' : \varphi', \theta_l', \theta_u') \ .$$

It then remains to show the statement for the latter step where, obviously, rule SSMT.6 was applied. Consequently, $\varphi = \varphi'$, $\theta_l = \theta_l'$, and $\theta_u = \theta_u'$. Due to rules SMT.1 and SMT.2 applied within the premise of SSMT.6, i.e. within $(M'', \varphi) \longrightarrow^*_{SMT} (M', \varphi)$, it holds that $\varphi \wedge M'' \equiv \varphi \wedge M'$. Let be $(Qx \in \mathcal{D}_x) \in \mathcal{Q}''$ and $(Qx \in \mathcal{D}_x') \in \mathcal{Q}'$ for some quantified variable x. Then, for each $v \in \mathcal{D}_x \setminus \mathcal{D}_x'$ the SMT formula $(\varphi \wedge M')[v/x]$ is unsatisfiable. As a consequence,

$$Pr(\mathcal{Q} : (\varphi \wedge M)) \quad = \quad Pr(\mathcal{Q}'' : (\varphi \wedge M'')) \quad = \quad Pr(\mathcal{Q}' : (\varphi' \wedge M'))$$

which proves the lemma. □

The following proofs of soundness and termination of the SSMT algorithm involve the principle of *Noetherian induction.* This proof method is based on well-founded strict partial orders. A *strict partial order* on some set A is a transitive and irreflexive binary relation $\succ$, i.e. for all $a_1, a_2, a_3 \in A$ it holds that if $a_1 \succ a_2$ and $a_2 \succ a_3$ then $a_1 \succ a_3$, and for all $a \in A$ it does not hold that $a \succ a$, i.e. $a \not\succ a$. A strict partial order $\succ$ on A is called *well-founded* if and only if there is no infinite descending chain $a_1 \succ a_2 \succ \ldots \succ a_n \succ \ldots$. An element $m \in A$ is called *minimal element* with respect to some strict partial order $\succ$ if and only if there does not exist an element $m' \in A$ such that $m \succ m'$. The idea of Noetherian induction is as follows. Let A be some set and $\succ$ be some well-founded strict partial order on A. Assume that we would like to prove that some property P holds for all elements of A. By the principle of Noetherian induction, it suffices to show the following implication for all $a \in A$:

- *if* property P holds for all $a' \in A$ with $a \succ a'$ *then* property P holds for a.

In case above implication is satisfied, it follows that property P is true for all $a \in A$.

In practical applications of this induction scheme, it is common to distinguish two special cases of above implication which are proven separately. First, all *minimal* elements $a \in A$ are considered. In this *base case*, we need to show that P holds for each minimal element of A. Second, all *non-minimal* elements $a \in A$ are examined. Here, we take an arbitrary non-minimal element $a \in A$ and assume that P is true for all "smaller" elements, i.e. for all $a' \in A$ such that $a \succ a'$. The latter gives us the *induction hypothesis.* In the *induction step*, we then conclude that property P holds for a using the induction hypothesis.

To prove soundness and termination of the SSMT algorithm by means of Noetherian induction, we define the following well-founded strict partial order on the set of SSMT layer states.

Definition 6.2 (Well-founded strict partial order $\succ_{\mathrm{SSMT}}$)
Let $\mathcal{S}_{SSMT}$ be the set of all possible, non-distinguished states of the SSMT layer. The binary relation $\succ_{SSMT}$ on $\mathcal{S}_{SSMT}$ is defined as follows. Let $S, S' \in \mathcal{S}_{SSMT}$ with $S = (M, \mathcal{Q} : \varphi, \theta_l, \theta_u)$ and $S' = (M', \mathcal{Q}' : \varphi', \theta_l', \theta_u')$ be two states of the SSMT layer. Then, $S \succ_{SSMT} S'$ if and only if

1. *for each $(Qx \in \mathcal{D}_x) \in \mathcal{Q}$ it holds that $\mathcal{D}_x \neq \emptyset$, and*

2. $\mathcal{Q} = \mathcal{Q}'' \odot Qx_1 \in \mathcal{D}_{x_1} \odot \ldots \odot Qx_n \in \mathcal{D}_{x_n}$ *and* $\mathcal{Q}' = Qx_1 \in \mathcal{D}'_{x_1} \odot \ldots \odot Qx_n \in \mathcal{D}'_{x_n}$ *such that for all* $i \in \{1, \ldots, n\} : \mathcal{D}_{x_i} \supseteq \mathcal{D}'_{x_i}$ *and if* $\mathcal{Q}'' = \varepsilon$ *then there exists some* $j \in \{1, \ldots, n\} : \mathcal{D}_{x_j} \supset \mathcal{D}'_{x_j}$.

Intuitively, state S' is "smaller" than state S if and only if all domains within quantifier prefix $\mathcal{Q}$ are non-empty, and prefix $\mathcal{Q}'$ arises from $\mathcal{Q}$ by removing some quantifiers from the left or by removing values from the domains of some quantified variables. Observe that $\mathcal{Q}$ need be non-empty of necessity, i.e. $\mathcal{Q} \neq \varepsilon$, as, otherwise, if $\mathcal{Q} = \varepsilon$ then $\mathcal{Q}'' = \varepsilon$ which immediately entails by definition the contradiction that $\mathcal{Q} \neq \varepsilon$. It is not hard to see that relation $\succ_{\text{SSMT}}$ is transitive and irreflexive as well as well-founded. The latter is due to the fact that the number of quantifiers in a prefix as well as the domains of quantified variables are both finite. Minimal elements with respect to $\succ_{\text{SSMT}}$ are states $(M, \mathcal{Q} : \varphi, \theta_l, \theta_u)$ where the quantifier prefix is empty, i.e. $\mathcal{Q} = \varepsilon$, or where $\mathcal{Q}$ contains an empty domain, i.e. $\mathcal{D}_x = \emptyset$ for some $(Qx \in \mathcal{D}_x) \in \mathcal{Q}$.

The next theorem now establishes *soundness* of the SSMT layer.

Theorem 6.2 (Soundness of the SSMT algorithm)
For each sequence

$$(M, \mathcal{Q} : \varphi, \theta_l, \theta_u) \quad \longrightarrow^*_{SSMT} \quad (pr, \varphi')$$

of SSMT rule applications with fixed $\tilde{p} \in \{0, 1\}$, *it holds that*

1. *if* $\tilde{p} = 0$ *then* $Pr(\mathcal{Q} : (\varphi \wedge M)) \geq pr$ *and*

2. *if* $\tilde{p} = 1$ *then* $Pr(\mathcal{Q} : (\varphi \wedge M)) \leq pr$.

Proof. We prove the theorem by Noetherian induction using the well-founded strict partial order $\succ_{\text{SSMT}}$ from Definition 6.2.

First of all, note that there must be a state $(M', \mathcal{Q}' : \varphi'', \theta'_l, \theta'_u)$ such that

$$(M, \mathcal{Q} : \varphi, \theta_l, \theta_u) \quad \longrightarrow^*_{SSMT} \quad (M', \mathcal{Q}' : \varphi'', \theta'_l, \theta'_u) \quad \longrightarrow_{SSMT} \quad (pr, \varphi') \quad .$$

Observe that if states $(M, \mathcal{Q} : \varphi, \theta_l, \theta_u)$ and $(M', \mathcal{Q}' : \varphi'', \theta'_l, \theta'_u)$ are different then the latter state is reachable by applications of rule SSMT.6 *only*. As explained after introduction of SSMT.6, each application of the latter rule preserves the condition in Remark 6.1, in particular for the SMT solver state (M', φ'').

By Lemma 6.5, we know that $\varphi = \varphi''$, $\theta_l = \theta'_l$, $\theta_u = \theta'_u$, and $Pr(\mathcal{Q} : (\varphi \wedge M)) = Pr(\mathcal{Q}' : (\varphi'' \wedge M'))$. It therefore suffices to consider the latter step in the base case as well as in the induction step, while the induction hypothesis holds for the whole sequence of SSMT rule applications. That is, we show the statement of the theorem for

$$(M', \mathcal{Q}' : \varphi, \theta_l, \theta_u) \quad \longrightarrow_{SSMT} \quad (pr, \varphi') \quad .$$

In the base case, we assume that state $(M', \mathcal{Q}' : \varphi, \theta_l, \theta_u)$ is minimal with respect to $\succ_{\text{SSMT}}$, i.e. $\mathcal{Q}' = \varepsilon$ or $\mathcal{D}_x = \emptyset$ for some $(Qx \in \mathcal{D}_x) \in \mathcal{Q}'$. Then, only rules SSMT.5, SSMT.7, SSMT.8, SSMT.9, and SSMT.10 could be applied.

For SSMT.5, the domain $\mathcal{D}_x$ of some quantified variable $x \in \mathit{Var}(\mathcal{Q}')$ is empty, i.e. $\mathcal{D}_x = \emptyset$, meaning that $\varphi \wedge M'$ is unsatisfiable within the domains of the variables and thus $Pr(\mathcal{Q}' : (\varphi \wedge M')) = 0$. We have that $pr = 0 = Pr(\mathcal{Q}' : (\varphi \wedge M'))$ from which above items 1 and 2 follow immediately.

For rules SSMT.7 and SSMT.10, the SMT formula $\varphi \wedge M'$ is unsatisfiable according to item 2a of Proposition 6.2 or rather of Remark 6.1. As a consequence of Definition 4.5, $Pr(\mathcal{Q}' : (\varphi \wedge M')) = 0$. As above, from $pr = 0 = Pr(\mathcal{Q}' : (\varphi \wedge M'))$ we conclude that items 1 and 2 hold.

The reasoning for SSMT.8 is very similar: $\varphi \wedge M'$ is satisfiable according to item 2b of Proposition 6.2 or rather of Remark 6.1. As $\mathcal{Q}' = \varepsilon$, $Pr(\mathcal{Q}' : (\varphi \wedge M')) = 1$ according to Definition 4.5. Items 1 and 2 immediately follow from the fact that $pr = 1 = Pr(\mathcal{Q}' : (\varphi \wedge M'))$.

We now consider rule SSMT.9. Concerning item 1, i.e. $\tilde{p} = 0$, we have that $pr = \tilde{p} = 0$ and thus $Pr(\mathcal{Q}' : (\varphi \wedge M')) \geq 0 = pr$. With regard to item 2, i.e. $\tilde{p} = 1$, from $pr = \tilde{p} = 1$ it clearly follows that $Pr(\mathcal{Q}' : (\varphi \wedge M')) \leq 1 = pr$.

For the induction step, let $(M', \mathcal{Q}' : \varphi, \theta_l, \theta_u)$ be an arbitrary state which is non-minimal with respect to $\succ_{\text{SSMT}}$, i.e. $\mathcal{Q}' \neq \varepsilon$ and $\mathcal{D}_x \neq \emptyset$ for all $(Qx \in \mathcal{D}_x) \in \mathcal{Q}'$. Then, only rules SSMT.1, SSMT.2, SSMT.3, SSMT.4, and SSMT.10 could be applied. As induction hypothesis, we assume that the statement of the theorem holds for all "smaller" states S, i.e. for all $S \in \mathcal{S}_{\text{SSMT}}$ such that $(M', \mathcal{Q}' : \varphi, \theta_l, \theta_u) \succ_{\text{SSMT}} S$.

Observe that $\varphi \Leftrightarrow \varphi'$ holds in the premises of SSMT.1, SSMT.2, SSMT.3, and SSMT.4, as well as $\varphi' \Leftrightarrow \varphi''$ and thus $\varphi \Leftrightarrow \varphi''$ for SSMT.1 and SSMT.2. This can be proven by simple induction with base cases SSMT.7, SSMT.8, and SSMT.9. The statement for the base cases is established by two facts. First, $\varphi \Leftarrow \varphi'$ holds since no SMT rule removes clauses from the SMT formula φ. Second, $\varphi \Rightarrow \varphi'$ follows immediately from application condition of rule SMT.4: each learnt conflict clause $c \in \varphi' \setminus \varphi$ that was added to SMT formula φ is implied by φ, i.e. $\varphi \Rightarrow c$. No other SMT rule adds clauses to φ.

Moreover, observe that the condition in Remark 6.1 is preserved for the SMT solver states $(M \odot \langle x = v \rangle, \varphi)$ and (M, φ') which are embedded in the corresponding SSMT layer states used within the premises of rules SSMT.1, SSMT.2, SSMT.3, and SSMT.4. That is, potential application of rules SSMT.7, SSMT.8, SSMT.9, and SSMT.10 is valid.

For rule SSMT.1, let be $\mathcal{Q}' = \exists x \in \mathcal{D}_x \odot \mathcal{Q}''$ and

$$\begin{array}{lll} (M' \odot \langle x = v \rangle, \mathcal{Q}'' : \varphi, \theta_l, \theta_u) & \longrightarrow_{SSMT} & (pr_1, \varphi_1) \quad , \\ (M', \exists x \in \mathcal{D}'_x \odot \mathcal{Q}'' : \varphi_1, \theta'_l, \theta_u) & \longrightarrow_{SSMT} & (pr_2, \varphi_2) \quad . \end{array}$$

Then, $pr = \max(pr_1, pr_2)$. Due to Definition 6.2,

$$\begin{array}{lll} (M', \exists x \in \mathcal{D}_x \odot \mathcal{Q}'' : \varphi, \theta_l, \theta_u) & \succ_{\text{SSMT}} & (M' \odot \langle x = v \rangle, \mathcal{Q}'' : \varphi, \theta_l, \theta_u) \quad , \\ (M', \exists x \in \mathcal{D}_x \odot \mathcal{Q}'' : \varphi, \theta_l, \theta_u) & \succ_{\text{SSMT}} & (M', \exists x \in \mathcal{D}'_x \odot \mathcal{Q}'' : \varphi_1, \theta'_l, \theta_u) \quad . \end{array}$$

From induction hypothesis and since $\varphi \Leftrightarrow \varphi_1$, we then conclude the following. If $\tilde{p} = 0$ then

$$\begin{array}{lll} Pr(\mathcal{Q}'' : (\varphi \wedge M' \odot \langle x = v \rangle)) & \geq & pr_1 \quad , \\ Pr(\exists x \in \mathcal{D}'_x \odot \mathcal{Q}'' : (\varphi \wedge M')) & \geq & pr_2 \quad , \end{array}$$

and if $\tilde{p} = 1$ then

$$\begin{array}{lll} Pr(\mathcal{Q}'' : (\varphi \wedge M' \odot \langle x = v \rangle)) & \leq & pr_1 \quad , \\ Pr(\exists x \in \mathcal{D}'_x \odot \mathcal{Q}'' : (\varphi \wedge M')) & \leq & pr_2 \quad . \end{array}$$

In conformity with Definition 4.5,

$$\begin{array}{rl} & Pr(\exists x \in \mathcal{D}_x \odot \mathcal{Q}'' : (\varphi \wedge M')) \\ = & \max(\ Pr(\mathcal{Q}'' : (\varphi \wedge M' \odot \langle x = v \rangle)),\ Pr(\exists x \in \mathcal{D}'_x \odot \mathcal{Q}'' : (\varphi \wedge M'))\ ) \end{array}$$

where $Pr(Qy \in \mathcal{D}_y \odot \mathcal{Q} : \psi) := 0$ whenever $\mathcal{D}_y = \emptyset$. For item 1, i.e. $\tilde{p} = 0$, it immediately follows that $Pr(\exists x \in \mathcal{D}_x \odot \mathcal{Q}'' : (\varphi \wedge M')) \geq \max(pr_1, pr_2) = pr$. Similarly for item 2, i.e. $\tilde{p} = 1$, $Pr(\exists x \in \mathcal{D}_x \odot \mathcal{Q}'' : (\varphi \wedge M')) \leq \max(pr_1, pr_2) = pr$.

For rule SSMT.3, we use the same assumptions as for SSMT.1 above whenever applicable. We have that $pr = \max(pr_1, \tilde{p})$. Concerning item 1, i.e. $\tilde{p} = 0$, we conclude that $Pr(\exists x \in \mathcal{D}_x \odot \mathcal{Q}'' : (\varphi \wedge M')) \geq pr_1$. As $Pr(\Phi) \geq 0$ for each SSMT formula Φ, $Pr(\exists x \in \mathcal{D}_x \odot \mathcal{Q}'' : (\varphi \wedge M')) \geq \max(pr_1, 0) = pr$. We now consider item 2, i.e. $\tilde{p} = 1$. As $Pr(\Phi) \leq 1$ is true for each SSMT formula Φ, we conclude that $Pr(\exists x \in \mathcal{D}_x \odot \mathcal{Q}'' : (\varphi \wedge M')) \leq \max(pr_1, 1) = pr$.

For rule SSMT.2, let be $\mathcal{Q}' = \text{Я}_{d_x} x \in \mathcal{D}_x \odot \mathcal{Q}''$ and

$$\begin{array}{llll} (M' \odot \langle x = v \rangle, \mathcal{Q}'' : \varphi, \theta'_l, \theta'_u) & \longrightarrow_{SSMT} & (pr_1, \varphi_1) & , \\ (M', \text{Я}_{d_x} x \in \mathcal{D}'_x \odot \mathcal{Q}'' : \varphi_1, \theta''_l, \theta''_u) & \longrightarrow_{SSMT} & (pr_2, \varphi_2) & . \end{array}$$

Then, $pr = p_v \cdot pr_1 + pr_2$. Due to Definition 6.2,

$$\begin{array}{llll} (M', \text{Я}_{d_x} x \in \mathcal{D}_x \odot \mathcal{Q}'' : \varphi, \theta_l, \theta_u) & \succ_{\text{SSMT}} & (M' \odot \langle x = v \rangle, \mathcal{Q}'' : \varphi, \theta'_l, \theta'_u) & , \\ (M', \text{Я}_{d_x} x \in \mathcal{D}_x \odot \mathcal{Q}'' : \varphi, \theta_l, \theta_u) & \succ_{\text{SSMT}} & (M', \text{Я}_{d_x} x \in \mathcal{D}'_x \odot \mathcal{Q}'' : \varphi_1, \theta''_l, \theta''_u) & . \end{array}$$

From induction hypothesis and since $\varphi \Leftrightarrow \varphi_1$, we then conclude the following. If $\tilde{p} = 0$ then

$$\begin{array}{llll} Pr(\mathcal{Q}'' : (\varphi \wedge M' \odot \langle x = v \rangle)) & \geq & pr_1 & , \\ Pr(\text{Я}_{d_x} x \in \mathcal{D}'_x \odot \mathcal{Q}'' : (\varphi \wedge M')) & \geq & pr_2 & , \end{array}$$

and if $\tilde{p} = 1$ then

$$\begin{array}{llll} Pr(\mathcal{Q}'' : (\varphi \wedge M' \odot \langle x = v \rangle)) & \leq & pr_1 & , \\ Pr(\text{Я}_{d_x} x \in \mathcal{D}'_x \odot \mathcal{Q}'' : (\varphi \wedge M')) & \leq & pr_2 & . \end{array}$$

In conformity with Definition 4.5,

$$\begin{array}{rl} & Pr(\text{Я}_{d_x} x \in \mathcal{D}_x \odot \mathcal{Q}'' : (\varphi \wedge M')) \\ = & p_v \cdot Pr(\mathcal{Q}'' : (\varphi \wedge M' \odot \langle x = v \rangle)) + Pr(\text{Я}_{d_x} x \in \mathcal{D}'_x \odot \mathcal{Q}'' : (\varphi \wedge M')) \end{array}$$

where $Pr(Qy \in \mathcal{D}_y \odot \mathcal{Q} : \psi) := 0$ whenever $\mathcal{D}_y = \emptyset$. For item 1, i.e. $\tilde{p} = 0$, it immediately follows that $Pr(\text{Я}_{d_x} x \in \mathcal{D}_x \odot \mathcal{Q}'' : (\varphi \wedge M')) \geq p_v \cdot pr_1 + pr_2 = pr$ as $p_v > 0$. Similarly for item 2, i.e. $\tilde{p} = 1$, $Pr(\text{Я}_{d_x} x \in \mathcal{D}_x \odot \mathcal{Q}'' : (\varphi \wedge M')) \leq p_v \cdot pr_1 + pr_2 = pr$ as $p_v > 0$.

For rule SSMT.4, we use the same assumptions as for SSMT.2 above whenever applicable. We have that $pr = p_v \cdot pr_1 + p_{remain} \cdot \tilde{p}$. Concerning item 1, i.e. $\tilde{p} = 0$, we conclude that $Pr(\text{Я}_{d_x} x \in \mathcal{D}_x \odot \mathcal{Q}'' : (\varphi \wedge M')) \geq p_v \cdot pr_1 = p_v \cdot pr_1 + p_{remain} \cdot 0 = pr$. For item 2, i.e. $\tilde{p} = 1$, observe that $Pr(\text{Я}_{d_x} x \in \mathcal{D}'_x \odot \mathcal{Q}'' : (\varphi \wedge M')) \leq p_{remain}$. Therefore, $Pr(\text{Я}_{d_x} x \in \mathcal{D}_x \odot \mathcal{Q}'' : (\varphi \wedge M')) \leq p_v \cdot pr_1 + p_{remain} \cdot 1 = pr$.

For rule SSMT.10, the same argument as in the base case applies. That is, $\varphi \wedge M'$ is unsatisfiable due to Remark 6.1. Hence, $Pr(\mathcal{Q}' : (\varphi \wedge M')) = 0 = pr$ which immediately shows items 1 and 2. Hence, the theorem follows. □

One might wonder why above Theorem 6.2 as well as its proof do not take the thresholds θ_l and θ_u into account. In particular, the issue of determining the thresholds for the recursive calls within the premises of the corresponding rules remained untouched. In fact, the returned probability results are always either underapproximations (if $\tilde{p} = 0$) or overapproximations (if $\tilde{p} = 1$), independently of the actual thresholds. That is, *thresholding* is "only" exploited as a heuristic approach to skip investigation of subproblems whenever their solutions cannot lead to "better" probability results with respect to θ_l and θ_u. More precisely, if the intermediate *lower* probability bound has already exceeded the *upper* threshold then skipping all remaining branches is reasonable and corresponds to the DPLL-SSAT case in this respect. However, if the current *upper* probability bound is greater than the *upper* threshold then it is not clear why omitting the remaining subproblem and thereby increasing the upper probability bound in the greatest possible way makes any sense at all. The motivation for the latter treatment complies with the actual idea of thresholding, namely as the intermediate upper probability bound already exceeds the upper threshold, so does the upper probability bound when additionally solving the remaining subproblem. With regard to the pragmatic perspective that approximate solutions are considered as "good enough" solutions, the above thresholding rule turns into a reasonable approach. A similar argument can be given if the intermediate *lower* probability bound is too small in the sense that the maximum possible probability of all remaining branches does not suffice to reach the *lower* threshold. Though the lower probability bound could be made tighter, i.e. greater, when solving the remaining subproblem, the lower threshold will be never reached. As we see in Section 6.7, thresholding is a very powerful pruning technique that may lead to tremendous performance gains, most often by multiple orders of magnitude.

Though being an obvious consequence of Theorem 6.2, Corollary 6.3 relates the probability result of the SSMT layer to the lower and the upper threshold.

Corollary 6.3
For each sequence

$$(M, \mathcal{Q} : \varphi, \theta_l, \theta_u) \quad \longrightarrow^*_{SSMT} \quad (pr, \varphi')$$

of SSMT rule applications with fixed $\tilde{p} \in \{0, 1\}$, it holds that

1. *if $\tilde{p} = 0$ and $pr > \theta_u$ then $Pr(\mathcal{Q} : (\varphi \wedge M)) > \theta_u$ and*
2. *if $\tilde{p} = 1$ and $pr < \theta_l$ then $Pr(\mathcal{Q} : (\varphi \wedge M)) < \theta_l$.*

Due to the approximative nature of the SiSAT algorithm stemming from the internal use of the incomplete SMT solver iSAT, the reverse directions of Corollary 6.3 do not hold in general. The approximation feature of the SSMT layer was introduced on account of approximate solutions considered by rule SSMT.9. Whenever SSMT.9 was not applied to solve some SSMT problem, i.e. each SMT subformula could be classified as satisfiable or unsatisfiable, then we can state a much stronger result that corresponds to the DPLL-SSAT case, confer Subsection 6.2.1. We have to admit that above assumption is rarely satisfied for very complex non-linear arithmetic problems involving transcendental functions. Recall that such problems are undecidable in general. However, avoiding rule SSMT.9 is always feasible when considering *decidable* theories $\mathcal{T}$ like linear arithmetic and when using an appropriate decision procedure for $\mathcal{T}$ as the underlying SMT solver instead of iSAT. The next Theorem 6.3 now strengthens the statement of Theorem 6.2 under the assumption that rule SSMT.9 is never executed.

Theorem 6.3 (Soundness of the SSMT algorithm avoiding SSMT.9)
For each sequence

$$(M, \mathcal{Q} : \varphi, \theta_l, \theta_u) \quad \longrightarrow^*_{SSMT} \quad (pr, \varphi')$$

of SSMT rule applications avoiding rule SSMT.9 *with (potentially variable)* $\tilde{p} \in \{0, 1\}$*, it holds that*

1. *if* $pr > \theta_u$ *then* $Pr(\mathcal{Q} : (\varphi \wedge M)) > \theta_u$*,*
2. *if* $pr < \theta_l$ *then* $Pr(\mathcal{Q} : (\varphi \wedge M)) < \theta_l$*, and*
3. *if* $\theta_l \leq pr \leq \theta_u$ *then* $Pr(\mathcal{Q} : (\varphi \wedge M)) = pr$*.*

Proof. The proof is very similar to the one of Theorem 6.2. We again employ Noetherian induction using the well-founded strict partial order $\succ_{\text{SSMT}}$, and restrict the proof obligation to step

$$(M', \mathcal{Q}' : \varphi, \theta_l, \theta_u) \quad \longrightarrow_{SSMT} \quad (pr, \varphi') \enspace .$$

In the base case, i.e. state $(M', \mathcal{Q}' : \varphi, \theta_l, \theta_u)$ is minimal with respect to $\succ_{\text{SSMT}}$ and, thus, only rules SSMT.5, SSMT.7, SSMT.8, and SSMT.10 could be applied, we observe that

$$Pr(\mathcal{Q}' : (\varphi \wedge M')) = pr$$

which immediately shows above items 1, 2, and 3.

In the induction step, state $(M', \mathcal{Q}' : \varphi, \theta_l, \theta_u)$ is non-minimal with respect to $\succ_{\text{SSMT}}$, and we assume that the statement of the theorem is true for all "smaller" states S, i.e. for which $(M', \mathcal{Q}' : \varphi, \theta_l, \theta_u) \succ_{\text{SSMT}} S$ holds. Then, only rules SSMT.1, SSMT.2, SSMT.3, SSMT.4, and SSMT.10 could be applied. We may again conclude that for above rules, it holds that $\varphi \Leftrightarrow \varphi'$ and $\varphi \Leftrightarrow \varphi''$, and that the condition in Remark 6.1 is satisfied for the SMT solver states $(M \odot \langle x = v \rangle, \varphi)$ and (M, φ').

For rule SSMT.1, let be $\mathcal{Q}' = \exists x \in \mathcal{D}_x \odot \mathcal{Q}''$ and

$$\begin{array}{lll} (M' \odot \langle x = v \rangle, \mathcal{Q}'' : \varphi, \theta_l, \theta_u) & \longrightarrow_{SSMT} & (pr_1, \varphi_1) \quad , \\ (M', \exists x \in \mathcal{D}'_x \odot \mathcal{Q}'' : \varphi_1, \theta'_l, \theta_u) & \longrightarrow_{SSMT} & (pr_2, \varphi_2) \quad . \end{array}$$

Then, $pr = \max(pr_1, pr_2)$. Due to Definition 6.2,

$$\begin{array}{lll} (M', \exists x \in \mathcal{D}_x \odot \mathcal{Q}'' : \varphi, \theta_l, \theta_u) & \succ_{\text{SSMT}} & (M' \odot \langle x = v \rangle, \mathcal{Q}'' : \varphi, \theta_l, \theta_u) \quad , \\ (M', \exists x \in \mathcal{D}_x \odot \mathcal{Q}'' : \varphi, \theta_l, \theta_u) & \succ_{\text{SSMT}} & (M', \exists x \in \mathcal{D}'_x \odot \mathcal{Q}'' : \varphi_1, \theta'_l, \theta_u) \quad . \end{array}$$

Thus, induction hypothesis applies to both states above. Since $pr_1 \leq \theta_u$, it follows from induction hypothesis that $Pr(\mathcal{Q}'' : (\varphi \wedge M' \odot \langle x = v \rangle)) \leq \theta_u$. If $pr > \theta_u$ then $pr_2 > \theta_u$. Then, $Pr(\exists x \in \mathcal{D}'_x \odot \mathcal{Q}'' : (\varphi \wedge M')) > \theta_u$ by induction hypothesis and due to $\varphi \Leftrightarrow \varphi_1$, and thus $Pr(\mathcal{Q}' : (\varphi \wedge M')) > \theta_u$ that shows item 1. If $pr < \theta_l$ then $pr_1 < \theta_l$ and $pr_2 < \theta_l$. Immediately by induction hypothesis, $Pr(\mathcal{Q}' : (\varphi \wedge M')) < \theta_l$ that proves item 2. Let now be $\theta_l \leq pr \leq \theta_u$. Then, $pr_1 \leq \theta_u$ and $pr_2 \leq \theta_u$. If $pr_1 \geq pr_2$ then $pr = pr_1$ and $pr_1 \geq \theta_l$. Thus, $pr_2 \leq \theta'_l = \max(pr_1, \theta_l) = pr_1$. By induction hypothesis and due to

$\varphi \Leftrightarrow \varphi_1$, $Pr(\mathcal{Q}'' : (\varphi \wedge M' \odot \langle x = v \rangle)) = pr_1$ and $Pr(\exists x \in \mathcal{D}'_x \odot \mathcal{Q}'' : (\varphi \wedge M')) \leq \theta'_l = pr_1$. As a consequence, $Pr(\mathcal{Q}' : (\varphi \wedge M')) = pr_1 = pr$. If $pr_1 < pr_2$ then $pr = pr_2$ and $pr_2 \geq \theta_l$. Consequently, $pr_2 \geq \theta'_l = \max(pr_1, \theta_l)$. Induction hypothesis as well as the fact that $\varphi \Leftrightarrow \varphi_1$ then give $Pr(\exists x \in \mathcal{D}'_x \odot \mathcal{Q}'' : (\varphi \wedge M')) = pr_2$. As $pr_1 \leq \theta'_l \leq \theta_u$ and $\theta_l \leq \theta'_l$, we have that $Pr(\mathcal{Q}'' : (\varphi \wedge M' \odot \langle x = v \rangle)) \leq \theta'_l \leq pr_2$. Instantaneously, $Pr(\mathcal{Q}' : (\varphi \wedge M')) = pr_2 = pr$. Hence, item 3 follows.

For rule SSMT.3, we use the same assumptions as for SSMT.1 above whenever applicable. We have that $pr = \max(pr_1, \tilde{p})$. Observe that $pr_1 > \theta_u$ or $pr_1 = 1$ holds. First, if $pr_1 > \theta_u$ then $pr > \theta_u$. By induction hypothesis, $Pr(\mathcal{Q}'' : (\varphi \wedge M' \odot \langle x = v \rangle)) > \theta_u$ from which $Pr(\mathcal{Q}' : (\varphi \wedge M')) > \theta_u$ follows. The latter shows items 1, 2, and 3. Second, if $pr_1 = 1$ then $pr = pr_1 = 1$ since $\tilde{p} \in \{0, 1\}$. If $pr > \theta_u$ then $Pr(\mathcal{Q}' : (\varphi \wedge M')) > \theta_u$ since $pr_1 > \theta_u$ and, thus, $Pr(\mathcal{Q}'' : (\varphi \wedge M' \odot \langle x = v \rangle)) > \theta_u$ by induction hypothesis. The latter proves item 1. If $pr < \theta_l$ then $\theta_l > 1$ and item 2 holds trivially. If $\theta_l \leq pr \leq \theta_u$ then $Pr(\mathcal{Q}' : (\varphi \wedge M')) = pr_1 = 1 = pr$ since $\theta_l \leq pr_1 \leq \theta_u$ and, thus, $Pr(\mathcal{Q}'' : (\varphi \wedge M' \odot \langle x = v \rangle)) = pr_1 = 1$ by induction hypothesis. This establishes item 3.

For rule SSMT.2, let be $\mathcal{Q}' = \rotatebox[origin=c]{180}{R}_{d_x} x \in \mathcal{D}_x \odot \mathcal{Q}''$ and

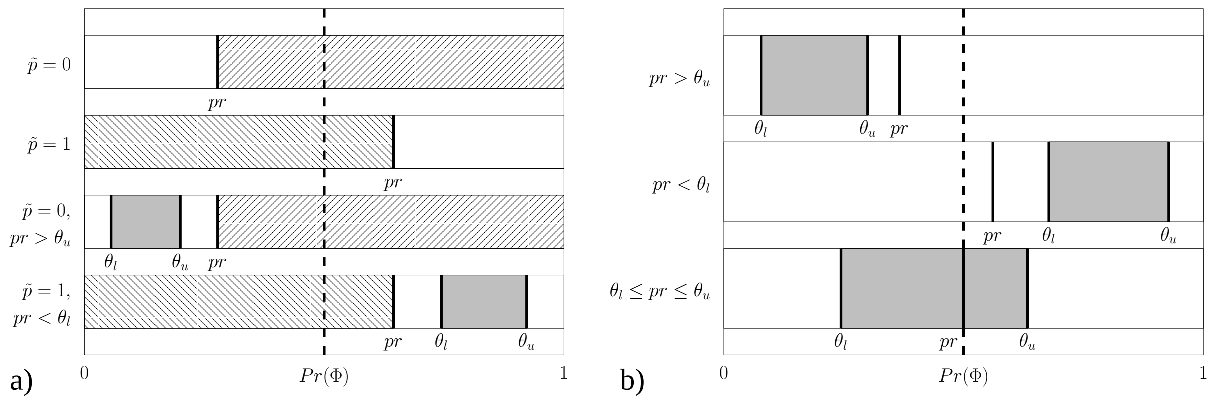
$$\begin{array}{lll} (M' \odot \langle x = v \rangle, \mathcal{Q}'' : \varphi, \theta'_l, \theta'_u) & \longrightarrow_{SSMT} & (pr_1, \varphi_1) \quad , \\ (M', \rotatebox[origin=c]{180}{R}_{d_x} x \in \mathcal{D}'_x \odot \mathcal{Q}'' : \varphi_1, \theta''_l, \theta''_u) & \longrightarrow_{SSMT} & (pr_2, \varphi_2) \quad . \end{array}$$

Then, $pr = p_v \cdot pr_1 + pr_2$. Due to Definition 6.2,

$$\begin{array}{lll} (M', \rotatebox[origin=c]{180}{R}_{d_x} x \in \mathcal{D}_x \odot \mathcal{Q}'' : \varphi, \theta_l, \theta_u) & \succ_{\mathrm{SSMT}} & (M' \odot \langle x = v \rangle, \mathcal{Q}'' : \varphi, \theta'_l, \theta'_u) \quad , \\ (M', \rotatebox[origin=c]{180}{R}_{d_x} x \in \mathcal{D}_x \odot \mathcal{Q}'' : \varphi, \theta_l, \theta_u) & \succ_{\mathrm{SSMT}} & (M', \rotatebox[origin=c]{180}{R}_{d_x} x \in \mathcal{D}'_x \odot \mathcal{Q}'' : \varphi_1, \theta''_l, \theta''_u) \quad . \end{array}$$

Thus, induction hypothesis applies to both states above. Due to application condition and the fact that $p_v > 0$, we conclude that $pr_1 \geq (\theta_l - p_{remain})/p_v = \theta'_l$ and $pr_1 \leq \theta_u/p_v = \theta'_u$. By induction hypothesis, $Pr(\mathcal{Q}'' : (\varphi \wedge M' \odot \langle x = v \rangle)) = pr_1$. If $pr > \theta_u$ then $pr_2 > \theta_u - p_v \cdot pr_1 = \theta''_u$. From induction hypothesis and from $\varphi \Leftrightarrow \varphi_1$, it follows that $Pr(\rotatebox[origin=c]{180}{R}_{d_x} x \in \mathcal{D}'_x \odot \mathcal{Q}'' : (\varphi \wedge M')) > \theta''_u$. In conformity with Definition 4.5, $Pr(\mathcal{Q}' : (\varphi \wedge M')) > p_v \cdot pr_1 + \theta''_u = \theta_u$ which shows item 1. If $pr < \theta_l$ then $pr_2 < \theta_l - p_v \cdot pr_1 = \theta''_l$. Due to induction hypothesis and $\varphi \Leftrightarrow \varphi_1$, $Pr(\rotatebox[origin=c]{180}{R}_{d_x} x \in \mathcal{D}'_x \odot \mathcal{Q}'' : (\varphi \wedge M')) < \theta''_l$. It follows that $Pr(\mathcal{Q}' : (\varphi \wedge M')) < p_v \cdot pr_1 + \theta''_l = \theta_l$ which establishes item 2. If $\theta_l \leq pr \leq \theta_u$ then $\theta''_l = \theta_l - p_v \cdot pr_1 \leq pr_2 \leq \theta_u - p_v \cdot pr_1 = \theta''_u$. Then, $Pr(\rotatebox[origin=c]{180}{R}_{d_x} x \in \mathcal{D}'_x \odot \mathcal{Q}'' : (\varphi \wedge M')) = pr_2$ by induction hypothesis and due to $\varphi \Leftrightarrow \varphi_1$. Item 3 immediately follows from the fact that $Pr(\mathcal{Q}' : (\varphi \wedge M')) = p_v \cdot pr_1 + pr_2 = pr$.

For rule SSMT.4, we use the same assumptions as for SSMT.2 above whenever applicable. We have that $pr = p_v \cdot pr_1 + p_{remain} \cdot \tilde{p}$. Observe that $p_v \cdot pr_1 + p_{remain} < \theta_l$ or $p_v \cdot pr_1 > \theta_u$ is true. In case $p_v \cdot pr_1 + p_{remain} < \theta_l$ holds, we know that $pr_1 < (\theta_l - p_{remain})/p_v = \theta'_l$. Thus, $Pr(\mathcal{Q}'' : (\varphi \wedge M' \odot \langle x = v \rangle)) < \theta'_l$ by induction hypothesis. Due to $\varphi \Leftrightarrow \varphi_1$, we obtain the conservative estimate $Pr(\rotatebox[origin=c]{180}{R}_{d_x} x \in \mathcal{D}'_x \odot \mathcal{Q}'' : (\varphi \wedge M')) \leq p_{remain}$. It follows that $Pr(\mathcal{Q}' : (\varphi \wedge M')) < p_v \cdot \theta'_l + p_{remain} = \theta_l$. As $\tilde{p} \in \{0, 1\}$ and $p_{remain} \geq 0$, we further have that $pr \leq p_v \cdot pr_1 + p_{remain} < \theta_l$. The above facts show items 1, 2, and 3. In case $p_v \cdot pr_1 > \theta_u$ is true, we have that $pr_1 > \theta_u/p_v = \theta'_u$. Induction hypothesis thus yields $Pr(\mathcal{Q}'' : (\varphi \wedge M' \odot \langle x = v \rangle)) > \theta'_u$. Instantaneously, $Pr(\mathcal{Q}' : (\varphi \wedge M')) > p_v \cdot \theta'_u = \theta_u$. As $\tilde{p} \in \{0, 1\}$ and $p_{remain} \geq 0$, it holds that $pr \geq p_v \cdot pr_1 > \theta_u$. The latter facts establish items 1, 2, and 3.



a)
$\tilde{p} = 0$
pr
$\tilde{p} = 1$
pr
$\tilde{p} = 0,$ $pr > \theta_u$
θ_l θ_u pr
$\tilde{p} = 1,$ $pr < \theta_l$
pr θ_l θ_u
0 $Pr(\Phi)$ 1
b)
$pr > \theta_u$
θ_l θ_u pr
$pr < \theta_l$
pr θ_l θ_u
$\theta_l \leq pr \leq \theta_u$
θ_l pr θ_u
0 $Pr(\Phi)$ 1

Figure 6.6: Interpretation of the probability result *pr* of the SSMT algorithm: a) under- and overapproximations according to Theorem 6.2 and Corollary 6.3, and b) strengthened statement when avoiding rule SSMT.9 according to Theorem 6.3.

With regard to rule SSMT.10, we observe that $Pr(\mathcal{Q}' : (\varphi \wedge M')) = pr$ holds which immediately shows items 1, 2, and 3. This completes the proof. □

Observe that Theorem 6.3 does *not* demand a fixed value $\tilde{p} \in \{0, 1\}$. That is, Theorem 6.3 is valid for both possible values of $\tilde{p}$ and, moreover, in all cases the value of $\tilde{p}$ changes during the SSMT proof search. By fixing $\tilde{p}$, we may further apply Theorem 6.2 in order to obtain safe under- or overapproximations of the satisfaction probability. An illustrative summary of Theorem 6.2, Corollary 6.3, and Theorem 6.3 is given by Figure 6.6.

After having proven soundness of the SSMT layer, namely by interpreting each sequence of SSMT rule applications that terminates in a distinguished state (pr, φ'), we now consider *termination* of the SSMT algorithm. For this purpose, Theorem 6.4 shows the following: starting with some (valid) SSMT layer state $(M, \mathcal{Q} : \varphi, \theta_l, \theta_u)$, each sequence of SSMT rule applications eventually leads to a final, i.e. distinguished, state (pr, φ').

Theorem 6.4 (Termination of the SSMT algorithm)
Let $(M, \mathcal{Q} : \varphi, \theta_l, \theta_u)$ be a state of the SSMT layer as specified at the beginning of this subsection. To achieve termination of the underlying SMT solver, we require that the minimum splitting width ε and the progress parameter δ are both strictly greater than zero and that $\varepsilon \geq 2\delta$. Then, each sequence of SSMT rule applications that starts in above state finally reaches a distinguished state (pr, φ'). More formally,

1. *there does not exist an infinite sequence $(M, \mathcal{Q} : \varphi, \theta_l, \theta_u) \longrightarrow_{SSMT} \ldots \longrightarrow_{SSMT} S \longrightarrow_{SSMT} \ldots$ of SSMT rule applications, and*
2. *each final state F, i.e. $(M, \mathcal{Q} : \varphi, \theta_l, \theta_u) \longrightarrow^*_{SSMT} F$ and there does not exist a state F' such that $F \longrightarrow_{SSMT} F'$, is a distinguished state of the shape (pr, φ').*

Proof. The proof of item 1 is by contradiction. To this end, we assume the contrary, namely that an infinite sequence *IS* of SSMT rule applications exists. We first observe that any application of one of the rules SSMT.1 to SSMT.5 and SSMT.7 to SSMT.10 immediately leads to a distinguished state of the shape (pr, φ'), which in turn means that no rule is then applicable to this state (pr, φ'). We thus conclude that each such infinite sequence *IS* must consist of applications of SSMT.6 only. Within the premise of SSMT.6 the SMT layer is employed, actually rules SMT.1 (unit propagation) and SMT.2 (ICP) only. That is, existence of an infinite sequence *IS* implies an infinite sequence IS' of SMT rule applications. However, such an IS' cannot exist due to termination of the SMT layer which is ensured by $\varepsilon > 0$, $\delta > 0$, and $\varepsilon \geq 2\delta$, confer Proposition 6.2 and Remark 6.1. This reveals the contradiction and, hence, item 1 holds.

The proof of item 2 is also by contradiction. That is, we assume the contrary, namely that state F is *not* of the shape (pr, φ'). Then, F must be of the shape $(M', \mathcal{Q}' : \varphi', \theta_l', \theta_u')$. To achieve a contradiction, namely that F is *not* a final state, we now show that there always exists some distinguished state F' of the shape (pr, ψ) such that $F \longrightarrow_{SSMT} F'$. The proof of the latter is by Noetherian induction using the well-founded strict partial order $\succ_{\text{SSMT}}$ from Definition 6.2.

In the base case, we assume that state F is minimal with respect to $\succ_{\text{SSMT}}$, i.e. $\mathcal{Q}' = \varepsilon$ or $\mathcal{D}_x = \emptyset$ for some $(Qx \in \mathcal{D}_x) \in \mathcal{Q}'$. If $\mathcal{Q}' = \varepsilon$ then one of SSMT.7, SSMT.8, and SSMT.9 can always be applied due to Proposition 6.2 or rather Remark 6.1. That is, $F \longrightarrow_{SSMT} F'$ where F' is of the shape (pr, ψ). If $\mathcal{D}_x = \emptyset$ for some $(Qx \in \mathcal{D}_x) \in \mathcal{Q}'$ then SSMT.5 can be clearly executed, leading to state $F' = (0, \varphi')$.

In the induction step, state F is non-minimal with respect to $\succ_{\text{SSMT}}$, i.e. $\mathcal{Q}' \neq \varepsilon$ and $\mathcal{D}_x \neq \emptyset$ for all $(Qx \in \mathcal{D}_x) \in \mathcal{Q}'$. As induction hypothesis, we assume that for all states S with $F \succ_{\text{SSMT}} S$ there is some distinguished state (pr', ψ') such that $S \longrightarrow_{SSMT} (pr', \psi')$. Let be $\mathcal{Q}' = Qx \in \mathcal{D}_x \odot \mathcal{Q}''$ with $Q \in \{\exists, \text{Я}_{d_x}\}$ as well as

$$\begin{aligned} F_1 &= (M' \odot \langle x = v \rangle, \mathcal{Q}'' : \varphi', \theta_{l,1}, \theta_{u,1}) \quad , \\ F_2 &= (M', Qx \in \mathcal{D}_x' \odot \mathcal{Q}'' : \varphi'', \theta_{l,2}, \theta_{u,2}) \quad . \end{aligned}$$

Due to Definition 6.2, $F \succ_{\text{SSMT}} F_1$ and $F \succ_{\text{SSMT}} F_2$. Thus, induction hypothesis applies to states F_1 and F_2. Owing to induction hypothesis, if $Q = \exists$ then one of rules SSMT.1 and SSMT.3 is applicable, and if $Q = \text{Я}_{d_x}$ then one of rules SSMT.2 and SSMT.4 can be executed. In any case, $F \longrightarrow_{SSMT} F'$ where F' is of the shape (pr, ψ). This completes the proof of item 2 and the theorem follows. □

With regard to an implementation of the SSMT layer using machine data types, it must be ensured that Proposition 6.2 or rather Remark 6.1 hold for the underlying SMT layer. Concerning this matter, the reader is referred to the explanatory note following Proposition 6.2.

6.4.3 Example of the SSMT algorithm

To conclude this section, we finally present a small example of the SSMT layer. To this end, consider the SSMT formula $\mathcal{Q} : \varphi$ with

$$\mathcal{Q} = \text{Я}_{[-1\to 0.4, 0\to 0.5, 1\to 0.1]} x_1 \in \{-1, 0, 1\}\ \text{Я}_{[8\to 0.7, 9\to 0.3]} x_2 \in \{8, 9\}\ \exists x_3 \in \{-2, -1, 0, 1, 2, 3\}$$

and

$$\begin{aligned} \varphi = &(x_1 \leq -1 \vee x_3 \geq 1) \wedge (x_1 \geq 0 \vee y_2 \leq -10) \wedge (x_2 \leq 8 \vee x_3 \leq 1) \\ &\wedge (x_3 \geq 2 \vee y_2 \geq 5) \wedge (x_3 \geq 2 \vee y_3 \geq -3) \wedge (y_2 = y_1^3) \wedge (y_3 = -y_2) \end{aligned}$$

$$\cfrac{\cfrac{\cfrac{S_4 \longrightarrow_{SSMT} (0,\varphi) \quad , \quad \cfrac{S_6 \longrightarrow_{SSMT} (1,\varphi)}{S_5 \longrightarrow_{SSMT} (1,\varphi)}}{S_3 \longrightarrow_{SSMT} (1,\varphi)} \quad , \quad \cfrac{S_8 \longrightarrow_{SSMT} S_9 \longrightarrow_{SSMT} (0,\varphi)}{S_7 \longrightarrow_{SSMT} (0,\varphi)}}{S_2 \longrightarrow_{SSMT} (0.7,\varphi)}}{S_0 \longrightarrow_{SSMT} S_1 \longrightarrow_{SSMT} (0.45,\varphi)}$$

Figure 6.7: Example of the SSMT layer: structure of SSMT rule applications.

where $y_1 \in [-2, 10]$, $y_2 \in [-15, 33]$, and $y_3 \in [-50, 50]$ are (non-quantified) real-valued variables. Let be given the thresholds $\theta_l = 0.48$ and $\theta_u = 0.72$, and let us aim at approximating the satisfaction probability safely from above, i.e. we fix $\tilde{p} = 1$. According to equation 6.17, the conjunctive system M_{dom}, encoding the domains of all variables $x \in \mathit{Var}(\varphi)$ symbolically, is given by the sequence $\langle x_1 \geq -1, x_1 \leq 1, x_2 \geq 8, x_2 \leq 9, x_3 \geq -2, x_3 \leq 3, y_1 \geq -2, y_1 \leq 10, y_2 \geq -15, y_2 \leq 33, y_3 \geq -50, y_3 \leq 50\rangle$. Then, the initial state of the SSMT layer is

$$S_0 \;=\; (M_{\text{dom}}, \mathcal{Q} : \varphi, 0.48, 0.72) \quad .$$

In what follows, we apply the SSMT layer rules to solve the above SSMT problem. The overall structure of rule applications is depicted in Figure 6.7.

Before we process the quantifier prefix $\mathcal{Q}$ by means of rules SSMT.1, SSMT.2, SSMT.3, and SSMT.4, we can exploit the deduction mechanisms of the underlying SMT solver iSAT using rule SSMT.6. For this purpose, first observe that clauses $(y_2 = y_1^3)$ and $(y_3 = -y_2)$ are initially unit. From bound $y_1 \geq -2$ and equation $y_2 = y_1^3$, we then infer by ICP that $y_2 \geq -8$ must hold. As a consequence, clause $(x_1 \geq 0 \vee y_2 \leq -10)$ becomes unit due to inconsistent constraint $y_2 \leq -10$. The latter allows for propagating unit constraint $x_1 \geq 0$. This implies that clause $(x_1 \leq -1 \vee x_3 \geq 1)$ also becomes unit and that $x_3 \geq 1$ must be satisfied. Observe that more ICP steps are feasible which we neglect, however, for the sake of simplicity. For instance, $y_2 \geq -8$ and $y_3 = -y_2$ entail that $y_3 \leq 8$. Above facts can be summarized by $(M_{\text{dom}}, \varphi) \longrightarrow^*_{SMT} (M_1, \varphi)$ with $M_1 = M_{\text{dom}} \odot \langle y_2 = y_1^3, y_3 = -y_2, y_2 \geq -8, x_1 \geq 0, x_3 \geq 1\rangle$. Above pruning steps permit to reduce the domain of randomized variable x_1 from $\{-1, 0, 1\}$ to $\{0, 1\}$ and the domain of existential variable x_3 from $\{-2, -1, 0, 1, 2, 3\}$ to $\{1, 2, 3\}$. That is,

$$\begin{aligned} \mathcal{Q}_1 &= \mathit{prune}(\mathcal{Q}, \sigma_{M_1}) \\ &= \rotatebox[origin=c]{180}{R}_{[-1 \to 0.4, 0 \to 0.5, 1 \to 0.1]} x_1 \in \{0, 1\} \; \rotatebox[origin=c]{180}{R}_{[8 \to 0.7, 9 \to 0.3]} x_2 \in \{8, 9\} \; \exists x_3 \in \{1, 2, 3\} \quad . \end{aligned}$$

The observations above thus justify to perform step

$$S_0 \longrightarrow_{SSMT} S_1$$

with

$$S_1 \;=\; (M_1, \mathcal{Q}_1 : \varphi, 0.48, 0.72)$$

using rule SSMT.6.

We now perform branching for the leftmost quantified variable in prefix $\mathcal{Q}_1$, i.e. for $\text{Я}_{[-1\to 0.4, 0\to 0.5, 1\to 0.1]} x_1 \in \{0,1\}$. As we are currently unaware of whether rule SSMT.2 or SSMT.4 applies, we first need to choose a value $v \in \{0,1\}$ for variable x_1 and to solve the corresponding SSMT subproblem in order to find out whether thresholding is feasible or not. Let us decide that $v = 0$. The resulting SSMT layer state thus is

$$S_2 \; = \; (M_2, \mathcal{Q}_2 : \varphi, \theta_{l,2}, \theta_{u,2})$$

with $M_2 = M_1 \odot \langle x_1 = 0 \rangle$, $\mathcal{Q}_2 = \text{Я}_{[8\to 0.7, 9\to 0.3]} x_2 \in \{8,9\} \; \exists x_3 \in \{1,2,3\}$, $\theta_{l,2} = (0.48 - 0.1)/0.5 = 0.76$, and $\theta_{u,2} = 0.72/0.5 = 1.44$. With regard to the thresholds $\theta_{l,2}$ and $\theta_{u,2}$, note that $p_v = 0.5$ and $p_{remain} = 0.1$.

We now select value 8 for randomized variable x_2 and consider the SSMT layer state

$$S_3 \; = \; (M_3, \exists x_3 \in \{1,2,3\} : \varphi, \theta_{l,3}, \theta_{u,3})$$

with $M_3 = M_2 \odot \langle x_2 = 8 \rangle$, $\theta_{l,3} = (0.76 - 0.3)/0.7 = {}^{23}\!/\!{}_{35} \approx 0.657$, and $\theta_{u,3} = 1.44/0.7 = {}^{72}\!/\!{}_{35} \approx 2.057$.

Next, existential variable x_3 is branched for by choosing value 1 which leads to state

$$S_4 \; = \; (M_4, \varepsilon : \varphi, {}^{23}\!/\!{}_{35}, {}^{72}\!/\!{}_{35})$$

with $M_4 = M_3 \odot \langle x_3 = 1 \rangle$. As the quantifier prefix is empty, one of the rules SSMT.7, SSMT.8, and SSMT.9 can be executed. From that fact $x_3 = 1$, it follows that clauses $(x_3 \geq 2 \vee y_2 \geq 5)$ and $(x_3 \geq 2 \vee y_3 \geq -3)$ are unit, propagating bounds $y_2 \geq 5$ and $y_3 \geq -3$, respectively. As a consequence, the constraint $(y_3 = -y_2) \in M_4$ becomes inconsistent. This conflict occurred without any decision step in the SMT layer, i.e. rule SMT.3 was not applied and, thus, the special marker symbol | was not added to M_4. Hence,

$$(M_4, \varphi) \; \longrightarrow^*_{SMT} \; (M_4 \odot \langle y_2 \geq 5, y_3 \geq -3 \rangle, \varphi) \; \longrightarrow_{SMT} \; \texttt{unsat}$$

is feasible and application of rule SSMT.7 gives

$$S_4 \; \longrightarrow_{SSMT} \; (0, \varphi) \;\; .$$

As thresholding fails, we have to apply rule SSMT.1 to handle state S_3. The SSMT layer state for the second recursion thus is

$$S_5 \; = \; (M_5, \exists x_3 \in \{2,3\} : \varphi, {}^{23}\!/\!{}_{35}, {}^{72}\!/\!{}_{35})$$

with $M_5 = M_4$.

We now select value 2 for x_3 yielding state

$$S_6 \; = \; (M_6, \varepsilon : \varphi, {}^{23}\!/\!{}_{35}, {}^{72}\!/\!{}_{35})$$

with $M_6 = M_5 \odot \langle x_3 = 2 \rangle$. Observe that, first, satisfiability of M_6 implies satisfiability of φ and, second, that M_6 and, thus, φ are actually satisfiable within the interval assignment σ_{M_6} with $\sigma_{M_6}(x_1) = [0,0], \sigma_{M_6}(x_2) = [8,8], \sigma_{M_6}(x_3) = [2,2], \sigma_{M_6}(y_1) = [-2,10], \sigma_{M_6}(y_2) = [-8,33], \sigma_{M_6}(y_3) = [-50,50]$. A solution of φ, for instance, is the assignment τ with

$\tau(x_1) = 0, \tau(x_2) = 8, \tau(x_3) = 2, \tau(y_1) = 0, \tau(y_2) = 0, \tau(y_3) = 0$. If the SMT solver iSAT is able to detect satisfiability of the quantifier-free subproblem, namely by means of the *strong satisfaction check* as formalized by theory layer rule TS.3, then rule SSMT.8 is applicable, i.e.

$$S_6 \longrightarrow_{SSMT} (1, \varphi) \ .$$

Otherwise, rule SSMT.9 must be feasible which also gives

$$S_6 \longrightarrow_{SSMT} (1, \varphi)$$

as $\tilde{p} = 1$. Concerning the returned SMT formula φ, we assume for simplicity that the SMT layer has not detected any conflict and, thus, has not learnt any conflict clause.

Since the recursive call on S_6 yielded the maximum possible probability 1, rule SSMT.3 is performed for state S_5, i.e.

$$S_5 \longrightarrow_{SSMT} (1, \varphi)$$

as $\tilde{p} = 1$.

Consequently, namely by application of SSMT.1,

$$S_3 \longrightarrow_{SSMT} (1, \varphi) \ .$$

Due to above fact, thresholding fails for state S_2, i.e. $0.7 \cdot 1 + 0.3 = 1 \geq \theta_{l,2} = 0.76$ and $0.7 \cdot 1 \leq \theta_{u,2} = 1.44$. Hence, rule SSMT.2 needs to be executed to handle S_2. The SSMT layer state for the second recursion is

$$S_7 \ = \ (M_7, \mathcal{Q}_7 : \varphi, \theta_{l,7}, \theta_{u,7})$$

with $M_7 = M_2$, $\mathcal{Q}_7 = \text{ꓤ}_{[8 \to 0.7, 9 \to 0.3]} x_2 \in \{9\} \ \exists x_3 \in \{1, 2, 3\}$, $\theta_{l,7} = 0.76 - 0.7 \cdot 1 = 0.06$, and $\theta_{u,7} = 1.44 - 0.7 \cdot 1 = 0.74$. With regard to the thresholds $\theta_{l,7}$ and $\theta_{u,7}$, note that $p_v \cdot pr = 0.7 \cdot 1$.

It just remains one possible value for x_2, namely 9. The corresponding state is

$$S_8 \ = \ (M_8, \exists x_3 \in \{1, 2, 3\} : \varphi, \theta_{l,8}, \theta_{u,8})$$

with $M_8 = M_7 \odot \langle x_2 = 9 \rangle$, $\theta_{l,8} = (0.06 - 0)/0.3 = 0.2$, and $\theta_{u,8} = 0.74/0.3 = 2^7\!/_{15}$. Due to $x_2 = 9$, clause $(x_2 \leq 8 \vee x_3 \leq 1)$ is unit propagating bound $x_3 \leq 1$. Instantaneously, clauses $(x_3 \geq 2 \vee y_2 \geq 5)$ and $(x_3 \geq 2 \vee y_3 \geq -3)$ become unit. Their unit constraints $y_2 \geq 5$ and $y_3 \geq -3$ then cause inconsistency of equation $(y_3 = -y_2) \in M_8$. That is, we can first perform rule SSMT.6:

$$S_8 \longrightarrow_{SSMT} S_9$$

with

$$S_9 \ = \ (M_8 \odot \langle y_2 \geq 5, y_3 \geq -3 \rangle, \exists x_3 \in \{1, 2, 3\} : \varphi, \theta_{l,8}, \theta_{u,8}) \ .$$

Then, application of rule SSMT.10 gives

$$S_9 \longrightarrow_{SSMT} (0, \varphi)$$

since

$$(M_8 \odot \langle y_2 \geq 5, y_3 \geq -3 \rangle, \varphi) \longrightarrow_{SMT} \texttt{unsat} \ .$$

Observe that thresholding is feasible for S_7 since $0.3 \cdot 0 + 0 < \theta_{l,7} = 0.06$. As a consequence, rule SSMT.4 need be performed for S_7, i.e.

$$S_7 \longrightarrow_{SSMT} (0, \varphi) \quad .$$

Having completed the recursions on S_3 and on S_7, we can deduce

$$S_2 \longrightarrow_{SSMT} (0.7, \varphi)$$

by rule SSMT.2.

We have thus found out that the recursion on S_2 gives probability result 0.7. As $p_v = 0.5$ and $p_{remain} = 0.1$, thresholding succeeds for S_1, i.e. $0.5 \cdot 0.7 + 0.1 = 0.45 < \theta_l = 0.48$, meaning that rule SSMT.4 must be applied to handle S_1:

$$S_1 \longrightarrow_{SSMT} (0.45, \varphi) \quad .$$

The latter step completes the SSMT proof search. Since $\tilde{p} = 1$, the SSMT layer has determined an upper bound, namely 0.45, on the maximum satisfaction probability of the given SSMT formula $\mathcal{Q} : \varphi$ within the given domains according to Theorem 6.2, i.e.

$$Pr(\mathcal{Q} : (\varphi \wedge M_{\mathrm{dom}})) \leq 0.45 \quad .$$

As a consequence and as stated in Corollary 6.3, the satisfaction probability is strictly less than the given lower threshold, i.e.

$$Pr(\mathcal{Q} : (\varphi \wedge M_{\mathrm{dom}})) < \theta_l = 0.48 \quad .$$

6.5 Algorithmic enhancements

In the previous section, we have presented the basic SiSAT algorithm addressing non-linear arithmetic SSMT problems. Recall that SiSAT adds an additional layer to the SMT solver iSAT in order to deal with the quantifier prefix, confer Figure 6.4, namely by traversing the Cartesian product of the domains of the quantified variables and by computing the satisfaction probabilities for the individual quantifiers. As indicated by Figure 6.5 a, each complete assignment to the quantified variables induces a quantifier-free SMT subproblem which is solved using the SMT solver iSAT. Due to the fact that the number of such complete assignments is *exponential* in the number of quantified variables, a naive SSMT algorithm traversing the whole quantifier tree is far from scalable. In order to mitigate this issue and to improve performance of SSMT solvers in practice, it is thus of utmost importance to devise powerful algorithmic optimizations that are able to prune the quantifier tree considerably, as illustrated in Figure 6.5 b.

The SSMT algorithm of Section 6.4 already incorporates some such optimization techniques, namely *unit propagation*, *interval constraint propagation*, and *conflict-driven clause learning*, all of them inherited from iSAT, as well as *thresholding* as known from SSAT solvers. In what follows, we elaborate on further algorithmic features to cut off potentially huge parts of the quantifier tree. Most of these features were published in [TF08, TF09, TEF11] by the author of this thesis and his co-authors. We remark that empirical results demonstrating the benefit of the proposed pruning rules are presented in Section 6.7.

6.5.1 Activity-based value branching heuristics

It is well-known that variable and value decision heuristics, i.e. the order in which possible assignments are probed, can improve the performance of SAT as well as of SMT solvers significantly. In the stochastic setting, i.e. in the SSAT and SSMT case, however, the selection of variables for branching is mainly dictated by the quantifier prefix: in general, the leftmost variable must be taken which is in accordance with the semantics of SSAT and of SSMT, confer Definitions 4.2 and 4.5, respectively. As already mentioned in Subsection 6.2.1, this restriction can be relaxed to some extent: instead of taking the leftmost variable, it is admissible to select one variable from the leftmost block of variables bound by the *same* quantifier. The rationale is that $Pr(\mathcal{Q} \odot Qx \in \mathcal{D}_x \odot \mathcal{Q}' : \varphi) = Pr(Qx \in \mathcal{D}_x \odot \mathcal{Q} \odot \mathcal{Q}' : \varphi)$ whenever all quantifiers in subprefix $\mathcal{Q}$ are the same as quantifier Q.[9] With regard to SSAT solvers, several sophisticated heuristics to select the next variable for branching were investigated in [LMP01].

In the SSMT case, the domains of quantified variables may be much larger than in the SSAT framework where only propositional variables are present. This fact motivates to devise powerful *value branching heuristics*. To improve runtime of the SSMT algorithm, we are interested in detecting values for branching such that other pruning rules will apply earlier. With regard to thresholding, it seems reasonable to prefer values for branching which lead to

- *high* satisfaction probabilities aiming at *exceeding the upper threshold* or
- *low* satisfaction probabilities aiming at *missing the lower threshold.*

Being targeted at such value selection methods, our idea is to devise a heuristic measure for estimating the satisfaction probabilities of SSMT subproblems. For that purpose, we introduce an *activity* $act_{v,x}$ as well as a *counter* $cnt_{v,x}$ for each value $v \in \mathcal{D}_x$ of each quantified variable x, where both $act_{v,x}$ and $cnt_{v,x}$ are initialized with 0. Whenever a new probability result pr for a branch "$x = v$", i.e. for the corresponding SSMT subproblem induced by substituting value v for variable x, is determined then probability pr is added to activity $act_{v,x}$ and counter $cnt_{v,x}$ is incremented by 1. That is, activity $act_{v,x}$ is the sum of the already computed satisfaction probabilities for branch "$x = v$" while counter $cnt_{v,x}$ keeps track of their number. The heuristic measure $hm(v, x)$ for estimating the satisfaction probability of branch "$x = v$" is then defined as the arithmetic mean of the already computed satisfaction probabilities, i.e. $hm(v,x) = \begin{cases} 0 & \text{if } cnt_{v,x} = 0, \\ act_{v,x}/cnt_{v,x} & \text{if } cnt_{v,x} \neq 0 \end{cases}$.

It remains to discuss how values are detected with the aid of above heuristic measure such that thresholding is more likely to succeed. For existential variables x, the thresholding rule applies if the probability result pr of branch "$x = v$" has exceeded the upper threshold θ_u (or has reached value 1), confer rule SSMT.3. Thus, it is reasonable to select values v with highest measure $hm(v, x)$. For randomized variables x, thresholding is performed if the weighted probability result $p_v \cdot pr$ is greater than the upper threshold θ_u or if the probability mass p_{remain} of the remaining branches is too small to reach the lower threshold θ_l, more precisely if $p_v \cdot pr + p_{remain} < \theta_l$, confer rule SSMT.4. Thus, we may

[9]We remark that the probability distributions of randomized quantifiers need not be the same.

pursue two strategies for selecting values of randomized variables. The first strategy aims at exceeding upper thresholds and thus takes values v for which the weighted measure $p_v \cdot hm(v, x)$ is maximal. Aiming at missing lower thresholds, the second strategy selects values v for which the term $p_v \cdot hm(v, x) + p_{remain}$ is minimal.

We finally remark that a very similar version of the activity-based value branching heuristics aiming at exceeding upper thresholds was published in [TF09, TEF11] by the author of this thesis together with his co-authors.

6.5.2 Purification

The idea of *purification* as a mechanism to prune the quantifier tree known from the DPLL-SSAT algorithm, confer Subsection 6.2.1, is adapted to the SSMT setting as described in [TF09, TEF11]. Let be given some SSMT formula $\mathcal{Q} : \varphi$ with matrix φ being in CF as well as some interval assignment σ to the variables in $Var(\varphi)$. We say that formula φ is *monotonic* or *antitonic* in a (quantified) variable $x \in \varphi$ with respect to σ if and only if it holds that whenever some point $\tau \in \sigma$ satisfies φ, i.e. τ satisfies all clauses $cl \in \varphi$, then so does each point $\tau' \in \sigma$ with $\tau'(y) = \tau(y)$ for $y \neq x$ and $\tau'(x) > \tau(x)$ or $\tau'(x) < \tau(x)$, respectively. Intuitively, each solution $\tau \in \sigma$ of formula φ does not depend on the value of x in the following sense: the assignment τ' that arises from τ by replacing value $\tau(x)$ by any greater or any smaller value $v \in \sigma(x)$ if φ is monotonic or antitonic in x, respectively, is also a solution of φ.

The above observation can be exploited to prune the quantifier tree. If x is an *existential* variable and if φ is monotonic or antitonic in x then we may reduce the current (finite) domain $\mathcal{D}_x$ of x to the singleton $\{\max(\mathcal{D}_x)\}$ or $\{\min(\mathcal{D}_x)\}$, respectively. Akin to the DPLL-SSAT case, purification is in general impossible for randomized variables, since all branches give some contribution.

Detecting monotonicity or antitonicity is hard as soon as arithmetic constraints are involved. The current routine to recognize whether some formula φ is monotonic or antitonic in some variable x works as follows. First of all, we can clearly neglect all clauses $cl \in \varphi$ which could be detected to be satisfied by each point in σ, i.e. for each $\tau \in \sigma : \tau \models cl$, since all these satisfied clauses cl are trivially monotonic and antitonic in x with respect to σ. If in all remaining clauses $cl \in \varphi$ all constraints $c \in cl$ which contain x and which are *not* inconsistent under σ are *simple inequality bounds* sharing the *same polarity* then φ is monotonic or antitonic. More precisely, if $\forall cl \in \varphi\ \forall c \in cl$ with $x \in Var(c)$ and $\neg(\sigma \sharp c)$ it holds that c is of the shape $x \sim k$ with k being a rational constant and with $\sim \in \{>, \geq\}$ or with $\sim \in \{<, \leq\}$ then φ is monotonic or antitonic in x with respect to σ, respectively.

We remark that this approach is well-motivated in the context of SSMT-based probabilistic bounded reachability analysis of (concurrent) probabilistic hybrid automata. Employing the reduction scheme to SSMT introduced in Section 5.3, quantified variables occur only in *simple bounds* in the matrix of the SSMT formula $PBMC_{\mathcal{S},Target}(k)$. More precisely, reduction steps 6 and 7 introduce predicates of the shape $(tr^i_j = tr) \Rightarrow g(tr)$ and $(tr^i_j = tr \wedge pc^{tr}_j = pc) \Rightarrow asgn(tr, pc)$, respectively, with tr^i_j, pc^{tr}_j being quantified variables and tr, pc being values. The latter predicates are then rewritten to $(tr^i_j < tr \vee tr^i_j > tr \vee g(tr))$ and $(tr^i_j < tr \vee tr^i_j > tr \vee pc^{tr}_j < pc \vee pc^{tr}_j > pc \vee asgn(tr, pc))$

by means of the generalized Tseitin transformation, confer Subsection 4.3.1, to obtain a matrix in CF. As a consequence, quantified variables occur only in *simple inequality bounds* in the matrix of the SSMT formula to be solved.

For an example of purification, consider the formula

$$\begin{array}{rcl}\varphi & = & (x \le 5 \ \vee \ z = \sin(y)) \ \wedge \ (x = -y \ \vee \ x \ge 3 \ \vee \ b \le 0) \\ & & \wedge \ (b \ge 1 \ \vee \ z \le -31) \ \wedge \ (x = z^2 \ \vee \ x \ge -17 \ \vee \ z \le 54)\end{array}$$

and the interval assignment σ with $\sigma(x) = [-30, -4]$, $\sigma(y) = [6, 20]$, $\sigma(z) = [-50, 100]$, and $\sigma(b) = [0, 1]$. The first clause can be neglected since it is satisfied by each point in σ as constraint $x \le 5$ is satisfied by each value $x \in \sigma(x) = [-30, -4]$. Our current routine is not able to detect whether formula φ is monotonic or antitonic in one of the variables x and y since constraint $x = -y$ is not inconsistent under σ and is not a simple inequality bound. The same holds for variable b as the remaining clauses contain the bounds $b \le 0$ and $b \ge 1$ of different polarity. However, φ is antitonic in variable z with respect to σ: constraint $x = z^2$ is inconsistent under σ as z^2 ranges in $[0, 100^2]$ and the remaining constraints $z \le -31$ and $z \le 54$ are simple inequality bounds sharing the same polarity. The purification rule is thus able to prune the domain of z from $[-50, 100]$ to $[-50, -50]$ provided that z is an existential (or non-quantified) variable.

6.5.3 Exploiting desired accuracy of probability result

Akin to the idea of thresholding, an engineer might not always be interested in the exact probability of satisfaction but just in a result of some *accuracy*. Recall that thresholding (in the strong sense of Theorem 6.3) aims at computing the exact satisfaction probability pr whenever pr lies within the interval specified by the lower and the upper threshold. In all other cases, i.e. if pr is outside this interval, only a witness value is required to decide whether pr is below the lower or above the upper threshold, confer Figure 6.6 b.

Though sharing the same motivation, the algorithmic enhancement of *exploiting some desired accuracy* $\alpha \ge 0$ *of the probability result* is different to thresholding. As a result of the SSMT algorithm, an interval $[lb, ub]$ is returned which

- encloses the exact satisfaction probability pr, i.e. $pr \in [lb, ub]$, and
- is of width at most α, i.e. $ub - lb \le \alpha$.

It is important to remark that due to the approximative nature of SiSAT, which is caused by the incompleteness of the underlying SMT tool iSAT, it is in general not possible to enclose the *exact* satisfaction probability by an interval of some given maximum width α. The latter can only be guaranteed if Theorem 6.3 applies (or in the context of SSAT). Instead of that, such returned intervals $[lb, ub]$ enclose under- or overapproximated results as being specified by Theorem 6.2.

Algorithmically, we make use of the following simple observations. Let $\Phi = Qx \in \mathcal{D}_x\, \mathcal{Q} : \varphi$ be an SSMT formula. Moreover, let be $\Phi_1 = \mathcal{Q} : \varphi[v/x]$ with $v \in \mathcal{D}_x$ as well as $\Phi_2 = Qx \in \mathcal{D}'_x\, \mathcal{Q} : \varphi$ with $\mathcal{D}'_x = \mathcal{D}_x \setminus \{v\}$. Intuitively, Φ_1 is the resulting SSMT subformula for branch "$x = v$" and Φ_2 is the SSMT problem for all remaining branches. For each α, it then holds that

$$Pr(\Phi) \in [lb, ub] \ \text{ with } \ ub - lb \le \alpha$$

if and only if there exist some lb_1, lb_2, ub_1, and ub_2 such that

$$Pr(\Phi_1) \in [lb_1, ub_1] \quad \text{and} \quad Pr(\Phi_2) \in [lb_2, ub_2]$$

with

- $lb = \max(lb_1, lb_2)$, $ub = \max(ub_1, ub_2)$, $ub_1 - lb_1 \leq \alpha$, and $ub_2 - lb_2 \leq \alpha$ if $Q = \exists$, and
- $lb = d_x(v) \cdot lb_1 + lb_2$, $ub = d_x(v) \cdot ub_1 + ub_2$, $ub_1 - lb_1 \leq \alpha_1 / d_x(v)$, and $ub_2 - lb_2 \leq \alpha_2$ for some α_1, α_2 with $\alpha = \alpha_1 + \alpha_2$ if $Q = \text{ᴚ}_{d_x}$.

We now elaborate on how these facts can be exploited during SSMT proof search. For some SSMT formula Φ as above and some accuracy α, the accuracies for the subproblems Φ_1 and Φ_2 are determined recursively as follows. If x is an existential variable, i.e. if $Q = \exists$, then the accuracies for Φ_1 and Φ_2 simply coincide with α, as suggested above. In the randomized case, i.e. if $Q = \text{ᴚ}_{d_x}$, determination of these accuracies is not that obvious. In conformity with above facts, we need to decompose accuracy α into α_1 and α_2 to obtain valid accuracies for the subproblems Φ_1 and Φ_2. This clearly gives rise to several heuristics ranging from the one extreme where the full accuracy α is occupied by α_1, i.e. $\alpha_1 = \alpha, \alpha_2 = 0$, to the other extreme where α_2 takes full accuracy, i.e. $\alpha_1 = 0, \alpha_2 = \alpha$. In the experiments that are presented in Section 6.7, we employ a "fair" distribution of accuracy α. More precisely, $\alpha_1 = \alpha / |\mathcal{D}_x|$ and $\alpha_2 = \alpha - \alpha_1$. As indicated above, α_1 cannot be used directly as the accuracy for subproblem Φ_1 but must be "normalized" by means of dividing α_1 by the probability $d_x(v)$ associated with value v. That is, the resulting accuracy is given by $\alpha_1 / d_x(v)$. A small optimization in distributing the accuracy can be devised for the randomized case, i.e. if $Q = \text{ᴚ}_{d_x}$, by making use of the "saved" accuracy after having solved the first subproblem Φ_1. More precisely, if $ub_1 - lb_1$ is strictly less than $\alpha_1 / d_x(v)$ then accuracy $\alpha_1 / d_x(v)$ was not fully utilized and some accuracy $\alpha_s = \alpha_1 - d_x(v) \cdot (ub_1 - lb_1)$ is left over. This saved accuracy α_s can thus be added to accuracy α_2 for subproblem Φ_2, i.e. $\alpha_2' = \alpha_2 + \alpha_s$ may be used as the accuracy for Φ_2. The latter optimization is also employed in the experiments of Section 6.7.

After having described the recursive computation of the accuracies for the SSMT subproblems, it remains to explain when *accuracy-based pruning* applies: if accuracy α for some SSMT problem Φ is at least 1 then we may skip solving of Φ and return the interval $[lb, ub]$ with $lb = 0$ and $ub = 1$ immediately. Note that the latter is in compliance with the problem specification above, i.e. if $\alpha \geq 1$ then $Pr(\Phi) \in [lb, ub] = [0, 1]$ and $ub - lb = 1 \leq \alpha$.

By the algorithmic enhancement of accuracy-based pruning, the SSMT algorithm must now be able to deal with probability intervals instead of probability values. This fact necessitates to slightly modify the calculation of the probabilities for the individual quantifiers. Obtaining probability intervals in the base cases has been described above. Note that whenever the exact probability pr is returned then the corresponding interval is the point interval $[pr, pr]$. Otherwise, i.e. if two probability intervals $[lb_1, ub_1]$ and $[lb_2, ub_2]$ for subproblems Φ_1 and Φ_2 are given, respectively, we apply usual interval arithmetic, confer [Moo66, Moo79, Moo80] as well as Subsections 6.3.1 and 6.3.2, to compute the probability interval $[lb, ub]$ for Φ, as already indicated above. More precisely, if $Q = \exists$

$$\Phi = \text{ꓤ}_{[0\to 0.85, 1\to 0.15]} x \in \{0,1\}\ \exists y \in \{0,1\} : \big((x = 0 \vee y = 0) \wedge (x = 1 \vee y = 1)\big)$$

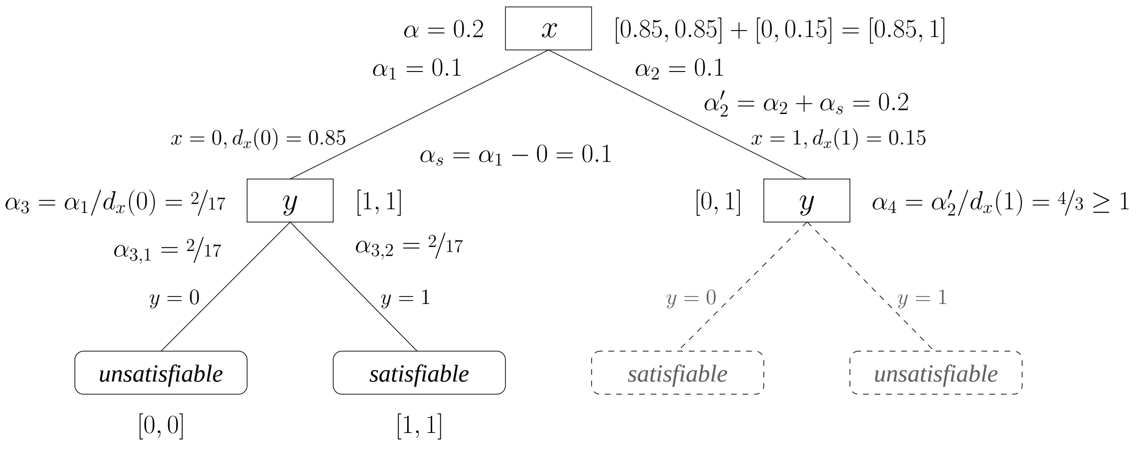


Figure 6.8: Example of pruning the search tree based on accuracy $\alpha = 0.2$ when solving SSMT formula Φ.

then $[lb, ub] = [\max(lb_1, lb_2), \max(ub_1, ub_2)]$ and if $Q = \text{ꓤ}_{d_x}$ then $[lb, ub] = [d_x(v) \cdot lb_1 + lb_2, d_x(v) \cdot ub_1 + ub_2]$.

A simple example of accuracy-based pruning is presented in Figure 6.8. The given accuracy $\alpha = 0.2$ is fairly distributed among both branches for randomized variable x, i.e. $\alpha_1 = 0.1$ for branch "$x = 0$" as well as $\alpha_2 = 0.1$ for branch "$x = 1$". The accuracy for the left subproblem then is $\alpha_3 = \alpha_1/d_x(0) = {}^{2}/_{17}$. As $\alpha_3 < 1$, accuracy-based pruning does not apply. The accuracies for the branches of existential variable y then coincide with α_3. Branch "$y = 0$" yields probability interval $[0, 0]$ while branch "$y = 1$" gives $[1, 1]$ such that the result for branch "$x = 0$" is the interval $[1, 1]$ of width 0. That is, the result for branch "$x = 0$" is exact meaning that *no* accuracy was utilized at all. As a consequence, we have saved full accuracy α_1, i.e. $\alpha_s = \alpha_1$. The latter allows to increase the accuracy for branch "$x = 1$" from $\alpha_2 = 0.1$ to $\alpha'_2 = \alpha_2 + \alpha_s = 0.2$. As the "normalized" accuracy for branch "$x = 1$" is at least 1, i.e. $\alpha_4 = \alpha'_2/d_x(1) = {}^{4}/_{3} \geq 1$, accuracy-based pruning succeeds permitting to skip investigation of the whole subproblem. The corresponding probability interval is $[0, 1]$. Taking into account the probabilities 0.85 and 0.15 of setting x to 0 and to 1, respectively, we achieve the weighted intervals $[0.85, 0.85]$ and $[0, 0.15]$. The sum of these intervals then yields the final probability interval $[0.85, 1]$ which contains the exact probability of satisfaction $Pr(\Phi) = 1$ and which is of width $0.15 \leq \alpha = 0.2$.

We finally elaborate on the combination of thresholding and accuracy-based pruning. Let $[lb, ub]$ be the (intermediate) probability interval of some subproblem. Then, thresholding may only be applied if the lower bound lb has exceeded the upper threshold θ_u or if the upper bound ub is too small in the sense that the maximum possible probability of all remaining branches does not suffice to reach the lower threshold θ_l.

$$\Phi = \exists x \in \{0,1,2\}\ \text{ꓤ}_{[0\to 0.1, 1\to 0.05, 2\to 0.7, 3\to 0.15]} y \in \{0,1,2,3\}\ \text{ꓤ}_{[0\to 0.25, 1\to 0.25, 2\to 0.25, 3\to 0.25]} z \in \{0,1,2,3\} :$$
$$\big((x \geq 0 \vee y \leq 0 \vee z \leq 2)\ \wedge\ (x \leq 1 \vee z \geq 1)\ \wedge\ (y \geq 1 \vee h \geq 0)\ \wedge\ (z \geq 3 \vee h < 0)\ \wedge\ (h = \sin(a))\big)$$

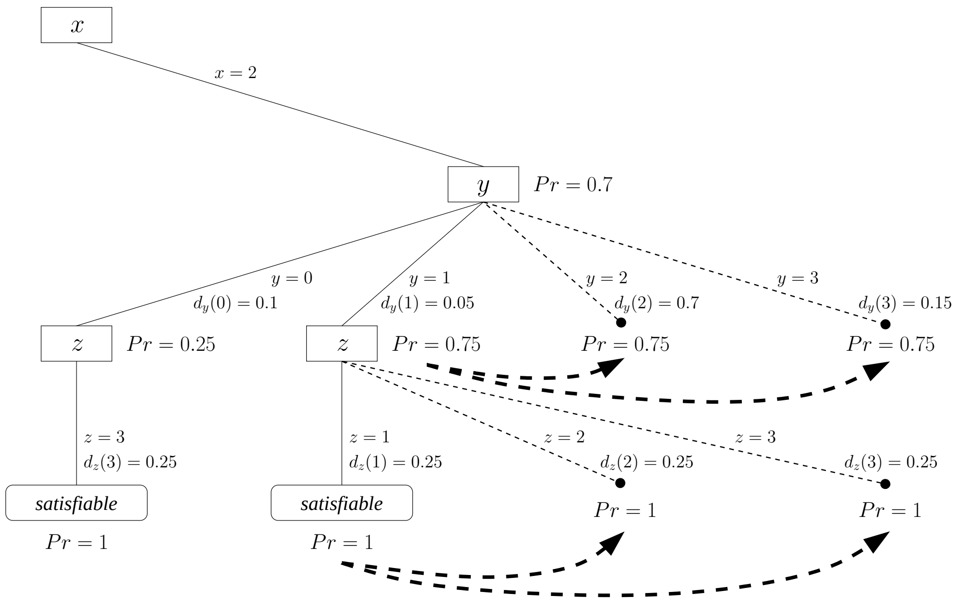


Figure 6.9: Example of solution-directed backjumping when solving SSMT formula Φ.

6.5.4 Solution-directed backjumping

The concept of *solution-directed backjumping* (SDB) has been originally introduced in the context of solving quantified Boolean formulae (QBFs) in order to improve performance of QBF solvers [GNT03], and then has been adapted to the SSAT framework by Majercik [Maj04]. In this subsection, we elaborate on how SDB can be integrated into the SiSAT algorithm addressing SSMT problems. We remark that the content of this subsection has appeared in slightly different form in the conference paper [TF08].

Intuitively, the overall idea of solution-directed backjumping is the following. Upon having found an (approximate) solution, SiSAT analyzes which quantified variables have no impact on the (approximate) solution and saves probing their alternative values by directly assigning the satisfaction probability of the current subproblem to all remaining subproblems.

In order to motivate this technique, we start the presentation of SDB with a simple example that is illustrated in Figure 6.9. Let be given the SSMT formula $\Phi = \mathcal{Q} : \varphi$ with

$$\begin{aligned}\mathcal{Q} \;=\;& \exists x \in \{0,1,2\}\\ & \text{ꓤ}_{[0\to 0.1, 1\to 0.05, 2\to 0.7, 3\to 0.15]} y \in \{0,1,2,3\}\\ & \text{ꓤ}_{[0\to 0.25, 1\to 0.25, 2\to 0.25, 3\to 0.25]} z \in \{0,1,2,3\}\end{aligned}$$

and

$$\begin{aligned}\varphi \;=\;& \quad (x \geq 0 \vee y \leq 0 \vee z \leq 2)\ \wedge\ (x \leq 1 \vee z \geq 1)\ \wedge\ (y \geq 1 \vee h \geq 0)\\ & \wedge\ \ (z \geq 3 \vee h < 0)\ \wedge\ (h = \sin(a))\end{aligned}$$

where $h \in [-1, 1]$ and $a \in [-100, 100]$ are real-valued variables. Assume that we first investigate value 2 for existential variable x. Due to clause $(x \leq 1 \vee z \geq 1)$, we may then prune the domain of randomized variable z from $\{0, 1, 2, 3\}$ to $\{1, 2, 3\}$ by unit propagation. For randomized variable y, branch "$y = 0$" is solved first. Due to clause $(y \geq 1 \vee h \geq 0)$ it follows that $h \geq 0$. Then, $z \geq 3$ can be deduced due to $(z \geq 3 \vee h < 0)$. That is, it remains only one possible value for randomized variable y, namely 3. The corresponding quantifier-free SMT subproblem is satisfiable as the conjunction of $h \geq 0$ and $h = \sin(a)$ can be satisfied, for instance by $h := \sin(2) \approx 0.9093$ and $a := 2$.

Next, we explore branch "$y = 1$" and then "$z = 1$". The corresponding quantifier-free SMT subproblem is satisfiable under the current assignment σ to the quantified variables with $\sigma(x) = 2$, $\sigma(y) = 1$, and $\sigma(z) = 1$, thus yielding satisfaction probability 1 for this base case. In order to apply SDB, we first search for an explanation of satisfiability of φ under σ. To this end, we select from each clause (at least) one constraint, namely

1. $x \geq 0$ from the first clause,
2. $z \geq 1$ from the second,
3. $y \geq 1$ from the third,
4. $h < 0$ from the fourth, and
5. $h = \sin(a)$ from the fifth.

Observe that the conjunction of above constraints is satisfiable under σ since constraints of items 1, 2, and 3 are immediately satisfied under σ and constraints of items 4 and 5 are satisfied, for instance, by assignments $h := \sin(4) \approx -0.7568$ and $a := 4$. The latter fact entails satisfiability of φ under σ and, therefore, the above selection of constraints serves as an explanation that φ is satisfiable under σ.

Now observe that the conjunction of above constraints remains satisfiable when replacing value $\sigma(z) = 1$ by any other value from the current domain $\{1, 2, 3\}$ of randomized variable z, since the only constraint talking about z, namely $z \geq 1$ of item 2, is satisfied by any value for z that is at least 1. In other words, the current value $\sigma(z) = 1$ has *no* impact on satisfiability of φ under σ with respect to the current domain of z. As a consequence, φ is satisfiable under each σ' with $\sigma'(x) = \sigma(x)$, $\sigma'(y) = \sigma(y)$, and $\sigma'(z) \in \{1, 2, 3\}$. That is, each of the remaining branches for z leads to a satisfiable SMT subproblem and thus to satisfaction probability 1. This allows us to skip these branches and to compute the satisfaction probability for z immediately, namely 0.75.

A similar argument can now be applied for randomized variable y. However, as y is followed by z within quantifier prefix $\mathcal{Q}$, satisfiability of φ under the corresponding assignment to the quantified variables must be ensured for all remaining values of y *and* of z. That is, to apply SDB also for y, the conjunction of above constraints must be satisfiable under each σ'' with $\sigma''(x) = \sigma'(x)$, $\sigma''(y) \in \{1, 2, 3\}$, and $\sigma''(z) \in \{1, 2, 3\}$, which actually is the case. All remaining branches for y hence yield the same satisfaction probability as the current one, namely 0.75. We again skip investigation of these branches and determine the probability of satisfaction for y directly, namely 0.7.

SDB is however not applicable for variable x. Though all remaining values for x, namely 0 and 1, satisfy constraint $x \geq 0$ of item 1, the domains of variables y and z have been increased during backtracking to their initial domains, i.e. to $\{0, 1, 2, 3\}$. This implies that constraints $y \geq 1$ (item 3) and $z \geq 1$ (item 2) are no longer satisfied by each of the remaining values of y and z. Indeed, both branches "$x = 0$" and "$x = 1$" yield a higher probability of satisfaction, namely 0.925.

In what follows, we formally introduce the algorithmic enhancement of SDB. To this end, we first define a reason for an (approximate) solution akin to a reason for a conflict, confer Subsection 6.3.2 and in particular rule SMT.4. More precisely, let φ be some SMT formula in CF and M be some conjunctive constraint system, the latter representing, among others, the initial variable domains as well as the current instantiation of the quantified variables. Then, a conjunctive system r of constraints from φ, i.e. $r \subseteq \{c \in cl : cl \in \varphi\}$ using set notation, is called a *reason for an (approximate) solution of* $\varphi \wedge M$ if and only if

- r contains from each clause at least one constraint, i.e. $\forall cl \in \varphi : cl \cap r \neq \emptyset$, and
- there exists an (approximate) solution of $r \wedge M$.

From the above definition, it immediately follows that existence of such a reason r actually implies existence of an (approximate) solution of $\varphi \wedge M$. We denote the set of all reasons for an (approximate) solution of $\varphi \wedge M$ by $sat\_reasons(\varphi, M)$.

We formalize *solution-directed backjumping* by the following rule SSMT.11. Instantiation of quantified variable x with value $v \in \mathcal{D}_x$ yields probability pr and SMT formula φ' potentially containing learnt conflict clauses. Note that thresholding has been "disabled" for the latter call by taking 0 as the lower and 1 as the upper threshold. This approach was devised to avoid any interference with thresholding. It is important to remark that thresholding is still possible, namely by selecting rule SSMT.3 or SSMT.4 instead of SSMT.11 if applicable.

The idea of SDB is now formalized as follows. Let us assume that there exists a reason r for an (approximate) solution of $\varphi \wedge M \odot M'$ for all M', i.e. for all possible assignments to the quantified variables in prefix $Qx \in \mathcal{D}_x \odot \mathcal{Q}$. That is, for each such assignment there is an (approximate) solution of $\varphi \wedge M \odot M'$ which in turn means that the probability result for each of the remaining branches "$x = v'$" is the same as for branch "$x = v$", namely pr. This gives us an argument for skipping investigation of the remaining subproblems and immediately returning the combined probability result pr'. One might wonder whether pr can be less than 1. This can actually be the case, namely, first, if r witnesses an *approximate* solution and if parameter $\tilde{p}$ was fixed to 0 (aiming at underapproximated probability results) and, second, if some domains in prefix $\mathcal{Q}$ were already pruned by rule SSMT.6 as in the motivating example.

$$\text{(SSMT.11)}\quad \frac{\begin{array}{c} v \in \mathcal{D}_x,\ (M \odot \langle x = v \rangle, \mathcal{Q} : \varphi, 0, 1) \longrightarrow^*_{SSMT} (pr, \varphi'), \\ \mathcal{Q} = Qx_1 \in \mathcal{D}_{x_1} \odot \ldots \odot Qx_m \in \mathcal{D}_{x_m}, \\ \exists r \subseteq \{c \in cl : cl \in \varphi\} : \forall v' \in \mathcal{D}_x \forall v_1 \in \mathcal{D}_{x_1} \ldots \forall v_m \in \mathcal{D}_{x_m} : \\ r \in sat\_reasons(\varphi, M \odot M') \text{ with } M' = \langle x = v', x_1 = v_1, \ldots, x_m = v_m \rangle, \\ pr' = \begin{cases} pr & ; \quad Q = \exists, \\ \sum_{v' \in \mathcal{D}_x} d_x(v') \cdot pr & ; \quad Q = \text{Я}_{d_x} \end{cases} \end{array}}{(M, Qx \in \mathcal{D}_x \odot \mathcal{Q} : \varphi, \theta_l, \theta_u) \longrightarrow_{SSMT} (pr', \varphi')}$$

While soundness of rule SSMT.11 was explained above, it is however not clear how its application condition can be checked in an efficient way. We employ the following routine to check applicability of rule SSMT.11.

In the base case, i.e. $\mathcal{Q} = \varepsilon$ and existence of a solution or of an approximate solution was certified by rule SSMT.8 or by rule SSMT.9, respectively, a reason r for this (approximate) solution of $\varphi \wedge M \odot \langle x = v \rangle$ is determined by selecting from each clause of φ (at least) one constraint. In case of SSMT.8, we simply take as the reason r the constraint system M_3 used in the premise of the underlying satisfiability checking rule SMT.6. Recall that the strong satisfaction check has succeeded on $M_1 \odot M_3$ where M_1 comprises the initial interval domains of all variables and the current instantiations of the quantified variables (as well as some deductions obtained from the latter), and M_3 contains from each clause at least one constraint. In case of SSMT.9, we take from each clause one constraint that is *not* inconsistent under interval assignment $\sigma_{M'}$. Success of the latter process is ensured by item 2c of Proposition 6.2 or rather Remark 6.1.

We then check the application condition of SSMT.11 successively for increasing $\mathcal{Q}$ until some check fails. To this end, we use reason r which has been determined above. That is, we have to prove whether r is actually a reason for an (approximate) solution of $\varphi \wedge M \odot M'$ for *all* M', i.e. for all assignments to the quantified variables $x, x_1, \ldots, x_m$ of which *exponentially* many exist in general. A brute-force approach, probing all these assignments, would clearly neutralize any benefit of SDB. To achieve an efficient check, we proceed as follows. By construction of r in the base case above, there is at least *one* $M'' = \langle x = w, x_1 = w_1, \ldots, x_m = w_m \rangle$ such that r is a reason for an (approximate) solution τ of $\varphi \wedge M \odot M''$. By definition, τ is an (approximate) solution of $r \wedge M \odot M''$. Though we are unaware of this (approximate) solution τ (only existence of τ is ensured), we know however that $\tau(x) = w, \tau(x_1) = w_1, \ldots, \tau(x_m) = w_m$ due to the shape of M''. Let

$$\begin{aligned} r_{\mathit{Var}(M'')} &= \{c \in r : \mathit{Var}(c) \cap \mathit{Var}(M'') \neq \emptyset\} \text{ and} \\ M_{\mathit{Var}(M'')} &= \{c \in M : \mathit{Var}(c) \cap \mathit{Var}(M'') \neq \emptyset\} \end{aligned}$$

be the subsystems of r and of M, respectively, that consist of all constraints involving some variables from $\mathit{Var}(M'') = \{x, x_1, \ldots, x_m\}$. Obviously, each assignment τ' with $\tau'(y) = \tau(y)$ for all $y \notin \mathit{Var}(M'')$ is an (approximate) solution of all constraints not containing variables in $\mathit{Var}(M'')$, i.e. of $r_{\mathit{remain}} \wedge M_{\mathit{remain}}$ with $r_{\mathit{remain}} = r \setminus r_{\mathit{Var}(M'')}$ and $M_{\mathit{remain}} = M \setminus M_{\mathit{Var}(M'')}$. If we can now show that each such τ' above with $\tau'(x) \in \mathcal{D}_x, \tau'(x_1) \in \mathcal{D}_{x_1}, \ldots, \tau'(x_m) \in \mathcal{D}_{x_m}$ is also an (approximate) solution of $r_{\mathit{Var}(M'')} \wedge M_{\mathit{Var}(M'')}$ then rule SSMT.11 may be executed, since then for each M' an (approximate) solution of $r \wedge M \odot M'$ exists. Observe that each τ' trivially satisfies the corresponding M' by construction.

The latter check is hard as soon as arithmetic constraints are involved. For the sake of efficiency, our routine tests for a stronger condition, namely whether each constraint c in $r_{\mathit{Var}(M'')}$ and in $M_{\mathit{Var}(M'')}$ is

1. a *simple bound* of the shape $z \sim q$ with $\sim \in \{<, \leq, =, \geq, >\}$ and with q being a rational constant, and

2. satisfied by each value $w_z \in \mathcal{D}_z$, i.e. whether $w_z \sim q$ holds for each $w_z \in \mathcal{D}_z$.

Observe that verification of condition 1 is straightforward. Checking condition 2 just calls for comparing the minimum or maximum value of $\mathcal{D}_z$ with constant q if $\sim \in \{<, \leq, \geq, >\}$ and for investigating whether $\mathcal{D}_z = \{q\}$ if $\sim \in \{=\}$.

With regard to soundness, if both of the above conditions are satisfied for all these constraints c then the application condition of rule SSMT.11 is satisfied permitting to apply solution-directed backjumping by means of execution of SSMT.11.

We remark that the above approach, which only succeeds if all these constraints c are simple bounds, is well-motivated in the context of SSMT-based probabilistic bounded reachability analysis of (concurrent) probabilistic hybrid automata. As already explained in Subsection 6.5.2, quantified variables occur only in simple bounds in the matrix of the SSMT formula $PBMC_{\mathcal{S},Target}(k)$, confer Section 5.3. This property is maintained after having transformed above formula into CF.

Any interference with thresholding was excluded on account of setting θ_l to 0 and θ_u to 1, confer rule SSMT.11. With regard to accuracy-based pruning, our routine performing SDB applies only after having found an (approximate) solution by means of rule SSMT.8 or rule SSMT.9 as explained above. That is, the corresponding probability results are always of accuracy 0 such that accuracy-based pruning is not feasible during solution-directed backjumping. In principle, SSMT.11 could permit that the recursive call returns a probability interval (instead of value pr). In order to guarantee that the combined probability interval then meets the desired accuracy, the accuracies for all branches of a randomized variable must be distributed *uniformly*.

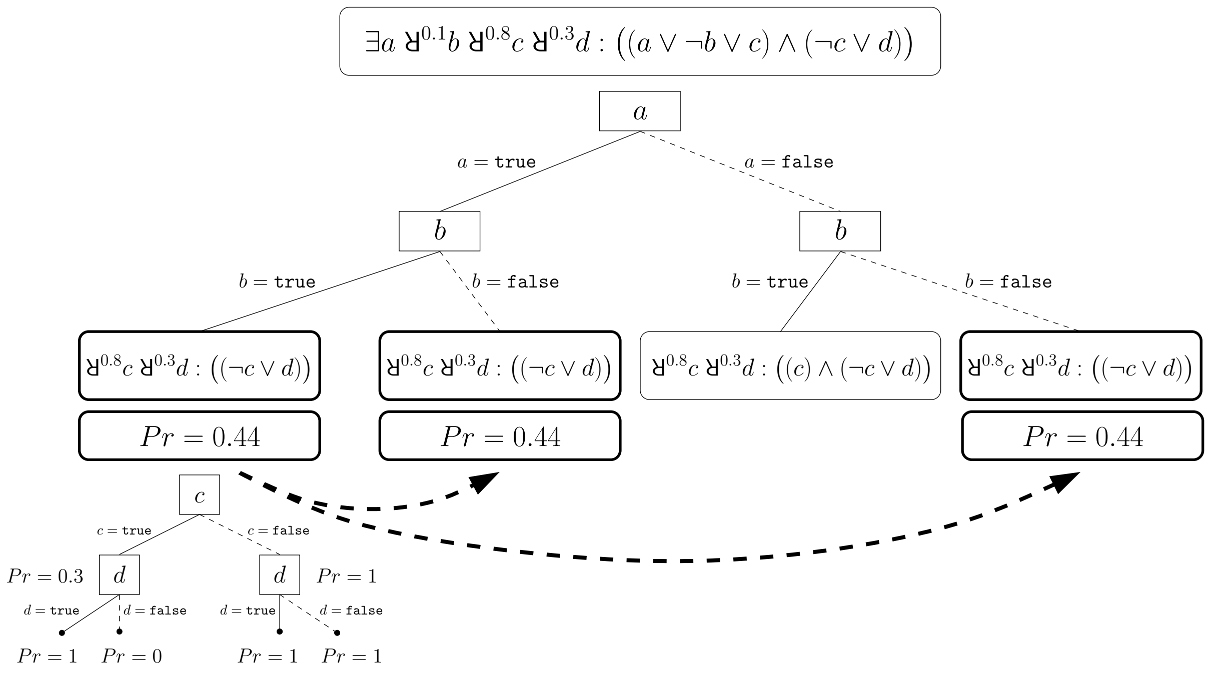
6.5.5 Caching probability results of subproblems

During SSMT proof search, it may happen that the *same* SSMT subformula has to be solved several times. In order to avoid such recomputations, it is a natural approach to *store* and *reuse probability results* of subproblems. This technique also known as *memoization* was already investigated in the context of SSAT by Majercik and Littman: as shown in [ML98b, ML98a], memoization can lead to tremendous performance gains but is also very memory intensive due to the huge number of subproblems to be memoized.

The main issue with respect to this optimization is to detect when subproblems are actually the same. Note that SiSAT as well as DPLL-SSAT visit each partial assignment to the quantified variables at most once. That is, memoization which would be implemented by means of a mapping from such partial assignments to probability results is pointless as these probability results cannot be used in future search. We therefore need to consider a more sophisticated mapping that takes account of the whole quantified formula $\mathcal{Q} : \varphi$ and the current domains of the variables.

In the SSAT case, where each variable can only take one of the truth values `true` or `false`, such a mapping can be simply realized by means of the *cleaning* of the propositional formula $\varphi[\tau(x_1)/x_1] \ldots [\tau(x_i)/x_i]$ in CNF, as defined in Section 2.2, with τ being a partial assignment to the quantified variables, i.e. by means of the formula that arises when removing the constants `true` and `false`. An illustrative example of memoization for SSAT is depicted in Figure 6.10.

In [TF09, TEF11], we have adapted the concept of memoization to the SSMT framework, where arithmetic constraints as well as variables with continuous domains are in-



$\exists a\, \reflectbox{R}^{0.1} b\, \reflectbox{R}^{0.8} c\, \reflectbox{R}^{0.3} d : \big((a \vee \neg b \vee c) \wedge (\neg c \vee d)\big)$
a
$a = \texttt{true}$
$a = \texttt{false}$
b
b
$b = \texttt{true}$
$b = \texttt{false}$
$b = \texttt{true}$
$b = \texttt{false}$
$\reflectbox{R}^{0.8} c\, \reflectbox{R}^{0.3} d : \big((\neg c \vee d)\big)$
$\reflectbox{R}^{0.8} c\, \reflectbox{R}^{0.3} d : \big((\neg c \vee d)\big)$
$\reflectbox{R}^{0.8} c\, \reflectbox{R}^{0.3} d : \big((c) \wedge (\neg c \vee d)\big)$
$\reflectbox{R}^{0.8} c\, \reflectbox{R}^{0.3} d : \big((\neg c \vee d)\big)$
$Pr = 0.44$
$Pr = 0.44$
$Pr = 0.44$
c
$c = \texttt{true}$
$c = \texttt{false}$
$Pr = 0.3$
d
d
$Pr = 1$
$d = \texttt{true}$
$d = \texttt{false}$
$d = \texttt{true}$
$d = \texttt{false}$
$Pr = 1$
$Pr = 0$
$Pr = 1$
$Pr = 1$

Figure 6.10: Example of caching and reusing probability results of subtrees: once subproblem $\reflectbox{R}^{0.8} c\, \reflectbox{R}^{0.3} d : (\neg c \vee d)$ has been solved, its satisfaction probability is cached and then reused whenever encountering the same subproblem again.

volved. The main idea remains the same as in the SSAT case. Intuitively, for some SSMT formula $\mathcal{Q} : \varphi$ and some partial assignment τ to the quantified variables, we determine the remaining SSMT formula $\mathcal{Q}' : \varphi'$ by removing the variables from $\mathcal{Q}$ for which τ is defined and by substituting the values $\tau(x)$ for the corresponding variables x in φ. Then, some constraints in φ' involving variables x have a definite truth value. For instance, $x \geq 3$ is `false` for $\tau(x) = -2$, and $x = y + z$ is `true` for $\tau(x) = -2$, $\tau(y) = 2$, and $\tau(z) = -4$. Observe that equations like $x = y + z$ need not be `true` or `false` for some τ in general since τ is partial. For instance, the truth value of $x = y + z$ is generally undefined if τ is not defined for some of the variables x, y, and z. If we adapt the notion of a cleaning to SMT formulae, then the cleaning of φ' arises by removing all clauses that contain some constraints that are `true` and by removing from all clauses all constraints that are `false`. Note that φ' and its cleaning are semantically equivalent. Assume that we have already determined the probability result *pr* of an SSMT formula $\mathcal{Q}' : \varphi'$ and we encounter some SSMT formula $\mathcal{Q}'' : \varphi''$ such that $\mathcal{Q}'' = \mathcal{Q}'$ and the cleanings of the substitutions φ'' and φ' of φ coincide. Then, $Pr(\mathcal{Q}'' : \varphi'') = Pr(\mathcal{Q}' : \varphi')$ which gives rise to reuse probability result *pr*.

Being a bit more formal, let $(M, \mathcal{Q} : \varphi, \theta_l, \theta_u)$ be some state of the SSMT layer with $\mathcal{Q} = Q_i x_i \in \mathcal{D}_{x_i} \odot \ldots \odot Q_n x_n \in \mathcal{D}_{x_n}$. That is to say, the quantified variables $x_1, \ldots, x_{i-1}$ were already instantiated with some values in M, i.e. $(x_j = v_j) \in M$ for each $j \in \{1, \ldots, i-1\}$. Let

$$\varphi_{M,iqv} \quad = \quad \{cl \in \varphi : \mathit{Var}(cl) \cap \{x_1, \ldots, x_{i-1}\} \neq \emptyset\}$$

be the set of all clauses from φ that contain already instantiated quantified variables, and let $\varphi_{M,r} = \varphi \setminus \varphi_{M,iqv}$ be the set of all other clauses. Furthermore, let

$$\varphi_{M,siqv} \quad \subseteq \quad \{cl \in \varphi_{M,iqv} : \exists c \in cl : \forall \tau \in \sigma_M : \tau \models c\}$$

be a subset of $\varphi_{M,iqv}$ comprising (not necessarily all) clauses cl which involve constraints that are satisfied by each point $\tau \in \sigma_M$, and let be $\varphi_{M,riqv} = \varphi_{M,iqv} \setminus \varphi_{M,siqv}$. It follows that $M \Rightarrow \varphi_{M,siqv}$ is valid and thus

(6.18) $$\varphi_{M,iqv} \wedge M \quad \equiv \quad \varphi_{M,siqv} \wedge \varphi_{M,riqv} \wedge M \quad \equiv \quad \varphi_{M,riqv} \wedge M \quad .$$

Since interval arithmetic is in general not able to decide whether $\forall \tau \in \sigma_M : \tau \models c$ holds, we only ensure soundness, i.e. each clause in $\varphi_{M,siqv}$ is indeed satisfied, but *not* completeness in the sense that $\varphi_{M,siqv}$ contains all satisfied clauses. From all remaining clauses, i.e. clauses which were not proven to be satisfied, we remove all inconsistent constraints and obtain

(6.19) $$\varphi'_{M,riqv} \quad = \quad \{cl \setminus cl' : cl \in \varphi_{M,riqv},\ cl' = \{c \in cl : \sigma_M \sharp c\}\} \quad .$$

It holds that $(\varphi_{M,riqv} \wedge M) \Rightarrow \varphi'_{M,riqv}$ and that $\varphi'_{M,riqv} \Rightarrow \varphi_{M,riqv}$. As a consequence,

(6.20) $$\varphi_{M,riqv} \wedge M \quad \equiv \quad \varphi'_{M,riqv} \wedge \varphi_{M,riqv} \wedge M \quad \equiv \quad \varphi'_{M,riqv} \wedge M \quad .$$

Equivalences 6.18 and 6.20 give

$$\varphi_{M,iqv} \wedge M \quad \equiv \quad \varphi'_{M,riqv} \wedge M$$

from which we deduce that

(6.21) $$\varphi \wedge M \quad \equiv \quad \varphi_{M,r} \wedge \varphi_{M,iqv} \wedge M \quad \equiv \quad \varphi_{M,r} \wedge \varphi'_{M,riqv} \wedge M$$

is true. Let $M_{iqv} = \{c \in M : \mathit{Var}(c) \cap \{x_1, \ldots, x_{i-1}\} \neq \emptyset\}$ be the conjunctive system containing all constraints from M which involve already instantiated quantified variables. Analogously, $M_r = \{c \in M : \mathit{Var}(c) \cap \{x_1, \ldots, x_{i-1}\} = \emptyset\}$ consists of all constraints from M not containing already instantiated quantified variables. Obviously, if the conjunctive system M_{iqv} is satisfied by each point in σ_M, i.e.

(6.22) $$\forall \tau \in \sigma_M : \tau \models M_{iqv} \quad ,$$

then

$$M[v_1, \ldots, v_{i-1}/x_1, \ldots, x_{i-1}], \quad \equiv \quad M_r$$

where $\sigma_M(x_j) = [v_j, v_j]$ with $j \in \{1, \ldots, i-1\}$. If moreover the SMT formula $\varphi'_{M,riqv}$ does not talk about already instantiated variables, i.e.

(6.23) $$\mathit{Var}(\varphi'_{M,riqv}) \cap \{x_1, \ldots, x_{i-1}\} = \emptyset \quad ,$$

then it follows that

(6.24) $$\left(\varphi'_{M,riqv} \wedge M\right) [v_1, \ldots, v_{i-1}/x_1, \ldots, x_{i-1}] \quad \equiv \quad \varphi'_{M,riqv} \wedge M_r$$

is true. Let us summarize the findings above: if conditions 6.22 and 6.23 are satisfied then

$$(\varphi \wedge M)\,[v_1,\ldots,v_{i-1}/x_1,\ldots,x_{i-1}] \quad\equiv\quad \varphi_{M,r} \wedge \varphi'_{M,riqv} \wedge M_r$$

according to equivalences 6.21 and 6.24 and due to $\mathit{Var}(\varphi_{M,r}) \cap \{x_1,\ldots,x_{i-1}\} = \emptyset$. Instantaneously,

$$\begin{array}{rcl} \mathit{Pr}\,(\mathcal{Q} : (\varphi \wedge M)) & = & \mathit{Pr}\,(\mathcal{Q} : (\varphi \wedge M)\,[v_1,\ldots,v_{i-1}/x_1,\ldots,x_{i-1}]) \\ & = & \mathit{Pr}\left(\mathcal{Q} : \left(\varphi_{M,r} \wedge \varphi'_{M,riqv} \wedge M_r\right)\right) \quad . \end{array}$$

With regard to the first equation, observe that variables $x_1,\ldots,x_{i-1}$ do not occur in $\mathcal{Q}$ and are thus interpreted as innermost existentially quantified. Due to the shape of M, for each solution τ of $\varphi \wedge M$ it holds that $\tau(x_j) = v_j$ for all $j \in \{1,\ldots,i-1\}$. Thus, the satisfaction probability remains the same after substituting values v_j for x_j.

In order to utilize the idea of caching and reusing probability results, we proceed as follows. Let $(M', \mathcal{Q}' : \psi', \theta'_l, \theta'_u)$ and $(M'', \mathcal{Q}'' : \psi'', \theta''_l, \theta''_u)$ be two states of the SSMT layer that occur when solving some given SSMT formula $\mathcal{Q} : \varphi$ (with respect to some thresholds θ_l, θ_u) by means of applying the SSMT layer rules. Recall that $\varphi \equiv \psi' \equiv \psi''$ as only implied conflict clauses were added to φ by the SSMT layer rules. That is, $\mathit{Pr}(\mathcal{Q}' : (\psi' \wedge M')) = \mathit{Pr}(\mathcal{Q}' : (\varphi \wedge M'))$ and $\mathit{Pr}(\mathcal{Q}'' : (\psi'' \wedge M'')) = \mathit{Pr}(\mathcal{Q}'' : (\varphi \wedge M''))$. As a consequence, if $\mathit{Pr}(\mathcal{Q}' : (\varphi \wedge M')) = \mathit{Pr}(\mathcal{Q}'' : (\varphi \wedge M''))$ then $\mathit{Pr}(\mathcal{Q}' : (\psi' \wedge M')) = \mathit{Pr}(\mathcal{Q}'' : (\psi'' \wedge M''))$.

A *sufficient condition* that $\mathit{Pr}(\mathcal{Q}' : (\varphi \wedge M')) = \mathit{Pr}(\mathcal{Q}'' : (\varphi \wedge M''))$ holds is established by the following list of items:

1. condition 6.22 holds for M' and M'', i.e. $\forall \tau \in \sigma_{M'} : \tau \models M'_{iqv}$ and $\forall \tau \in \sigma_{M''} : \tau \models M''_{iqv}$,

2. condition 6.23 holds for $\varphi'_{M',riqv}$ and $\varphi'_{M'',riqv}$, confer equation 6.19, i.e. $\mathit{Var}(\varphi'_{M',riqv}) \cap QV_{M'} = \emptyset$ and $\mathit{Var}(\varphi'_{M'',riqv}) \cap QV_{M''} = \emptyset$ where $QV_{M'}$ and $QV_{M''}$ are the sets of all quantified variable which are instantiated by M' and M'', respectively,

3. the prefixes coincide, i.e. $\mathcal{Q}' = \mathcal{Q}''$,

4. the SMT formulae $\varphi'_{M',riqv}$ and $\varphi'_{M'',riqv}$, confer equation 6.19, are equal with respect to their set representations, i.e. they contain the same clauses while two clauses are the same if and only if they share the same constraints,

5. the conjunctive systems M'_r and M''_r (comprising all constraints from M' and M'', respectively, without already instantiated variables) contain the same equations[10], i.e. $\{e \in M'_r : e \text{ is an equation}\} = \{e \in M''_r : e \text{ is an equation}\}$, and

6. the interval assignments $\sigma_{M'_r}$ and $\sigma_{M''_r}$ are the same, i.e. for each variable $x \in \mathit{Var}(M'_r) \cup \mathit{Var}(M''_r) : \sigma_{M'_r}(x) = \sigma_{M''_r}(x)$.

Above items 1 and 2 ensure that

$$\begin{array}{rcl} \mathit{Pr}\,(\mathcal{Q}' : (\varphi \wedge M')) & = & \mathit{Pr}\left(\mathcal{Q}' : \left(\varphi_{M',r} \wedge \varphi'_{M',riqv} \wedge M'_r\right)\right) \quad \text{and} \\ \mathit{Pr}\,(\mathcal{Q}'' : (\varphi \wedge M'')) & = & \mathit{Pr}\left(\mathcal{Q}'' : \left(\varphi_{M'',r} \wedge \varphi'_{M'',riqv} \wedge M''_r\right)\right) \quad . \end{array}$$

[10]Recall that the term *equation* refers to a constraint involving an arithmetic operator, confer Definition 4.3.

Item 3 entails that $\varphi_{M',r}$ and $\varphi_{M'',r}$ are equal with respect to their set representations since $\varphi_{M',iqv}$ and $\varphi_{M'',iqv}$ are. Thus,

$$\varphi_{M',r} \quad \equiv \quad \varphi_{M'',r} \quad .$$

An obvious consequence of item 4 is that

$$\varphi'_{M',riqv} \quad \equiv \quad \varphi'_{M'',riqv}$$

holds. From items 5 and 6 we deduce that

$$M'_r \quad \equiv \quad M''_r$$

as each solution τ of M'_r satisfies all equations as well as all bounds in M''_r due to item 5 as well as item 6, respectively. Since no other constraints occur in M''_r, τ is also a solution of M''_r. Analogously, the reverse direction holds.

We may thus conclude that

$$Pr\left(\mathcal{Q}' : \left(\varphi_{M',r} \wedge \varphi'_{M',riqv} \wedge M'_r\right)\right) \quad = \quad Pr\left(\mathcal{Q}'' : \left(\varphi_{M'',r} \wedge \varphi'_{M'',riqv} \wedge M''_r\right)\right)$$

is true from which

$$Pr\left(\mathcal{Q}' : (\varphi \wedge M')\right) \quad = \quad Pr\left(\mathcal{Q}'' : (\varphi \wedge M'')\right)$$

follows.

Let *pr* be the probability result for state $(M', \mathcal{Q}' : \psi', \theta'_l, \theta'_u)$, i.e.

$$(M', \mathcal{Q}' : \psi', \theta'_l, \theta'_u) \quad \longrightarrow^*_{SSMT} \quad (pr, \psi) \quad .$$

In order to cache result *pr* for above SSMT layer state, we first check whether conditions 6.22 and 6.23 are satisfied according to above items 1 and 2. If so, we store a mapping from the prefix $\mathcal{Q}'$, the SMT formula $\varphi'_{M',riqv}$, and the conjunctive system M'_r (as per items 3, 4, 5, and 6) as well as the current thresholds θ'_l and θ'_u to probability result *pr*, i.e. we cache the entry

$$\left(\mathcal{Q}', \varphi'_{M',riqv}, M'_r, \theta'_l, \theta'_u\right) \quad \to \quad pr \quad .$$

In case we visit some state $(M'', \mathcal{Q}'' : \psi'', \theta''_l, \theta''_u)$ for which conditions 6.22 and 6.23 hold and for which a cached entry $(\mathcal{Q}', \varphi'_{M',riqv}, M'_r, \theta'_l, \theta'_u) \to pr$ exists such that items 3, 4, 5, and 6 are fulfilled, we may *reuse* probability result *pr* if it is in conformity with the thresholds. More precisely, we may directly assign *pr* to $(M'', \mathcal{Q}'' : \psi'', \theta''_l, \theta''_u)$, i.e. *without* solving the same subproblem again, whenever

1. $pr \in [\theta'_l, \theta'_u]$,
2. $[\theta''_l, \theta''_u] \subseteq [\theta'_l, \theta'_u]$,
3. $pr < \theta'_l \leq \theta''_l$, or
4. $pr > \theta'_u \geq \theta''_u$.

In case of item 1, pr is the exact (or rather under- or overapproximated) probability result. Thus, pr can be reused regardless of thresholds θ_l'' and θ_u''. Item 2 specifies the fact that the cached result is "more precise than required". That is to say, if $pr \in [\theta_l', \theta_u']$ then it is "exact" and if $pr < \theta_l'$ or $pr > \theta_u'$ then clearly $pr < \theta_l''$ or $pr > \theta_u''$, respectively. Items 3 and 4 finally reflect the situations where pr misses and exceeds the cached lower and upper thresholds which are at most and at least as large the current lower and upper thresholds, respectively.

If accuracy-based pruning, as presented in Subsection 6.5.3, is applied then we may only reuse the cached probability interval $[lb, ub]$ if its width meets the accuracy α required for the current subproblem, i.e. $ub - lb \leq \alpha$.

For a simple example of memoization for the SSMT case, consider the SSMT formula $\mathcal{Q} : \varphi$ where

$$\mathcal{Q} \;=\; \exists x_1 \in \{-10, \ldots, 10\} \odot \rotatebox[origin=c]{180}{R}_{d_{x_2}} x_2 \in \{-10, \ldots, 10\} \odot \mathcal{Q}'$$

and

$$\varphi \;=\; (x_1 \geq 2 \vee x_2 \leq 1) \;\wedge\; (x_1 = -x_2 \vee c_1 \vee c_2) \;\wedge\; \psi$$

with $\mathcal{Q}'$ being a quantifier prefix, c_1 and c_2 being constraints not involving variables x_1 and x_2, and ψ being an SMT formula in CF also not comprising x_1 and x_2. Let further be given the SSMT layer state $(M, \mathcal{Q}' : \varphi, 0.1, 0.9)$ with

$$M \;=\; M_{\mathrm{dom}} \odot \langle x_1 = 4, x_2 = 3 \rangle$$

where the conjunctive system M_{dom} encodes the initial domains of all variables symbolically as defined in equation 6.17. We assume that SSMT proof search for above state yields probability result 0.6, i.e.

$$(M, \mathcal{Q}' : \varphi, 0.1, 0.9) \;\longrightarrow^{*}_{SSMT}\; (0.6, \psi') \;\;.$$

Let constraints c_1 and c_2 be neither inconsistent under σ_M nor satisfied by each point in σ_M. Then, we obtain

$$\begin{array}{rcll}
\varphi_{M,iqv} & = & (x_1 \geq 2 \vee x_2 \leq 1) \;\wedge\; (x_1 = -x_2 \vee c_1 \vee c_2) & , \\
\varphi_{M,siqv} & = & (x_1 \geq 2 \vee x_2 \leq 1) & , \\
\varphi_{M,riqv} & = & (x_1 = -x_2 \vee c_1 \vee c_2) & , \\
\varphi'_{M,riqv} & = & (c_1 \vee c_2) &
\end{array}$$

as well as

$$\begin{array}{rcll}
M_{iqv} & = & \langle x_1 \geq -10, x_1 \leq 10, x_2 \geq -10, x_2 \leq 10, x_1 = 4, x_2 = 3 \rangle & , \\
M_r & = & M'_{\mathrm{dom}} &
\end{array}$$

where the bounds $x_1 \geq -10, x_1 \leq 10, x_2 \geq -10, x_2 \leq 10 \in M_{\mathrm{dom}}$ encode the initial domains of x_1 and x_2, and where the conjunctive system M'_{dom} arises from M_{dom} by removing bounds $x_1 \geq -10, x_1 \leq 10, x_2 \geq -10, x_2 \leq 10$. It is not hard to see that conditions 6.22 and 6.23 are satisfied. We may thus cache the entry

$$(M, \mathcal{Q}' : \varphi, 0.1, 0.9) \;\to\; 0.6 \;\;.$$

If we visit, for instance, the SSMT layer state $(M', \mathcal{Q}' : \varphi, \theta_l', \theta_u')$ with

$$M' \quad = \quad M_{\text{dom}} \odot \langle x_1 = -9, x_2 = -1 \rangle$$

in future search then we need not solve the corresponding SSMT problem but we may reuse probability result 0.6 since the sufficient condition for $Pr(\mathcal{Q}' : (\varphi \wedge M)) = Pr(\mathcal{Q}' : (\varphi \wedge M'))$ is satisfied, i.e. above items 1 to 6 hold. However, for state $(M'', \mathcal{Q}' : \varphi, \theta_l', \theta_u')$ with

$$M'' \quad = \quad M_{\text{dom}} \odot \langle x_1 = 1, x_2 = -1 \rangle$$

the cached probability result must not be reused since constraint $x_1 = -x_2$ is satisfied by each point in $\sigma_{M''}$. That is, $\varphi'_{M'',riqv} = \emptyset \neq \varphi'_{M,riqv}$ which violates item 4.

With regard to implementation, it is worth mentioning that a cached entry

$$\left(\mathcal{Q}', \varphi'_{M',riqv}, M_r', \theta_l', \theta_u'\right) \;\to\; pr$$

can be represented in a *more concise* way. For this purpose, we assume that the original matrix φ is known in each SSMT solver state. Note that this requires no extra memory if φ is stored in a global data structure.

At first, we aim at storing only a sufficient subsystem of M_r'. To this end, observe that all equations in M_r' that are satisfied by each point in $\sigma_{M_r'}$ can be neglected since they are redundant. That is, $M_r' \equiv \widehat{M_r'}$ where $\widehat{M_r'}$ arises from M_r' by removing all such redundant equations. We now show that each "non-redundant" equation $e \in M_r'$, i.e. e is not satisfied by each point in $\sigma_{M_r'}$, can be deduced from the SMT formula $\varphi_{M',r}$, the SMT formula $\varphi'_{M',riqv}$, and the interval assignment $\sigma_{M_r'}$. The rationale is as follows.

- If some $e \in M_r'$ was deduced from some clause $cl \in \varphi_{M',r}$ and from some subsystem of M' then e can also be deduced using the strongest bounds of M_r', which are taken into account by $\sigma_{M_r'}$, since all clause in $\varphi_{M',r}$ do not contain already instantiated variables.

- If some $e \in M_r'$ was deduced from some clause $cl \in \varphi_{M',riqv}$ and from some subsystem of M' then e can also be deduced from clause $cl' \in \varphi'_{M',riqv}$ which arises from cl when removing all constraints that are inconsistent under $\sigma_{M'}$. It must be true that such clauses cl' are singletons and thus unit.

- If some $e \in M_r'$ was deduced from some clause $cl \in \varphi_{M',siqv}$ and from some subsystem of M' then e is satisfied by each point in $\sigma_{M'}$. This is due to the facts that all clauses in $\varphi_{M',siqv}$ are satisfied by each point $\sigma_{M'}$ and that e is the unit constraint of cl, i.e. all other constraints in cl are false. As e does not comprise any already instantiated quantified variable, e is also satisfied by each point in $\sigma_{M_r'}$.

We remark that each equation in M_r' must be deduced according to one of the above items since $\varphi = \varphi_{M',r} \wedge \varphi_{M',riqv} \wedge \varphi_{M',siqv}$. Provided that $\varphi'_{M',riqv}$ is given, it thus suffices to store $\sigma_{M_r'}$ (taking account of the strongest bounds of M_r') only instead of the whole system M_r'. That is, from $\varphi_{M',r}$, $\varphi'_{M',riqv}$, and $\sigma_{M_r'}$ we can deduce each equation in $\widehat{M_r'}$, while $\varphi_{M',r}$ can be constructed from $\mathcal{Q}'$ and the original matrix φ.

A second optimization is devised as follows. Due to condition 6.23, all clauses of $\varphi'_{M',riqv}$ can be constructed from $\varphi_{M',riqv}$ and $\sigma_{M'_r}$, namely by removing all constraints involving already instantiated quantified variables as well as all constraints being inconsistent under $\sigma_{M'_r}$. As $\varphi_{M',riqv} \subseteq \varphi$ consists of original clauses only, it suffices to cache *unique identifiers*, like pointers or indices, to all clauses in $\varphi_{M',riqv}$.

6.5.6 Caching solutions

The algorithmic enhancement of *caching solutions*, as published in [TEF11] by the author of this thesis together with his co-authors, is motivated by the application of SSMT to probabilistic bounded model checking of probabilistic hybrid automata. In order to solve the probabilistic bounded reachability problem for some system $\mathcal{S}$ of concurrent probabilistic hybrid automata, i.e. to compute $\mathcal{P}^k_{\mathcal{S},Target}(\imath)$ from Definition 5.3 or rather $P^k_{\mathcal{S},Target}(\imath)$ from Lemma 5.2, we need to take account of anchored system runs r of length at most k reaching the target states. Of course, such runs r may be of much smaller length $k' < k$. This circumstance is considered in the definition of $P^k_{\mathcal{S},Target}(s)$ by simply skipping all suffixes of r, i.e. by returning probability 1 instantaneously, whenever s is a target state, confer Lemma 5.2. However, this handling is not preserved directly after reduction from probabilistic bounded reachability to SSMT, i.e. after having transformed the problem of computing $P^k_{\mathcal{S},Target}(\imath)$ into the problem of computing the maximum satisfaction probability of the SSMT formula $PBMC_{\mathcal{S},Target}(k)$, confer Section 5.3. Recall that $P^k_{\mathcal{S},Target}(\imath) = Pr(PBMC_{\mathcal{S},Target}(k))$ according to Theorem 5.1. As enforced by reduction step 10 in Section 5.3, whenever a target state t is visited in less than k steps then the system remains in t until step depth k is reached by performing self loops. That is, all target states are sinks as illustrated in Figure 6.11 a. The above observation motivates to cache and reuse solutions, which is formalized in the following proposition.

Proposition 6.3 (Soundness of reusing cached assignments)
Let us abbreviate by $CHOICE_{\mathcal{S}}(k_1,k_2)$ the quantifier prefix $\bigodot_{j=k_1}^{k_2} CHOICE_{\mathcal{S}}(j)$. For some SSMT formula $PBMC_{\mathcal{S},Target}(k) = CHOICE_{\mathcal{S}}(1,k) : BMC_{\mathcal{S},Target}(k)$ as constructed in Section 5.3, let τ be an assignment to the quantified variables $x \in Var(CHOICE_{\mathcal{S}}(1,k'))$ with $k' \le k$ and $\tau(x) \in \mathcal{D}_x$ such that

$$Pr\left(CHOICE_{\mathcal{S}}(k'+1,k) : BMC_{\mathcal{S},Target}(k)[\tau(\vec{x})/\vec{x}]\right) = 1$$

where $BMC_{\mathcal{S},Target}(k)[\tau(\vec{x})/\vec{x}]$ arises from $BMC_{\mathcal{S},Target}(k)$ by substituting values $\tau(x)$ for all variables $x \in Var(CHOICE_{\mathcal{S}}(1,k'))$. Then, for each $k'' \ge k$ it holds that

$$Pr\left(CHOICE_{\mathcal{S}}(k'+1,k'') : BMC_{\mathcal{S},Target}(k'')[\tau(\vec{x})/\vec{x}]\right) = 1 \quad .$$

Proof. In order to prove the proposition, it suffices to show that for each assignment τ' to the quantified variables $y \in Var(CHOICE_{\mathcal{S}}(k'+1,k))$ which yields a satisfiable SMT formula, i.e.

$$Pr\left(\varepsilon : BMC_{\mathcal{S},Target}(k)[\tau(\vec{x})/\vec{x}][\tau'(\vec{y})/\vec{y}]\right) = 1 \quad ,$$

it holds that each assignment τ'' to the quantified variables $z \in Var(CHOICE_{\mathcal{S}}(k+1,k''))$ also leads to a satisfiable SMT formula, i.e.

$$Pr\left(\varepsilon : BMC_{\mathcal{S},Target}(k'')[\tau(\vec{x})/\vec{x}][\tau'(\vec{y})/\vec{y}][\tau''(\vec{z})/\vec{z}]\right) = 1 \quad .$$

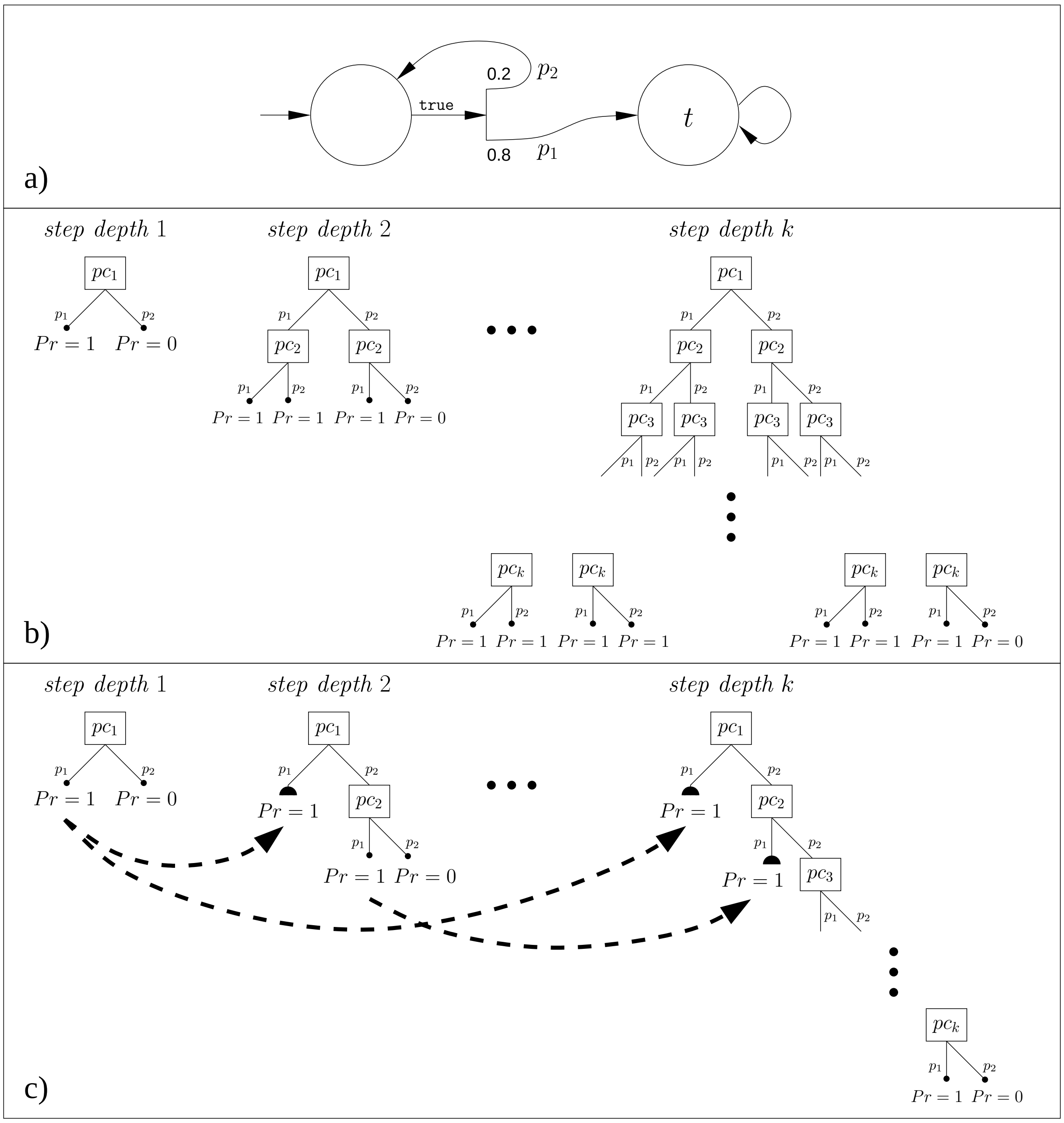


Figure 6.11: Example of caching and reusing solutions: a) a simple probabilistic automaton with t being the target state to be reached, b) illustration of the quantifier prefixes of the corresponding SSMT encodings for step depths 1 to k, and c) exploiting cached assignments leading to subproblems of satisfaction probability 1 when solving SSMT encodings of larger step depths. Note that each randomized variable pc_i encodes the probabilistic choice at step i.

Due to construction of $BMC_{\mathcal{S},Target}(k'')$, confer reduction step 10, the above sufficient condition actually holds since quantified variables are not involved in the predicates $TARGET(j-1)$ and $SELF\_LOOP_{\mathcal{S}}(j-1, j)$. $\square$

Though considering target states as sinks within the SSMT encoding $PBMC_{\mathcal{S},Target}(k)$ is sound due to Proposition 6.3, this fact potentially causes unnecessary effort when solving

SSMT formulae $\mathit{PBMC}_{\mathcal{S},\mathit{Target}}(k'')$ of larger depth $k'' \geq k$. The rationale is that even though an assignment τ to the quantified variables $x \in \mathit{Var}(\mathit{CHOICE}_{\mathcal{S}}(1,k'))$ with $k' \leq k$ already leads to satisfaction probability 1, as in Proposition 6.3, each assignment τ^* to the quantified variables $x^* \in \mathit{Var}(\mathit{CHOICE}_{\mathcal{S}}(k'+1,k''))$ must be explored in worst case, as depicted in Figure 6.11 b. The number of these assignments τ^* clearly is exponential in the number of variables in $\mathit{Var}(\mathit{CHOICE}_{\mathcal{S}}(k'+1,k''))$. With the aid of Proposition 6.3, we may avoid this pointless overhead by *caching* and *reusing* assignments τ that yield satisfaction probability 1 after substitution, as indicated in Figure 6.11 c.

Such assignments τ are stored when solving some SSMT formula $\mathit{PBMC}_{\mathcal{S},\mathit{Target}}(k)$, namely whenever a satisfiable (quantifier-free) SMT subproblem was found. Observe that in such cases, $k' = k$ in Proposition 6.3. With regard to implementation, we use a tree-like data structure to store and access assignments τ efficiently. When solving SSMT problems $\mathit{PBMC}_{\mathcal{S},\mathit{Target}}(k'')$ of larger depth $k'' \geq k$, we directly assign satisfaction probability 1 to the current subproblem if the current assignment to the quantified variables was cached beforehand, which is in accordance with Proposition 6.3.

An optimization of the caching-solutions technique allows to compress stored assignments in size. Assume that we have cached assignments $\tau_1, \ldots, \tau_n$ to the quantified variables $x_1, \ldots, x_m$. Whenever

1. $\tau_1(x_i) = \ldots = \tau_n(x_i)$ for all $i \in \{1, \ldots, m-1\}$ and

2. a) x_m is an existential variable or

   b) x_m is a randomized variable and $\{\tau_1(x_m), \ldots, \tau_n(x_m)\} = \mathcal{D}_{x_m}$

then we may replace the assignments $\tau_1, \ldots, \tau_n$ by the assignment τ to the quantified variables $x_1, \ldots, x_{m-1}$ with $\tau(x_i) = \tau_1(x_i)$ for all $i \in \{1, \ldots, m-1\}$.[11] That is to say, the assignment τ already induces an SSMT subproblem of satisfaction probability 1. Soundness of the latter operation follows directly from the semantics of SSMT, confer Definition 4.5. With regard to the latter, recall that existential quantifiers aim at maximizing the satisfaction probability and randomized quantifiers at computing the weighted sum, and observe that one value in the domain of x_m leads to probability 1 if x_m is existential and that all values do if x_m is randomized. Multiple applications of this optimization lead to very compact representations of the set of assignments found so far, facilitating earlier pruning of the quantifier tree.

Concerning the issue of *approximate* solutions, confer rule SSMT.9, the algorithmic enhancement of caching solutions is in conformity with Theorem 6.2. Finally, it is important to remark that caching and reusing solutions is *only* valid for families of SSMT formulae $Q_1 x_1 \in \mathcal{D}_{x_1} \ldots Q_k x_k \in \mathcal{D}_{x_k} : \varphi(k)$ for which $\mathit{Pr}(Q_{i+1} x_{i+1} \in \mathcal{D}_{x_{i+1}} \ldots Q_k x_k \in \mathcal{D}_{x_k} : \varphi(k)[\tau(x_1)/x_1] \ldots [\tau(x_i)/x_i]) = 1$ implies that $\mathit{Pr}(Q_{i+1} x_{i+1} \in \mathcal{D}_{x_{i+1}} \ldots Q_{k'} x_{k'} \in \mathcal{D}_{x_{k'}} : \varphi(k')[\tau(x_1)/x_1] \ldots [\tau(x_i)/x_i]) = 1$ for all $k' \geq k$. For the family $\mathit{PBMC}_{\mathcal{S},\mathit{Target}}(k)$, this has been shown by Proposition 6.3.

[11] Employing a tree-like data structure, such replacements are realized efficiently by removing all vertices representing $\tau_1(x_m), \ldots, \tau_n(x_m)$.

6.6 SSMT-based probabilistic bounded model checker SiSAT

In this section, we present the SSMT solver SiSAT which is built upon the SMT tool iSAT[12]. The SiSAT tool was first described in [TF08] while its current version was basically explained in [TEF11]. As the development of SiSAT is an ongoing project, this section reflects the current state of affairs. For potentially later stages of development, we refer the reader to the website

`http://sisat.gforge.avacs.org`

on which the SiSAT tool, a user manual, as well as some input files are available.

In Subsection 6.6.1, we first elaborate on the input language of SiSAT, while then Subsection 6.6.2 briefly explains the tool usage and algorithmic core. Finally, an example of modeling a probabilistic hybrid automaton using the SiSAT language is presented in Subsection 6.6.3.

6.6.1 Input language

The command-line tool SiSAT provides two different modes of operation: On the one hand, it can be used as a solver for *single SSMT formulae* and, on the other hand, as a probabilistic bounded model checker for *probabilistic transition systems*. The distinction between both modes is made by two different input file formats which are described below.

Single SSMT formula. A valid input of SiSAT is a single SSMT formula given in the input file format consisting of the following three parts:

1. `DECL`: This part contains declarations of all *non-quantified variables* which are interpreted as innermost existentially quantified. Supported variable types are *integer*, *floating-point*, and *Boolean*[13] encoded by `int`, `float`, and `boole`, respectively. Note that the domain of each integer and floating-point variable has to be bounded by an interval. For instance, `int [-100, 100] a;`, `float [-100, 100] b;`, and `boole c;` declare the integer variable a with interval domain $[-100, 100]$, the floating-point variable b with interval domain $[-100, 100]$, and the Boolean variable c, respectively. Furthermore, you can define *symbolic constants* in this section, for instance, `define v = 5.2;`.

2. `PREFIX`: This part specifies the *prefix* of quantified variables. Note that the order of variables in this part is the same as the order in the resulting prefix. For instance, $\exists x \in \{1, 2, 3\}$ is encoded as `E. x {1,2,3}:` and $\text{Я}_{[1\to 0.6, 2\to 0.1, 3\to 0.3]} y \in \{1, 2, 3\}$ as

[12]Confer `http://isat.gforge.avacs.org`.

[13]Recall that the formal definition of SSMT does not allow Boolean variables explicitly, confer Definition 4.4. For the sake of convenience, Boolean variables are however supported by the SiSAT input language. Internally, a Boolean variable b is represented by an integer variable b' with interval domain $[0, 1]$, and propositional literals b and $\neg b$ are encoded by $b' \geq 1$ and $b' \leq 0$, respectively, as pointed out in Subsection 4.3.1.

```
DECL
    -- Declaration of non-quantified variables.
    int   [-100, 100] a;
    float [-100, 100] b;
    boole             c;
    -- Definition of symbolic constant v.
    define v = 5.2;

PREFIX
    -- Declaration of quantified variables.
    E. x {1, 2, 3}:
    R. y p = [1 -> 0.6, 2 -> 0.1, 3 -> 0.3]:

EXPR
    -- Quantifier-free SMT formula.
    (x * y <= 4);
    (x = 1) -> (y <= 2 and  c);
    (x = 2) -> (y  = 1 and !c);
    (x = 3) -> (y  = 3 and v*(a - b^2) <= 4.5);
    c <-> ((0.2*a + sin(b))^3 >= -0.5);
    (y  = 1 or max(a,b)  < -5.31);
    (y >= 2 or min(a,b)  >  6.7);
```

Figure 6.12: Input file format of SiSAT: example of a single SSMT formula. Note that a line comment starts with the comment delimiter `--`. We further remark that symbols `;` and `!` stand for conjunction with lowest precedence and for negation, respectively.

`R. y p = [1 -> 0.6, 2 -> 0.1, 3 -> 0.3]:`.[14] We remark that the quantified variables are implicitly declared by `PREFIX` and thus must not appear within `DECL`.

3. `EXPR`: This section contains the *matrix*, i.e. the quantifier-free SMT formula, which is an arbitrary Boolean combination of non-linear arithmetic constraints. The syntax is the same as for the iSAT tool. To not stress the reader with a plethora of technicalities, we refer to the iSAT manual[15] comprising a complete list of the supported Boolean and arithmetic operators.

An example of a single SSMT formula in SiSAT's concrete syntax is shown in Figure 6.12.

Probabilistic transition system. Another valid input of SiSAT is a predicative encoding of a probabilistic transition system. The corresponding input file format includes a section for the declaration of variables to be used in the predicates as well as a section for the definition of the sequence of quantified variables to encode non-deterministic and probabilistic choices of a system step. Finally, three formula sections are provided to describe the initial states, the transition relation, and the target states. More precisely,

[14] We remark that SiSAT also supports *universal* quantifiers which are, however, neglected within this thesis as they are *not* needed when addressing the problem of computing *maximum* reachability probabilities.

[15] `http://isat.gforge.avacs.org/documentation/quickstartguide.pdf`

a SiSAT input file encoding a probabilistic transition system consists of the following five sections:

1. `DECL`: This section contains the declarations of all *discrete and continuous system variables* to be used in the individual predicates. As in the single SSMT formula mode, supported variable types are *integer*, *floating-point*, and *Boolean* encoded by `int`, `float`, and `boole`, respectively, where the domain of each floating-point and integer variable has to be bounded by an interval. For instance, `float [0, 1000] x;` declares the floating-point variable x with interval domain $[0, 1000]$, and `boole b;` the Boolean variable b. This section again permits the definition of *symbolic constants*, like `define f = 2.7`.

2. `INIT`: In this section, the set of possible *initial states* is encoded by an arbitrary Boolean combination of arithmetic and propositional constraints over the system variables from section `DECL`, as in `x = 0.6; !b;`.

3. `DISTR`: This part consists of a sequence of *quantified variables* to specify the pattern of non-deterministic and probabilistic choices for each system step. Existential and randomized variables are encoded as in the single SSMT formula mode. For instance, the existential variable $\exists tr \in \{1, 2\}$ is encoded as `E. tr {1,2}:` and the randomized variable $\text{Я}_{[1\to 0.6, 2\to 0.4]} pc \in \{1, 2\}$ as `R. pc p = [1 -> 0.6, 2 -> 0.4]:`.

4. `TRANS`: This section contains the *transition relation* predicate in which system variables from section `DECL` may occur in both primed and unprimed form. An unprimed variable name denotes the value of the variable in the current state while a primed variable represents the value of that variable in the post-state, i.e. after the transition step has taken place. For instance, `(tr = 1 and pc = 1) -> (x' = x + f and (b' <-> !b));` expresses that whenever both tr and pc carry value 1 then the value of x is incremented by constant f and Boolean variable b is toggled.

5. `TARGET`: Finally, the formula of this part characterizes the set of *target states* to be reached, for instance `x > 3.5;`.

An example of a predicative encoding of a probabilistic transition system in the concrete syntax is shown in Figure 6.13.

Given some step depth $k \in \mathbb{N}$, the semantics of a probabilistic transition system encoded in the SiSAT input language is defined by the maximum probability of satisfaction of the following SSMT formula:

$$\left(\bigodot_{j=1}^{k} \texttt{DISTR}(j)\right) : \left(\texttt{INIT}(0) \wedge \left(\bigwedge_{j=1}^{k} \texttt{TRANS}(j-1, j)\right) \wedge \texttt{TARGET}(k)\right) \tag{6.25}$$

where $\bigodot$ denotes concatenation, confer Section 2.1, and

- $\texttt{DISTR}(j)$ gives the sequence of quantified variables within the `DISTR` part where each variable v is substituted by its representative v_j at depth j,

- $\texttt{INIT}(0)$ gives the formula within the `INIT` section where each variable v is substituted by its representative v_0 at depth 0,

```
DECL
   -- Declaration of discrete and continuous system variables.
   boole          b;
   float [0, 1000] x;
   -- Definition of symbolic constant f.
   define f = 2.7;

INIT
   -- Initial states predicate.
   !b and x = 0.6;

DISTR
   -- Declaration of quantified variables to encode
   -- non-deterministic and probabilistic choices.
   E. tr {1, 2}:
   R. pc p = [1 -> 0.6, 2 -> 0.4]:

TRANS
   -- Transition relation predicate.
   (tr = 1 and pc = 1) -> (( b' <-> !b) and x' = x + f);
   (tr = 1 and pc = 2) -> (( b' <->  b) and x' = x);
   (tr = 2 and pc = 1) -> (  b'         and x' = x + 0.5*f);
   (tr = 2 and pc = 2) -> ( !b'         and x' = x);

TARGET
   -- Target states predicate.
   x > 3.5;
```

Figure 6.13: Input file format of SiSAT: example of a probabilistic transition system.

- TRANS($j-1,j$) gives the formula within the TRANS section where each undecorated variable v is substituted by its representative v_{j-1} at depth $j-1$ and each primed variable v' is replaced by v_j for depth j, and
- TARGET(k) gives the formula within the TARGET section where each variable v is substituted by its representative v_k at depth k.

Observe that a probabilistic transition system is more general than a system of concurrent discrete-time probabilistic hybrid automata from Definition 3.1 since, first, the formula of the INIT part potentially characterizes several initial states and not a unique one and, second, the transition relation predicate of the TRANS section potentially connects some system state to several post-states for the same assignment to the quantified variables, violating Property 3.1. As a consequence, the results of Chapter 5 do not apply in general to these richer systems. The rationale for this more general framework, on the one hand, is that enforcing such essentially semantic conditions is hard or even impossible without imposing deep syntactic restrictions. On the other hand, providing more generality might be useful when considering other application scenarios or when extending the scope of quantified variables to continuous domains as it was done theoretically for existential variables in [FTE10a]. The latter extension of SSMT would allow for the symbolic reachability analysis of *continuous-time* probabilistic hybrid systems which is touched upon in

Chapter 10.

As opposed to the SSMT formula $\mathit{PBMC}_{\mathcal{S},\mathit{Target}}(k)$, being the result of the reduction from probabilistic bounded reachability for PHAs to SSMT from Section 5.3, the SSMT formula 6.25, defining the semantics of a probabilistic transition system, does not ensure in general that target states are sinks. This design choice is due to providing the possibility of other analysis problems different to probabilistic bounded reachability from Definition 5.3, like, for instance, computing the maximum probability of reaching the target states in *exactly* k steps instead of *within* k steps.

In order to address the probabilistic bounded state reachability problem for a system $\mathcal{S}$ of concurrent discrete-time probabilistic hybrid automata by means of the SiSAT tool, we can use the predicates introduced by the reductions steps of Section 5.3 but we need to regain the original variable names. For that purpose, we denote

- by $\mathit{CHOICE}_{\mathcal{S}}$ the sequence of quantified variables which arises from $\mathit{CHOICE}_{\mathcal{S}}(1)$ from reduction step 13 by omitting the indices 1 from the variable names,
- by $\mathit{INIT}_{\mathcal{S}}$ and TARGET the formulae which arise from the initial state predicate $\mathit{INIT}_{\mathcal{S}}(0)$ from reduction step 5 and from the target states predicate $\mathit{TARGET}(0)$ from reduction step 9, respectively, by substituting the original names of the system variables for the indexed names of their representatives at depth 0, and
- by $\mathit{TRANS}_{\mathcal{S}}$ and $\mathit{SELF\_LOOP}_{\mathcal{S}}$ the formulae which arise from the transition relation predicate $\mathit{TRANS}_{\mathcal{S}}(0,1)$ from reduction step 8 and from the self loop predicate $\mathit{SELF\_LOOP}_{\mathcal{S}}(0,1)$ from reduction step 10, respectively, by substituting the original as well as the primed names of the system variables for the indexed names of their representatives at depth 0 as well as at depth 1, respectively, and by omitting the indices 1 from the names of quantified variables.

It then remains to create a SiSAT input file such that

1. the `DECL` section declares all discrete and continuous variables of system $\mathcal{S}$, i.e. all variables in $D_1,\dots,D_n$ and $R_1,\dots,R_n$,
2. the `INIT` section consists of $\mathit{INIT}_{\mathcal{S}}$,
3. the `DISTR` section consists of $\mathit{CHOICE}_{\mathcal{S}}$,
4. the `TRANS` section consists of the predicate

   $$(\neg \mathit{TARGET} \Rightarrow \mathit{TRANS}_{\mathcal{S}}) \wedge (\mathit{TARGET} \Rightarrow \mathit{SELF\_LOOP}_{\mathcal{S}}) \quad ,$$

   and
5. the `TARGET` section consists of TARGET.

Employing the above encoding scheme, it then holds that both SSMT formula 6.25 and SSMT formula $\mathit{PBMC}_{\mathcal{S},\mathit{Target}}(k)$ coincide for each $k \in \mathbb{N}$. As an immediate consequence, Theorem 5.1 as well as Corollary 5.1 can be applied to SSMT formula 6.25.

```
This is SiSAT 1.0.

Usage: sisat --i <inputfile.hys> [options]

General iSAT options:
...
  --msw=[real]    : Minimum splitting width of an interval
                    (must be > 2*prabs, default: 0.01)
  --prabs=[real] : Neglect deductions with absolute progress
                    less than prabs (default: 0.001)
...
  --strongsat     : Enables check for strong satisfaction
...

BMC-related options:
  --start-depth   : BMC starting depth (default: 0)
  --max-depth     : BMC maximal number of unwindings
                    (default: 2147483647)

SiSAT options:
  --lt=[real]     : Lower threshold for satisfaction probability
                    (values: 0..1; default: 0)
  --ut=[real]     : Upper threshold for satisfaction probability
                    (values: 0..1; default: 1)
  --no-appr-sol   : Enables that approximate solutions yield
                    probability interval [0,1] (default: [1,1])
  --heu-quant=
       exceed-ut : Aims at exceeding upper threshold (default)
       miss-lt    : Aims at missing lower threshold
       natural    : Natural order
       random     : Random choice
  --no-pur-quant : Disables purification rule for quantified
                    variables (default: enabled)
  --no-sdb        : Disables solution-directed backjumping
                    (default: enabled)
  --pr-cach       : Enables caching of probability results
                    of subproblems (default: disabled)
  --sol-cach      : Enables caching solutions (PBMC)
                    (default: disabled)
  --accur=[real] : Accuracy of probability result
                    (values: 0..1; default: 0)
  --dont-stop     : Search will be continued even if a
                    counter-example was found
```

Figure 6.14: Excerpt of the help menu of SiSAT, which is printed when calling the tool without specifying an input file.

6.6.2 Tool usage and algorithmic core

Tool usage. SiSAT is a command-line tool handling input files of the formats described in Subsection 6.6.1. For an overview of the tool options available, confer the excerpt of the help menu of SiSAT shown in Figure 6.14.

Front end. After having read the specified input file and thereby having detected the desired mode of operation, i.e. addressing a single SSMT formula or a probabilistic transition system, SiSAT first rewrites the quantifier-free formula part(s) into conjunctive form by means of the generalized Tseitin transformation mentioned in Subsection 4.3.1.

In case the input is a probabilistic transition system, SiSAT automatically constructs the SSMT formulae 6.25 defining the semantics of the probabilistic transition system.

```
This is SiSAT 1.0.

Settings.
Minimum splitting width     : 0.01
Absolute bound progress     : 0.001
Relative bound progress     : 0
Variable decision order     : natural
Strong satisfaction check   : enabled
Target thresholds           : 0 / 1
Quant value decision order : exceeding upper threshold
Purification (quant vars)   : enabled
Solution-dir. backjumping   : enabled
Caching probabilities       : disabled
Caching solutions (PBMC)    : disabled
Required accuracy           : 0
Probability 1 for approximate solutions is enabled.

SOLVING:
    k = 0
RESULT:
    Probability of satisfaction lies in [0.69999999,0.70000001]

Statistics.
Number of variables                     : 32
    Existential variables               : 1
    Universal variables                 : 0
    Random variables                    : 1
    Boolean variables                   : 12
    Integer variables                   : 2
    Real variables                      : 16
Number of complex bounds                : 12
Number of problem clauses               : 52
Number of decisions (current / total)   : 105 / 105
(of these:) Number of decisions on
            quantified variables        : 4 / 4
(of these:) Number of decisions on
            non-singleton domains       : 3 / 3
...
SAT / UNKNOWN                           : 100% (2 / 0)
Probability of satisfaction in          : [0.69999999,0.70000001]
Accuracy of result                      : 0

Time solver                             : 0.02 / 0.02 sec
```

Figure 6.15: Abridged output of SiSAT after having solved the single SSMT formula from Figure 6.12 using the default settings and option `--strongsat`.

This procedure works iteratively, i.e. it successively builds the formulae of depths $s, s+1, s+2, \ldots$, where the start depth s and, if desired, the final depth can be specified by input parameters, confer the help menu of SiSAT shown in Figure 6.14.

The SSMT problems constructed above are then handed over to the back end of the tool, i.e. to the actual SSMT solving engine. When addressing probabilistic transition systems, the SSMT formulae of increasing step depths are processed one after another.

Back end. The SSMT solving engine of the SiSAT tool implements the SSMT algorithm described in Section 6.4. It furthermore makes use of the algorithmic enhancements introduced in Section 6.5. The latter features can be enabled or disabled by input parameters, confer the help menu of SiSAT shown in Figure 6.14.

One worth mentioning implementation detail is the calculation of probability results,

```
...
SOLVING:
    k = 0
RESULT:
    Probability of satisfaction lies in [0,0]
...
SOLVING:
    k = 1
RESULT:
    Probability of satisfaction lies in [0,0]
...
SOLVING:
    k = 2
RESULT:
    Probability of satisfaction lies in [0.35999999,0.36000001]
...
SOLVING:
    k = 3
RESULT:
    Probability of satisfaction lies in [0.64799999,0.64800001]
...
SOLVING:
    k = 4
RESULT:
    Probability of satisfaction lies in [0.82079999,0.82080001]
...
SOLVING:
    k = 5
RESULT:
    Probability of satisfaction lies in [0.91295999,0.91296001]

Statistics.
Number of variables                     : 117
    Existential variables               : 5
    Universal variables                 : 0
    Random variables                    : 5
    Boolean variables                   : 82
    Integer variables                   : 0
    Real variables                      : 25
Number of complex bounds                : 16
Number of problem clauses               : 292
...
SAT / UNKNOWN                           : 100% (692 / 0)
Probability of satisfaction in          : [0.91295999,0.91296001]
Accuracy of result                      : 0

Time solver                             : 1.59 / 1.91 sec
```

Figure 6.16: Abridged output of SiSAT when called on the probabilistic transition system from Figure 6.13 up to step depth 5 using the default settings and option `--strongsat`.

circumventing the problem of numerical instabilities, as described in [TF09]. Internally, SiSAT employs floating-point numbers to represent probabilities, both probabilities of randomized quantifiers and intermediate probability results. In order to cope with numerical problems, like converting decimal numbers to floating-point representation while parsing the input file as well as rounding errors, each probability value is safely enclosed by a, as small as possible, floating-point interval. To calculate with such enclosing probability intervals, we apply usual interval arithmetic, confer [Moo66, Moo79, Moo80] as well as Subsections 6.3.1 and 6.3.2. In principle, aforementioned numerical issues can be avoided by using exact number types like rationals but at the price of potentially high computa-

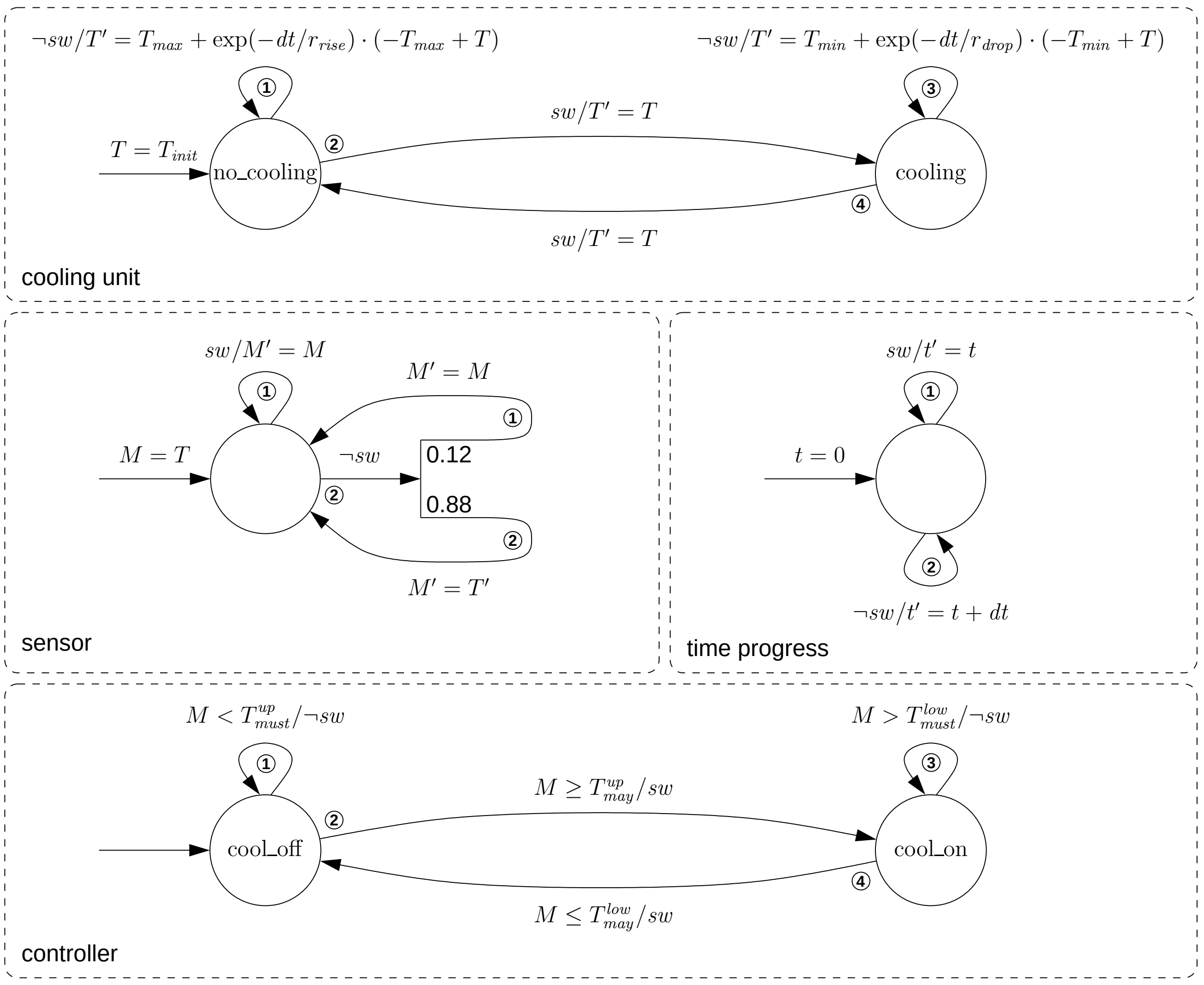


Figure 6.17: Cooling system $\mathcal{S}_{cool}$ consisting of four concurrent automata. The encircled numbers enumerate the (non-deterministic) transitions and, if applicable, the probabilistic transition alternatives of the individual automata.

tional cost. In addition to the fact of efficient treatment of floating-point data types, the introduction of probability intervals immediately allows for the algorithmic enhancement of *accuracy-based pruning*, confer Subsection 6.5.3.

As a side note, probability intervals are also offered in the input language of SiSAT permitting to model systems with uncertain probabilities. However, we do not elaborate further on the latter fact as it is outside the scope of this thesis.

Examples of the output of SiSAT when called on the single SSMT formula from Figure 6.12 and on the probabilistic transition system from Figure 6.13 are presented in Figures 6.15 and 6.16, respectively.

6.6.3 Example of modeling a system of probabilistic hybrid automata within the SiSAT input language

In this subsection, we exemplify the encoding of a system of concurrent PHAs into the SiSAT input language by means of the cooling system $\mathcal{S}_{cool}$ from Figure 6.17.

Explanation of cooling system $\mathcal{S}_{cool}$. The overall idea of the cooling system $\mathcal{S}_{cool}$, consisting of the four concurrent automata cooling unit, sensor, time progress, and controller, is as follows.

The automaton cooling unit describes a simple cooling unit where cooling can be switched off or on. In case cooling is disabled or enabled, the continuous evolution of the temperature T approaches the maximum temperature T_{max} or the minimum temperature T_{min}, respectively, according to the ordinary differential equations (ODEs)

$$\frac{\mathrm{d}T(t)}{\mathrm{d}t} = \frac{-T(t) + T_{max}}{r_{rise}} \tag{6.26}$$

or

$$\frac{\mathrm{d}T(t)}{\mathrm{d}t} = \frac{-T(t) + T_{min}}{r_{drop}} \tag{6.27}$$

with constant $r_{rise} > 0$ and $r_{drop} > 0$. As the discrete-time probabilistic hybrid system model from Definition 3.1 does not support ODEs directly, we deal with the solution functions of above ODEs.[16] That is, the value $T(t_0 + dt)$ of temperature T after dt time units when starting with some initial temperature $T(t_0)$ and when following ODE 6.26 or ODE 6.27 is given by

$$T(t_0 + dt) = T_{max} + e^{(-dt/r_{rise})} \cdot (-T_{max} + T(t_0))$$

or by

$$T(t_0 + dt) = T_{min} + e^{(-dt/r_{drop})} \cdot (-T_{min} + T(t_0)) \quad ,$$

respectively. To distinguish the two different modes of operation, the automaton cooling unit consists of the two locations no_cooling and cooling meaning that cooling is disabled and enabled, respectively. Continuous transition steps of duration dt are then encoded by transitions 1 and 3 with the associated guard conditions that flag sw is `false`, confer Figure 6.17. Mode switches, keeping the value of T, are feasible by taking transitions 2 and 4 provided that flag sw is `true`. Initially, the automaton resides in location no_cooling and the temperature T takes the initial temperature value T_{init}. We see later on that cooling unit is controlled by the automaton controller by means of setting the flag sw. Due to the latter fact, the behavior of cooling unit is fully *deterministic*.

The next automaton sensor is responsible for measuring the temperature T of cooling unit each dt time units. The measured value is made available by variable M. Due to the behavior of cooling unit, time advances by dt if flag sw is `false`, and time remains constant if sw is `true`. Thus, the temperature is sensed each time sw is `false`, and is kept constant otherwise. For technical reasons, the sensor works correctly only with probability 0.88 and thus fails with probability 0.12. In case the measurement is successful, variable M carries the value of T after the step, i.e. $M' = T'$. Otherwise, i.e. measurement fails, the most recently sampled value remains available after the transition, i.e. $M' = M$. Initially, the value of M is equal to the value of T, i.e. to the initial temperature T_{init}. Since the flag sw is set by automaton controller, the behavior of sensor is purely *probabilistic*.

[16] We remark that Chapter 10 elaborates on an extension of the system model to continuous-time involving ODEs.

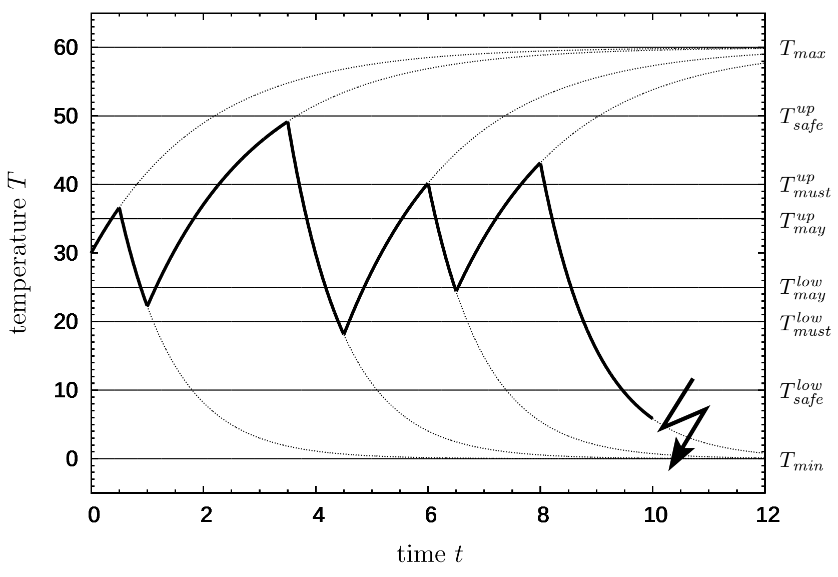


Figure 6.18: Illustration of the behavior of cooling system $\mathcal{S}_{cool}$: potential evolution of temperature T over time t leaving the safe region.

The automaton time progress keeps track of the time t elapsed so far. Initially, t is zero. If cooling unit performs a continuous transition step, i.e. sw is `false`, then t is incremented by the duration dt of the step and otherwise, i.e. sw is `true`, t keeps its old value.

Finally, the automaton controller is in charge of controlling the cooling unit, namely by setting flag sw, such that the temperature remains within the safe region $[T_{safe}^{low}, T_{safe}^{up}] \subseteq [T_{min}, T_{max}]$. The locations of controller are designed for indicating the mode of automaton cooling unit. More precisely, controller resides in cool_off or cool_on if and only if cooling unit resides in no_cooling or cooling, respectively. The control loop of controller relies on the measured temperature M as well as on four parameters T_{must}^{up}, T_{may}^{up}, T_{must}^{low}, and T_{may}^{low} with

$$T_{safe}^{low} \leq T_{must}^{low} \leq T_{may}^{low} \leq T_{may}^{up} \leq T_{must}^{up} \leq T_{safe}^{up} .$$

Whenever the current location is cool_off and the current value of M is at least T_{may}^{up} then the controller may switch on cooling in the cooling unit by performing transition 2 and thus by setting flag sw to `true`. As long as M is strictly below T_{must}^{up}, controller may also decide to let cooling be disabled by executing transition 1. Only if $M \geq T_{must}^{up}$, controller must enforce a mode switch in cooling unit. An analogous behavior applies for location cool_on. As the initial location of cooling unit is no_cooling, controller initially resides in cool_off. Since the value of variable M is determined by automaton sensor, the behavior of controller is purely *non-deterministic*.

The reader might expect the presence of location invariants within the system model. Recall that the semantics of a system of concurrent PHAs, as formalized in Definition 3.2, is defined by runs connecting system states by (discrete) transitions steps. It is however feasible to simulate violation of classical invariants and thus necessitation of leaving a location by blocking a potential self loop for that location through violation of the transition guard of that self loop. For instance, an invariant $M \leq T_{must}^{up}$ of location cool_off in automaton controller is simulated in a way that cool_off cannot be entered by any transition if $M > T^{up}_{must}$. This is obvious for transition 1. With regard to transition 4, observe that $T^{low}_{may} \leq T^{up}_{must}$.

Note that there is a circular dependency between automata cooling unit, sensor, and controller. That is, cooling unit depends on controller by means of flag sw, sensor reads variable T from cooling unit, and controller needs the measurement M from sensor. Illustrating the behavior of the cooling system $\mathcal{S}_{cool}$, a potential evolution of temperature T leaving the safe region $[T^{low}_{safe}, T^{up}_{safe}]$ is shown in Figure 6.18, where the parameters are as follows: $T_{init} = 30$, $T_{max} = 60$, $T_{min} = 0$, $T^{up}_{must} = 40$, $T^{up}_{may} = 35$, $T^{low}_{may} = 25$, $T^{low}_{must} = 20$, $T^{up}_{safe} = 50$, $T^{low}_{safe} = 10$, $r_{rise} = 2$, $r_{drop} = 1$, and $dt = 0.5$.

SiSAT encoding of $\mathcal{S}_{cool}$. Figure 6.18 indicates that there are actually system runs violating the safety requirement of keeping the temperature within $[T^{low}_{safe}, T^{up}_{safe}]$. To address analysis questions like "What is the worst-case probability of reaching an unsafe state within a bounded number of system steps?" or "Can the worst-case probability of reaching an unsafe state ever exceed some acceptable threshold value?", we may employ the probabilistic bounded model checker SiSAT. To this end, we now demonstrate how the cooling system $\mathcal{S}_{cool}$ can be encoded into the SiSAT input language, where the target states predicate is given by $T > T^{up}_{safe} \vee T < T^{low}_{safe}$.

We first define the symbolic constants and declare the system variables in the DECL section, where the values of the system parameters are as above:

```
DECL
   -- Definition of symbolic constants.
   define T_init      = 30;
   define T_max       = 60;
   define T_min       =  0;
   define T_up_must   = 40;
   define T_up_may    = 35;
   define T_low_may   = 25;
   define T_low_must  = 20;
   define T_up_safe   = 50;
   define T_low_safe  = 10;
   define r_rise      =  2;
   define r_drop      =  1;
   define dt          =  0.5;
   -- Declaration of system variables.
   -- Cooling unit:
   define NO_COOLING = 0;
   define COOLING    = 1;
   int [NO_COOLING, COOLING] loc_cu;
   float [T_min, T_max] T;
   -- Reciprocals of r_rise and r_drop.
   float [0, 1] reci_r_rise, reci_r_drop;
   -- Sensor:
   float [T_min, T_max] M;
```

```
    -- Controller:
    define COOL_OFF = 0;
    define COOL_ON  = 1;
    int [COOL_OFF, COOL_ON] loc_cntr;
    boole sw;
    -- Time progress:
    define time_max = 10000;
    float [0, time_max] t;
    -- Encoding of target states.
    boole target;
```

In order to encode the locations of the automata cooling unit and controller, we introduce the integer variables `loc_cu` and `loc_cntr`, respectively. Note that the symbolically encoded values `NO_COOLING` and `COOLING` as well as `COOL_OFF` and `COOL_ON` of the latter variables reflect the corresponding locations immediately. As SiSAT, or rather iSAT, does not support the arithmetic operation *division*[17] directly, we express the reciprocals of `r_rise` and `r_drop` by multiplication in the `TRANS` section later on. The latter fact requires the introduction of the auxiliary variables `reci_r_rise` and `reci_r_drop`. Recall that the domain of each integer and floating-point variable in a SiSAT input file has to be bounded by an interval. The domains of variables `T` and `M` can be simply derived from parameters `T_min` and `T_max`. As both `r_rise` and `r_drop` are at least 1 in our setting, a safe interval domain for their reciprocals is `[0, 1]`. Due to the facts that time variable `t` is initialized with `0` and that the values of `t` are monotonically increasing, a valid lower bound of `t` is `0`. As the duration `dt` of a time step is fixed to `0.5`, a valid lower bound of `t` is subject to the step depth k, namely $0.5 \cdot k$. In our encoding, we use the parameter `time_max` as the upper bound of `t` and set `time_max` to value `10000` which is valid up to step depth 20000. For the sake of readability, the Boolean variable `target` is used to encode the target states predicate. That is, `target` is `true` if and only if the current system state is a target state.

The initial state predicate is specified within the `INIT` section:

```
INIT
    -- Global initial system state.
    loc_cu   = NO_COOLING;
    T        = T_init;
    M        = T;
    loc_cntr = COOL_OFF;
    t        = 0;
    -- Encoding target states.
    target <-> (T > T_up_safe or T < T_low_safe);
```

The `DISTR` section characterizes the pattern of non-deterministic and probabilistic choices of transitions and transition alternatives for a system step. For this purpose, we take one

[17]Recall that all arithmetic operators need to be total in order to obviate the issue with undefined values of partial operations, confer Subsection 4.3.1.

existential variable for each automaton encoding the non-deterministic transition selection, and one randomized variable for each probabilistic choice in each automaton encoding the probabilistic selection of transition alternatives, as it is indicated by reduction steps 3, 4, and 13 in Section 5.3:

```
DISTR
   -- Non-deterministic choices of transitions.
   E. tr_cu   {1, 2, 3, 4}:
   E. tr_snsr {1, 2}:
   E. tr_cntr {1, 2, 3, 4}:
   E. tr_tp   {1, 2}:
   -- Probabilistic choice of transition alternatives.
   R. pc_snsr p = [1 -> 0.12, 2 -> 0.88]:
```

The domains of above quantified variables arise from the enumeration of the transitions and transition alternatives, as shown in Figure 6.17. By Definition 3.1, each transition is formally associated with a non-empty set of probabilistic transition alternatives, even though this set is a singleton. The latter implies that reduction step 4 actually introduces one randomized variable per transition. For the sake of simplicity and without loss of generality, we omit the introduction of such redundant randomized variables of the shape $Ⴘ_{[v \to 1]} x \in \{v\}$. We moreover remark that the automata cooling unit, sensor, and time progress do *not* exhibit non-deterministic behavior as the guard predicates of the outgoing transitions from each location are mutually exclusive. This fact would enable a potentially more efficient encoding scheme omitting the introduction of unnecessary existential variables. However, checking mutual exclusion of guard predicates is a hard problem in general. We thus blindly apply the encoding scheme of Section 5.3 also for automata without non-determinism.

The transition relation predicate is specified in the TRANS section. For each automaton, the guard conditions and assignments of the transitions and transition alternatives are encoded as shown by reduction steps 6 and 7. The Boolean variable target is used to enable self loops whenever the system has already reached a target state, confer reduction step 10.

```
TRANS
   -- Defining reciprocals of r_rise and r_drop.
   r_rise * reci_r_rise = 1;
   r_drop * reci_r_drop = 1;

   -- Transition relation of cooling unit.
   (!target and tr_cu = 1) -> (
       loc_cu  = NO_COOLING and !sw and
       loc_cu' = NO_COOLING and
       T' = T_max + exp(-dt*reci_r_rise)*(-T_max + T));
   (!target and tr_cu = 2) -> (
       loc_cu  = NO_COOLING and sw and
       loc_cu' = COOLING and T' = T);
```

```
    (!target and tr_cu = 3) -> (
        loc_cu  = COOLING and !sw and
        loc_cu' = COOLING and
        T' = T_min + exp(-dt*reci_r_drop)*(-T_min + T));
    (!target and tr_cu = 4) -> (
        loc_cu  = COOLING and sw and
        loc_cu' = NO_COOLING and T' = T);
    -- Transition relation of sensor.
    (!target and tr_snsr = 1) -> ( sw and M' = M);
    (!target and tr_snsr = 2) -> (!sw);
    (!target and tr_snsr = 2 and pc_snsr = 1) -> (M' = M);
    (!target and tr_snsr = 2 and pc_snsr = 2) -> (M' = T');
    -- Transition relation of controller.
    (!target and tr_cntr = 1) -> (
        loc_cntr  = COOL_OFF and M < T_up_must and
        loc_cntr' = COOL_OFF and !sw);
    (!target and tr_cntr = 2) -> (
        loc_cntr  = COOL_OFF and M >= T_up_may and
        loc_cntr' = COOL_ON  and  sw);
    (!target and tr_cntr = 3) -> (
        loc_cntr  = COOL_ON  and M > T_low_must and
        loc_cntr' = COOL_ON  and !sw);
    (!target and tr_cntr = 4) -> (
        loc_cntr  = COOL_ON  and M <= T_low_may and
        loc_cntr' = COOL_OFF and  sw);
    -- Transition relation of time progress.
    (!target and tr_tp = 1) -> ( sw and t' = t);
    (!target and tr_tp = 2) -> (!sw and t' = t + dt);

    target -> (loc_cu' = loc_cu and T' = T and M' = M and
               loc_cntr' = loc_cntr and !sw and t' = t);

    -- Encoding target states.
    target' <-> (T' > T_up_safe or T' < T_low_safe);
```

Observe that constraints `r_rise * reci_r_rise = 1;` and `r_drop * reci_r_drop = 1;` express the reciprocals of `r_rise` and `r_drop` by multiplication.

The `TARGET` section finally consists of the target states predicate. We may reuse the Boolean variable `target` here as `target` is `true` if and only if the corresponding system state is a target state, as ensured by the `INIT` and `TRANS` sections.

```
TARGET
   target;
```

Alternative SiSAT encoding of $\mathcal{S}_{cool}$. Recall that the basic SiSAT approach as being presented in Section 6.4 implements an explicit traversal through the tree spanned by the quantifier prefix of the given SSMT formula, confer Figure 6.5 a. As the size of such trees is exponential in the number of quantified variables, the naive SiSAT algorithm traversing the whole quantifier tree is far from scalable. In order to improve performance of SiSAT, Section 6.5 has elaborated on several algorithmic enhancements to prune the quantifier tree, as indicated by Figure 6.5 b.

As already mentioned in Subsection 4.4.2, an additional way to improve solving time of SSMT tools is to provide "better" SSMT problem encodings in the sense of reducing the potential search space. In what follows, we discuss how the basic SiSAT encoding of the cooling system $\mathcal{S}_{cool}$ can modified with the objective of speeding up SiSAT's proof search.

The main idea we have in mind is to "disable" quantifiers whenever they are "not needed". A bit more precisely, whenever some partial assignment to the variables implies that the truth value of the remaining formula does not depend on some quantified variables $x_1, \ldots, x_i$ then we aim at fixing the values for $x_1, \ldots, x_i$ such that SiSAT need *not* probe *all* the assignments to $x_1, \ldots, x_i$ *but* just *one* of them.

With regard to the basic SiSAT encoding of $\mathcal{S}_{cool}$, observe that the quantified variables does not play any role if the Boolean variable `target` is `true`. In other words, in case a target state is reached then the variables representing the post-state are frozen by means of a self loop, in fact, independent of the actual values of the quantified variables. We may exploit this observation for each existential variable tr, namely by adding some fresh value v to its domain and then by enforcing that tr carries value v if and only if the Boolean literal `target` holds.

A randomized variable pc_t encoding the probabilistic choice after selection of transition t may be "disabled" whenever transition t is not executed, i.e. whenever a value different to t was assigned to the corresponding existential variable tr, the latter encoding the non-deterministic choice of transitions in the corresponding automaton. Adapting this to the cooling system $\mathcal{S}_{cool}$, we aim at "disabling" randomized variable `pc_snsr` if and only if existential variable `tr_snsr` takes a value which is not equal to `2`. As pointed out in Subsection 4.4.2, the same approach as for existential variables mentioned above is not directly feasible for randomized variables since this would lead to incorrect probability results. However, a solution to this issue was suggested in Subsection 4.4.2 by means of randomized quantification involving *dependent probability distributions*. That is, "disabling" variable `pc_snsr` if and only if `tr_snsr` is not equal to `2` can be realized by

$$\text{Я}_{[(\texttt{tr\_snsr} \neq 2) \to d_{off}, (\texttt{tr\_snsr} = 2) \to d_{on}]} \texttt{pc\_snsr} \in \{0, 1, 2\}$$

where d_{off} gives the distribution $[0 \to 1]$, and d_{on} represents the distribution $[1 \to 0.12, 2 \to 0.88]$ as derived from the system model $\mathcal{S}_{cool}$. From the semantics of SSMT involving such dependent distributions, being formalized in Definition 4.7, it follows that if $\texttt{tr\_snsr} \neq 2$ holds, i.e. if transition 2 is *not* selected in automaton sensor, then distribution d_{off} is "activated", the latter forcing variable `pc_snsr` to take value 0 with probability 1. If $\texttt{tr\_snsr} = 2$ is true, i.e. transition 2 is actually selected, then distribution d_{on} is "activated", the latter dealing with the probabilistic transition alternatives. We remark that no extra treatment is necessary for randomized variable `pc_snsr` in case the target states are reached. The rationale is as follows: if variable `target` is `true` then `tr_snsr` is set to some value $v \neq 2$ and thus distribution d_{off} is selected for `pc_snsr`.

Due to the fact that the SSMT algorithm described in Sections 6.4 and 6.5 as well as the SiSAT tool do *not* yet support the extension of SSMT involving dependent probability distributions, as introduced in Subsection 4.4.2, we aim at a pragmatic approach to some similar concept.

For that purpose, we relax the requirement that the probabilities in a randomized quantifier must add up to 1, i.e. for $Ɐ_{d_x} x \in \mathcal{D}_x$ we do *not* demand that $\sum_{v \in \mathcal{D}_x} d_x(v) = 1$ holds but rather $\sum_{v \in \mathcal{D}_x} d_x(v) \geq 1$. With regard to this relaxed definition of SSMT, it is important to remark that d_x is no longer a probability distribution, and that the maximum probability of satisfaction as formalized by Definition 4.5 is no longer bounded from above by value 1 but can be a value strictly greater than 1. Though this relaxation seems to be unreasonable, it becomes beneficial when ensuring valid probability distributions in randomized quantifiers by means of predicates in the matrix of the SSMT formula. Intuitively, whenever a randomized variable x becomes leftmost in the prefix then the assignment to its preceding variables has triggered some constraints in the matrix which exclude values of the domain of x from potential solutions such that all the remaining values have a probability mass at most 1. Definition 6.3 formally introduces this property of well-definedness:

Definition 6.3 (Well-definedness of relaxed SSMT formulae)
An SSMT formula $\mathcal{Q} : \varphi$, potentially being relaxed as mentioned before, is called well-defined *if and only if for each $(Ɐ_{d_x} x \in \mathcal{D}_x) \in \mathcal{Q}$ with $\mathcal{Q} = \mathcal{Q}' \odot Ɐ_{d_x} x \in \mathcal{D}_x \odot \mathcal{Q}''$ and for each assignment τ to the quantified variables in $Var(\mathcal{Q}') = \{x_1, \ldots, x_i\}$, it holds that*

$$\sum_{v \in \mathcal{V}} d_x(v) \leq 1$$

where $\mathcal{V} = \{\tau'(x) \in \mathcal{D}_x : \tau' \models \varphi[\tau(x_1)/x_1] \ldots [\tau(x_i)/x_i]\}$ denotes the set of all values $\tau'(x)$ in the domain $\mathcal{D}_x$ of x which are part of a model τ' of SMT formula $\varphi[\tau(x_1)/x_1] \ldots [\tau(x_i)/x_i]$.

Clearly, each "classical" SSMT formula conforming to Definition 4.4 is well-defined. It is moreover not hard to see that application of the classical SSMT semantics of Definition 4.5 to *well-defined* relaxed SSMT formulae Φ always gives maximum probabilities of satisfaction that range within the interval $[0, 1]$, i.e. $0 \leq Pr(\Phi) \leq 1$.

Without going into detail, we remark that Theorem 6.2, stating soundness of the SSMT algorithm, and thus Corollary 6.3, as well as Theorem 6.3, strengthening soundness when avoiding rule SSMT.9, and Theorem 6.4, evincing termination of the SSMT algorithm, also apply to *well-defined* relaxed SSMT formulae. The only difference to the "classical" case is that an *overapproximated* probability result, i.e. if parameter $\tilde{p}$ is fixed to 1, might exceed value 1, while the latter still establishes a correct result according to Theorem 6.2. Such results exceeding value 1 are caused by *incompleteness* of the underlying SMT solver iSAT, more precisely, by *approximate* solutions obtained by rule SSMT.9. That is, for some randomized variable it might happen during SSMT proof search that the probability mass of values yielding positive (overapproximated) probability results is strictly greater than 1. With regard to the SSMT tool SiSAT, this issue is avoided by bounding each intermediate probability result from above by 1. We finally mention that all algorithmic enhancements of Section 6.5 can also be employed on *well-defined* relaxed SSMT formulae.

Let us return to our goal of "disabling" randomized variable `pc_snsr` if and only if existential variable `tr_snsr` is not equal to 2. Instead of using the randomized quantifier

$$Ʉ_{[(\texttt{tr\_snsr} \neq 2) \to d_{off}, (\texttt{tr\_snsr} = 2) \to d_{on}]} \texttt{pc\_snsr} \in \{0, 1, 2\}$$

involving dependent probability distributions, we may deploy the "relaxed" randomized quantifier

$$Ʉ_{[0 \to 1, 1 \to 0.12, 2 \to 0.88]} \texttt{pc\_snsr} \in \{0, 1, 2\}$$

whenever we ensure that if $\texttt{tr\_snsr} \neq 2$ then randomized variable `pc_snsr` may only carry value 0 and otherwise, i.e. if $\texttt{tr\_snsr} = 2$, possible values for `pc_snsr` are only 1 and 2. The latter can be simply enforced by adding corresponding predicates to the `TRANS` section, as we see later on.

We are now prepared to modify the basic SiSAT encoding of the cooling system $\mathcal{S}_{cool}$ in such a way that a quantifier becomes "disabled" whenever the concrete value of its corresponding quantified variable becomes irrelevant.

At first, we define the symbolic constant `OFF` within the `DECL` section which is used to "disable" quantified variables. It is important to note that the concrete value represented by `OFF` may *not* be present already in the domains of the quantified variables. In our case, we define `OFF` to be `0`. That is, the `DECL` part of the SiSAT input file is extended as follows:

```
DECL
...
    -- Symbolic constant for "disabling" quantifiers.
    define OFF = 0;
```

While the initial system state remains the same, i.e. the `INIT` section does not change, we need to add the fresh value `OFF` to the domains of all quantified variables, as described above. This is done within the `DISTR` part of the SiSAT input file:

```
DISTR
    -- Non-deterministic choices of transitions.
    E. tr_cu   {OFF, 1, 2, 3, 4}:
    E. tr_snsr {OFF, 1, 2}:
    E. tr_cntr {OFF, 1, 2, 3, 4}:
    E. tr_tp   {OFF, 1, 2}:
    -- Probabilistic choice of alternatives.
    R. pc_snsr p = [OFF -> 1, 1 -> 0.12, 2 -> 0.88]:
```

In order to "disable" quantified variables correctly such that the resulting relaxed SSMT formulae are well-defined according to Definition 6.3, we add corresponding predicates to the `TRANS` section, as it was explained above:

```
TRANS
...
    -- "Disable" existential quantifiers
    -- iff target states are reached.
    target <-> tr_cu   = OFF;
    target <-> tr_snsr = OFF;
    target <-> tr_cntr = OFF;
    target <-> tr_tp   = OFF;
    -- "Disable" randomized quantifier
    -- if corresponding transition is not taken.
    tr_snsr != 2 -> pc_snsr  = OFF;
    -- "Enable" randomized quantifier
    -- if corresponding transition is taken.
    tr_snsr  = 2 -> pc_snsr != OFF;
```

With respect to "disabling" randomized variable `pc_snsr`, observe that the aforementioned issue concerning overapproximated probability results strictly greater than 1 is of no relevance for the SiSAT encoding above. The rationale is that satisfaction of both predicates `tr_snsr != 2 -> pc_snsr = OFF;` and `tr_snsr = 2 -> pc_snsr != OFF;` under each assignment to variables `tr_snsr` and `pc_snsr` can always be decided by SiSAT since both `tr_snsr` and `pc_snsr` range over finite domains. In fact, SiSAT is potentially able to appropriately prune the domain of randomized variable `pc_snsr` immediately after existential variable `tr_snsr` was assigned a value, namely by means of the deduction rule SSMT.6.

Finally, the `TARGET` section is left unchanged, and thus we have completed the alternative SiSAT encoding of the cooling system $\mathcal{S}_{cool}$.

Analysis results. After having encoded the cooling system $\mathcal{S}_{cool}$ into the SiSAT input language, we are now capable of employing the SSMT-based probabilistic bounded model checker SiSAT in order to address analysis questions like

1. "What is the maximum probability of reaching the unsafe states, i.e. states where temperature T has left the safe region $[T^{low}_{safe}, T^{up}_{safe}]$, within k system steps?" or
2. "Can this maximum probability ever exceed some acceptable threshold value θ?".

To this end, SiSAT was called on the *alternative* SiSAT encoding of $\mathcal{S}_{cool}$, while all experiments were performed on a 2.4 GHz AMD Opteron machine with 128 GByte physical memory running Linux. Concerning the issue of *approximate* solutions, the pragmatic perspective of considering such results as "good enough" solutions in practice was realized, i.e. parameter $\tilde{p}$ was fixed to 1 in each application of rule SSMT.9, where the minimum splitting width ε was set to 0.01. Recall that the minimum splitting width ε specifies the accuracy of an approximate solution σ to some extent, more precisely, each interval in σ has a width below ε, confer item 2c of Proposition 6.2.

Giving attention to analysis question 1, SiSAT was able to compute the maximum probability of reaching the target (i.e. unsafe) states from step depth 0 up to depth 31 within a time limit of 25 hours. Concerning the solver settings of the latter run, we made use of some algorithmic enhancements described in Section 6.5, namely *activity-based value branching heuristics* preferring values with highest (weighted) measure, *purification*,

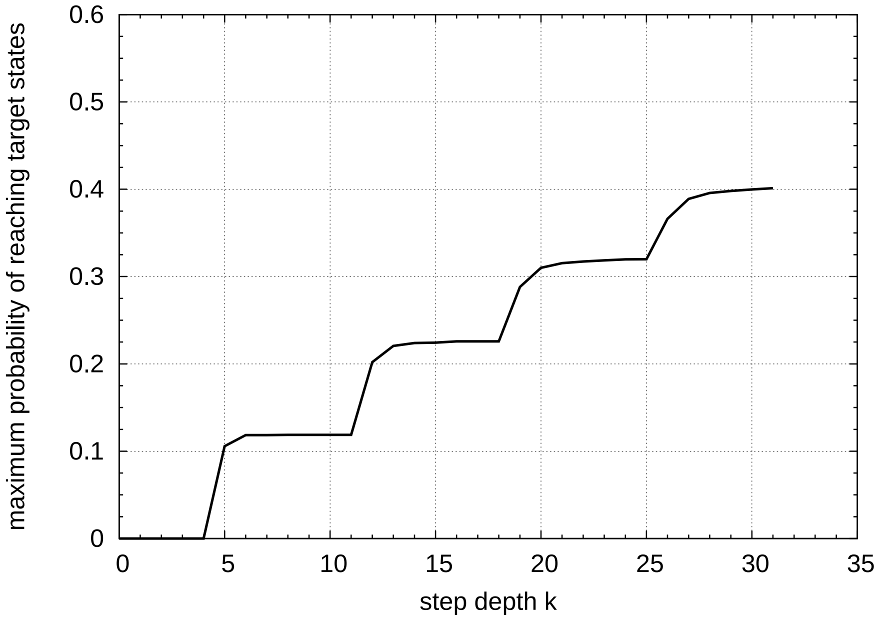


Figure 6.19: Analysis of cooling system $\mathcal{S}_{cool}$: evolution of the maximum probability of reaching the unsafe states over step depth k.

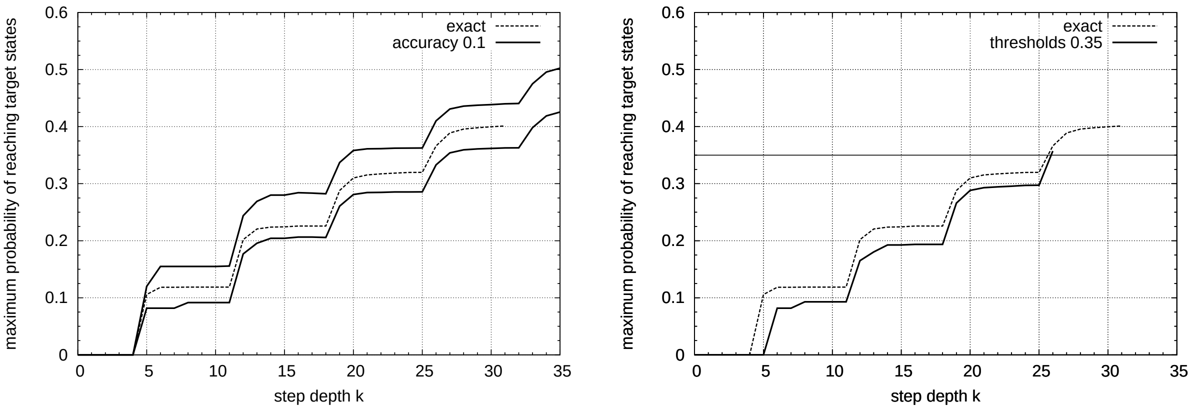


Figure 6.20: Analysis of cooling system $\mathcal{S}_{cool}$: probability results when having used accuracy-based pruning with accuracy $\alpha = 0.1$ (left) and thresholding with thresholds $\theta_l = \theta_u = 0.35$ (right).

solution-directed backjumping, as well as *caching solutions.* The evolution of the maximum reachability probability over step depth k is plotted in Figure 6.19.

From an engineering perspective, sometimes it might be reasonable to relax analysis question 1 a bit, namely in a way of *not* targeting at the *exact* reachability probability but just at a result of some sufficient *accuracy.* In such cases, we may exploit the optimization of *accuracy-based pruning* with some desired accuracy α. When having used the latter feature with $\alpha = 0.1$ in addition to the algorithmic enhancements listed above, SiSAT solved the probabilistic reachability problem up to step depth 35 within the same time limit of 25 hours. The corresponding results are depicted on the left of Figure 6.20.

Similar to the relaxed version of analysis question 1, the *exact* probability result is *not* called for by analysis question 2, but rather whether this probability can ever *exceed* some *threshold* value θ. Taking this into account, the idea of *thresholding* becomes applicable, potentially improving performance of SSMT proof search. Concerning the cooling system $\mathcal{S}_{cool}$, SiSAT in the above setting without accuracy-based pruning needed about 49 minutes to find out that the maximum probability of reaching the unsafe states exceeds threshold value $\theta = 0.35$, the latter being the case at step depth 26. Having called SiSAT with the same solver options but furthermore having activated *thresholding* with $\theta_l = \theta_u = \theta = 0.35$, solving time has been reduced by roughly 30% to about 34 minutes. The corresponding witness values are plotted on the right of Figure 6.20. With regard to the computation of these witness values, parameter $\tilde{p}$ was fixed to 0 in all applications of rules SSMT.3 and SSMT.4. This setting reflects the computational effort to some extent, namely by exhibiting how much of the actual probability result had to be computed in order to reveal that threshold $\theta = 0.35$ cannot be reached or is exceeded, confer the right of Figure 6.20.

A very detailed study of the runtime and solving behavior of the SSMT solver SiSAT by taking into consideration the various algorithmic enhancements follows next in Section 6.7.

6.7 Experimental results

This section gives a detailed account of the runtime and solving behavior of the SSMT-based probabilistic bounded model checker SiSAT, being described in Section 6.6, with a special focus on an experimental evaluation of the algorithmic enhancements presented in Section 6.5. As a benchmark, we use the SiSAT encodings of the cooling system $\mathcal{S}_{cool}$, confer Subsection 6.6.3, being representative for a vast number of similar case studies. We remark that a more sophisticated case study, namely the analysis of the networked automation system (NAS) introduced in Section 3.1, is investigated in Chapter 8.

All experiments of this section were performed on a 2.4 GHz AMD Opteron machine with 128 GByte physical memory running Linux. As in Subsection 6.6.3, we considered *approximate* solutions as “good enough” solutions in practice and thus used $\tilde{p} = 1$ in each application of rule SSMT.9 with the minimum splitting width $\varepsilon = 0.01$. For the following experiments, we performed several SiSAT runs with different solver settings. In each such run, the tool was called to solve the corresponding probabilistic bounded reachability problem for step depths 0 to 40 within a time limit of 25 hours and with a memory limit of 16 GByte. In case the memory-intensive enhancement of *caching probability results of subproblems*, explained in Subsection 6.5.5, was employed, the memory limit was raised to 64 GByte.

To evaluate the proof search of SiSAT with respect to different solver settings, the corresponding *solving times* are of paramount importance. However, it is also of interest to take account of other statistics of the proof search. As motivated in Section 6.5, the main goal of the algorithmic enhancements is to prune the quantifier tree considerably, confer Figure 6.5 b. To assess the latter fact, the *number of (actual) branching steps*, i.e. number of applications of rules SSMT.1 to SSMT.4 with $|\mathcal{D}_x| \geq 2$, as well as the *number of detected (approximate) solutions* are meaningful figures. For reasons of clarity and comprehensibility and due to the large range of the measured values, we make use of a *logarithmic* scale with respect to base 10 when presenting the experimental data graphically. In order to avoid the issue with value 0 on a logarithmic scale arising from the fact that the logarithm of 0 is not defined, solving times less than 0.01 seconds are mapped to value 0.01, and value 0 is represented by 0.5 when considering the number of branching steps and detected (approximate) solutions.

We first investigate the impact of each of the various algorithmic enhancements compared to the naive SSMT algorithm in Subsections 6.7.1 to 6.7.8. Thereafter, we also take account of combinations of these optimizations in Subsection 6.7.9. For the latter experiments, the *basic* SiSAT encoding of the cooling system $\mathcal{S}_{cool}$ was used, i.e. the version without "relaxed" randomized quantifiers. In Subsection 6.7.10, we then examine the tool behavior on the *alternative* SiSAT encoding of $\mathcal{S}_{cool}$ supporting the idea of "relaxed" randomized quantifiers. Subsection 6.7.11 finally draws conclusions from the experimental results.

6.7.1 Naive algorithm

The naive SiSAT algorithm, implementing a brute-force approach without any of the algorithmic enhancements being deployed, was only able to solve the probabilistic bounded reachability problem for the first 8 step depths. The corresponding experimental results are plotted in Figure 6.21. These data reveal a strong exponential growth in the solving time as well as in the number of branching steps and of detected (approximate) solutions over the step depth. For instance, the solving time has increased from depth 6 to depth 7 by a factor of about 144, and from depth 7 to 8 by a factor of about 147. Assuming a growth factor of at least 140, we may extrapolate the solving times for larger step depths k: the solver would need more than 40 days for $k = 9$, more than 15 years for $k = 10$, and even more than 2100 years for $k = 11$.

Inspecting Figure 6.21 a bit more precisely, the solving behavior gets particularly worse from step depth 5 on. A plausible explanation of this is as follows. Observe that cooling system $\mathcal{S}_{cool}$ can reach the target states only after 5 system steps, as proven by Figure 6.19, exhibiting positive reachability probabilities only from depth 5 on, and by Figure 6.21, where the bottom graph shows a positive number of detected (approximate) solutions only from depth 5 on. Regarding the basic SiSAT encoding of $\mathcal{S}_{cool}$, whenever a target state is reached then a self loop is executed meaning that the remaining non-quantified system variables keep their values independent of the actual values of the non-instantiated quantified variables. That is, in such cases all these latter quantified variables become "unconstrained" and thus iSAT's deduction mechanisms, being lifted to SiSAT by means of rule $\mathsf{SSMT.6}$, never apply to these variables. This implies that the SSMT algorithm must explore all the values of these "unconstrained" variables in worst case. For the basic SiSAT encoding of $\mathcal{S}_{cool}$, these are $4 \cdot 2 \cdot 4 \cdot 2 \cdot 2 = 128$ possible assignments to the quantified variables for the system step immediately after the target states have been reached, and thus 128^n assignments for n successive iterations of this self loop.

Concerning the latter issue, note that whenever a target state is reached then each of the assignments to the remaining quantified variables leads to a solution of the formula and thus to satisfaction probability 1. As a consequence, each value of an existential variable among the remaining quantified variables yields satisfaction probability 1. The

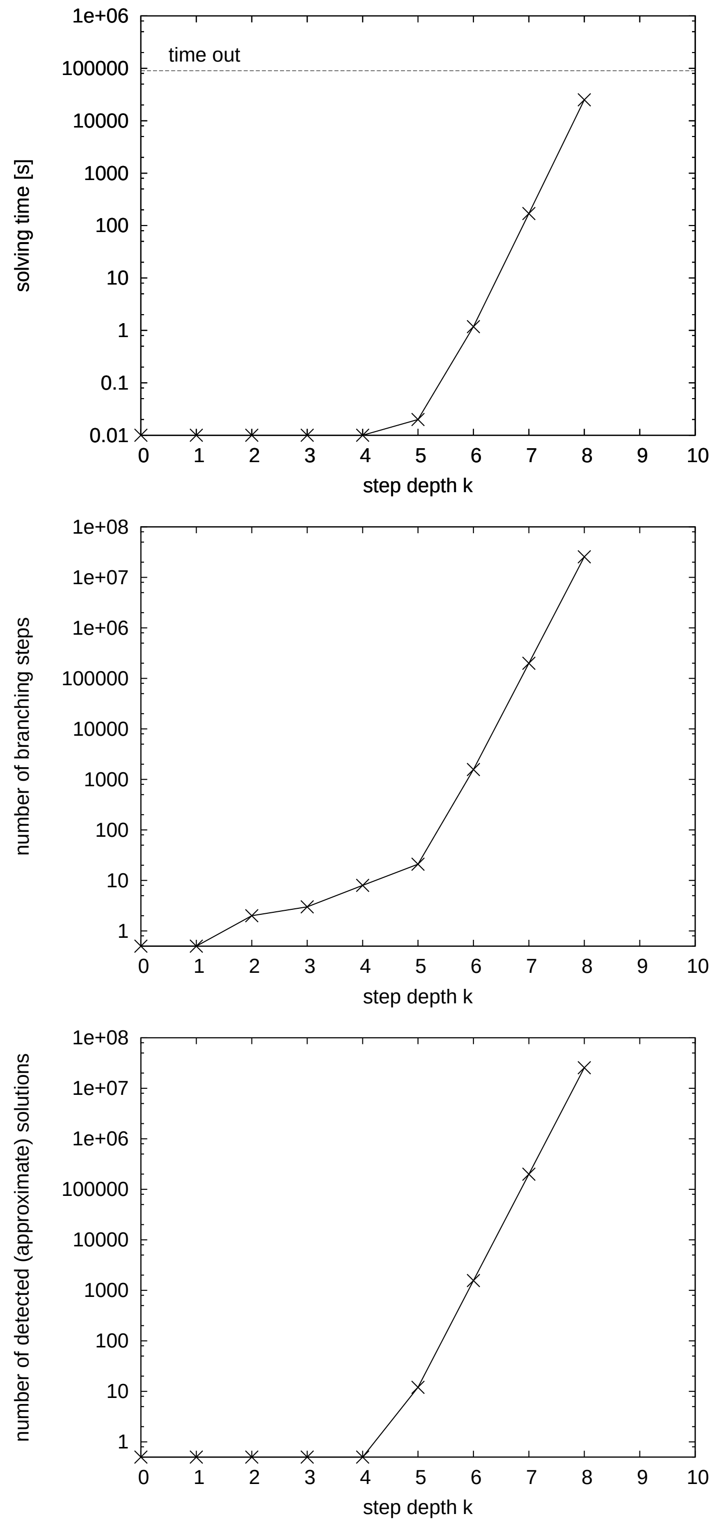


Figure 6.21: Evaluation of the SiSAT tool: *naive algorithm.* Solving times (top), numbers of branching steps (middle), and numbers of detected (approximate) solutions (bottom) for the corresponding step depths when no algorithmic enhancements were applied.

latter fact in turn means that above issue may not be present for existential variables since rule SSMT.3 skips investigation of all the remaining branches whenever an intermediate probability result $pr = 1$ was computed. However, this situation does not arise in our experiments due to the following reason. As mentioned at the end of Subsection 6.6.2, SiSAT internally represents probability values, i.e. both probabilities of randomized quantifiers and intermediate probability results, as safely enclosing, as small as possible floating-point intervals in order to avoid numerical instabilities. The probability values 0.12 and 0.88 occurring in the SiSAT encoding of $\mathcal{S}_{cool}$ are *not* representable as floating-point numbers and thus are enclosed by non-point intervals $[l_{0.12}, u_{0.12}]$ and $[l_{0.88}, u_{0.88}]$, respectively, with $l_{0.12} < 0.12 < u_{0.12}$ and $l_{0.88} < 0.88 < u_{0.88}$. Therefore, the sum of values 0.12 and 0.88, which clearly is 1, is also represented as a non-point interval $[l_{0.12+0.88}, u_{0.12+0.88}]$ with $l_{0.12+0.88} \leq l_{0.12} + l_{0.88} < 1 < u_{0.12} + u_{0.88} \leq u_{0.12+0.88}$. We remark that in this special case where an upper bound u of a probability interval is strictly greater than 1, u is actually reduced to the safe upper bound 1. That is, each value of the existential variables among the remaining quantified variables yields a probability interval $[l, 1]$ with $l < 1$ instead of $[1, 1]$. Hence, rule SSMT.3 may not be applied as suggested above. This issue obviously can be avoided by using exact number types like rationals but at the price of potentially high computational cost. Since some algorithmic optimizations as well as the alternative SiSAT encoding supporting "relaxed" randomized quantifiers can cope with the above situation, as we see later on, it renders unnecessary the introduction of exact number types.

6.7.2 Impact of thresholding

The first algorithmic enhancement we investigate is *thresholding* which is based on two input parameters, namely on a lower threshold θ_l and on an upper threshold θ_u. As explained in Section 6.4, the idea of thresholding when solving some SSMT formula Φ is as follows. If the maximum probability of satisfaction $Pr(\Phi)$ lies in the interval $[\theta_l, \theta_u]$ then we aim at computing the *actual* satisfaction probability. In all other cases, i.e. $Pr(\Phi) \notin [\theta_l, \theta_u]$, the actual probability of satisfaction is not of interest but only some *witness* value pr is desired such that $pr < \theta_l$ if and only if $Pr(\mathcal{Q} : \varphi) < \theta_l$ and $pr > \theta_u$ if and only if $Pr(\mathcal{Q} : \varphi) > \theta_u$. This relaxation is exploited during SSMT proof search by means of skipping investigation of the remaining branches whenever the upper or lower threshold for some subtree is already exceeded by processed branches or can no longer be reached by all remaining branches, respectively.

The presence of such thresholds is motivated, for instance, in (bounded) model checking of probabilistic systems where the problem is to decide whether the probability of reaching the target states is below some acceptable threshold value θ, like in Definition 5.4. Regarding this, several SiSAT runs were performed where both the lower and upper thresholds were set to the same threshold value $\theta \in \{0, 0.1, 0.2, \ldots, 0.9, 1\}$, i.e. $\theta_l = \theta_u = \theta$.

As mentioned in Subsection 6.5.1, the order in which possible assignments to the quantified variables are probed may influence the performance of the proof search, as some values trigger earlier applications of thresholding. We therefore used two different static value orderings for the quantified variables, namely a *descending* and an *ascending* order. We remark that Subsection 6.7.3 investigates more sophisticated and dynamic value ordering heuristics. For technical reasons, SiSAT reads the values of the domains of quantified variables from right to left such that the domains are sorted in *descending* order after having parsed the SiSAT input file from Subsection 6.6.3. To achieve an *ascending* order, we may simply rearrange the domains of the quantified variable in the input file. That is, we have changed the `DISTR` section to:

```
DISTR
    -- Non-deterministic choices of transitions.
    E. tr_cu   {4, 3, 2, 1}:
    E. tr_snsr {2, 1}:
    E. tr_cntr {4, 3, 2, 1}:
    E. tr_tp   {2, 1}:
    -- Probabilistic choice of alternatives.
    R. pc_snsr p = [2 -> 0.88, 1 -> 0.12]:
```

The corresponding experimental results for both value orderings and for the different threshold values θ are plotted in Figure 6.22, where "naive" denotes the naive SSMT algorithm from Subsection 6.7.1 with thresholding being disabled.[18]

The graphs of Figure 6.22 give two diverse impressions. While the benefit of thresholding seems to be negligible for the ascending order (except for $\theta = 0$), the results for the descending order indicate that thresholding can lead to tremendous performance gains. In the extreme case where θ is 0, both value orderings show a very impressive behavior: for step depth 8, speed-ups of six orders of magnitude were obtained compared to the naive algorithm. Moreover, SiSAT was able to solve the problem for step depths 0 up to 40 in less than 2 seconds. The rationale for these speed-ups is as follows: instead of enumerating all solutions as in the naive approach, the algorithm needs to find only *one* solution to decide that the satisfaction probability is strictly greater than threshold value 0. Concerning the descending order, thresholding for $\theta = 1$ was also very powerful: the speed-up at step depth 8 again amounts to six orders of magnitude compared to the naive approach, while all SSMT problems, i.e. for all depths 0 to 40, were solved within 27 seconds. An explanation for the latter behavior is that after refuting some assignments to the quantified variables, SiSAT was able to recognize that the probability mass of the remaining assignments is too small to reach threshold value 1.

In addition to the value ordering, the power of thresholding clearly depends on the given threshold values as well as on the actual probability of satisfaction. Because of that, a more elaborated and more appropriate view is given by Figure 6.23 which plots the solving time over the threshold value θ for step depths 6, 7, and 8. We remark that the latter step depths were chosen since SiSAT solved the corresponding SSMT problems for all of the different threshold values.

With respect to the descending order, the speed-ups obtained for threshold value 0 and for all thresholds greater than or equal to 0.5 are of two orders of magnitude at step depth 6, of (at least) three orders at depth 7, and even of six orders at depth 8, while the tool is just about 6 times faster compared to the naive algorithm for both thresholds 0.1 and 0.4 at all depths 6, 7, and 8. Concerning threshold value 0.3, the speed-up further

[18] Recall that thresholding can be "disabled" by setting $\theta_l = 0$ and $\theta_u = 1$.

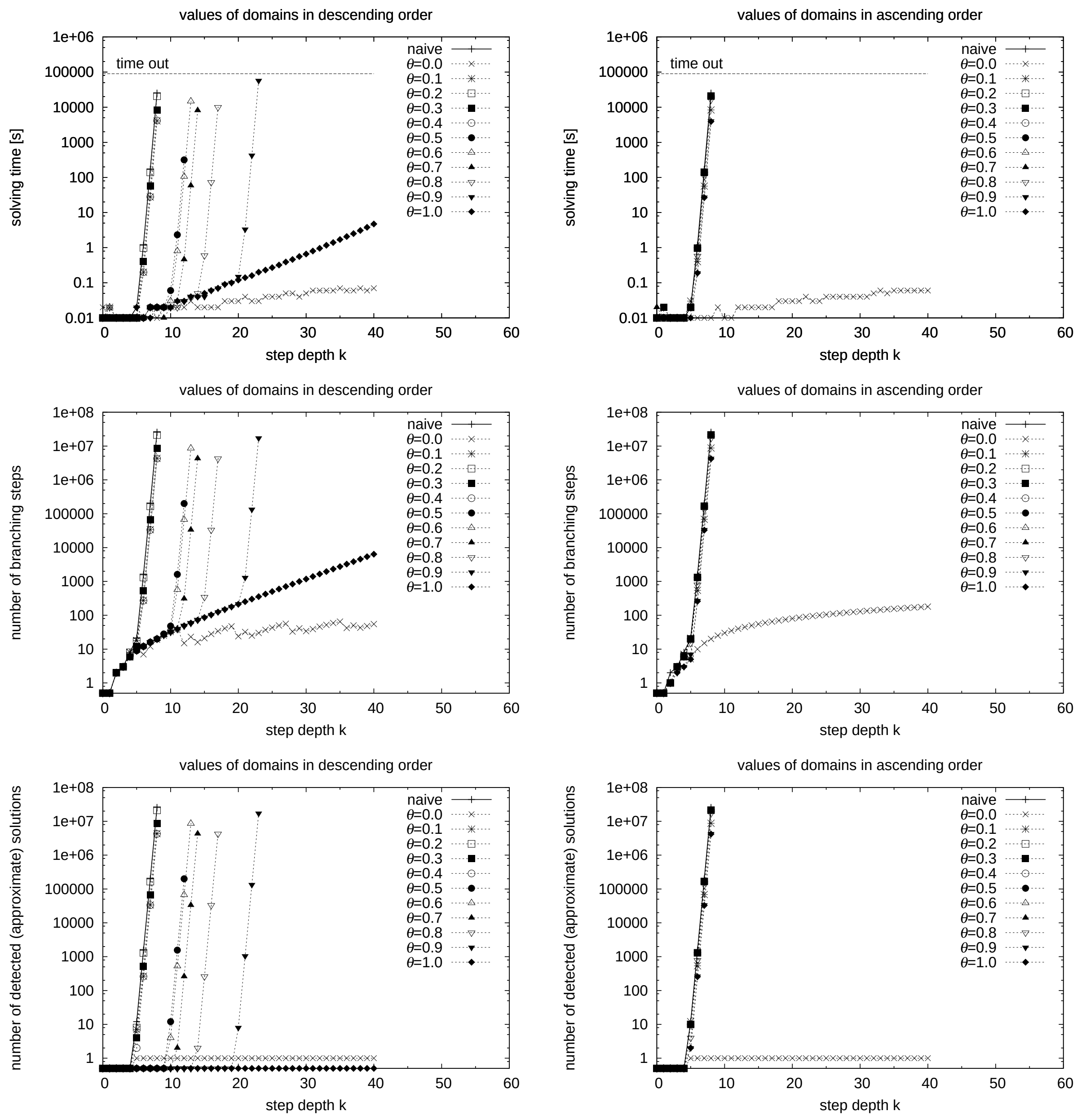
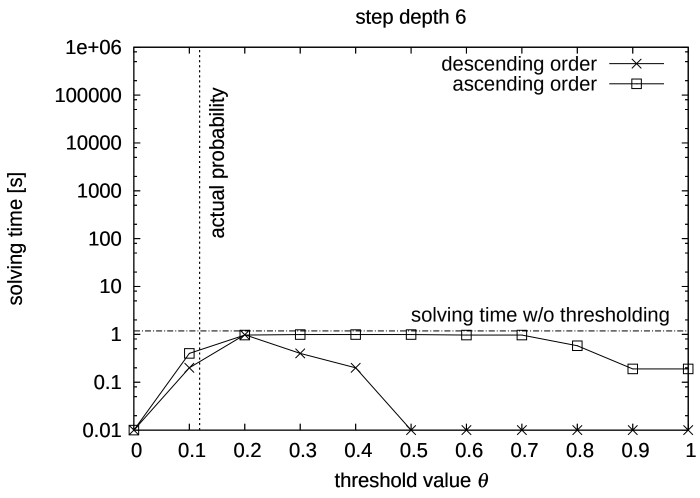
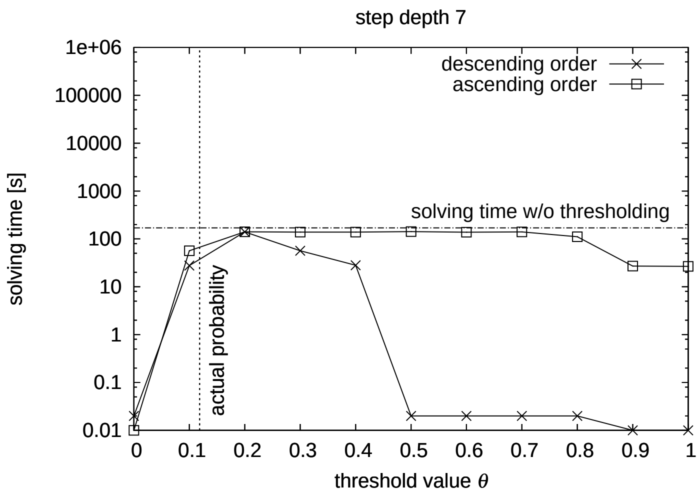


Figure 6.22: Evaluation of the SiSAT tool: *thresholding*. Solving times (top), numbers of branching steps (middle), and numbers of detected (approximate) solutions (bottom) for the corresponding step depths when thresholding with different threshold values θ was applied, where the values of the domains of the quantified variables were sorted in descending (left) and in ascending (right) order.

reduces to a factor of about 3 (at all these depths), while the benefit of thresholding for threshold 0.2 is almost vanished, namely to a speed-up factor of about 1.2. Considering the ascending order, the same speed-ups as for the descending order were obtained for threshold value 0, while SiSAT is about 3 times faster than the naive SSMT approach for threshold 0.1. The performance for all thresholds 0.2 to 0.8 is notably poor, where a speed-up factor of roughly 1.2 was observed. Only for thresholds 0.9 and 1, solving times





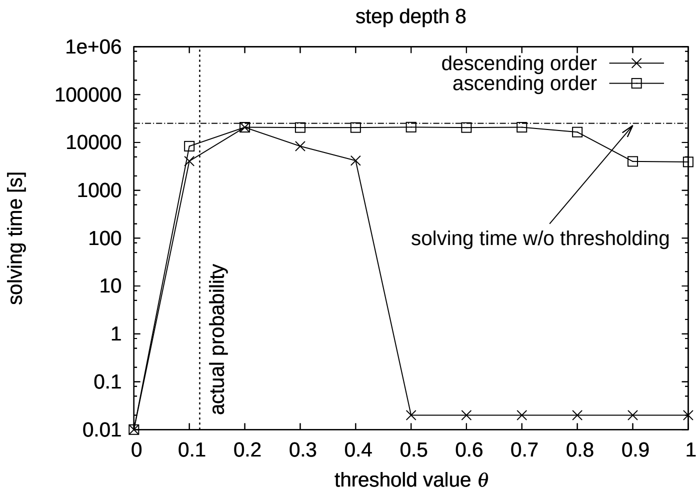


Figure 6.23: Evaluation of the SiSAT tool: *thresholding*. Solving times for the corresponding threshold values θ at step depths 6 (top), 7 (middle), and 8 (bottom) when thresholding with different threshold values and with two different static value orderings was applied.

improve to a speed-up factor of about 6.

Though the solving times for both value orderings differ strongly, in particular for threshold values $\theta \geq 0.3$, Figure 6.23 suggests the assumption that thresholding is becoming more powerful when the threshold value is moving away from the actual satisfaction probability, while its impact is becoming negligible if the threshold value is approaching the actual satisfaction probability. It moreover seems that thresholding is not as effective if the threshold is greater than the actual probability of satisfaction as it is in the reverse case. Considering the descending order, though the *distance* between threshold 0.2 and the actual satisfaction probability, which lies in the interval $[0.11847935, 0.11866184]$ for all depths 6, 7, and 8, is more than 4 times *greater* than the distance between threshold 0.1 and the actual probability, the *solving time* is about 5 times *slower*.

The latter circumstance can be attributed to the presence of *existential* quantifiers. Recall that thresholding is feasible for existential as well as randomized variables whenever the intermediate (weighted) probability result has exceeded the upper threshold, confer rules SSMT.3 and SSMT.4, which we refer to as "positive" thresholding in the sequel. That is, if the probability result of some branch of an existential variable is greater than the upper threshold then investigation of all the remaining branches is skipped by rule SSMT.3. Most likely, such latter situations were discovered when solving the SSMT problem for threshold value 0.1 since the actual satisfaction probability is greater than 0.1. However, a similar rule being able to recognize that the probability results of all the remaining branches cannot be large enough to reach the lower threshold, as being realized for randomized variables by rule SSMT.4 and being referred to as "negative" thresholding, is infeasible for existential variables due to computing the maximum of the probability results over all branches. Consequently, all of the remaining branches of an existential variable must be explored to decide that a lower threshold cannot be reached, as it is the case when solving the SSMT problem for threshold value 0.2.

The above fact clearly holds for all thresholds greater than the actual satisfaction probability. The reason why the SSMT problems where the domains of the quantified variables are sorted in descending order were solved much faster for threshold values $\theta \geq 0.5$ is as follows: if the lower threshold is sufficiently far away from the actual probability then "negative" thresholding for randomized variables potentially applies much earlier, i.e. on a much lower level in the search tree, such that processing the existential variables are of no significant consequence with respect to solving time.

As already observed above, the effectiveness of thresholding is very sensitive to the value ordering. In the remainder of this subsection, we try to analyze the diverse performance of thresholding for the two different static value orderings.

First of all, we need to clarify that sorting the domains of the quantified variables in descending order *cannot* lead to a powerful branching heuristics in general, and in particular not in the context of SSMT-based probabilistic bounded model checking, though the experimental results might suggest such an approach. The rationale is that the values of the quantified variables enumerate transitions and transition alternatives in the basic SiSAT encoding of the cooling system $\mathcal{S}_{cool}$, confer Figure 6.17, and this enumeration may be completely arbitrary. That is, if enumerating the transitions and transition alternatives in the reverse order then a descending order would be equivalent to the current ascending order, the latter exhibiting a poor performance as illustrated by Figures 6.22 and 6.23.

As a consequence, the reason why the one value ordering outperforms the other one on the SiSAT encoding of $\mathcal{S}_{cool}$ cannot be on the syntactic level but must be related to the effects of the selected transitions and transition alternatives on the system behavior. Observe that the descending and ascending orders prefer values 2 and 1, respectively, for randomized variable `pc_snsr` for each transition step, i.e. for each of the k copies of this variable with k being the step depth.

Concerning the tool behavior for threshold value 0.1 at step depths 6, 7, and 8, the descending order performs slightly better than the ascending one, confer Figure 6.23. This can be explained as follows.

We first consider the *ascending* order preferring value 1, which is associated with probability 0.12. Though selecting transition alternative 1 in automaton sensor increases the chance of reaching the target states in the short run, since transition alternative 1 models a failure of measuring the temperature T, the resulting satisfaction probability pr when selecting value 1 for the first copy of randomized variable `pc_snsr` is not large enough to exceed threshold 0.1. The latter would be the case if and only if $pr > 0.1/0.12 = 0.8\overline{3}$, since value 1 is associated with probability 0.12. Though the probability results are increasing for the following copies of `pc_snsr` when taking value 1, the corresponding upper thresholds are also growing simultaneously. However, once the other value 2 for `pc_snsr` is investigated at some lower levels in the search tree, the upper thresholds are potentially reduced, confer rule SSMT.2, and thus thresholding becomes more likely after selection of value 1 at some deeper level. The latter fact explains why thresholding with threshold 0.1 for the ascending order is nevertheless more efficient than the naive approach.

We next examine the *descending* order preferring value 2, which is associated with probability 0.88. Though taking transition alternative 2 does not increase the chance of reaching the target states in the short run, the resulting satisfaction probability pr carries more weight due to the higher probability of 0.88. That is, “positive” thresholding applies for the first copy of `pc_snsr` when selecting value 2 if and only if $pr > 0.1/0.88 = 0.11\overline{36}$. Moreover, the upper thresholds are *not* growing at deeper levels of the search tree as much as they are doing when preferring value 1, which is due to the higher probability of 0.88, such that thresholding is more likely at deeper levels.

The above consideration sheds a bit more light on the observation that the descending order performs slightly better than the ascending one for threshold value 0.1.

We finally analyze the divergent solving times for both static value orderings with respect to threshold values 0.2 to 1 at step depths 6, 7, and 8. The descending order again performs better than the ascending one, in particular for all threshold values $\theta \geq 0.5$ with the highest speed-ups of six orders of magnitude being observed for thresholds 0.5, 0.6, and 0.7 at step depth 8.

The reason for this behavior again is due to preferring a particular value when branching for randomized variable `pc_snsr`. If selecting value 1 or 2 first then “negative” thresholding, confer rule SSMT.4, is applicable if and only if the thresholding conditions $0.12 \cdot pr + 0.88 < \theta_l$ or $0.88 \cdot pr + 0.12 < \theta_l$, respectively, are satisfied with pr being the probability result of branch “`pc_snsr` = 1” or “`pc_snsr` = 2”. Obviously, the latter conditions hold if and only if $pr < (\theta_l - 0.88)/0.12 =: ub_1(\theta_l)$ or $pr < (\theta_l - 0.12)/0.88 =: ub_2(\theta_l)$, respectively. The functions $ub_1(\theta_l)$ and $ub_2(\theta_l)$, mapping lower thresholds θ_l to strict upper bounds which must not be violated by pr in order that “negative” thresholding be

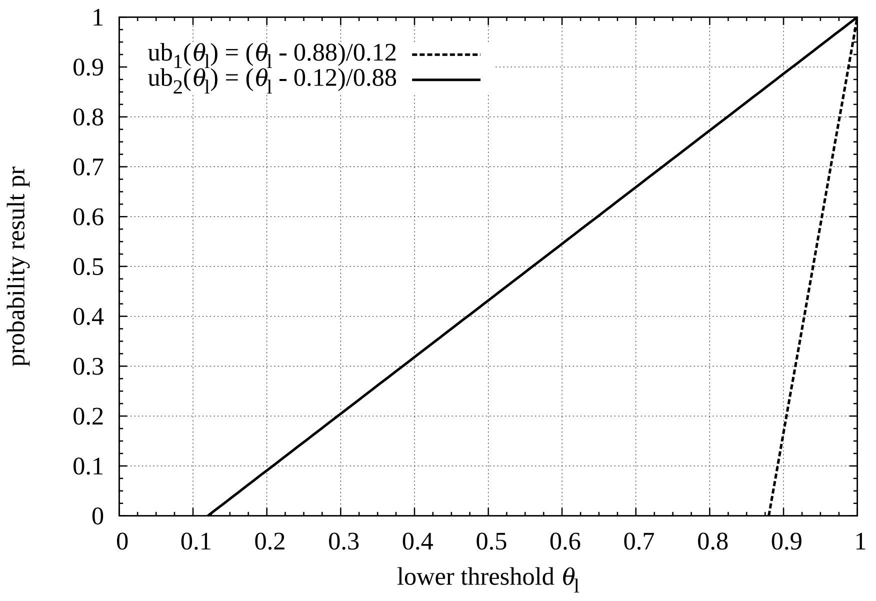


Figure 6.24: Graphical representation of functions $ub_1(\theta_l)$ and $ub_2(\theta_l)$ mapping lower thresholds θ_l to strict upper bounds which must not be violated by the probability result pr of the first branch in order that “negative” thresholding be feasible for θ_l.

feasible for θ_l, are plotted in Figure 6.24.

The graphs of Figure 6.24 illustrate well why the descending order outperforms the ascending one with respect to threshold values 0.2 to 1. If preferring value 1, as done by the *ascending* order, then “negative” thresholding is never applicable for each lower threshold $\theta_l \leq 0.88$ whatever the probability result pr for branch “pc_snsr $= 1$” actually amounts to, i.e. $0.12 \cdot pr + 0.88 < \theta_l$ is never satisfied for $\theta_l \leq 0.88$. With regard to the latter fact, confer the dashed line in Figure 6.24 representing the corresponding upper bounds $ub_1(\theta_l)$ for pr. If however preferring value 2, as done by the *descending* order, then “negative” thresholding is more likely to succeed. For instance, if $\theta_l = 0.7$ then pr may be relatively large, namely less than $pr = 0,65\overline{90}$, confer the solid line in Figure 6.24 representing the corresponding upper bounds $ub_2(\theta_l)$ for pr.

As shown by Figure 6.23, thresholding with thresholds $\theta \geq 0.2$ for the ascending order is nevertheless more efficient than the naive approach, in particular for larger thresholds 0.9 and 1. This can be attributed to the following: once the other value 2 is investigated at some lower levels in the search tree, the lower thresholds are potentially increased at some deeper levels such that “negative” thresholding becomes more likely after selection of value 1. Let us briefly elaborate on how lower thresholds can grow during SSMT proof search: first, the lower threshold $\theta_l' = \max(pr, \theta_l)$ for the second branch of rule SSMT.1 is increased if and only if $pr > \theta_l$ and, second, the lower threshold $\theta_l' = (\theta_l - p_{remain})/p_v$ for the first branch of rule SSMT.2 as well as of rule SSMT.4 is increased if and only if $p_{remain} < (1 - p_v) \cdot \theta_l$, for instance, in case $p_{remain} = 0$, $p_v < 1$, and $\theta_l > 0$.

Concluding this subsection, we have shown that *thresholding* can lead to tremendous performance gains, sometimes by orders of magnitude, while its effectiveness is very sensitive to the value ordering.

6.7.3 Impact of thresholding and activity-based value branching heuristics

In the previous subsection, we have seen that the effectiveness of thresholding strongly depends on the order in which possible assignments to the quantified variables are probed, as some values trigger earlier applications of thresholding. We have investigated two different static value orderings for the quantified variables, namely a *descending* and an *ascending* order. While thresholding for the descending order has led to tremendous performance gains on the considered example, the benefit of thresholding for the ascending order was negligible in many cases. We furthermore have discussed that such static value orderings cannot result in powerful branching heuristics in general. That is to say, the performance of thresholding using the descending order on the basic SiSAT encoding of the cooling system $\mathcal{S}_{cool}$ cannot be attributed to the static value ordering alone but rather to the semantic effects of the selected values.

Since it is hard to extract such semantic effects of values from the given SSMT formula prior to actually solving the SSMT formula, we have devised an *activity-based value branching heuristics* in Subsection 6.5.1 with the objective of monitoring the semantic effects of values during the SSMT proof search. That is, for each value v in the domain of each quantified variable x, a heuristic measure $hm(v, x)$ is maintained for the purpose of estimating the satisfaction probabilities of the corresponding SSMT subproblems. More precisely, $hm(v, x)$ keeps track of the arithmetic mean of the already computed probability results for branch "$x = v$". Using this heuristic measure, the algorithm is potentially able to detect promising values for branching such that thresholding is more likely to succeed.

While for *existential* variables it is only reasonable to select values with *highest* measure as only "positive" thresholding is available, we may pursue two strategies for *randomized* variables x:

1. If aiming at exceeding upper thresholds then values v with *highest* weighted measure $p_v \cdot hm(v, x)$ are preferred.

2. If aiming at missing lower thresholds then values v are selected for which the term $p_v \cdot hm(v, x) + p_{remain}$ is *minimal*, where p_v is the probability of setting x to v and p_{remain} is the probability mass of the remaining values for x.

In order to evaluate the activity-based value branching heuristics, we have rerun the experiments of Subsection 6.7.2 for both of above strategies. In particular, we performed the experiments on both static value orderings since the heuristic measure $hm(v, x)$ is initially 0 for all values v and for all quantified variables x. That is to say, for both of the strategies 1 and 2, it may happen that the maximum or minimum, respectively, is associated with several values. In such cases, the first value is taken which in turn means that the initial value ordering potentially produces an effect on the performance of the SSMT proof search. The corresponding solving times of SiSAT are plotted in Figure 6.25.

With regard to the SiSAT input file where the domains are initially sorted in *descending* order, we observe that application of the activity-based value branching heuristics for both of above strategies yields roughly the same solving times as in the solver setting without this dynamic branching heuristics. When running SiSAT on the input file where the domains are in *ascending* order initially, the performance of the tool is not enhanced

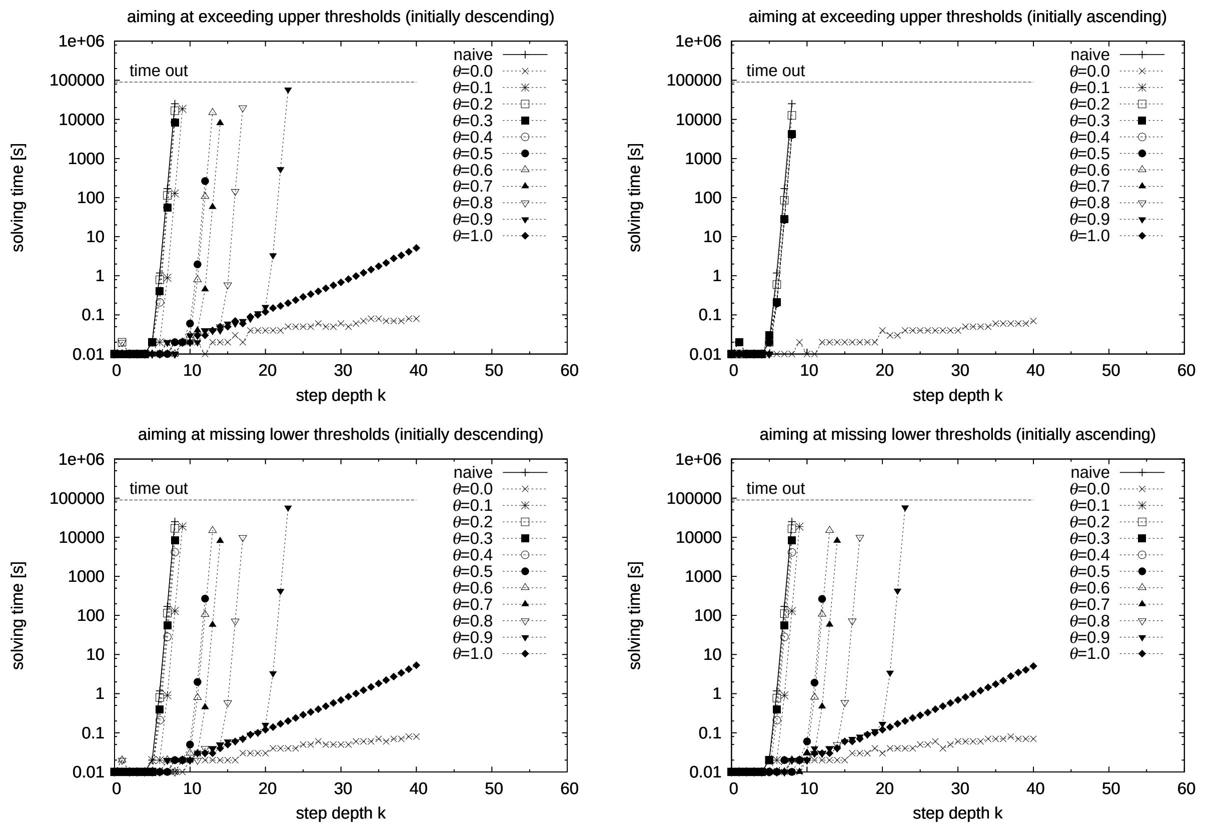


Figure 6.25: Evaluation of the SiSAT tool: *thresholding* together with *activity-based value branching heuristics*. Solving times for the corresponding step depths when thresholding with different threshold values θ and activity-based value branching heuristics with strategies of exceeding upper thresholds (top) and of missing lower thresholds (bottom) were applied, where the values of the domains of the quantified variables were initially sorted in descending (left) and in ascending (right) order.

considerably for above strategy 1. However, if employing above strategy 2 then similar speed-ups as for the descending order could be achieved.

These results confirm the intended purpose of the activity-based value branching heuristics, at least on the considered example: on the one hand, the heuristics does not worsen the tool behavior on the well-performing descending value ordering and, on the other hand, tremendous speed-ups are obtained for the poorly performing ascending order when aiming at missing lower thresholds. With regard to the diverse performance of strategies 1 and 2 in case the domains are in ascending order initially, we remark that the maximum probabilities of reaching the target states are less than 0.4 for all step depths up to 30, while the reachability probability at depth 31 lies in the interval $[0.40115867, 0.40115868]$ and thus slightly exceeds value 0.4, confer Figure 6.19. This explains why strategy 2, which aims at missing lower thresholds, outperforms strategy 1, which is designed to perform well when upper thresholds are exceeded.

To analyze the behavior of both strategies in more detail, the solving time is plotted over the threshold value θ for step depths 6, 7, and 8 in Figures 6.26 and 6.27, where the domains of the quantified variables are initially sorted in descending and in ascending

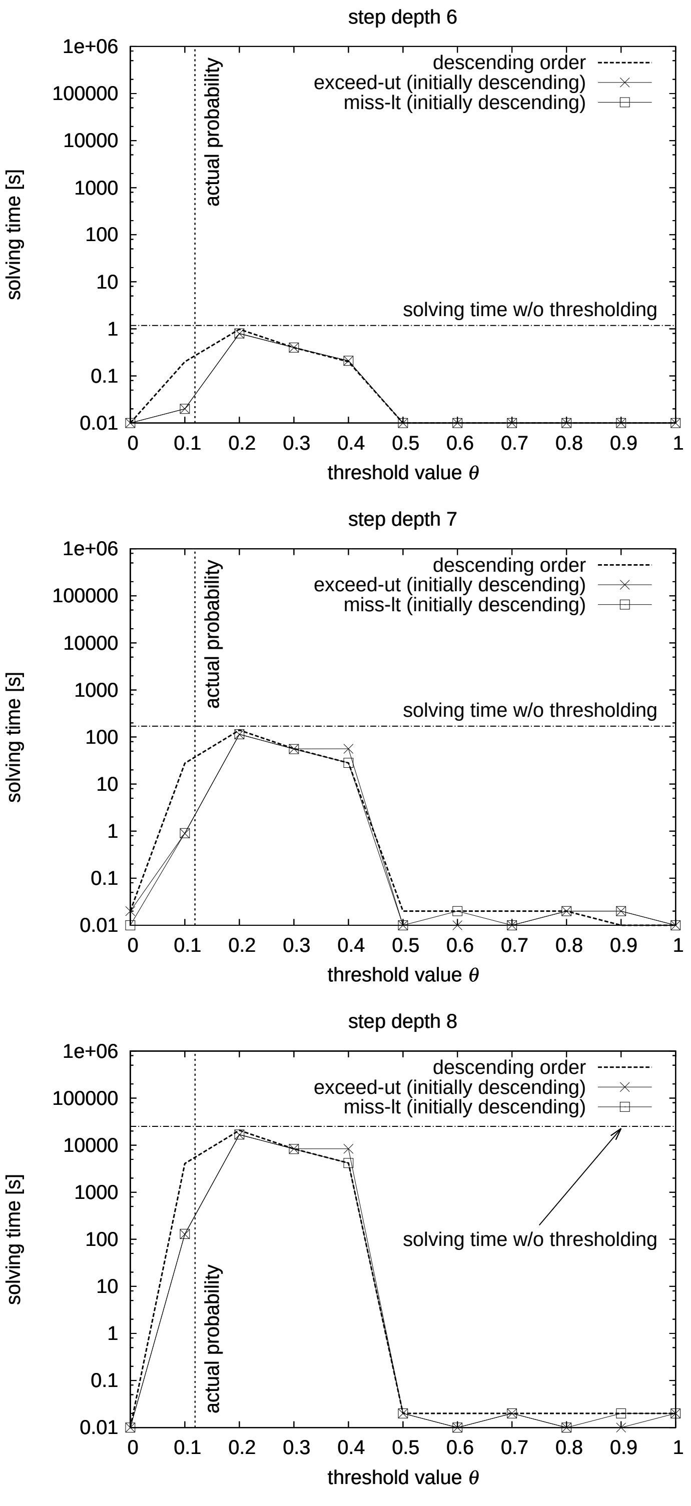


Figure 6.26: Evaluation of the SiSAT tool: *thresholding* together with *activity-based value branching heuristics*. Solving times for the corresponding threshold values θ at step depths 6 (top), 7 (middle), and 8 (bottom) when thresholding and activity-based branching heuristics with strategies of exceeding upper thresholds (exceed-ut) and of missing lower thresholds (miss-lt) were applied, where the domains of the quantified variables were initially sorted in *descending* order.

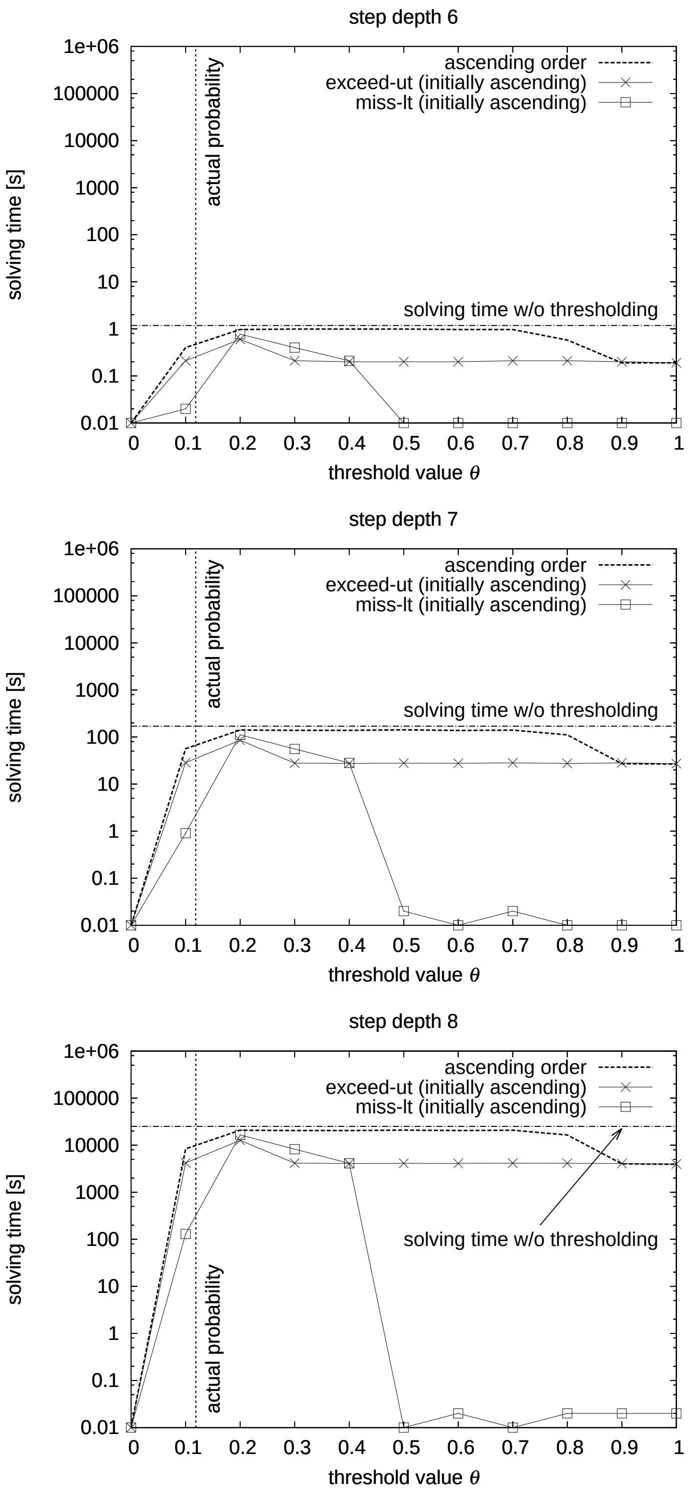


Figure 6.27: Evaluation of the SiSAT tool: *thresholding* together with *activity-based value branching heuristics*. Solving times for the corresponding threshold values θ at step depths 6 (top), 7 (middle), and 8 (bottom) when thresholding and activity-based branching heuristics with strategies of exceeding upper thresholds (exceed-ut) and of missing lower thresholds (miss-lt) were applied, where the domains of the quantified variables were initially sorted in *ascending* order.

order, respectively.

In case the domains of the quantified variables are initially sorted in *descending* order, the performance of SiSAT employing the activity-based value branching heuristics could not be increased considerably for all thresholds, confer Figure 6.26. For threshold value 0.1, however, both strategies yield speed-ups of a factor of 10 as well as of more than 30 at step depth 6 as well as at depths 7 and 8, respectively, compared to the static value ordering and even of a factor of 59 as well as of about 190 compared to the naive algorithm. It is remarkable that strategy 2, though being designed to prefer values such that lower thresholds will be missed, works well for $\theta = 0.1$ on this example. Another notable observation is that strategy 1 is also competitive for all threshold values $\theta \geq 0.2$, except for $\theta = 0.4$ at step depths 7 and 8 where the solving time is twice as large as in both other settings.

Let us now consider the case in which the domains of the quantified variables are initially sorted in *ascending* order, confer Figure 6.27. With regard to threshold value $\theta = 0.1$, though strategy 1 yields a speed-up factor of about 2 compared to the static value ordering at all step depths 6 to 8, a speed-up factor of about 20 was achieved for strategy 2 at depth 6 and of more than 60 at depths 7 and 8, again compared to the static ordering. The latter circumstance can be attributed to the fact that the heuristic measures $hm(v,x)$ are defined to be 0 initially. As mentioned above, whenever strategy 1 or 2 cannot determine a unique value then the first value is taken. As a consequence, the initial value ordering has an impact on the dynamic branching heuristics. In our example, when branching for randomized variable `pc_snsr` on condition that $hm(1, \texttt{pc\_snsr}) = 0$ and $hm(2, \texttt{pc\_snsr}) = 0$ then strategy 1 prefers value 1 since a unique value cannot be determined and the domains are in ascending order initially. On the same assumption, strategy 2 however selects value 2 since $0.88 \cdot 0 + 0.12 < 0.12 \cdot 0 + 0.88$.

As discussed in the latter part of Subsection 6.7.2, due to selection of value 2 when branching for randomized variable `pc_snsr`, in particular on earlier levels of the search tree, "positive" thresholding becomes more likely. It thus seems that the disadvantage of taking the "worse" value 1 at lower levels of the search tree cannot be compensated for by strategy 1 during the remaining proof search. One might argue that strategy 1 thus should take more into account the probabilities p_v of the values v for randomized variable x whenever the corresponding heuristic measures $hm(v,x)$ still carry their initial value 0, i.e. if their counters $cnt_{v,x}$ are still 0. For instance, the latter could be realized by defining $hm(v,x) = 1$ if $cnt_{v,x} = 0$. Such a modification would clearly help in the considered example of the cooling system $\mathcal{S}_{cool}$ but is as arbitrary as the current initialization of $hm(v,x)$. Moreover, there obviously are examples of SSMT problems where this modified initialization does not enhance proof search, namely if values associated with high probabilities lead to very small satisfaction probabilities but values associated with low probabilities to very high probability results. For instance, let us assume that value 2 associated with high probability 0.88 just yields satisfaction probability 0.07 but value 1 associated with rather low probability 0.12 gives the high probability result 0.95. Let furthermore be given an upper threshold $\theta_u = 0.1$. Then, "positive" thresholding becomes applicable after investigating value 1 first since $0.12 \cdot 0.95 = 0.114 > \theta_u = 0.1$, while the same is not true when taking value 2 first since $0.88 \cdot 0.07 = 0.0616 < \theta_u = 0.1$.

That is to say, an arbitrary choice for the initial value of the heuristic measure $hm(v,x)$

never performs well in general. Concerning the latter issue, a promising direction for future research is to devise powerful heuristics for determining "good" initial values of $hm(v, x)$ for all v and x. One idea here might be the following. First, some, preferably relevant assignments to the quantified variables are selected such that each value v of each quantified variable x should be covered by at least one of the assignments. Then, all these assignments are traversed in a depth first search manner according to the quantifier prefix, thereby computing the resulting probabilities at all visited nodes of the search tree spanned by the selected assignments. These latter probabilities can then be exploited to determine potentially good choices for the initial values of the heuristic measure $hm(v, x)$, for instance by taking the arithmetic mean of all the probabilities for branch "$x = v$". Note that the runtime of the above approach as well as the significance of the initial values of the heuristic measures obtained most likely depend on the number of selected assignments. More precisely, it can be expected that the fewer assignments considered the shorter runtime achieved but the more assignments considered the more significant initial values of the heuristic measures obtained. It thus will be important to find a suitable number of assignments such that the runtime is acceptable and the determined initial values are still helpful.

It remains to consider the performance of the activity-based value branching heuristics for threshold values $\theta \geq 0.2$ in case the domains of the quantified variables are initially sorted in ascending order, confer Figure 6.27. Here the behavior is as expected: strategy 2, aiming at missing lower thresholds, outperforms strategy 1 in almost all cases, the latter aiming at exceeding upper thresholds. An outlier is observed for threshold value 0.3, where strategy 1 solved the problem in half the time of strategy 2 at all depths 6, 7, and 8. Compared to the static value ordering, strategy 2 yields tremendous speed-ups, for instance, of six orders of magnitude for threshold $\theta = 0.6$ and of five orders for $\theta = 1$ both observed at step depth 8. Also strategy 1 is rather powerful with respect to the static ordering for thresholds 0.3 up to 0.8, for instance, with a speed-up factor of about 5 for threshold $\theta = 0.6$ at all depths 6 to 8.

To summarize this subsection, we conclude that the suggested *activity-based value branching heuristics* can further improve thresholding, in some cases by orders of magnitude.

6.7.4 Impact of purification

The next algorithmic enhancement we consider is *purification*, as introduced in Subsection 6.5.2. Recall that the naive SiSAT approach shows a particularly bad performance from step depth 5 on. This behavior has been attributed to the following: after 5 system steps, the cooling system $\mathcal{S}_{cool}$ reaches the target states, and whenever a target state is reached then a self loop is executed meaning that the remaining non-quantified system variables keep their values independent of the actual values of the non-instantiated quantified variables. This implies that the naive SSMT algorithm must explore all assignments to these "unconstrained" quantified variables in worst case, while the number of such assignments is exponential in the number of iterations of this self loop, confer Subsection 6.7.1.

The idea of purification can now be exploited in above situations since the matrix

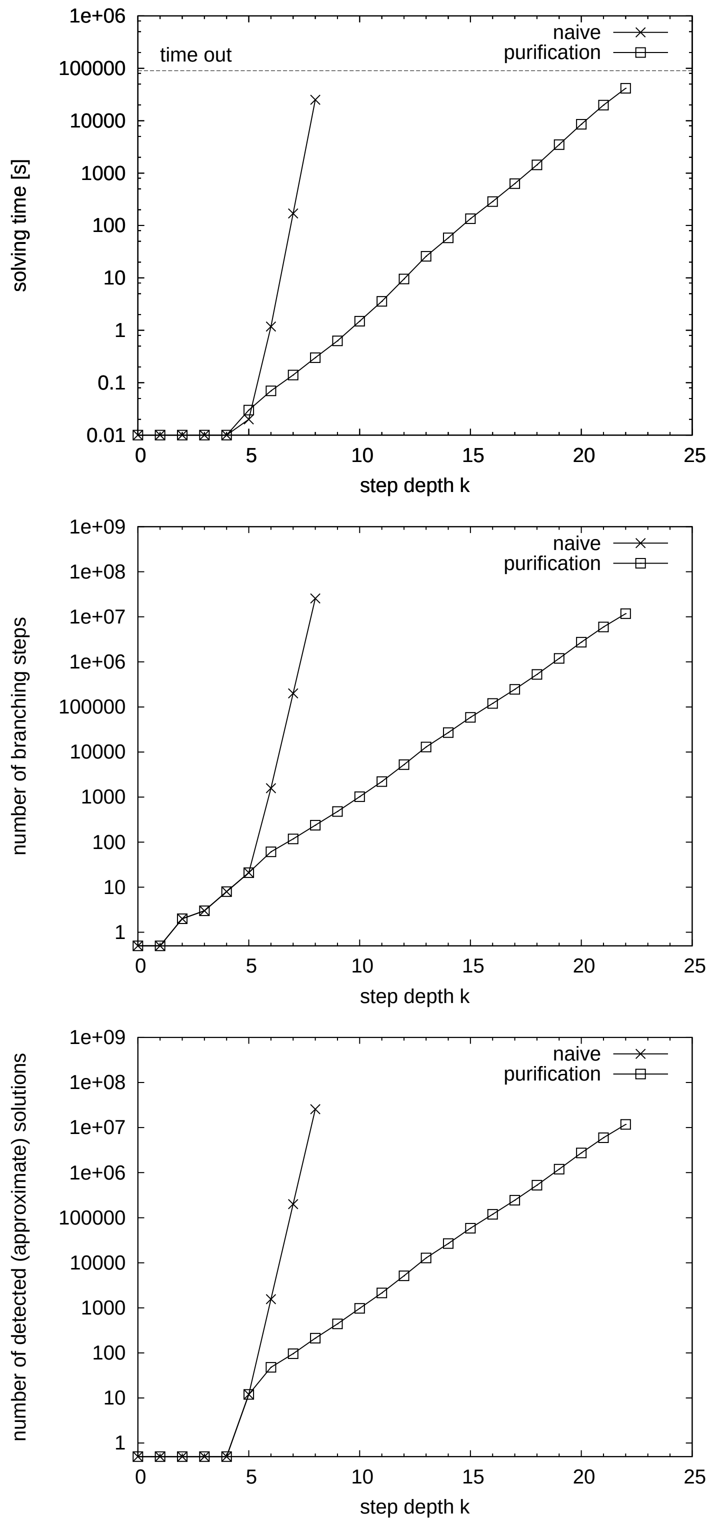


Figure 6.28: Evaluation of the SiSAT tool: *purification*. Solving times (top), numbers of branching steps (middle), and numbers of detected (approximate) solutions (bottom) for the corresponding step depths when purification was applied.

of the SSMT formula is *monotonic* and, simultaneously, *antitonic* in each such above "unconstrained" quantified variable with respect to the current interval assignment. The rationale is as follows. Consider the `TRANS` section of the basic SiSAT encoding of $\mathcal{S}_{cool}$ from Subsection 6.6.3: effects of transitions and transition alternatives are encoded by predicates of the shape

$$(\neg target \ \wedge \ x_1 = v_1 \ \wedge \ \ldots \ \wedge \ x_n = v_n) \ \Rightarrow \ \psi$$

where the Boolean variable *target* is `true` if and only if the target states are reached, the variables $x_1, \ldots, x_n$ are quantified variables, and ψ is a transition guard or assignment predicate. In our case, the number n of quantified variables is actually at most 2. Above predicates are rewritten into conjunctive form by the front end of SiSAT yielding clauses of the shape

$$(target \ \vee \ x_1 < v_1 \ \vee \ x_1 > v_1 \ \vee \ \ldots \ \vee \ x_n < v_n \ \vee \ x_n > v_n \ \vee \ \psi')$$

where ψ' is some disjunction of constraints, introduced by the generalized Tseitin transformation, confer Subsection 4.3.1, encoding predicate ψ.

It is not hard to see that the matrix of the SSMT formula is monotonic as well as antitonic in all quantified variables encoding the current system step with respect to each interval assignment σ that satisfies *target*, i.e. $\forall \tau \in \sigma : \tau \models target$. Exploiting the optimization of purification, it thus is feasible to prune the current domain $\mathcal{D}_x$ of each *existential* variable x among the above variables to a singleton $\{v\}$, where either $v = \max(\mathcal{D}_x)$ or $v = \min(\mathcal{D}_x)$ subject to whether checking for monotonicity or antitonicity, respectively, is executed first.

Recall that purification is in general impossible for randomized variables. In Subsection 6.7.6, we however see that another algorithmic enhancement, namely solution-directed backjumping, can additionally cope with randomized variables in above situations.

The corresponding experimental results, being presented in Figure 6.28, confirm our theoretical considerations above: up to step depth 5, both the naive SSMT approach and the algorithm exploiting purification exhibit identical solving behavior, while the solving time at depth 5 when using purification is slightly larger due to the additional overhead of checking for monotonicity and antitonicity. From step depth 6 on, the number of both branching steps and detected (approximate) solutions is decreasing extremely if purification is enabled, namely by one order of magnitude at depth 6, by three orders at depth 7, and by at least four orders at depth 8.[19] This heavy pruning of the quantifier tree clearly is reflected in a significant improvement of the corresponding solving times, yielding speedups compared to the naive algorithm by the same orders of magnitude as mentioned before. We finally emphasize that SiSAT with purification solved the probabilistic reachability problem up to step depth 22 within 25 hours, while the naive algorithm was just able to cope with the problem up to depth 8 within the same time limit.

6.7.5 Impact of exploiting desired accuracy of probability result

We next investigate the impact of *accuracy-based pruning* which was described in Subsection 6.5.3. The idea of this algorithmic enhancement, being similar to the one of

[19]The number of detected (approximate) solutions has actually been decreased by five orders of magnitude at step depth 8.

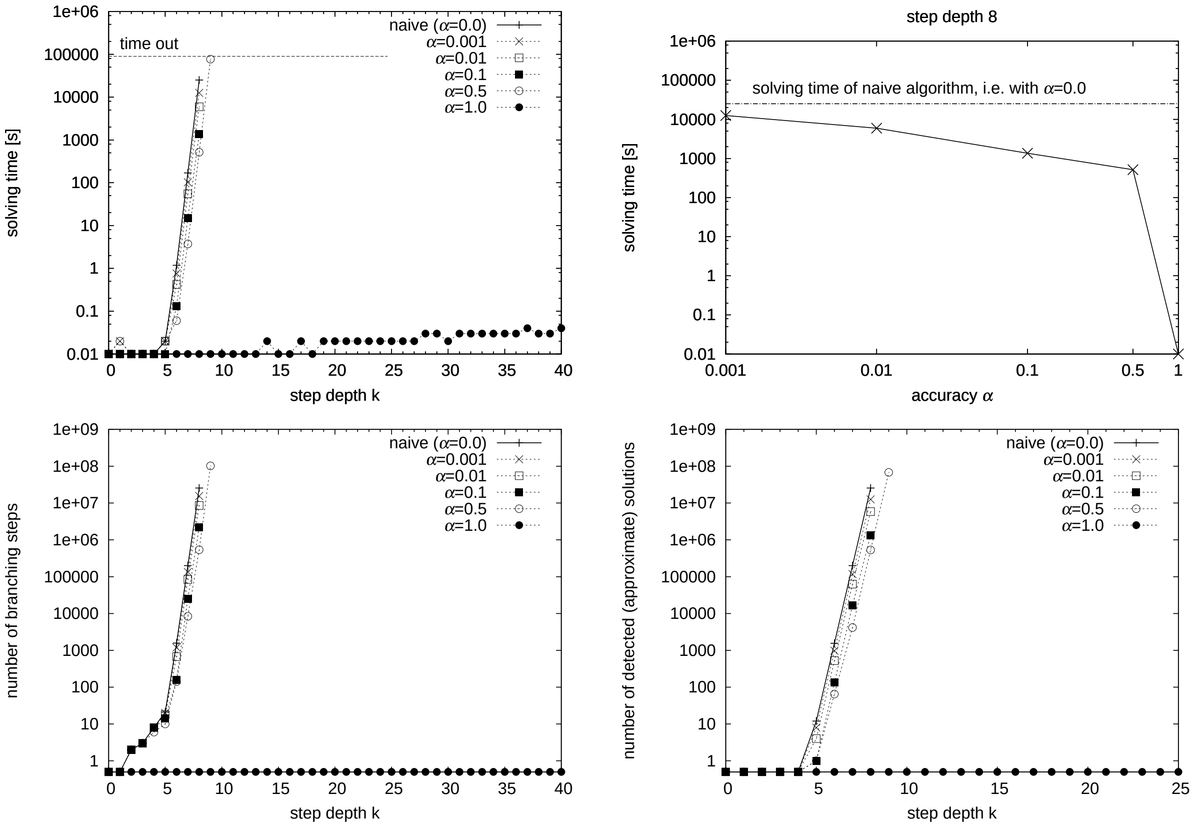


Figure 6.29: Evaluation of the SiSAT tool: *accuracy-based pruning*. Solving times (top left), numbers of branching steps (bottom left), and numbers of detected (approximate) solutions (bottom right) for the corresponding step depths as well as solving times for the corresponding accuracies α at step depth 8 (top right) when accuracy-based pruning with different values for accuracy α was applied.

thresholding to some extent, is to exploit situations in which the exact maximum probability of satisfaction is not requested for but just for a result of some predefined accuracy. More precisely, for a given *accuracy* $\alpha \geq 0$, the SSMT algorithm should return an interval $[lb, ub]$ of width at most α, i.e. $ub - lb \leq \alpha$, containing the exact satisfaction probability pr, i.e. $pr \in [lb, ub]$.

Figure 6.29 depicts the experimental results when utilizing accuracy-based pruning with different accuracies $\alpha \in \{0, 0.001, 0.01, 0.1, 0.5, 1\}$. Observe that accuracy-based pruning can be “disabled” by setting $\alpha = 0$, thus resulting in the naive SSMT approach from Subsection 6.7.1.

When looking at the results, attention might be attracted to the remarkable performance if $\alpha = 1$. However, this behavior is not a surprise at all: accuracy-based pruning immediately applies for the first quantified variable as $\alpha \geq 1$ such that no branching step must be performed to return the trivial result, namely the interval $[0, 1]$.

As it was to be expected, the experiments reveal that accuracy-based pruning becomes less beneficial when α approaches value 0: while a speed-up of six orders of magnitude compared to the naive algorithm was achieved for the meaningless case $\alpha = 1$ at step depth 8, the performance of accuracy-based pruning is getting worse for accuracies 0.5, 0.1, 0.01, and 0.001, observing speed-up factors of about 48, 18, 4, and 2, respectively, confer the top-right subfigure of Figure 6.29.

That is to say, in order to attain a major advantage in solving time, the accuracy has to be rather large which in turn causes the disadvantage of less precise results. Nevertheless, the optimization of accuracy-based pruning may help whenever just a rough insight into the evolution of the reachability probabilities over the step depth is asked for, as illustrated on the left of Figure 6.20.

6.7.6 Impact of solution-directed backjumping

We next evaluate the idea of *solution-directed backjumping* (SDB) which was introduced in Subsection 6.5.4 and formalized by rule SSMT.11. Recall the intuition behind SDB: upon having detected an (approximate) solution, quantified variables are determined which have no impact on the (approximate) solution. For such variables, investigation of their alternative values is skipped by means of assigning the probability result of the current subproblem directly to all remaining subproblems, confer Figure 6.9.

As explained in Subsections 6.7.1 and 6.7.4, whenever a target state is reached in the cooling system $\mathcal{S}_{cool}$ after k system steps, i.e. if the Boolean variable `target` at depth k is `true`, then all quantified variables encoding the selection of transitions and transition alternatives for all the remaining k' system steps become "unconstrained". Moreover, each assignment to these "unconstrained" quantified variables leads to an (approximate) solution of the formula, while the number of such assignments is exponential in k'.

We have seen that the optimization of purification is beneficial in above situations but is limited to existential variables, confer Subsection 6.7.4. SDB is furthermore able to cope with *randomized* variables in aforementioned cases. As explained above, quantified variables becoming "unconstrained" have no impact on the resulting (approximate) solution, and thus SDB can be applied for all these variables, thereby skipping investigation of exponentially many assignments.

The experimental results of Figure 6.30 reflect the theoretical considerations above. Compared to the naive SSMT algorithm, the number of detected (approximate) solutions could be reduced by one order of magnitude at step depth 6, by three orders at depth 7, and even by five orders at depth 8. The speed-ups are also considerable: one order of magnitude at depth 6, two orders at depth 7, and four orders at depth 8. Furthermore, SiSAT with SDB was able to solve the probabilistic reachability problem up to step depth 22 within the time limit of 25 hours, while the naive approach could only cope with the problem up to depth 8 within the same time. For a convenient comparison with the related optimization of purification, Figure 6.30 also shows the results for purification.

We have seen that solution-directed backjumping is a powerful mechanism to improve performance of the SSMT proof search significantly if quantified variables occur in the given SSMT formula which have no impact on detected (approximate) solutions.

6.7.7 Impact of caching probability results of subproblems

The next algorithmic enhancement to be evaluated is *caching probability results of subproblems*, as introduced in Subsection 6.5.5. The overall idea here is to avoid recomputations

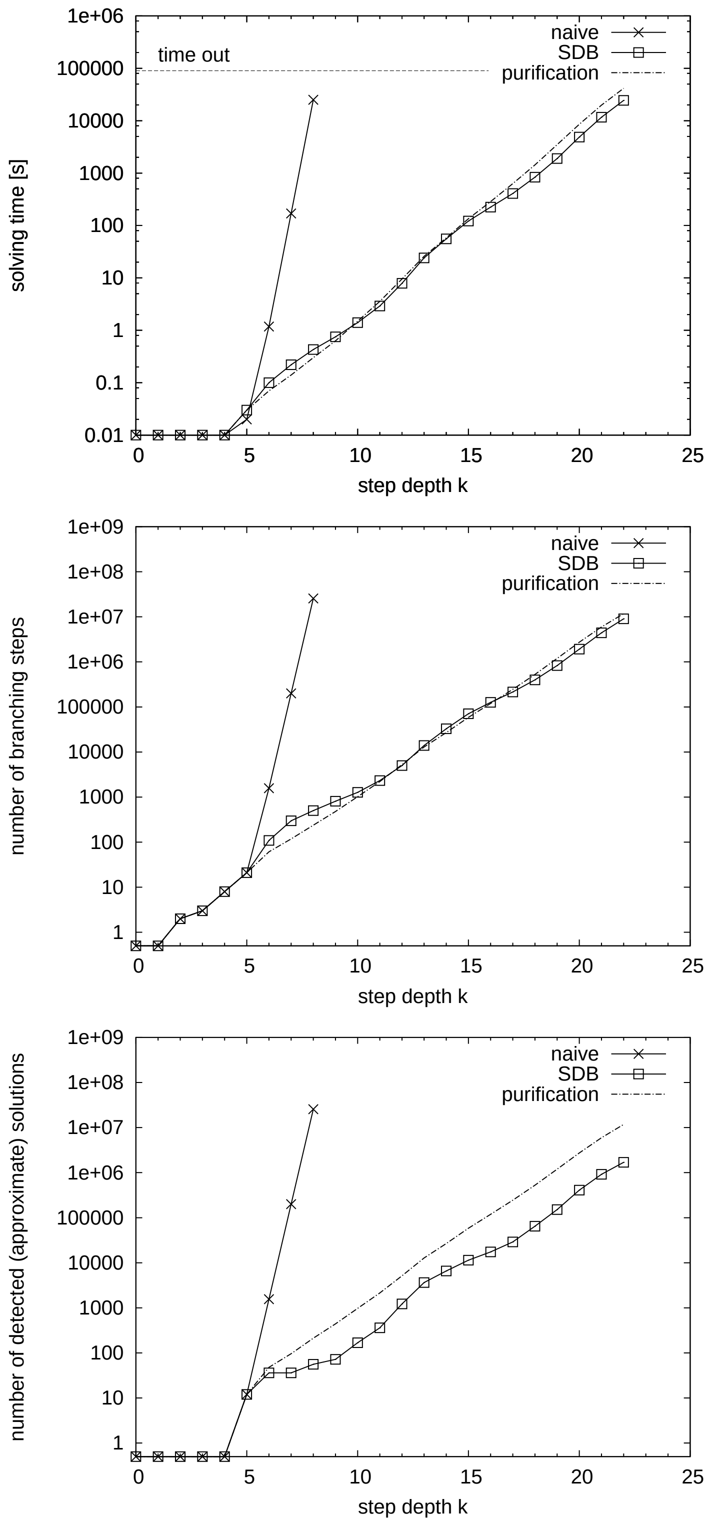


Figure 6.30: Evaluation of the SiSAT tool: *solution-directed backjumping*. Solving times (top), numbers of branching steps (middle), and numbers of detected (approximate) solutions (bottom) for the corresponding step depths when solution-directed backjumping was applied. For convenience, the results for purification are also shown.

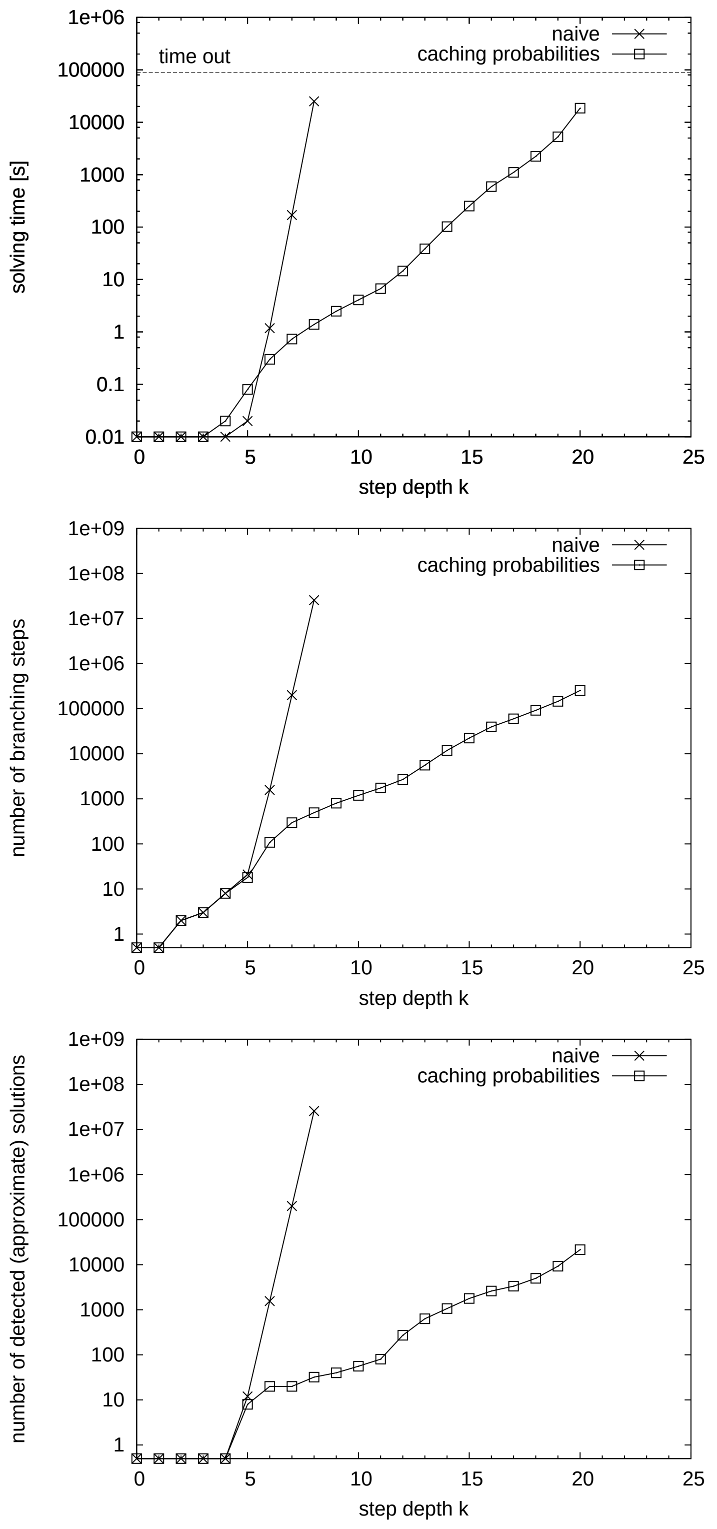


Figure 6.31: Evaluation of the SiSAT tool: *caching probability results of subproblems.* Solving times (top), numbers of branching steps (middle), and numbers of detected (approximate) solutions (bottom) for the corresponding step depths when caching probability results was applied.

of the same SSMT subformulae by means of storing and reusing probability results of subproblems. Intuitively, assume that we have already computed and cached the probability result *pr* of an SSMT subformula Φ and we encounter some SSMT subformula Ψ such that Φ and Ψ are the *same* in the sense of items 1 to 6 of Subsection 6.5.5. We may then conclude that $Pr(\Phi) = Pr(\Psi)$ which gives rise to reuse *pr* as the probability result of subproblem Ψ, in fact without solving Ψ.

The rationale why the optimization of caching and reusing probability results is expected to work for the basic SiSAT encoding of the cooling system $\mathcal{S}_{cool}$ is again due to reaching the target states at some step depth k and thereafter performing self loops for the remaining k' system steps, the latter being discussed for purification in Subsection 6.7.4 and for solution-directed backjumping in Subsection 6.7.6. More precisely, let us assume that we encounter an SSMT subproblem Φ which encodes k' iterations of a self loop as mentioned before. That is, the current partial assignment to the quantified variables corresponds to an anchored run of $\mathcal{S}_{cool}$ that reaches the target states after k system steps. As observed earlier, all non-instantiated quantified variables $x_1, \ldots, x_n$ occurring in Φ are “unconstrained” as they are contained in satisfied clauses only. That is, for each $i \in \{1, \ldots, n\}$, if taking any values $v_1, \ldots, v_i$ for quantified variables $x_1, \ldots, x_i$ then the resulting SSMT subproblems Φ_i which arise after substituting values $v_1, \ldots, v_i$ for variables $x_1, \ldots, x_i$, respectively, are the same in the above sense and thus have the same probabilities of satisfaction. The latter fact is potentially exploited by the algorithmic enhancement of caching and reusing probability results such that only one subproblem Φ_i is actually solved for each $i \in \{1, \ldots, n\}$ in best case. Observe that the number of all subproblems Φ_i for some i is exponential in i.

The latter theoretical investigation is substantiated by the experimental results presented by Figure 6.31. The SSMT proof search could be enhanced considerably if the optimization of caching and reusing probability results was utilized: compared to the naive SSMT algorithm, the number of branchings steps as well as of detected (approximate) solutions could be reduced by two as well as by three orders of magnitude at step depth 7 and by four as well as by five orders at depth 8. This clearly has led to positive effects on the solving time such that speed-ups of two and of four orders of magnitude were obtained at step depths 7 and 8, respectively. Moreover, SiSAT employing above enhancement was able to solve the probabilistic reachability problem up to step depth 20 within the time limit of 25 hours, while the naive approach could only cope with the problem up to depth 8 within the same time.

It is finally worth to mention that the algorithmic enhancement of caching probability results of subproblems is actually very memory intensive. As stated at the beginning of Section 6.7, whenever the feature of caching probabilities was activated then the corresponding experiments were conducted with the higher memory limit of 64 GByte. Though the memory limit was not exceeded in the experiments of this subsection, the amount of required memory was strongly increasing for higher step depths and has almost reached the memory limit. This high memory consumption is clearly caused by the number of subproblems which is exponential in the number of quantifiers and thus in the step depth. Runtime can also be an issue, namely the more entries are cached the more time is needed to find matching entries.

One idea for future work alleviating the above disadvantages is to maintain a storage for cached entries of a predefined, fixed amount of memory. This clearly implies that the number of cached entries is limited. In order to avoid situations where "promising" entries are detected "too late", i.e. if the storage is exhausted, it makes sense to define some heuristics for evaluating the benefit of cached entries such that more beneficial entries are kept in the storage, while less beneficial ones can be removed when necessary.

6.7.8 Impact of caching solutions

We next examine the impact of *caching solutions*, the latter being introduced in Subsection 6.5.6. The idea of this algorithmic enhancement is to cache (approximate) solutions when solving the SSMT encoding $PBMC_{\mathcal{S},\mathit{Target}}(k)$ of a probabilistic reachability problem for step depth k. Intuitively, all anchored systems runs reaching the target states within k system steps are stored. When solving SSMT problems $PBMC_{\mathcal{S},\mathit{Target}}(k')$ of larger step depths $k' > k$, the cached (approximate) solutions are reused by means of directly assigning satisfaction probability 1 to the current SSMT subproblem whenever the current partial assignment to the quantified variables coincides with a solution cached beforehand. This treatment is sound due to the following reason: whenever a target state t is visited within k system steps, the system remains in t until step depth k' is reached by performing self loops, confer Proposition 6.3.

As already discussed in the previous Subsections 6.7.1, 6.7.4, 6.7.6, and 6.7.7, all non-instantiated quantified variables occurring in SSMT formula $PBMC_{\mathcal{S},\mathit{Target}}(k')$ become "unconstrained" in each such self loop after having reached the target states within k system steps. Let τ be a cached (approximate) solution of $PBMC_{\mathcal{S},\mathit{Target}}(k)$. Then, each extension τ' of τ, i.e. if $\tau(x)$ is defined then $\tau'(x) = \tau(x)$, yields satisfaction probability 1 when solving $PBMC_{\mathcal{S},\mathit{Target}}(k')$ for $k' > k$. If the cached solution τ is not reused when solving $PBMC_{\mathcal{S},\mathit{Target}}(k')$ then the SSMT algorithm must traverse all such extensions τ'. Observe that the number of these (partial) assignments τ' is in general exponential in the number of the remaining system steps, i.e. exponential in $k' - k$. If the optimization of caching and reusing solutions is applied then this pointless overhead can be avoided.

As shown by Figure 6.32, the algorithmic enhancement of caching and reusing solutions improves the SSMT proof search significantly. While the naive approach has detected about $2 \cdot 10^5$ (approximate) solutions at step depth 7, no new (approximate) solution was detected when caching solutions. At step depth 8, the number of detected (approximate) solutions could be reduced by six orders of magnitude. With respect to solving time, speed-ups of one order, of three orders, and of five orders of magnitude were obtained at step depths 6, 7, and 8, respectively, compared to the naive SSMT algorithm. Moreover, SiSAT employing the optimization of caching solutions was able to solve the probabilistic reachability problem up to step depth 16 in less than 76 seconds.

Owing to this rather large efficiency gains up to step depth 16, one might wonder why the SSMT problem for depth 17 could not be solved within the remaining time limit of more than a day. Compared to the enhancements of purification, solution-directed backjumping, and caching probability results of subproblems, SiSAT with caching solutions being enabled was more than 9 times, more than 7 times, and more than 19 times faster, respectively, at step depth 16 but was incapable of solving the problem for depth 17. This circumstance might be attributed to the fact that the optimization of caching solutions

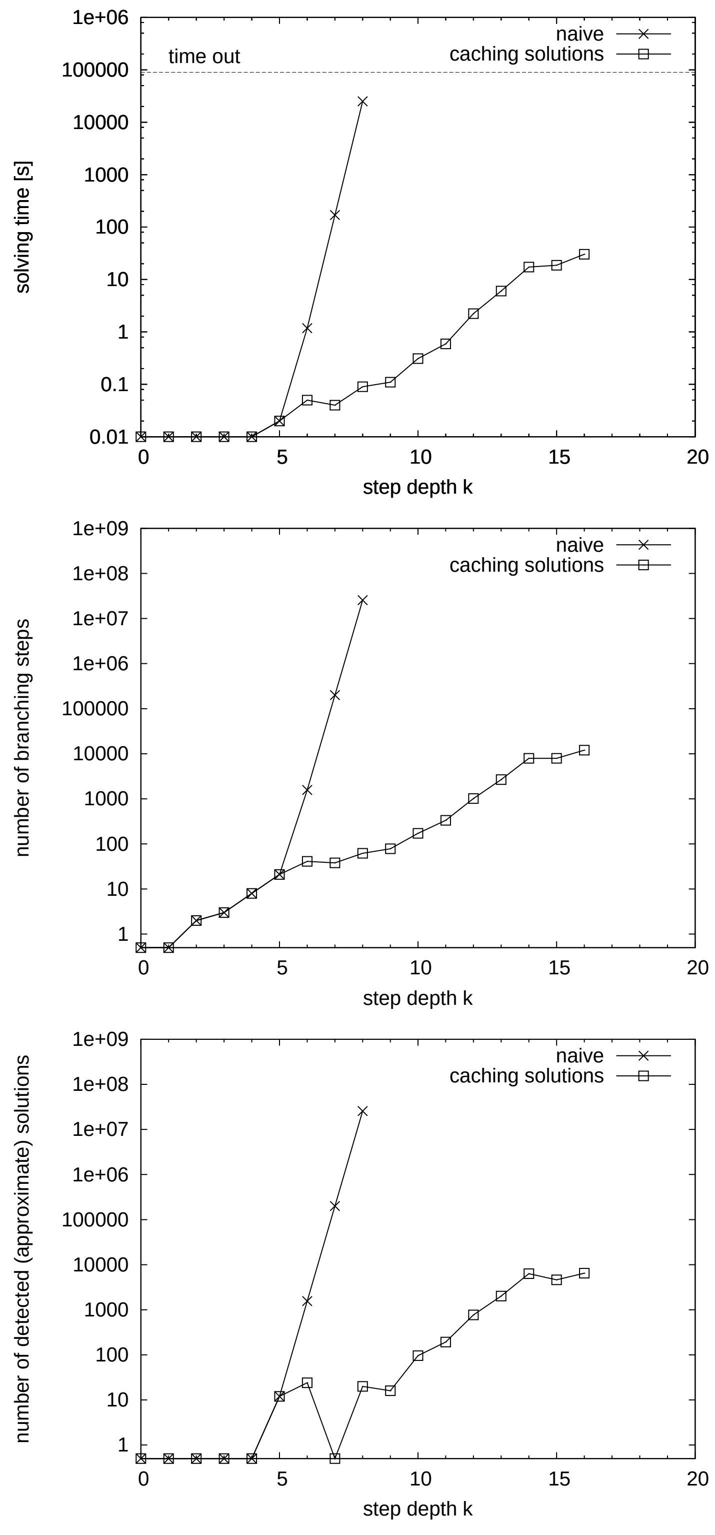


Figure 6.32: Evaluation of the SiSAT tool: *caching solutions*. Solving times (top), numbers of branching steps (middle), and numbers of detected (approximate) solutions (bottom) for the corresponding step depths when caching solutions was applied.

is more restricted in its application than the other enhancements above. More precisely, the probability result directly assigned to some subtree by means of a reused solution is always 1 while the other enhancements can cope with arbitrary probability results.

6.7.9 Impact of combinations of algorithmic enhancements

After having evaluated the impact of each single algorithmic enhancement on the basic SiSAT encoding of the cooling system $\mathcal{S}_{cool}$ in the previous subsections, we now investigate whether *combinations* thereof can lead to further performance gains.

Let us first consider the experimental results shown by Figure 6.33. As already observed in Subsection 6.7.4, *purification* has improved solving time by four orders of magnitude at step depth 8 compared to the naive SSMT algorithm, and was able to solve the probabilistic reachability problem up to step depth 22 within the time limit of 25 hours, while the naive approach could just cope with the problem up to depth 8 within the same time, confer the graph labeled with “+purification”. Roughly the same behavior was obtained when employing solution-directed backjumping (SDB) alone, confer Subsection 6.7.6.

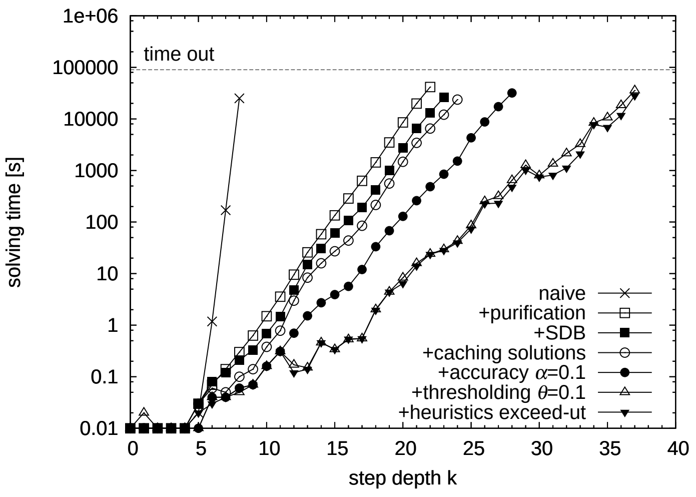
When having used *purification and SDB together*, solving time could be improved even further to a speed-up of five orders of magnitude with respect to the naive procedure at step depth 8, confer the graph labeled with “+SDB”. Moreover, the probabilistic reachability problem could be solved up to step depth 23 now. Concerning step depth 22, the combined approach exhibits speed-up factors of about 3 and of about 2 compared to SiSAT that applies purification alone and SDB alone, respectively.

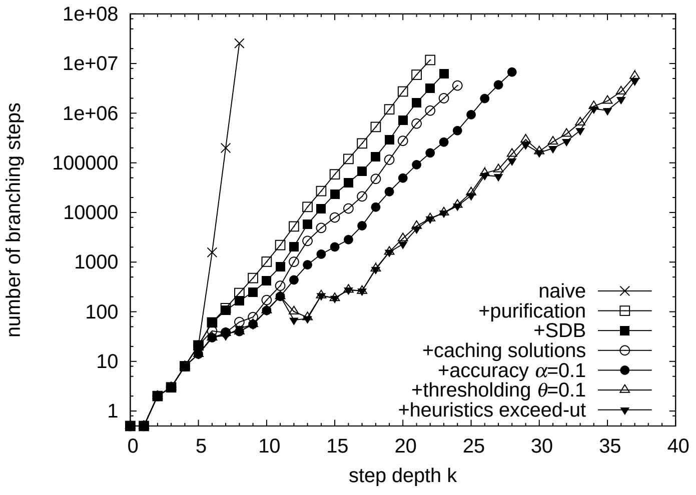
When having activated the optimization of *caching solutions* in addition to purification and SDB, solving time could be improved further by a factor of 2 at step depth 23, and the reachability problem could be solved for one more step depth within the time limit, confer the graph labeled with “+caching solutions”.

Performance gains were also achieved when *accuracy-based pruning* with accuracy $\alpha = 0.1$ was activated in above solver setting. For instance, a speed-up factor of more than 15 was observed at step depth 24 compared to the previous setting, while the probabilistic reachability problem could be coped with up to step depth 28 within the time limit, confer the graph labeled with “+accuracy $\alpha = 0.1$”.

When having enhanced the latter solver setting by *thresholding* with $\theta_l = \theta_u = 0.1$, solving time was reduced significantly, for instance, by a factor of more than 49 at step depth 28 and thus by more than an order of magnitude, confer the graph labeled with “+thresholding $\theta = 0.1$”. Moreover, the probabilistic reachability problem was solved up to step depth 37 within the time limit. Moderate speed-ups were furthermore observed when having enabled the *activity-based value branching heuristics* aiming at exceeding upper thresholds, for instance, by factors of almost 2 at step depth 32 and of about 1.25 at depth 37, confer the graph labeled with “+heuristics exceed-ut”.

Finally, we investigate the impact of *caching probability results of subproblems* in combination with other algorithmic enhancements. As shown in Subsection 6.7.7, the optimization of caching probabilities alone has improved the naive SSMT proof search by orders of magnitude, confer Figure 6.31. While consecutive activation of the above algorithmic enhancements has led to performance gains in our experiments, as illustrated in Figure 6.33, the same was not true for the feature of caching probabilities. Some of the





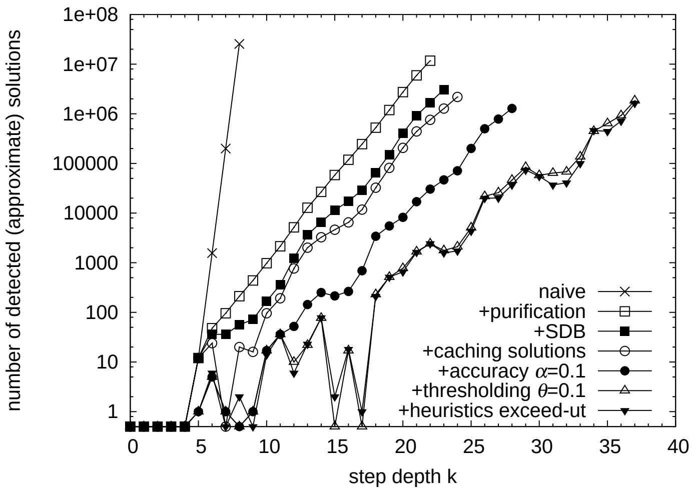


Figure 6.33: Evaluation of the SiSAT tool: *combinations of algorithmic enhancements*. Solving times (top), numbers of branching steps (middle), and numbers of detected (approximate) solutions (bottom) for the corresponding step depths when different combinations of the enhancements were applied. The naive SSMT algorithm is denoted by naive while +feature means the use of enhancement feature in addition to the solver setting stated in the preceding line.

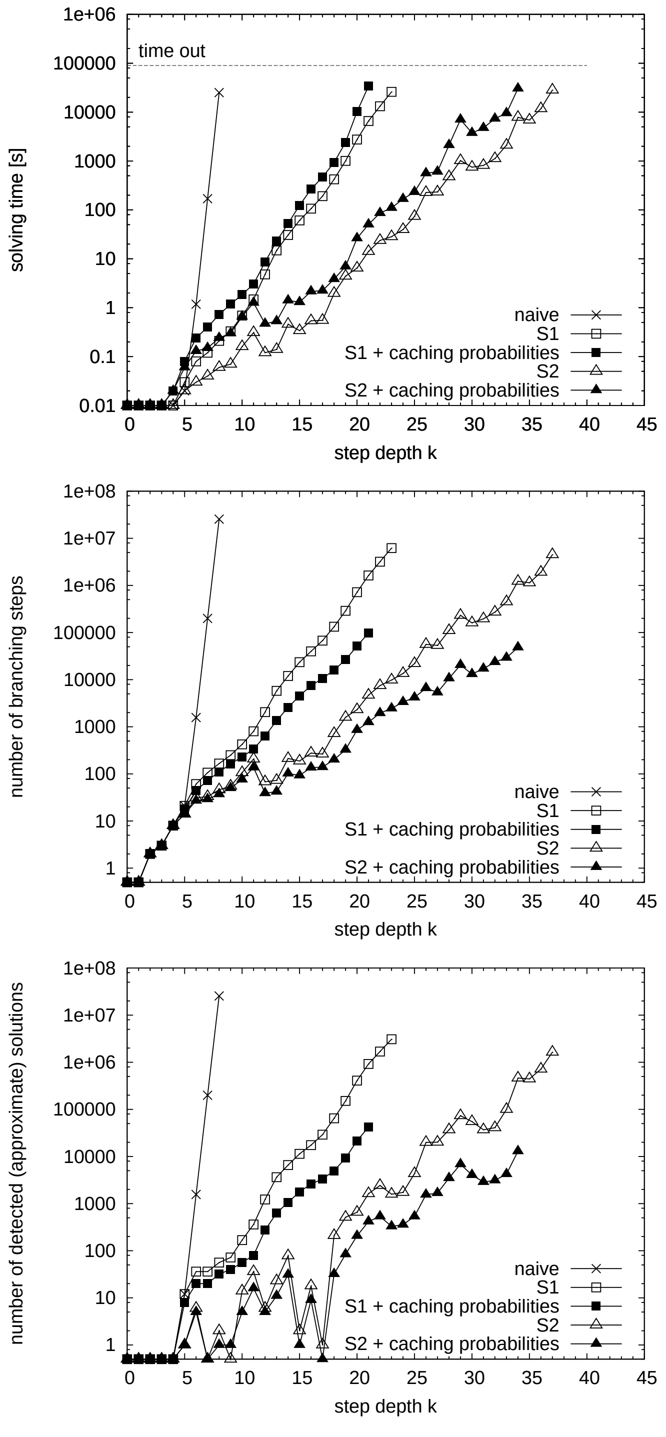


Figure 6.34: Evaluation of the SiSAT tool: *combinations of algorithmic enhancements*. Solving times (top), numbers of branching steps (middle), and numbers of detected (approximate) solutions (bottom) for the corresponding step depths when two combinations of algorithmic enhancements with and without caching probability results were applied. The solver setting S1 involves purification and SDB, while setting S2 extends S1 by caching solutions, accuracy-based pruning with $\alpha = 0.1$, thresholding with $\theta_l = \theta_u = 0.1$, and activity-based branching heuristics aiming at exceeding upper thresholds.

corresponding experimental results are presented in Figure 6.34.

With regard to solver setting S1, where purification and SDB were enabled, SiSAT was just able to solve the reachability problem up to step depth 21 instead of 23 within the time limit when caching probability results was used additionally. Considering step depth 21, solving time has increased by a factor of more than 5 but, at the same time, the numbers of branching steps and of detected (approximate) solutions could be reduced by factors greater than 16 and 21, respectively. The same noteworthy behavior was observed for solver setting S2, which extends S1 by caching solutions, accuracy-based pruning with $\alpha = 0.1$, thresholding with $\theta_l = \theta_u = 0.1$, and activity-based branching heuristics aiming at exceeding upper thresholds: when having employed the feature of caching probabilities additionally, the reachability problem could be solved up to step depth 34 only instead of 37. At step depth 34, solving time has increased by a factor of about 4, while the numbers of branching steps and of detected (approximate) solutions could be reduced by factors greater than 25 and 35, respectively.

That is, the algorithmic feature of caching probability results of subproblems has actually pruned the quantifier tree considerably, even in combination with other enhancements. However, the latter fact did not reflect in a positive effect on the solving time but yields a noticeable slow-down. This striking behavior was clearly caused by the immense number of cached probability results which is in general exponential in the number of quantifiers and thus in the step depth. Though the memory limit of 64 GByte was not exceeded in the experiments reported above, the amount of required memory was strongly increasing for higher step depths and has almost reached the memory limit. Indeed, we have also observed cases in which the memory limit became exhausted, for instance, when purification, SDB, caching solutions, and caching probabilities were enabled simultaneously. In the latter solver setting, the memory limit of 64 GByte was exceeded after about 16.8 hours when solving the probabilistic reachability problem at step depth 23.

As indicated by the experimental results, such a high memory consumption can worsen solving time since the more probability entries are cached the more time is needed to find matching entries. At the end of Subsection 6.7.7, we have suggested an idea for future work in order to alleviate the above disadvantages.

6.7.10 Impact of alternative SiSAT encoding exploiting relaxation of SSMT

While we have investigated the impact of the algorithmic enhancements on the *basic* SiSAT encoding of the cooling system $\mathcal{S}_{cool}$ in the previous subsections, we now consider the alternative SiSAT encoding of $\mathcal{S}_{cool}$ which has been introduced in Subsection 6.6.3. Recall that the main idea of the latter was to "disable" each existential or randomized quantifier whenever the concrete value of its corresponding quantified variable becomes irrelevant. To this end, we have introduced a *relaxation* of the SSMT framework that supports the concept of "relaxed" randomized quantifiers. In brief, the sum of the probabilities in a "relaxed" randomized quantifier may exceed value 1. When addressing *well-defined* relaxed SSMT formulae, valid probability distributions in randomized quantifiers however can be ensured, confer Definition 6.3. The latter property of well-definedness holds for each

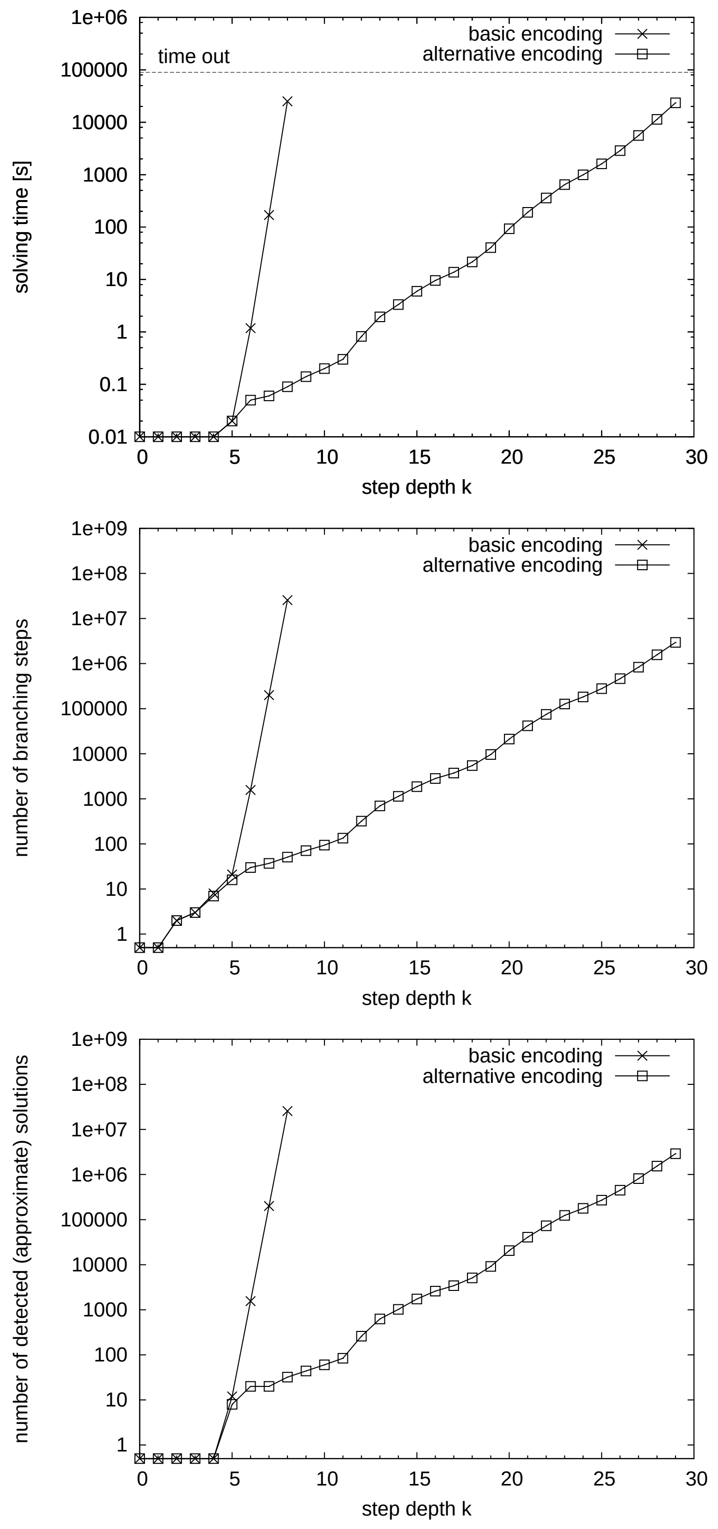


Figure 6.35: Evaluation of the SiSAT tool: *basic* versus *alternative* SiSAT encoding. Solving times (top), numbers of branching steps (middle), and numbers of detected (approximate) solutions (bottom) for the corresponding step depths and for the corresponding SiSAT encodings when no algorithmic enhancements were applied in the SiSAT runs.

SSMT formula 6.25 constructed from the alternative SiSAT encoding of $\mathcal{S}_{cool}$,[20] confer Subsection 6.6.3.

The overall goal of “disabling” quantifiers as realized by the alternative SiSAT encoding of $\mathcal{S}_{cool}$ is to reduce the potential search space and thus to speed up the SSMT proof search. As explained in Subsection 6.7.1, the main drawback of the *basic* SiSAT encoding of $\mathcal{S}_{cool}$ is that whenever $\mathcal{S}_{cool}$ has reached a target state then a self loop is executed such that the remaining non-quantified system variables keep their values independent of the actual values of the non-instantiated quantified variables. As a consequence, iSAT’s deduction mechanisms, being lifted to SiSAT by means of rule SSMT.6, are not applicable to these “unconstrained” quantified variables during SSMT proof search. This implies that the naive SSMT algorithm must explore all assignments to these “unconstrained” quantified variables in worst case, while the number of such assignments is exponential in the number of iterations of this self loop.

If being called on the *alternative* SiSAT encoding of $\mathcal{S}_{cool}$ then the naive SSMT approach might however be able to avoid the pointless overhead mentioned before: since the TRANS section of the alternative encoding contains additional predicates which are responsible for “disabling” quantifiers, confer Subsection 6.6.3, all non-instantiated quantified variables are forced to carry value OFF whenever the target states are reached. Due to the latter predicates, iSAT’s deduction mechanisms, as lifted to SSMT by means of rule SSMT.6, apply to the non-instantiated quantified variables once the target states are visited, thereby reducing their domains to singletons.

The above theoretical argument is confirmed by the experimental results illustrated in Figure 6.35. The naive SSMT algorithm exhibits an impressive behavior on the alternative SiSAT encoding compared to the basic one: the numbers of branching steps and of detected (approximate) solutions as well as the solving time could be reduced by five orders of magnitude at step depth 8. Moreover, the probabilistic reachability problem constructed from the alternative SiSAT encoding could be solved up to step depth 29 within the time limit of 25 hours instead of just up to depth 8 when using the basic encoding.

In what follows, we investigate the impact of the algorithmic enhancements on the alternative SiSAT encoding of $\mathcal{S}_{cool}$. As shown by Figure 6.36, none of the features *purification*, *solution-directed backjumping*, and *caching probability results of subproblems* has improved the performance of SiSAT when called on the alternative encoding. On the contrary, slow-downs were observed: while solving times with purification and SDB enabled have just slightly grown by factors of about 1.2 and 1.5 at step depth 29, respectively, the feature of caching probability results has revealed a slow-down of two orders of magnitude compared to the naive algorithm at step depth 21. In addition to that, the probabilistic reachability problem could only be solved up to step depth 21 when having employed the latter feature. Furthermore note that the naive SSMT procedure called on the alternative SiSAT encoding has performed better than SiSAT enhanced by any combination of purification, SDB, and caching probability results and called on the basic encoding, confer Figures 6.28, 6.30, 6.31, 6.33, and 6.34.

[20]Recall that the semantics of a probabilistic transition system encoded in the SiSAT input language for a given step depth k is defined by the maximum probability of satisfaction of the SSMT formula 6.25, confer Subsection 6.6.1.

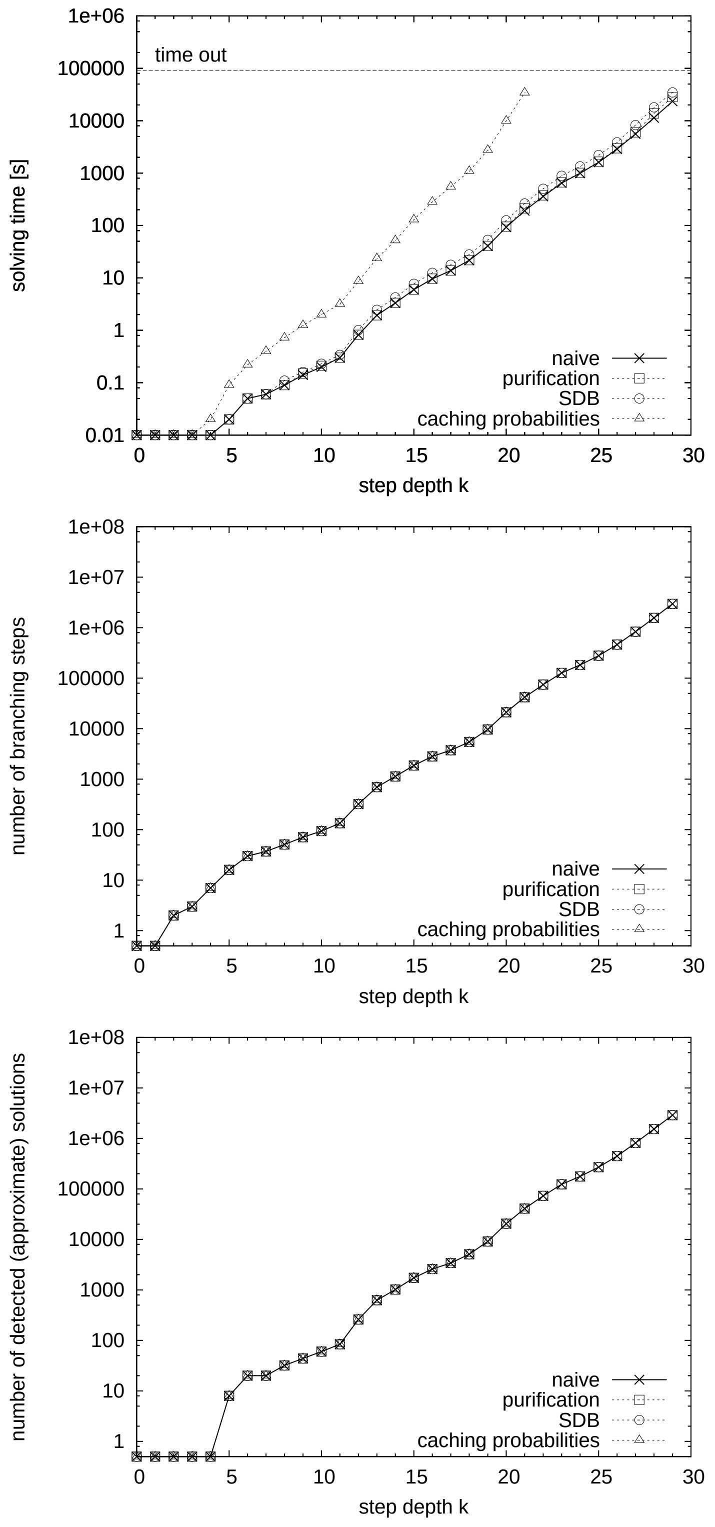


Figure 6.36: Evaluation of the SiSAT tool: *purification*, *solution-directed backjumping*, and *caching probability results of subproblems* for the *alternative* SiSAT encoding. Solving times (top), numbers of branching steps (middle), and numbers of detected (approximate) solutions (bottom) for the corresponding step depths and for the corresponding enhancements.

A plausible explanation of the latter behavior might be that the idea of "disabling" quantifiers as well as iSAT's deduction mechanisms, lifted to SiSAT by rule SSMT.6, together are strong enough to cope with the issue of "unconstrained" quantified variables as described above. This conjecture is substantiated by the fact that the numbers of branching steps and of detected (approximate) solutions have remained constant when having applied any of the above algorithmic enhancements, confer Figure 6.36. The latter observation thus suggests that each of the features purification, SDB, and caching probability results of subproblems has failed in pruning the search tree when applied to the alternative SiSAT encoding. The additional overhead of checking the application conditions of these features has then caused the observed penalties in solving time. As purification just works for existential variables, its slow-down factors are a bit smaller than these of SDB, the latter being additionally applicable to randomized variables. The rather bad performance of caching probability results can be attributed to the immense number of cached probability results which is in general exponential in the number of quantifiers, as mentioned in Subsections 6.7.7 and 6.7.9. That is, even though no cached probability entry was ever reused in the experiments reported above, as indicated by Figure 6.36, the application condition had to be checked which particularly involves the time-consuming search for matching entries in the cache.

We finally examine the impact of the remaining algorithmic enhancements on the alternative SiSAT encoding of $\mathcal{S}_{cool}$. The corresponding experimental results are depicted in Figure 6.37. When having enhanced the naive SSMT procedure by *caching solutions*, solving time has improved by a factor of about 2 at step depth 29, and the probabilistic reachability problem could be solved for two more step depths within the time limit, confer the graph labeled with "+caching solutions". When having further used *accuracy-based pruning* with accuracy $\alpha = 0.1$, a speed-up of an order of magnitude was achieved at step depth 31, confer the graph labeled with "+accuracy $\alpha = 0.1$". Moreover, the probabilistic reachability problem could be tackled up to step depth 34 within the time limit of 25 hours. Further performance gains were observed when having enabled the optimization of *thresholding* with $\theta_l = \theta_u = 0.1$ additionally: solving time was reduced significantly, for instance, by two orders of magnitude at step depth 34 compared to the previous solver setting and even by three orders at depth 29 compared to the naive algorithm, confer the graph labeled with "+thresholding $\theta = 0.1$". Furthermore, the probabilistic reachability problem could be solved for all step depths 0 to 40 with a total solving time of less than 53 minutes. Moderate speed-ups were obtained when having additionally applied the *activity-based value branching heuristics* aiming at exceeding upper thresholds, for instance, by factors of 4 at step depth 12 and of about 1.2 at depth 40 compared to the previous solver setting, confer the graph labeled with "+heuristics exceed-ut". We remark that the overall solving time for the probabilistic reachability problem up to step depth 40 was improved to slightly less than 45 minutes.

6.7.11 Summary

In the previous subsections, it was shown that the algorithmic enhancements presented in Section 6.5 are able to improve the performance of the naive SSMT algorithm described in Section 6.4 significantly, sometimes by several orders of magnitude. As a benchmark,

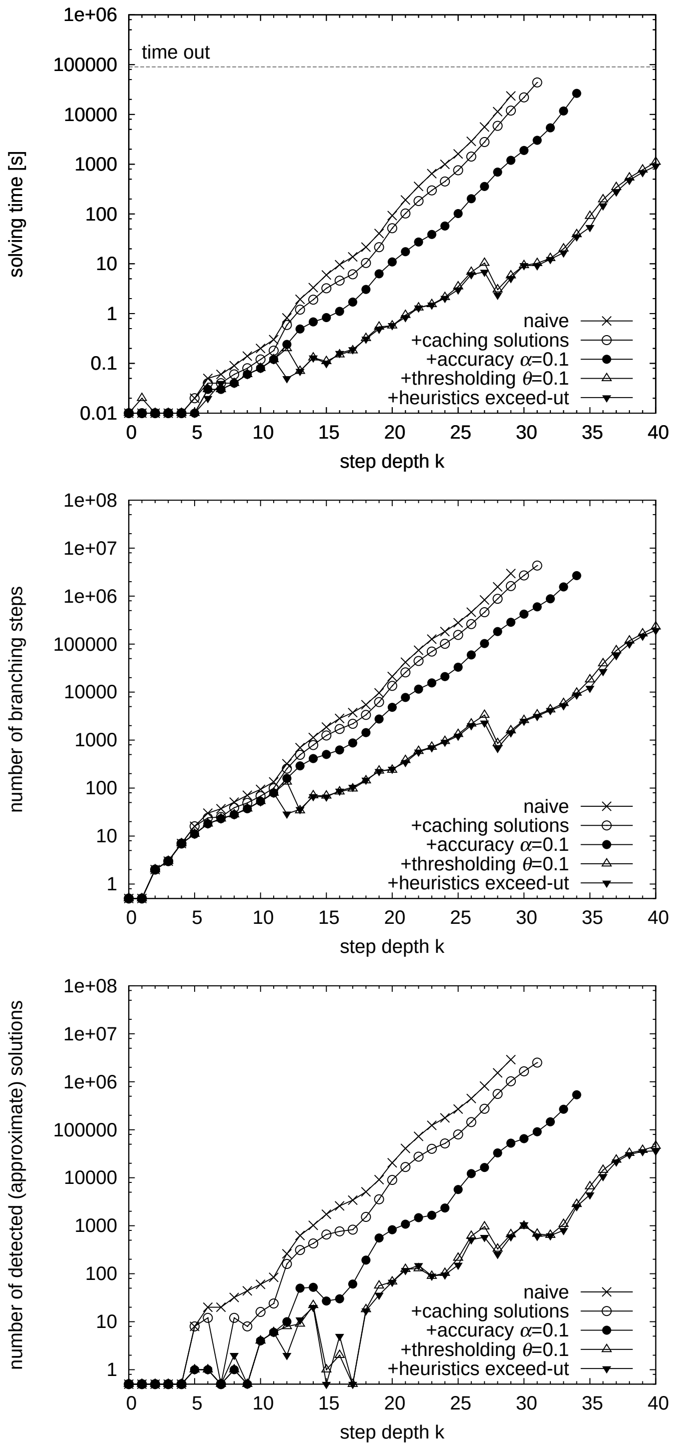


Figure 6.37: Evaluation of the SiSAT tool: *combinations of algorithmic enhancements* for the *alternative* SiSAT encoding. Solving times (top), numbers of branching steps (middle), and numbers of detected (approximate) solutions (bottom) for the corresponding step depths when different combinations of the algorithmic enhancements were applied. The naive SSMT algorithm is denoted by naive while +feature means the use of enhancement feature in addition to the solver setting stated in the preceding line.

we have used the SiSAT encodings of the cooling system $\mathcal{S}_{cool}$ which were introduced in Subsection 6.6.3. This example of a system of concurrent probabilistic hybrid automata can be seen as being representative for a vast number of similar case studies. We remark that a more sophisticated case study, namely the analysis of the networked automation system (NAS) introduced in Section 3.1, is investigated in Chapter 8.

The algorithmic enhancements as well as combinations thereof were evaluated on two SiSAT encodings, namely on the *basic* one, as formally introduced in Section 5.3, and on an *alternative* encoding, the latter realizing the idea of "disabling" quantifiers whenever they are "not needed", i.e. whenever the concrete values of their corresponding quantified variables become irrelevant, in order to aid the SSMT solver in pruning the search tree.

On the basic SiSAT encoding of $\mathcal{S}_{cool}$, isolated application of each of the algorithmic enhancements has yielded performance gains, except for very few, negligible outliers. The same is true for combined activation of these enhancements with the exception of *caching probability results of subproblems*. Though the latter feature was actually capable of pruning the quantifier tree considerably, noticeable slow-downs in solving time were observed. We have argued that this rather curious behavior was caused by the immense number of cached probability results which is in general exponential in the number of quantifiers and thus in the step depth. That is, the more probability entries are cached, the more time is consumed finding matching entries. At the end of Subsection 6.7.7, we have suggested an idea for future work in order to alleviate the above disadvantages.

Though the experiments were conducted on a concrete example of a system of concurrent probabilistic hybrid automata, namely the cooling system $\mathcal{S}_{cool}$ from Figure 6.17, we have explained in detail why similar performance gains can be expected for *basic* SiSAT encodings of other systems of probabilistic hybrid automata, for instance, due to the issue that non-instantiated quantified variables become "unconstrained" whenever the target states are reached, the latter being outlined first in Subsection 6.7.1.

The main goal of the alternative SiSAT encoding of $\mathcal{S}_{cool}$ was to "disable" quantifiers whenever they are "not needed" and thus to address the above issue with such "unconstrained" quantified variables. It turned out that this alternative encoding has a significant advantage over the basic encoding: even the naive SSMT procedure called on the alternative SiSAT encoding has performed better than SiSAT being enhanced by any combination of purification, solution-directed backjumping (SDB), and caching probability results when being called on the basic encoding. We have moreover observed that each of the features purification, SDB, and caching probability results has worsened solving time when called on the alternative encoding, while the other optimizations however were able to improve further the performance of SiSAT.

When aiming at the exact maximum probability of reaching the target states, SiSAT in its best setting for the *basic* encoding of $\mathcal{S}_{cool}$, i.e. with *purification*, *solution-directed backjumping*, and *caching solutions* being enabled, could solve the probabilistic reachability problem up to step depth 24 within the time limit of 25 hours, while the tool in its best setting for the *alternative* encoding, i.e. having applied *caching solutions* only, was able to address the problem up to depth 31 in the same time. The speed-up obtained for the alternative encoding compared to the basic one at step depth 24 amounts to a factor of more than 52 and thus to more than one order of magnitude.

When having employed *accuracy-based pruning* with accuracy $\alpha = 0.1$, *thresholding*

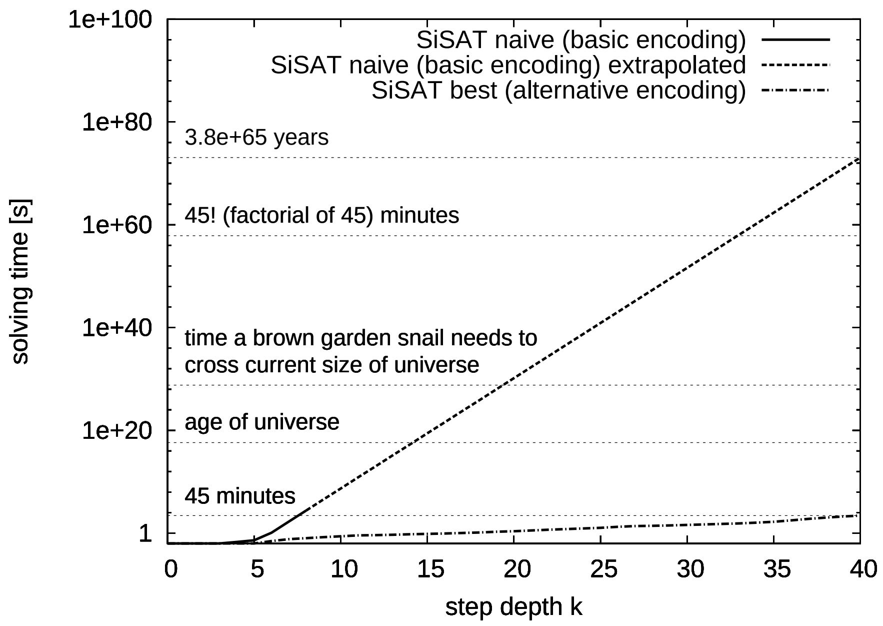


Figure 6.38: Evaluation of the SiSAT tool: *overall impact.* Accumulated solving times up to the corresponding step depths for the naive SSMT algorithm called on the basic encoding (solid line) and for SiSAT called on the alternative encoding when having enabled caching solutions, accuracy-based pruning with $\alpha = 0.1$, thresholding with $\theta_l = \theta_u = 0.1$, and activity-based branching heuristics aiming at exceeding upper thresholds (dashed-dotted line) as well as extrapolated accumulated solving times for the naive approach and for the basic encoding using a growth factor of 140 (dashed line).

with $\theta_l = \theta_u = 0.1$, and *activity-based value branching heuristics* aiming at exceeding upper thresholds in addition to above solver settings, SiSAT could solve the problem up to step depths 37 and 40 within the time limit when called on the *basic* and *alternative* encodings, respectively. The speed-up obtained for the alternative encoding compared to the basic one at step depth 37 amounts to a factor of more than 101 and thus to two orders of magnitude.

With respect to efficiency, the above observations suggest the use of the *alternative* encoding scheme for the SSMT-based probabilistic bounded reachability analysis of probabilistic hybrid automata, while the features purification, SDB, and caching probability results should *not* be employed during SSMT proof search.

It is important to point out that the advice of avoiding the latter algorithmic enhancements should not be taken as a general rule, but just be followed in above scenarios. Due to the expressive power of the SSMT framework, problems from application areas other than the analysis of probabilistic systems can be described as SSMT formulae. When solving such SSMT problems, the features of purification, SDB, and caching probability results might be able to prune the search space owing to some structural properties which cannot be covered by the concept of “disabling” quantifiers. In such cases, noticeable performance gains may be expected from other optimizations, as obtained for the basic SiSAT encoding of $\mathcal{S}_{cool}$.

We conclude this section and thereby the whole Chapter 6 by demonstrating the overall impact of the algorithmic enhancements as well as of the alternative SiSAT encoding on solving time.

In Subsection 6.7.1, we have observed a strong exponential growth in the solving time of the naive SSMT algorithm when called on the basic SiSAT encoding of $\mathcal{S}_{cool}$. This behavior has been attributed to the fact that whenever a target state is reached in $\mathcal{S}_{cool}$ then all non-instantiated quantified variables encoding the selection of transitions and transition alternatives for all the remaining n system steps become “unconstrained”. This means that the naive SSMT algorithm must explore all assignments to these “unconstrained” quantified variables, while the number of such assignments is exponential in n. We have furthermore observed that the solving time has increased from step depth 6 to 7 by a factor of about 144, and from depth 7 to 8 by a factor of about 147. Due to the above argument concerning the formula structure, it cannot be expected that this growth in solving time will decrease at some higher step depth. It is thus reasonable to assume a growth factor of at least 140 in order to extrapolate the solving times for larger step depths. When extrapolating the solving time in such a way, we obtain an accumulated solving time for the probabilistic reachability problem from step depth 0 up to 40 of about $3.8 \cdot 10^{65}$ years (assuming a sufficiently large and long-lived computer).

As mentioned at the end of Subsection 6.7.10, SiSAT in its best setting called on the alternative encoding was able to solve the reachability problem for all step depths 0 to 40 in slightly less than 45 minutes, equivalent to about $8.6 \cdot 10^{-5}$ years. Compared to the extrapolated overall solving time of the naive SSMT approach called on the basic SiSAT encoding, SiSAT in its best setting for the alternative encoding thus has improved the overall solving time by a factor of about $4.4 \cdot 10^{69}$ and therefore by *sixty nine* orders of magnitude. An illustrative presentation of this rather large difference in time is given by Figure 6.38.

7 SSMT-Based Expected-Value Analysis of Probabilistic Hybrid Automata

In the previous chapters, we have presented a symbolic approach to *probabilistic* bounded reachability analysis of probabilistic hybrid automata. The latter procedure, being based on the logical framework of stochastic satisfiability modulo theories (SSMT), is potentially able to *falsify* safety properties of the shape "the worst-case probability of reaching the bad system states is at most 1‰".

However, industrial applications often call for quantitative measures distinct from reachability probabilities since these and related figures frequently tend to 1 in the long run and thus are not sufficiently discriminative in practice when applied to systems with unbounded lifetime, as it is the case for the cooling system $\mathcal{S}_{cool}$ described in Subsection 6.6.3 and depicted in Figure 6.17. Motivated by the latter fact, this chapter is devoted to a symbolic, state-exploratory method for computing *expected values* of discrete-time probabilistic hybrid systems like, for instance, mean time to failure (MTTF). The suggested method builds upon SSMT-based probabilistic bounded model checking but has fundamentally different properties: instead of targeting at *falsification*, the resulting procedure turns into a *verification* approach being able to verify that a probabilistic hybrid system meets safety requirements of the shape "the MTTF is always at least 20 minutes".

In Section 7.1, we first establish the formal system model by slightly extending the concept of concurrent discrete-time probabilistic hybrid automata from Section 3.3 with a notion of costs, and then define the cost expectation for such systems as well as the corresponding cost-expectation model-checking problem. In order to address a step-bounded variant of the latter problem symbolically, Section 7.2 extends the semantics of an SSMT formula Φ with respect to the conditional expectation of a designated free variable in Φ. Thereafter, we show how the step-bounded cost-expectation problem can be reduced to the extended version of the SSMT problem in Section 7.3. In order to complete the symbolic verification procedure, Section 7.4 elaborates on an algorithm to cope with the extended semantics of SSMT. Demonstrating applicability of the suggested approach, experimental results finally are presented in Section 7.5.

We remark that major parts of this chapter are based on the conference paper [FTE10b] by Fränzle, Teige, and Eggers.

7.1 Cost expectation for probabilistic hybrid automata with costs

In order to facilitate the expected-value analysis of probabilistic hybrid systems, we slightly extend the concept of a system of concurrent discrete-time probabilistic hybrid automata from Section 3.3 by a notion of *step costs*.

Definition 7.1 (System of concurrent PHAs with cost function)
A system of concurrent discrete-time probabilistic hybrid automata (PHAs) with cost function *is given by a pair* $(\mathcal{S}, \mathit{cost})$ *of a system* $\mathcal{S}$ *of* n *concurrent discrete-time probabilistic hybrid automata as in Definition 3.1 and of a function*

$$\mathit{cost} : \mathit{NChoice} \times \mathit{PChoice} \times \mathit{States}_{\mathcal{S}} \to \mathbb{R}_{\geq 0}$$

that associates to each non-deterministic choice $\mathit{tr} \in \mathit{NChoice}$ *of a global transition and each probabilistic selection* $\mathit{pc} \in \mathit{PChoice}(\mathit{tr})$ *of a global transition alternative a non-negative cost value* $\mathit{cost}(\mathit{tr}, \mathit{pc}, s)$ *that depends on the current state* $s \in \mathit{States}_{\mathcal{S}}$*. For technical reasons, we demand that whenever the first two arguments of function cost are fixed then the resulting function* $\mathit{cost}(\mathit{tr}, \mathit{pc}) : \mathit{States}_{\mathcal{S}} \to \mathbb{R}_{\geq 0}$ *with* $\mathit{cost}(\mathit{tr}, \mathit{pc})(s) = \mathit{cost}(\mathit{tr}, \mathit{pc}, s)$ *for each* $s \in \mathit{States}_{\mathcal{S}}$ *is syntactically represented by a term in arithmetic theory* $\mathcal{T}$ *over the discrete variables in* $D_1, \ldots, D_n$ *and the continuous variables in* $R_1, \ldots, R_n$*.*

For the sake of simplicity, namely to avoid technicalities in the definition of a scheduler, we demand that the following two semantic conditions are satisfied:

1. *System* $\mathcal{S}$ *is* non-blocking *in the sense that for each state* $s \in \mathit{States}_{\mathcal{S}}$ *there exists a global transition* $\mathit{tr} = (\mathit{tr}^1, \ldots, \mathit{tr}^n) \in \mathit{NChoice}$ *such that the guards of all local transitions are mutually consistent, i.e.*

   $$\mathit{val}(s) \wedge g(\mathit{tr}^1) \wedge \ldots \wedge g(\mathit{tr}^n)$$

   is satisfiable[1] *where predicate* $\mathit{val}(s)$ *represents state* s *symbolically, i.e.*

   $$\mathit{val}(s) = \bigwedge_{v \in \bigcup_{i=1}^{n} (D_i \cup R_i)} v = s(v)$$

   with $s(v)$ *being the value of variable* v *in state* s*. Such global transition* tr *is called* enabled in state s.

2. *System* $\mathcal{S}$ *is* executable *in the sense that for each state* $s \in \mathit{States}_{\mathcal{S}}$*, for each global transition* $\mathit{tr} \in \mathit{NChoice}$ *which is enabled in state* s*, and for each probabilistic transition alternative* $\mathit{pc} \in \mathit{PChoice}(\mathit{tr})$*, system* $\mathcal{S}$ *does not deadlock, i.e.* $\mathit{Post}(s, \mathit{tr}, \mathit{pc}) \neq \bot$*.*

Note that checking above semantic conditions 1 and 2 is undecidable in general due to the potentially infinite state space of a system of PHAs and due to potential undecidability of arithmetic theory $\mathcal{T}$. Recall that we consider the undecidable theory of non-linear arithmetic over the reals and integers involving transcendental functions within this thesis, confer Section 4.3. In several cases, it however is reasonable to expect that a system designer is able to ensure above semantic conditions.

The executable behavior of a system $(\mathcal{S}, \mathit{cost})$ of concurrent PHAs with cost function is defined by the *runs* $r = \langle s_0, (\mathit{tr}_1, \mathit{pc}_1), s_1, \ldots, s_{i-1}, (\mathit{tr}_i, \mathit{pc}_i), s_i, \ldots, (\mathit{tr}_k, \mathit{pc}_k), s_k \rangle \in (\mathit{States}_{\mathcal{S}} \cup \{\bot\}) \times ((\mathit{NChoice} \times \mathit{PChoice}) \times (\mathit{States}_{\mathcal{S}} \cup \{\bot\}))^*$ of system $\mathcal{S}$, as formalized in Definition 3.2. The *cost of step* i in r is given by $\mathit{cost}(t_i, p_i, s_{i-1})$.

[1]Recall that transition guards may contain primed variables referring to the post-state.

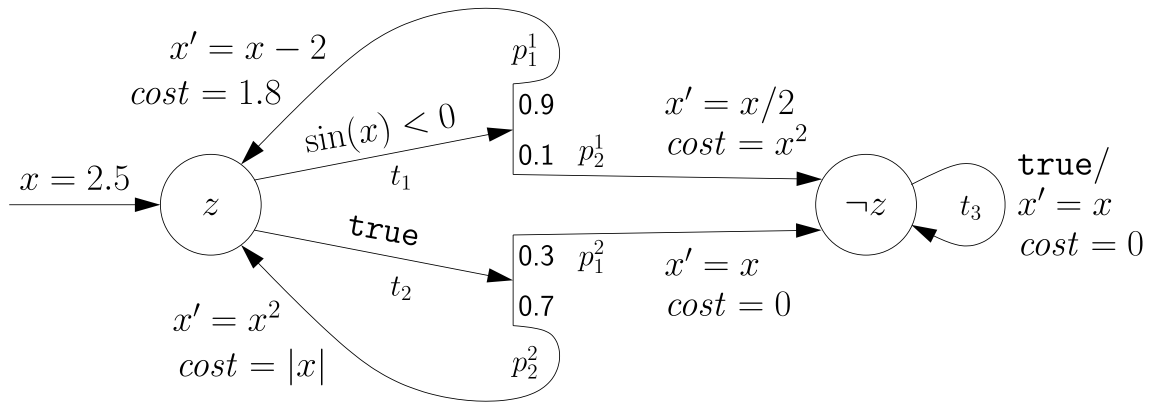


Figure 7.1: A single probabilistic hybrid automaton $(\mathcal{A}, \mathit{cost})$ with cost function. The arithmetic terms representing the functions $\mathit{cost}(\mathit{tr}, \mathit{pc})$ after having fixed the first two arguments are associated to the corresponding transitions tr and, if existent, transition alternatives pc, for instance, $\mathit{cost}(t_1, p_2^1) = x^2$. (This figure is a slight modification of Figure 2 from [FTE10b].)

For an example, consider the single, i.e. non-concurrent, probabilistic hybrid automaton $(\mathcal{A}, \mathit{cost})$ with cost function from Figure 7.1. The unique initial state of $\mathcal{A}$ is $s_0 = (z, x = 2.5)$. Transition t_1 cannot be taken since the guard is not satisfied due to $\sin(2.5) > 0.59$. Automaton $\mathcal{A}$ is thus forced to take transition t_2. If the subsequent probabilistic choice triggers transition alternative p_2^2 with associated probability 0.7 then $\mathcal{A}$ enters successor state $s_1 = (z, x = 6.25)$. The cost of the latter step is $\mathit{cost}(t_2, p_2^2, s_0) = |x| = 2.5$. The guard of t_1 is now satisfied since $\sin(6.25) < 0$. Let $\mathcal{A}$ now execute t_1 followed by probabilistic choice of alternative p_1^1 with associated probability 0.9. The cost of the latter step is given by $\mathit{cost}(t_1, p_1^1, s_1) = 1.8$, and the post-state is $s_2 = (z, x = 4.25)$. If thereafter selecting t_2 non-deterministically and p_1^2 probabilistically then $\mathcal{A}$ performs a step to $s_3 = (\neg z, x = 4.25)$ at zero cost, i.e. $\mathit{cost}(t_2, p_1^2, s_2) = 0$. Altogether, the latter anchored system run of length 3 yields an accumulated step cost of $2.5 + 1.8 + 0 = 4.3$.

In this chapter, we are interested in the *expected value of the accumulated cost* when a system $(\mathcal{S}, \mathit{cost})$ of concurrent PHAs with cost function reaches a target state. Due to the presence of non-deterministic selection between several transitions, we assume that the dynamics of $\mathcal{S}$ is controlled by a *scheduler* resolving this non-determinism. As opposed to the case of probabilistic bounded state reachability, where we have allowed the rather general notion of history-dependent, deterministic schedulers, confer Definition 5.1, we do here restrict ourselves to *stationary Markovian deterministic* schedulers that depend on the current system state only, confer, for instance, [BHKH05].

Definition 7.2 (stationary Markovian deterministic scheduler)
Let $(\mathcal{S}, \mathit{cost})$ be a system of concurrent PHAs with cost function, as in Definition 7.1. Then, a stationary Markovian deterministic scheduler*, or* simple scheduler*, $\sigma : \mathit{States}_{\mathcal{S}} \to \mathit{NChoice}$ for $(\mathcal{S}, \mathit{cost})$ maps a state $s \in \mathit{States}_{\mathcal{S}}$ to a global transition $\sigma(s) \in \mathit{NChoice}$ which is enabled in state s.*

Note that above simple schedulers σ for $(\mathcal{S}, \mathit{cost})$ are always well-defined, i.e. for each state $s \in \mathit{States}_{\mathcal{S}}$ there is at least one global transition which is enabled in state s according to semantic condition 1 from Definition 7.1.

After having elaborated on the issue of schedulers, we are now prepared to introduce the cost expectation for systems of concurrent PHAs with costs.

Definition 7.3 (Cost expectation)
Let $(\mathcal{S}, cost)$ be a system of n concurrent PHAs with cost function, Target be a $\mathcal{T}$-predicate over variables in $D_1, \ldots, D_n$ and $R_1, \ldots, R_n$ defining the target states, and σ be a stationary Markovian deterministic scheduler for $(\mathcal{S}, cost)$. Then, the cost expectation for $(\mathcal{S}, cost)$ under scheduler σ *is the least (with respect to the product order) solution of the equation system*

$$\left(CE_{(\mathcal{S},cost),\sigma,Target}(z) = \begin{cases} 0 & \text{if} \quad z \models Target \\ \sum\limits_{pc \in PChoice(tr)} p(tr,pc) \cdot \begin{pmatrix} & cost(tr,pc,z) \\ + & CE_{(\mathcal{S},cost),\sigma,Target}(z') \end{pmatrix} & \\ \qquad \text{if} \quad z \not\models Target \end{cases} \right)_{z \in States_{\mathcal{S}}}$$

with $tr = \sigma(z)$ being the global transition selected by σ and $z' = Post(z, tr, pc)$ being the corresponding unique post-state. We demand that for each state $z \in States_{\mathcal{S}}$, $CE_{(\mathcal{S},cost),\sigma,Target}(z)$ ranges within the set of all non-negative real numbers extended by ∞ representing infinity, i.e. $CE_{(\mathcal{S},cost),\sigma,Target}(z) \in \mathbb{R}_{\geq 0} \cup \{\infty\}$. In order to calculate with ∞, we assume the smallest monotone extension of addition and multiplication, i.e. we define $a + \infty = \infty + a = \infty$ for all $a \in \mathbb{R}_{\geq 0} \cup \{\infty\}$, $a \cdot \infty = \infty \cdot a = \infty$ for all $a \in \mathbb{R}_{\geq 0} \cup \{\infty\}$ with $a \neq 0$, and $0 \cdot \infty = \infty \cdot 0 = 0$.

For a particular state $s \in States_{\mathcal{S}}$, the cost expectation for reaching the target states from state s under scheduler σ *is $CE_{(\mathcal{S},cost),\sigma,Target}(s)$, while the* infimal cost expectation for reaching the target states from state s *is given by*

$$CE_{(\mathcal{S},cost),Target}(s) \quad = \quad \inf_{\sigma' \in \Upsilon} CE_{(\mathcal{S},cost),\sigma',Target}(s)$$

with Υ being the set of all stationary Markovian deterministic schedulers for $(\mathcal{S}, cost)$.

Observe that global transition $tr = \sigma(z) \in NChoice$ always exists due to semantic condition 1 from Definition 7.1. It furthermore holds that $z' = Post(z, tr, pc) \neq \bot$ for each probabilistic transition alternative $pc \in PChoice(tr)$ due to semantic condition 2 from Definition 7.1.

We remark that existence of the least solution of the equation system above is guaranteed due to the following. First, note that the Cartesian product $(\mathbb{R}_{\geq 0} \cup \{\infty\})^{|States_{\mathcal{S}}|}$ together with product order is a complete partially ordered set. Second, let function

$$f : (\mathbb{R}_{\geq 0} \cup \{\infty\})^{|States_{\mathcal{S}}|} \to (\mathbb{R}_{\geq 0} \cup \{\infty\})^{|States_{\mathcal{S}}|}$$

be defined by

$$f(C) = \left[\begin{array}{c} \left(\begin{array}{ll} 0 & \text{if} \quad z_1 \models \mathit{Target} \\ \sum_{pc_1 \in \mathit{PChoice}(tr_1)} p(tr_1, pc_1) \cdot \left(\begin{array}{ll} & \mathit{cost}(tr_1, pc_1, z_1) \\ + & C_{j_1} \end{array} \right) & \\ & \text{if} \quad z_1 \not\models \mathit{Target} \end{array} \right) \\ \vdots \\ \left(\begin{array}{ll} 0 & \text{if} \quad z_i \models \mathit{Target} \\ \sum_{pc_i \in \mathit{PChoice}(tr_i)} p(tr_i, pc_i) \cdot \left(\begin{array}{ll} & \mathit{cost}(tr_i, pc_i, z_i) \\ + & C_{j_i} \end{array} \right) & \\ & \text{if} \quad z_i \not\models \mathit{Target} \end{array} \right) \\ \vdots \end{array} \right]$$

such that $tr_i = \sigma(z_i)$, $z_{j_i} = \mathit{Post}(z_i, tr_i, pc_i)$, and C_i gives the i-th element of vector C for all $i \geq 1$. Note that C may contain uncountably many elements as the state space $\mathit{States}_{\mathcal{S}}$ of $\mathcal{S}$ is potentially uncountable.

It then holds that function f is monotone and continuous with respect to product order, i.e. application of f preserves product order and the suprema of all ascending chains. From Kleene's fixed point theorem [Kle52], existence of a least fixed point of f then follows, with the latter corresponding to the least solution of above equation system.

We finally state the corresponding decision problem, which we call *cost-expectation model checking* and which is defined to be the problem of deciding whether the cost expectation for a system $(\mathcal{S}, \mathit{cost})$ of concurrent PHAs with cost function is acceptable, where acceptability is defined by a threshold value θ to be exceeded irrespective of the actual scheduler. An example is a setting where *costs* of steps correspond to their *durations* and the *target states* denote *system failures*. The expected cost then is the mean time to failure (MTTF) of $(\mathcal{S}, \mathit{cost})$, and the threshold θ can be interpreted as a requirement on the MTTF of the design under inspection. Adopting a demonic interpretation of non-determinism, the latter has to be guaranteed irrespective of the actual scheduler.

Definition 7.4 (Cost-expectation model checking)
Let be given a system $(\mathcal{S}, \mathit{cost})$ of n concurrent PHAs with cost function, a $\mathcal{T}$-predicate Target over variables in $D_1, \ldots, D_n$ and $R_1, \ldots, R_n$ defining the target states, and a target threshold $\theta \geq 0$. Then, the cost-expectation model-checking problem (CEMC) *is to decide whether*

$$CE_{(\mathcal{S},\mathit{cost}),\mathit{Target}}(\imath) \quad \geq \quad \theta$$

holds with $\imath \models \bigwedge_{i=1}^{n} \mathit{init}_i$ being the (unique) global initial state of $\mathcal{S}$.

7.2 Conditional expectation for SSMT

In order to address a step-bounded variant of the cost-expectation model-checking problem from Section 7.1 symbolically, we propose a conservative extension of the semantics of SSMT formulae which adds considerably to the expressiveness of SSMT. The new semantics is based on the *maximum conditional expectation* of a designated free variable y in an

$$\Phi = \reflectbox{R}_{[0\to 0.5, 1\to 0.5]} x_1 \in \{0,1\}\ \exists x_2 \in \{0,1\}\ \reflectbox{R}_{[0\to 0.8, 1\to 0.2]} x_3 \in \{0,1\} : \big((x_1 = 1 \vee x_2 = 1 \vee x_3 = 0) \wedge (x_1 = 1 \vee x_2 = 0 \vee x_3 = 1) \wedge (y = 4\cdot x_1 + (x_2+x_3)^2)\big)$$

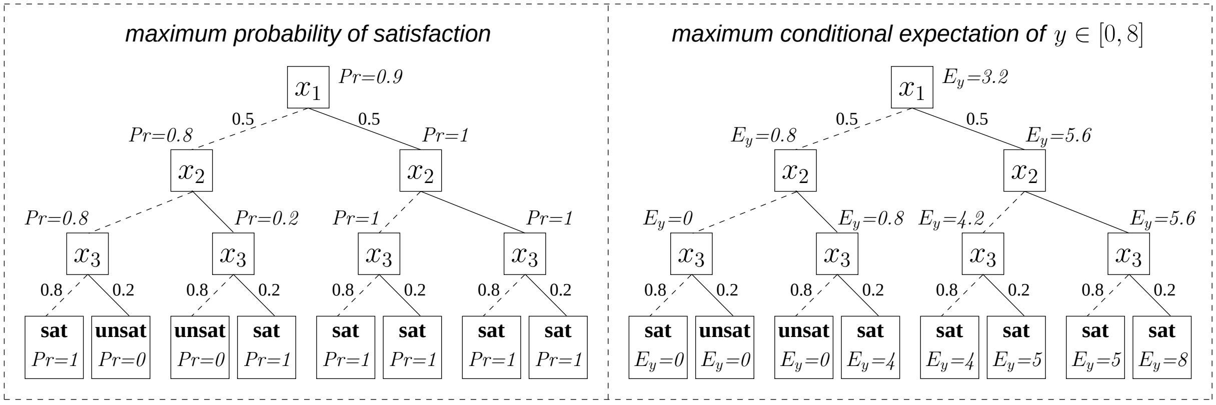


Figure 7.2: The two different semantics of SSMT: maximum probability of satisfaction of SSMT formula Φ (left) and maximum conditional expectation of y given Φ (right). Dashed and solid lines denote variable assignments 0 and 1, respectively, while the probabilities of these variable assignments are associated to the corresponding lines whenever applicable. (This figure is a slight modification of Figure 1 from [FTE10b].)

SSMT formula. Intuitively, instead of assigning a probability 0 or 1 to a quantifier-free formula φ as in the "classical" SSMT semantics constituting the maximum probability of satisfaction, the extended semantics calls for the maximum value $\tau(y)$ of the designated variable y over all solutions $\tau \models \varphi$ of the quantifier-free formula φ. The semantic rules for existential and randomized quantifiers then remain the same as in the "classical" setting.

With regard to syntax, the only difference to SSMT formulae Φ from Definition 4.4 is that we require a designated free variable y in Φ with its domain given by a bounded interval. Then, the semantics is formalized as follows:

Definition 7.5 (Maximum conditional expectation for SSMT)
Let $\Phi = Q_1 x_1 \in \mathcal{D}_{x_1} \odot \ldots \odot Q_n x_n \in \mathcal{D}_{x_n} : \varphi$ be an SSMT formula as in Definition 4.4, and let y be a free variable in Φ with its domain dom(y) *being given by a bounded interval $[l_y, u_y] \subset \mathbb{R}$, i.e. $y \notin \{x_1, \ldots, x_n\}$ and $y \in Var(\varphi)$. Then, the* maximum conditional expectation of y given Φ, *denoted as $E_y(\Phi)$, is recursively defined as follows:*

$$\begin{aligned}
E_y(\varepsilon : \varphi) &= \max\nolimits_{\tau \models \varphi} \tau(y) \ , \\
E_y(\exists x \in \mathcal{D}_x \odot \mathcal{Q} : \varphi) &= \max\nolimits_{v \in \mathcal{D}_x} E_y(\mathcal{Q} : \varphi[v/x]) \ , \\
E_y(\reflectbox{R}_{d_x} x \in \mathcal{D}_x \odot \mathcal{Q} : \varphi) &= \textstyle\sum_{v \in \mathcal{D}_x} d_x(v) \cdot E_y(\mathcal{Q} : \varphi[v/x]) \ ,
\end{aligned}$$

where ε denotes the empty and $\mathcal{Q}$ an arbitrary quantifier prefix.

Observe that the semantic rules for the cases in which the quantifier prefix is non-empty are identical to the corresponding rules of the "classical" semantics of SSMT, i.e. of the maximum probability of satisfaction, confer Definition 4.5. If the quantifier prefix is empty, the classical scheme is generalized by determining the maximum value $\tau(y) \in [l_y, u_y]$ of the random variable y over all satisfying assignments τ of φ, instead of just checking for satisfiability of φ. Whenever no such satisfying assignment exists, we follow the order-theoretic convention that the maximum over the empty subset of the ordered set $[l_y, u_y]$ is the minimum domain value l_y. For an example, confer Figure 7.2.

Observe that the *maximum conditional expectation* is a conservative extension of the classical semantics based on *maximum probability of satisfaction* in the sense that

$$Pr(\mathcal{Q} : \varphi) \quad = \quad E_y(\mathcal{Q} : (\varphi \wedge (y = 1)))$$

holds for each SSMT formula $\mathcal{Q} : \varphi$ with y being a fresh variable ranging over the real-valued interval $[0, 1]$, i.e. $y \notin \mathit{Var}(\varphi)$ and $\mathrm{dom}(y) = [0, 1]$.

7.3 Reducing step-bounded cost expectation to SSMT

In order to facilitate the automatic verification of cost-expectation model-checking problems (CEMC) from Definition 7.4, we aim at solving corresponding SSMT formulae, the latter being interpreted by the extended semantics from Definition 7.5. Akin to probabilistic state reachability, confer Chapter 5, our verification procedure is based on an SSMT encoding of the step-bounded behavior of a system of concurrent PHAs with cost function. Therefore, we first introduce the notion of step-bounded cost expectation in Subsection 7.3.1, which provides a lower bound on the infimal cost expectation for reaching the target states from the initial state. In order to cope with the problem of computing the step-bounded cost expectation symbolically, we then suggest a reduction from step-bounded cost expectation to SSMT in Subsection 7.3.2.

Together with an SSMT algorithm for computing the maximum conditional expectation, which is presented in Section 7.4, the proposed approach constitutes a symbolic verification procedure for the general CEMC problem with target threshold θ in the sense that once a lower expectation bound $lb \geq \theta$ is computed for the SSMT formula encoding the corresponding step-bounded cost expectation then the general CEMC problem is decided to be true.

7.3.1 Step-bounded cost expectation

The following definition of the step-bounded cost expectation does not rely on a mutually recursive equation defining the cost expectation for each state as in Definition 7.3 but on a well-founded recursion over the remaining step depth.

Definition 7.6 (*k*-step minimum cost expectation)
Let $(\mathcal{S}, cost)$ be a system of n concurrent PHAs with cost function, Target be a $\mathcal{T}$-predicate over variables in $D_1, \ldots, D_n$ and $R_1, \ldots, R_n$ defining the target states, and $k \in \mathbb{N}$ be a step bound. Then, the k-step minimum cost expectation for reaching the target states from a state $s \in \mathit{States}_{\mathcal{S}}$ and for cost seeds $c \in \mathbb{R}_{\geq 0} \cup \{\infty\}$ and $d : \mathit{States}_{\mathcal{S}} \to (\mathbb{R}_{\geq 0} \cup \{\infty\})$, *denoted $CE^k_{(\mathcal{S},cost),\,Target}(s, c, d)$, is defined as follows:*

$$CE^k_{(\mathcal{S},cost),\,Target}(s, c, d)$$

$$= \begin{cases} c & \text{if} \quad s \models \mathit{Target}\ , \\ c + d(s) & \text{if} \quad s \not\models \mathit{Target} \text{ and } k = 0\ , \\ \min\limits_{tr \in En(s)} \sum\limits_{pc \in PChoice(tr)} p(tr, pc) \cdot CE^{k-1}_{(\mathcal{S}, cost), \mathit{Target}}(s', c + cost(tr, pc, s), d) & \\ & \text{if} \quad s \not\models \mathit{Target} \text{ and } k > 0\ , \end{cases}$$

with $En(s) = \{tr \in NChoice : tr$ is enabled in $s\}$ and $s' = Post(s, tr, pc)$.

The k-step minimum cost expectation can be used for obtaining safe estimates of the infimal cost expectation, as the following lemma shows.

Lemma 7.1 (Estimates of infimal cost expectation)
Let $(\mathcal{S}, cost)$ be a system of concurrent PHAs with cost function, Target be a $\mathcal{T}$-predicate defining the target states, and $k \in \mathbb{N}$ be a step bound. It then holds that

$$CE_{(\mathcal{S}, cost), \mathit{Target}}(s) \quad \geq \quad CE^{k}_{(\mathcal{S}, cost), \mathit{Target}}(s, 0, \mathbf{0})$$

where $\mathbf{0}$ is the constant function assigning 0 to all states. Furthermore, the sequence $CE^{k}_{(\mathcal{S}, cost), \mathit{Target}}(s, 0, \mathbf{0})$ is (not necessarily strictly) monotonically increasing.

Likewise, if some function $C : States_{\mathcal{S}} \to (\mathbb{R}_{\geq 0} \cup \{\infty\})$ satisfies $C \geq CE_{(\mathcal{S}, cost), \mathit{Target}}$, i.e. $C(z) \geq CE_{(\mathcal{S}, cost), \mathit{Target}}(z)$ for all $z \in States_{\mathcal{S}}$, then

$$CE_{(\mathcal{S}, cost), \mathit{Target}}(s) \quad \leq \quad CE^{k}_{(\mathcal{S}, cost), \mathit{Target}}(s, 0, C) \quad .$$

Proof. We show by induction on k that

$$CE^{k}_{(\mathcal{S}, cost), \mathit{Target}}(s, c, CE_{(\mathcal{S}, cost), \mathit{Target}}) \quad = \quad c + CE_{(\mathcal{S}, cost), \mathit{Target}}(s) \tag{7.1}$$

is true for arbitrary $c \in \mathbb{R}_{\geq 0} \cup \{\infty\}$. It is not hard to see that $CE^{k}_{(\mathcal{S}, cost), \mathit{Target}}(s, c, d)$ is monotonic in d. Due to the latter fact and since $C \geq CE_{(\mathcal{S}, cost), \mathit{Target}} \geq \mathbf{0}$, both inequalities $CE_{(\mathcal{S}, cost), \mathit{Target}}(s) \geq CE^{k}_{(\mathcal{S}, cost), \mathit{Target}}(s, 0, \mathbf{0})$ as well as $CE_{(\mathcal{S}, cost), \mathit{Target}}(s) \leq CE^{k}_{(\mathcal{S}, cost), \mathit{Target}}(s, 0, C)$ follow from equation 7.1 by taking $c = 0$.

For the base case of the induction, i.e. for $k = 0$, it is immediate from definition of $CE^{k}_{(\mathcal{S}, cost), \mathit{Target}}$ that $CE^{0}_{(\mathcal{S}, cost), \mathit{Target}}(s, c, CE_{(\mathcal{S}, cost), \mathit{Target}}) = c + CE_{(\mathcal{S}, cost), \mathit{Target}}(s)$ holds.

For $k > 0$, we first obtain from Definition 7.6 that

$$\begin{aligned} & CE^{k}_{(\mathcal{S}, cost), \mathit{Target}}(s, c, CE_{(\mathcal{S}, cost), \mathit{Target}}) \\ &= \begin{cases} c & \text{if} \quad s \models \mathit{Target}\ , \\ \min\limits_{tr \in En(s)} \sum\limits_{pc \in PChoice(tr)} p(tr, pc) \cdot CE^{k-1}_{(\mathcal{S}, cost), \mathit{Target}}(s', c + cost(tr, pc, s), CE_{(\mathcal{S}, cost), \mathit{Target}}) & \\ & \text{if} \quad s \not\models \mathit{Target}\ , \end{cases} \end{aligned}$$

while application of induction hypothesis yields

$$\begin{aligned} & CE^{k}_{(\mathcal{S}, cost), \mathit{Target}}(s, c, CE_{(\mathcal{S}, cost), \mathit{Target}}) \\ &= \begin{cases} c & \text{if} \quad s \models \mathit{Target}\ , \\ \min\limits_{tr \in En(s)} \sum\limits_{pc \in PChoice(tr)} p(tr, pc) \cdot (c + cost(tr, pc, s) + CE_{(\mathcal{S}, cost), \mathit{Target}}(s')) & \\ & \text{if} \quad s \not\models \mathit{Target}\ . \end{cases} \end{aligned}$$

Using Definition 7.3, we have that

$$\begin{aligned}
&CE^{k}_{(\mathcal{S},cost),\,Target}(s,\, c,\, CE_{(\mathcal{S},cost),\,Target}) \\
&= \begin{cases} c & \text{if} \quad s \models Target\ , \\ \min\limits_{tr\in En(s)} \sum\limits_{pc\in PChoice(tr)} p(tr,pc)\cdot \Big(c + cost(tr,pc,s) + \inf\limits_{\sigma'\in\Upsilon} CE_{(\mathcal{S},cost),\sigma',\,Target}(s')\Big) \\ & \text{if} \quad s \not\models Target \end{cases}
\end{aligned}$$

with Υ being the set of all stationary Markovian deterministic schedulers for $(\mathcal{S}, cost)$. Observe that the set of all global transitions that are enabled in state s coincides with the set of all transitions that are selected by stationary Markovian deterministic schedulers in state s, i.e. $En(s) = \{\sigma(s) \in NChoice : \sigma \in \Upsilon\}$. Due to the latter fact, we conclude that

$$\begin{aligned}
&CE^{k}_{(\mathcal{S},cost),\,Target}(s,\, c,\, CE_{(\mathcal{S},cost),\,Target}) \\
&= \begin{cases} c & \text{if} \quad s \models Target\ , \\ \inf\limits_{\sigma\in\Upsilon} \sum\limits_{pc\in PChoice(tr')} p(tr',pc)\cdot \Big(c + cost(tr',pc,s) + \inf\limits_{\sigma'\in\Upsilon} CE_{(\mathcal{S},cost),\sigma',\,Target}(s'')\Big) \\ & \text{if} \quad s \not\models Target \end{cases} \\
&= \begin{cases} c & \text{if} \quad s \models Target\ , \\ \inf\limits_{\sigma\in\Upsilon} \inf\limits_{\sigma'\in\Upsilon} \sum\limits_{pc\in PChoice(tr')} p(tr',pc)\cdot \big(c + cost(tr',pc,s) + CE_{(\mathcal{S},cost),\sigma',\,Target}(s'')\big) \\ & \text{if} \quad s \not\models Target \end{cases} \\
&= \begin{cases} c & \text{if} \quad s \models Target\ , \\ \inf\limits_{\sigma\in\Upsilon} \sum\limits_{pc\in PChoice(tr')} p(tr',pc)\cdot \big(c + cost(tr',pc,s) + CE_{(\mathcal{S},cost),\sigma,\,Target}(s'')\big) \\ & \text{if} \quad s \not\models Target \end{cases}
\end{aligned}$$

where $tr' = \sigma(s)$ and $s'' = Post(s, tr', pc)$. Again using Definition 7.3, we finally achieve

$$\begin{aligned}
CE^{k}_{(\mathcal{S},cost),\,Target}(s,\, c,\, CE_{(\mathcal{S},cost),\,Target}) &= \inf_{\sigma\in\Upsilon}(c + CE_{(\mathcal{S},cost),\sigma,\,Target}(s)) \\
&= c + CE_{(\mathcal{S},cost),\,Target}(s) \quad .
\end{aligned}$$

The monotonic increase of the sequence $CE^{k}_{(\mathcal{S},cost),\,Target}(s, 0, \mathbf{0})$ can be shown by a straightforward induction. □

Consequently, a *verification* procedure for the CEMC problem from Definition 7.4 can be derived from Lemma 7.1. More precisely, the *step-bounded cost-expectation model-checking problem* (step-bounded CEMC), which is to decide whether

$$CE^{k}_{(\mathcal{S},cost),\,Target}(\imath, 0, \mathbf{0}) \quad \geq \quad \theta$$

holds for some given step depth $k \in \mathbb{N}$ and some given threshold $\theta \geq 0$, with $\imath$ being the unique initial state, is addressed for increasing k until some depth k' is found for which the step-bounded CEMC problem is decided to be true, i.e. $CE^{k'}_{(\mathcal{S},cost),\,Target}(\imath, 0, \mathbf{0}) \geq \theta$ holds. It then follows that also the general CEMC problem is decided to be true since

$$CE_{(\mathcal{S},cost),\,Target}(\imath) \quad \geq \quad CE^{k'}_{(\mathcal{S},cost),\,Target}(\imath, 0, \mathbf{0}) \quad \geq \quad \theta \quad .$$

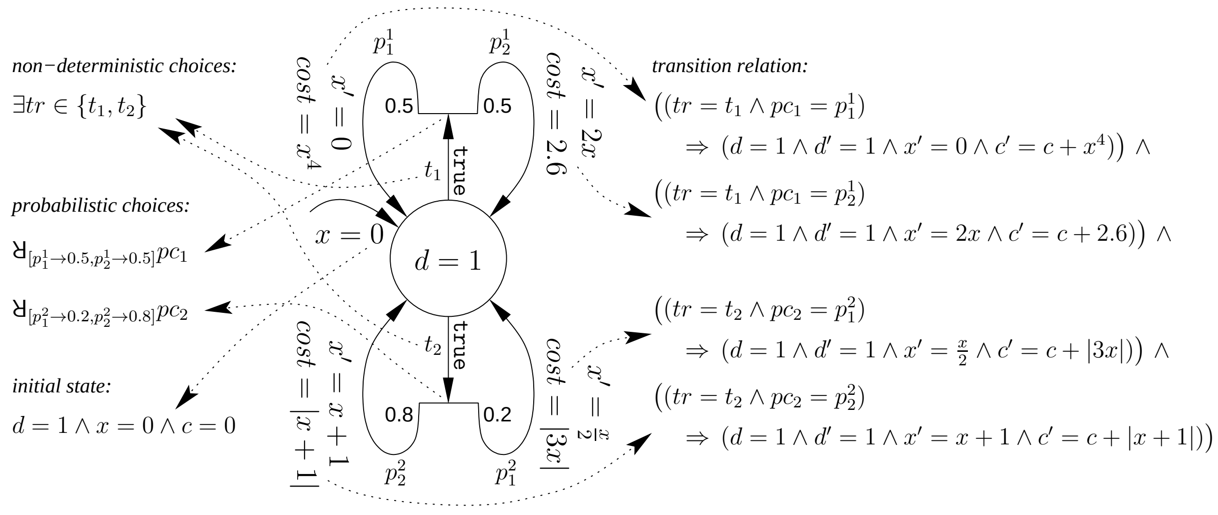

non−deterministic choices:
$\exists tr \in \{t_1, t_2\}$
probabilistic choices:
$Я_{[p_1^1 \to 0.5, p_2^1 \to 0.5]} pc_1$
$Я_{[p_1^2 \to 0.2, p_2^2 \to 0.8]} pc_2$
initial state:
$d = 1 \wedge x = 0 \wedge c = 0$
p_1^1
p_2^1
$x' = 0$
$cost = x^4$
$x' = 2x$
$cost = 2.6$
0.5
0.5
true
t_1
$x = 0$
$d = 1$
t_2
true
$x' = x + 1$
$cost = |x + 1|$
$x' = \frac{x}{2}$
$cost = |3x|$
0.8
0.2
p_2^2
p_1^2
transition relation:
$((tr = t_1 \wedge pc_1 = p_1^1)$
$\Rightarrow (d = 1 \wedge d' = 1 \wedge x' = 0 \wedge c' = c + x^4)) \wedge$
$((tr = t_1 \wedge pc_1 = p_2^1)$
$\Rightarrow (d = 1 \wedge d' = 1 \wedge x' = 2x \wedge c' = c + 2.6)) \wedge$
$((tr = t_2 \wedge pc_2 = p_1^2)$
$\Rightarrow (d = 1 \wedge d' = 1 \wedge x' = \frac{x}{2} \wedge c' = c + |3x|)) \wedge$
$((tr = t_2 \wedge pc_2 = p_2^2)$
$\Rightarrow (d = 1 \wedge d' = 1 \wedge x' = x + 1 \wedge c' = c + |x + 1|))$

Figure 7.3: Example of the SSMT encoding of step-bounded cost expectation. The real-valued variable c is used to accumulate the costs of the individual transition steps within the SSMT encoding. Note that the domains of the randomized variables pc_1 and pc_2 are omitted for the sake of clarity. (This figure is a slight modification of Figure 3 from [FTE10b].)

Moreover, Lemma 7.1 in principle provides the opportunity to additionally compute *upper* bounds on the infimal cost expectation. To realize the latter, guessing some function C is required such that $C \geq CE_{(\mathcal{S},cost),Target}$. It seems that determining a non-trivial such function C is not straightforward in general. Though the constant function assigning ∞ to all states can always be used, it cannot be expected that this choice leads to meaningful results in general. The issue of determining some "good" function $C \geq CE_{(\mathcal{S},cost),Target}$ might be an interesting direction for future research. In the remainder of this chapter, we devote our attention to an SSMT-based verification approach to the CEMC problem from Definition 7.4.

7.3.2 SSMT encoding of step-bounded cost expectation

In order to establish a *symbolic* verification procedure for the CEMC problem from Definition 7.4, we now explain how the k-step minimum cost expectation for reaching the target states from the initial state, i.e. $CE^k_{(\mathcal{S},cost),Target}(\imath, 0, \mathbf{0})$, can be encoded as an SSMT formula. For this purpose, let $(\mathcal{S}, cost)$ be a system of n concurrent PHAs with cost function, $\imath \models \bigwedge_{i=1}^n init_i$ be the unique initial state of $\mathcal{S}$, *Target* be a $\mathcal{T}$-predicate over variables in $D_1, \ldots, D_n$ and $R_1, \ldots, R_n$ defining the target states, and $k \in \mathbb{N}$ be a step bound.

The overall idea of the SSMT encoding scheme basically is the same as the reduction from probabilistic bounded reachability to SSMT, the latter being formalized in Section 5.3. The intuition is illustrated by Figure 7.3. As in the case of probabilistic bounded reachability, the non-deterministic selection of transitions as well as the probabilistic choices of transition alternatives are encoded by existential and randomized quantification, respectively. In order to accumulate the costs of the individual transition steps, we introduce a fresh real-valued variable $c \in [0, u_c]$. The initial state predicate forces that $c = 0$ holds initially, while the transition relation predicate is responsible for incrementing the accumulated cost by the cost of the current transition step, i.e. $c' = c + \mathit{cost}(tr, pc)$ whenever $tr \in \mathit{NChoice}$ and $pc \in \mathit{PChoice}(tr)$ were selected. Once the target states are reached, the SSMT encoding must ensure that the accumulated cost remains unchanged in the subsequent system steps, i.e. $c' = c$, which is in conformity with Definition 7.6. A more fundamental difference to the SSMT encoding of probabilistic bounded reachability is that *all* system runs need to be considered and not only those which reach the target states within k transition steps. That is, the SSMT encoding $\Phi(k)$ of the k-step minimum cost expectation must *not* enforce reachability of the target states within k steps. This is due to the fact that $\mathit{CE}^0_{(\mathcal{S},\mathit{cost}),\mathit{Target}}(s, c, \mathbf{0}) = c$ regardless of whether s is a target state or not, confer Definition 7.6.

Assuming such SSMT encoding $\Phi(k)$ of step depth k, we are able to determine the k-step minimum cost expectation as follows. Observe that we are interested in minimizing the expected value of the accumulated cost at depth k, the latter being encoded by copy c_k of variable c. However, the *maximum* conditional expectation $E_{c_k}(\Phi(k))$ of c_k does *not* correspond to the k-step *minimum* cost expectation. We thus aim at the maximum conditional expectation $E_{ic}(\Phi(k))$ of the *additive inverse* $ic \in [-u_c, 0]$ of c_k, i.e. $ic = -c_k$. Finally, the additive inverse of this result, i.e. $-E_{ic}(\Phi(k))$, then obviously gives the k-step minimum cost expectation. The latter statement relies on the simple observation that $\min(a_1, \ldots, a_m) = -\max(-a_1, \ldots, -a_m)$.

It is important to remark that variable c must range over an interval domain $[0, u_c]$ which calls for an upper bound u_c. This is required by Definition 7.5 as variable $ic \in [-u_c, 0]$ is the designated variable in the SSMT formula $\Phi(k)$. In practice, this need not be a huge restriction as, first, u_c can be chosen arbitrarily large and, second, in cases where the maximum value $\mathit{cost}_{\max}$ of the individual transition costs exists and is known, which is most often the case in industrial applications when considering entities like time, position, or velocity, the value of u_c can be safely set to $k \cdot \mathit{cost}_{\max}$ with k being the step bound from Definition 7.6.

We now introduce the formalized reduction scheme thereby reusing essential parts of the reduction scheme from Section 5.3.

At first, all *variables* according to reduction steps 1 to 4 are introduced. In order to *accumulate the costs* of the individual transition steps, we further take $k+1$ fresh real-valued variables c_j for $0 \leq j \leq k$, each with real-valued interval domain $[0, u_c]$. The value of c_j encodes the accumulated cost at depth j. For the *additive inverse* of variable c_k, we introduce variable ic with real-valued interval domain $[-u_c, 0]$. The *initial state* predicate is given by

$$\mathit{INIT}_{(\mathcal{S},\mathit{cost})}(0) := \mathit{INIT}_{\mathcal{S}}(0) \wedge c_0 = 0$$

where $\mathit{INIT}_{\mathcal{S}}(0)$ is the predicate introduced by reduction step 5. With regard to the *transition relation* predicate encoding a transition step from depth $j-1$ to j, we conjoin the predicate $\mathit{TRANS}_{\mathcal{S}}(j-1, j)$ of reduction step 8 with a constraint system for incrementing the accumulated cost by the cost of the current transition step, i.e.

$$\begin{array}{rcl}\mathit{TRANS}_{(\mathcal{S},\mathit{cost})}(j-1, j) & := & \mathit{TRANS}_{\mathcal{S}}(j-1, j) \\ & & \wedge \bigwedge_{i=1}^{n} \bigwedge_{tr \in Tr_i} \bigwedge_{pc \in PC_{tr}} \left(\begin{array}{l} (tr^i_j = tr \wedge pc^{tr}_j = pc) \Rightarrow \\ c_j = c_{j-1} + \mathit{cost}(tr, pc)[d^1_{1,j-1}, d^1_{1,j}, \ldots, x^n_{m_n,j-1}, x^n_{m_n,j} / \\ \qquad d^1_1, d'^1_1, \ldots, x^n_{m_n}, x'^n_{m_n}] \end{array} \right)\end{array}$$

where in term $cost(tr, pc)$, confer Definition 7.1, each undecorated variable v is substituted by its representative v_{j-1} at depth $j-1$, and each primed variable v' is replaced by v_j for depth j. The *self loop* predicate $SELF\_LOOP_{\mathcal{S}}(j-1, j)$ of reduction step 10 is slightly extended such that the value of the accumulated cost is preserved:

$$SELF\_LOOP_{(\mathcal{S},cost)}(j-1,j) := SELF\_LOOP_{\mathcal{S}}(j-1,j) \wedge c_j = c_{j-1}$$

We are now prepared to compile the matrix $BMC_{(\mathcal{S},cost),\,Target}(k)$ of the overall SSMT formula:

$$\begin{array}{rcl} BMC_{(\mathcal{S},cost),\,Target}(k) & := & INIT_{(\mathcal{S},cost)}(0) \\ & \wedge & \displaystyle\bigwedge_{j=1}^{k} \left(\begin{array}{l} (\neg TARGET(j-1) \Rightarrow TRANS_{(\mathcal{S},cost)}(j-1,j)) \\ \wedge(TARGET(j-1) \Rightarrow SELF\_LOOP_{(\mathcal{S},cost)}(j-1,j)) \end{array} \right) \\ & \wedge & ic = -c_k \end{array}$$

with $TARGET(j-1)$ being the target states predicate from reduction step 9. The predicate $ic = -c_k$ ensures that variable ic actually represents the additive inverse of variable c_k. Observe that above SMT formula $BMC_{(\mathcal{S},cost),\,Target}(k)$ has as its models both runs reaching the target states, in which case the accumulated cost is kept constant after having reached the target states, and runs not reaching the target within k transition steps. The rationale is that the accumulated costs of the latter runs still provide safe lower bounds on the accumulated costs of their extensions.

Finally, we construct the SSMT formula $CEMC_{(\mathcal{S},cost),\,Target}(k)$ from the corresponding quantifier prefixes $CHOICE_{\mathcal{S}}(j)$ of reduction step 13, encoding the non-deterministic and probabilistic selection of transitions and transition alternatives by existential and randomized quantification, respectively, and from the SMT formula $BMC_{(\mathcal{S},cost),\,Target}(k)$:

$$CEMC_{(\mathcal{S},cost),\,Target}(k) := \left(\bigodot_{j=1}^{k} CHOICE_{\mathcal{S}}(j) \right) : BMC_{(\mathcal{S},cost),\,Target}(k)$$

Given the structural similarity between step-bounded minimum cost expectation for PHAs and maximum conditional expectation for SSMT, the above reduction is correct in the following sense.

Proposition 7.1 (Correctness of reduction)
Let be given a system $(\mathcal{S}, cost)$ of n concurrent PHAs with cost function, a $\mathcal{T}$-predicate Target over variables in $D_1, \dots, D_n$ and $R_1, \dots, R_n$ defining the target states, and a step bound $k \in \mathbb{N}$. Then,

$$CE^{k}_{(\mathcal{S},cost),\,Target}(\imath, 0, \mathbf{0}) \quad = \quad -E_{ic}(CEMC_{(\mathcal{S},cost),\,Target}(k))$$

holds with $\imath \models \bigwedge_{i=1}^{n} init_i$ being the (unique) global initial state of $\mathcal{S}$.

Above Proposition 7.1 is presented without a formal proof. We remark that such a proof is very similar to the one of Theorem 5.1, the latter establishing correctness of the reduction from probabilistic bounded reachability to SSMT. That is to say, a potential proof of Proposition 7.1 is based on a rather technical induction over the step bound $k \in \mathbb{N}$. Some particular attention must be devoted to the duality between *minimization* of the costs in $CE^{k}_{(\mathcal{S},cost),Target}(\imath, 0, \mathbf{0})$ and *maximization* of the expectation in $-E_{ic}(CEMC_{(\mathcal{S},cost),Target}(k))$. As already mentioned above, this issue is handled by considering the *additive inverse* of the *maximum* expectation of the *negative* accumulated cost. Soundness of the latter treatment can be simply derived from the fact that $\min(a_1, \ldots, a_m) = -\max(-a_1, \ldots, -a_m)$.

Note that the maximum conditional expectation $E_{ic}(CEMC_{(\mathcal{S},cost),Target}(k))$ of variable *ic* given SSMT formula $CEMC_{(\mathcal{S},cost),Target}(k)$ can be determined by a procedure for computing maximum conditional expectations for SSMT formulae. Such an SSMT algorithm is investigated in the next section.

7.4 SSMT algorithm for conditional expectation

In order to complete the symbolic verification procedure for the CEMC problem from Definition 7.4, it remains to describe how the SSMT encoding $CEMC_{(\mathcal{S},cost),Target}(k)$ of the k-step minimum cost expectation can be solved algorithmically. For this purpose, we elaborate on an SSMT algorithm for computing the maximum conditional expectation of a designated free variable in a given SSMT formula.

The key idea of this algorithm is mainly the same as the one of the SSMT procedure for determining the maximum probability of satisfaction, the latter being explained in Section 6.4. The algorithm implements a backtracking search that mimics the extended semantics of SSMT, confer Definition 7.5, thereby explicitly traversing the tree spanned by the assignments to the quantified variables and recursively computing the individual conditional expectations for all subtrees, as indicated on the right of Figure 7.2. Upon each complete assignment to the quantified variables, intuitively, in each leaf of the quantifier tree, the algorithm has to maximize the value $\tau(y)$ of the designated variable y over all solutions τ of the induced quantifier-free SMT subformula.

Recall that the number of such complete assignments is exponential in the number of quantified variables and that a naive SSMT approach traversing the whole quantifier tree is far from scalable. With regard to the SSMT procedure for determining the maximum probability of satisfaction, we have proposed a range of algorithmic enhancements in order to prune the quantifier tree considerably, as illustrated in Figure 6.5 b, and thus to improve performance of SSMT solvers in practice, confer Sections 6.4, 6.5, and 6.7. Concerning the SSMT algorithm addressing the maximum conditional expectation, we remark that the majority of the algorithmic enhancements from Sections 6.4 and 6.5 can be adapted to this more general case. Since this adaptation calls for a plethora of technical issues, we just consider the feature of *thresholding* for the sake of convenience.

In what follows, we explain the SSMT algorithm addressing the maximum conditional expectation which is presented in Figure 7.4. In addition to an SSMT formula $\mathcal{Q} : \varphi$ and a non-quantified variable $y \in Var(\varphi)$ with interval domain $[l_y, u_y]$, the algorithm $\text{MaxExp}(\mathcal{Q} : \varphi, y, \theta_l, \theta_u)$ further requires as inputs two rational constants θ_l and θ_u with $\theta_l \leq \theta_u$ which are called *lower threshold* and *upper threshold*, respectively. Akin to the case of computing the maximum probability of satisfaction, the idea of the additional parameters θ_l and θ_u is as follows. If the maximum conditional expectation $E_y(\mathcal{Q} : \varphi)$ lies in the interval $[\theta_l, \theta_u]$ then we aim at computing the exact expectation, i.e. the result of

```
MaxExp($\mathcal{Q} : \varphi, y, \theta_l, \theta_u$)
  input: SSMT formula $\mathcal{Q} : \varphi$, variable $y$ with interval domain $[l_y, u_y]$ such that
         $y \notin \mathit{Var}(\mathcal{Q})$ and $y \in \mathit{Var}(\varphi)$, rational constants $\theta_l, \theta_u$ with $\theta_l \leq \theta_u$.

  if $\mathcal{Q} = \varepsilon$ then return MaxSol($\varphi, y$).
  $e_y := l_y$.
  if $\mathcal{Q} = \exists x \in \mathcal{D}\ \mathcal{Q}'$ then
    while $\mathcal{D} \neq \emptyset$ do
      if $e_y > \theta_u$ or $e_y = u_y$ then return $e_y$.
      select $v \in \mathcal{D}$, $\mathcal{D} := \mathcal{D} - \{v\}$.
      $\theta'_l := \max(\theta_l, e_y)$.
      $e'_y :=$ MaxExp($\mathcal{Q}' : \varphi[v/x], y, \theta'_l, \theta_u$).
      $e_y := \max(e_y, e'_y)$.
    return $e_y$.
  if $\mathcal{Q} = \text{Ꙇ}_d x \in \mathcal{D}\ \mathcal{Q}'$ then
    while $\mathcal{D} \neq \emptyset$ do
      if $e_y > \theta_u$ or $e_y = u_y$ then return $e_y$.
      $p_{remain} := \sum_{w \in \mathcal{D}} d(w)$.
      if $e_y + p_{remain} \cdot (u_y - l_y) < \theta_l$ then return $e_y$.
      select $v \in \mathcal{D}$, $\mathcal{D} := \mathcal{D} - \{v\}$.
      $\theta'_l := l_y + (\theta_l - e_y - (p_{remain} - d(v)) \cdot (u_y - l_y))/d(v)$.
      $\theta'_u := l_y + (\theta_u - e_y)/d(v)$.
      $e'_y :=$ MaxExp($\mathcal{Q}' : \varphi[v/x], y, \theta'_l, \theta'_u$).
      $e_y := e_y + d(v) \cdot (e'_y - l_y)$.
    return $e_y$.
```

Figure 7.4: SSMT algorithm for computing the maximum conditional expectation of variable y given SSMT formula $\mathcal{Q} : \varphi$, thereby exploiting the algorithmic enhancement of thresholding. (This figure is substantially based on Algorithm 1 from [FTE10b] but slightly differs in presentation.)

MaxExp($\mathcal{Q} : \varphi, y, \theta_l, \theta_u$) should be equal to $E_y(\mathcal{Q} : \varphi)$. Otherwise, i.e. $E_y(\mathcal{Q} : \varphi) \notin [\theta_l, \theta_u]$, the exact expectation is not called for but only some witness value $pr =$ MaxExp($\mathcal{Q} : \varphi, y, \theta_l, \theta_u$) suffices such that $pr < \theta_l$ if and only if $E_y(\mathcal{Q} : \varphi) < \theta_l$ and $pr > \theta_u$ if and only if $E_y(\mathcal{Q} : \varphi) > \theta_u$.

It is important to remark that the above requirement can only be guaranteed if arithmetic theories are considered for which the optimization problem is decidable like, for instance, linear arithmetic. Recall that we address the undecidable theory of non-linear arithmetic over the reals and integers involving transcendental functions within this thesis, confer Section 4.3. We therefore relax the above conditions and only demand that, first, if $E_y(\mathcal{Q} : \varphi) \in [\theta_l, \theta_u]$ then the result must be a safe upper estimate, i.e. $E_y(\mathcal{Q} : \varphi) \leq$ MaxExp($\mathcal{Q} : \varphi, y, \theta_l, \theta_u$) and, second, if $E_y(\mathcal{Q} : \varphi) > \theta_u$ then also MaxExp($\mathcal{Q} : \varphi, y, \theta_l, \theta_u$) $> \theta_u$. From the above, it then follows that if MaxExp($\mathcal{Q} : \varphi, y, \theta_l, \theta_u$) $\leq \theta_l$ then $E_y(\mathcal{Q} : \varphi) \leq \theta_l$.

The thresholds θ_l and θ_u are exploited during proof search in order to improve efficiency by skipping some recursive calls of MaxExp($\mathcal{Q} : \varphi, y, \theta_l, \theta_u$). This algorithmic enhancement is called *thresholding* and is motivated, for instance, by the CEMC problem from Definition 7.4, where the question is whether the infimal cost expectation is below or above some acceptable threshold value θ. To address the step-bounded CEMC problem, it thus suffices to set both the lower and upper threshold to θ, i.e. $\theta_l = \theta_u = \theta$. Conversely, the algorithm can still be used for targeting at the exact expectation, namely by setting the lower and upper threshold to the respective domain limits of the designated variable y, i.e. $\theta_l = l_y$ and $\theta_u = u_y$.

In detail, the algorithm MaxExp($\mathcal{Q} : \varphi, y, \theta_l, \theta_u$) from Figure 7.4 works as follows. Whenever the quantifier prefix $\mathcal{Q}$ is empty, i.e. if $\mathcal{Q} = \varepsilon$, then the subroutine MaxSol(φ, y) is called. The latter routine seeks for a solution τ of the quantifier-free SMT problem φ which maximizes the value of variable y, and then returns the value $\tau(y)$ according to the extended semantics of SSMT from Definition 7.5. If φ is unsatisfiable, i.e. if no solution exists, then the minimum domain value l_y is given back. For maximizing the value of y in quantifier-free SMT problems, we employ bifurcation search in interval-based SMT solving, confer Section 6.3, which yields safe upper estimates on the actual maximum due to the conservative approximation provided by interval arithmetic. Observe that this approach is in conformity with the relaxed requirement on the output of the algorithm.

In case the quantifier prefix $\mathcal{Q}$ is non-empty, the algorithm MaxExp($\mathcal{Q} : \varphi, y, \theta_l, \theta_u$) descends recursively through the quantifiers in $\mathcal{Q}$, yet prunes branches by *thresholding*, which skips subproblems when detecting that an upper threshold already is exceeded or that a lower threshold can no longer be reached. This optimization builds on the monotonicity argument that the intermediate values e_y^i, i.e. the value of e_y after the i-th iteration of the corresponding while loop, confer Figure 7.4, of the final expectation result e_y are never decreasing, i.e. $e_y^{i+1} \geq e_y^i$, and do all establish lower bounds on e_y, i.e. $e_y \geq e_y^i$. Such monotonicity cannot be expected from a straightforward implementation of the semantics from Definition 7.5 as, for a randomized quantifier $\text{Я}_{d_x} x \in \mathcal{D}_x$, the terms of the sum $\sum_{v \in \mathcal{D}_x} d_x(v) \cdot E_y(\mathcal{Q} : \varphi[v/x])$ may be both positive and negative.

To achieve the above monotonicity argument, the return value e_y is initialized with the minimum domain value l_y of the designated variable y. For an existential quantifier, i.e. if $\mathcal{Q} = \exists x \in \mathcal{D}\ \mathcal{Q}'$, we replace the provisional return value by the actual value e'_y of the current subproblem whenever the latter is larger, which is a monotonic process. In the randomized case, i.e. if $\mathcal{Q} = \text{Я}_d x \in \mathcal{D}\ \mathcal{Q}'$, we accumulate the non-negative increases $d(v) \cdot (e'_y - l_y) \geq 0$ with respect to the minimum domain value l_y of the individual expectations $e'_y \geq l_y$. The increasing partial sums for e_y finally yield the desired expectation since $l_y + \sum_{v \in \mathcal{D}} d(v) \cdot (e_y^v - l_y) = l_y + \sum_{v \in \mathcal{D}} d(v) \cdot e_y^v - \sum_{v \in \mathcal{D}} d(v) \cdot l_y = \sum_{v \in \mathcal{D}} d(v) \cdot e_y^v$ due to $\sum_{v \in \mathcal{D}} d(v) = 1$.

With such monotonic estimates, we are able to safely conclude that an upper threshold θ_u is exceeded or that the maximum possible domain value u_y is reached, namely by checking whether the intermediate expectation e_y is greater than θ_u or equal to u_y, respectively, i.e. whether $e_y > t_u$ or $e_y = u_y$ holds. In case of success, we skip investigation of all remaining subproblems and return the witness value e_y. For randomized variables, we can also safely determine that the lower threshold θ_l can no longer be reached. This detection depends on the probability mass p_{remain} of all values $w \in \mathcal{D}$ not yet explored.

Note that the maximum possible expectation result of a subproblem is the maximum domain value u_y. As a consequence, the maximum possible increase of a subproblem is $d(v) \cdot (u_y - l_y)$ and thus of all remaining branches is $p_{remain} \cdot (u_y - l_y)$. From the latter, we can derive a safe upper bound of the final expectation result, namely $e_y + p_{remain} \cdot (u_y - l_y)$. If this upper bound is strictly less than the lower threshold θ_l then θ_l is unreachable and the witness value e_y can be returned without exploring any remaining branch.

We finally explain how the thresholds are determined for analyzing the subordinate quantifiers. In the existential case, we use the lower threshold $\theta'_l = \max(\theta_l, e_y)$ for solving the induced subproblem, where e_y is the current intermediate expectation, since subproblems with expectation less than e_y do not lead to a modification of the current intermediate expectation. The upper threshold θ_u remains unchanged, since if the result e'_y of the subproblem is above θ_u then the same holds for the updated expectation result $e_y := \max(e_y, e'_y) > \theta_u$.

In the randomized case, the lower threshold $\theta'_l = l_y + (\theta_l - e_y - (p_{remain} - d(v)) \cdot (u_y - l_y))/d(v)$ is used for the induced subproblem. The rationale is as follows: $e'_y < \theta'_l$ if and only if $e'_y < l_y + (\theta_l - e_y - (p_{remain} - d(v)) \cdot (u_y - l_y))/d(v)$ if and only if $e_y + d(v) \cdot (e'_y - l_y) + (p_{remain} - d(v)) \cdot (u_y - l_y)) < \theta_l$ with e'_y being the result of the induced subproblem. That is to say, the result e'_y of the subproblem is below the lower threshold θ'_l if and only if the intermediate expectation result after the current iteration of the while loop, i.e. $e_y + d(v) \cdot (e'_y - l_y)$, suffices to prove that the lower threshold θ_l is unreachable. The corresponding upper threshold is $\theta'_u = l_y + (\theta_u - e_y)/d(v)$ with the reason that $e'_y > \theta'_u$ if and only if $e'_y > l_y + (\theta_u - e_y)/d(v)$ if and only if $e_y + d(v) \cdot (e'_y - l_y) > \theta_u$. That is, the result e'_y of the subproblem is above the upper threshold θ'_u if and only if the intermediate expectation result after the current iteration of the while loop, i.e. $e_y + d(v) \cdot (e'_y - l_y)$, exceeds the upper threshold θ_u.

With regard to a symbolic verification procedure for the CEMC problem from Definition 7.4, we finally remark that the above SSMT algorithm MaxExp is able to *safely verify* that $CE_{(\mathcal{S},cost),Target}(\imath) \geq \theta$ holds. This is due to the following. Let us assume that the result of the SSMT algorithm satisfies

$$\mathrm{MaxExp}(CEMC_{(\mathcal{S},cost),Target}(k), ic, -\theta, -\theta) \ \leq\ -\theta \quad .$$

The relaxed requirement on the result of the algorithm then ensures that inequality

$$E_{ic}(CEMC_{(\mathcal{S},cost),Target}(k)) \ \leq\ -\theta$$

holds. The latter is true if and only if

$$-E_{ic}(CEMC_{(\mathcal{S},cost),Target}(k)) \ \geq\ \theta \quad .$$

From Lemma 7.1, i.e. from $CE_{(\mathcal{S},cost),Target}(\imath) \geq CE^k_{(\mathcal{S},cost),Target}(\imath, 0, \mathbf{0})$, and from Proposition 7.1, i.e. from $CE^k_{(\mathcal{S},cost),Target}(\imath, 0, \mathbf{0}) = -E_{ic}(CEMC_{(\mathcal{S},cost),Target}(k))$, we finally conclude that

$$CE_{(\mathcal{S},cost),Target}(\imath) \ \geq\ CE^k_{(\mathcal{S},cost),Target}(\imath, 0, \mathbf{0}) \ =\ -E_{ic}(CEMC_{(\mathcal{S},cost),Target}(k)) \ \geq\ \theta$$

holds which decides the general CEMC problem to be true.

```
This is SiSAT 1.0.

Usage: sisat --i <inputfile.hys> [options]

General iSAT options:
...

BMC-related options:
...

SiSAT options:
...
  --cond-exp=var : Specifies the variable for which the conditional
                   expectation should be computed. For transition
                   systems, the variable instance of the last step
                   depth is taken.
  --lt-exp=[real]: Lower threshold for conditional expectation
                   (default: minimum value of domain of 'var')
  --ut-exp=[real]: Upper threshold for conditional expectation
                   (default: maximum value of domain of 'var')
```

Figure 7.5: Excerpt of the extended help menu of SiSAT, which is printed when calling the tool without specifying an input file.

It is moreover possible to compute *safe lower bounds* of the infimal cost expectation $CE_{(\mathcal{S},cost),Target}(\imath)$ for reaching the target states from the initial state. To this end, we need to set the lower and upper thresholds to the minimum and maximum domain values of variable $ic \in [-u_c, 0]$, respectively, i.e. $\theta_l = -u_c$ and $\theta_u = 0$. It then holds that

$$E_{ic}(CEMC_{(\mathcal{S},cost),Target}(k)) \;\leq\; \mathrm{MaxExp}(CEMC_{(\mathcal{S},cost),Target}(k), ic, -u_c, 0)$$

and thus

$$-E_{ic}(CEMC_{(\mathcal{S},cost),Target}(k)) \;\geq\; -\mathrm{MaxExp}(CEMC_{(\mathcal{S},cost),Target}(k), ic, -u_c, 0) \quad .$$

Using the same arguments as above, we infer that

$$CE_{(\mathcal{S},cost),Target}(\imath) \;\geq\; -\mathrm{MaxExp}(CEMC_{(\mathcal{S},cost),Target}(k), ic, -u_c, 0) \quad .$$

7.5 Experimental results

In the previous sections of this chapter, we have introduced a symbolic verification procedure for the CEMC problem from Definition 7.4 based on extended SSMT solving. In this section, we finally demonstrate practical applicability of the suggested approach on a small case study. In order to achieve an automatic verification procedure, the SSMT algorithm for computing the maximum conditional expectation of a designated free variable in a given SSMT formula, as presented in Section 7.4, has been integrated into the SSMT-based probabilistic bounded model checker SiSAT, the latter tool being described in Section 6.6. While the input language of SiSAT remains unchanged, further tool options have been added, which are listed in the excerpt of the extended help menu of SiSAT shown in Figure 7.5.

With regard to the case study used in the experiments, we were interested in the mean time to failure (MTTF) analysis of the cooling system $\mathcal{S}_{cool}$, the latter being described in Subsection 6.6.3 and depicted in Figure 6.17. To this end, observe that the cost of an individual transition step corresponds to its duration, which is 0 for a discrete transition step and $dt > 0$ for a step modeling continuous evolution, and that a system failure occurs whenever temperature T has left the safe region $[T^{low}_{safe}, T^{up}_{safe}]$. Let *cost* be the corresponding cost function indicated above, *Target* be a $\mathcal{T}$-predicate describing above system failures, and $\imath$ be the unique initial state of $\mathcal{S}_{cool}$. Then, the k-step minimum cost expectation $CE^k_{(\mathcal{S}_{cool}, cost), Target}(\imath, 0, \mathbf{0})$ gives the worst-case, i.e. minimum, MTTF of $\mathcal{S}_{cool}$ for k system steps, which we also call k-step minimum MTTF, and thus a lower estimate of the worst-case, i.e. infimal, MTTF. This enables us to potentially verify that the cooling system $\mathcal{S}_{cool}$ meets, for instance, the safety requirement that its MTTF is always at least 8 time units.

We remark that Chapter 8 deals with a more sophisticated case study involving the computation of expected values for the networked automation system (NAS) introduced in Section 3.1.

SiSAT encoding of $(\mathcal{S}_{cool}, cost)$**.** The SSMT encoding of the k-step minimum cost expectation for $(\mathcal{S}_{cool}, cost)$ is based on the *alternative* SiSAT encoding of $\mathcal{S}_{cool}$ from Subsection 6.6.3. Recall that the usage of the alternative encoding of $\mathcal{S}_{cool}$, supporting the idea of “disabling” quantifiers, has led to tremendous performance gains of multiple orders of magnitude compared to the basic encoding, confer Section 6.7. In order to describe the step-bounded minimum cost expectation as in Proposition 7.1, the alternative SiSAT encoding of $\mathcal{S}_{cool}$ has to be adapted according to the reduction scheme of Subsection 7.3.2.

Observe that we do not need to introduce a fresh variable for accumulating the costs or rather the durations of the individual transition steps since variable `t` encodes the time elapsed so far, i.e. `t` accumulates the durations 0 and `dt` of the discrete and continuous steps, respectively. To represent the additive inverse of `t`, we add the fresh variable `it` to the `DECL` part:

```
DECL
...
   define dt          =  0.5;
...
   -- Time progress:
   define time_max = 20;
   float [0, time_max] t;
   -- Additive inverse of t.
   define time_min = -20;
   float [time_min , 0] it;
...
```

Observe that the interval domain of variable `t` is bounded from above by value 20 and thus the domain of the additive inverse `it` is bounded from below by -20. Since the cost of a transition step is at most `dt` which is fixed to 0.5 time units, the domains of `t` and `it` are valid up to step depth $k = 20/0.5 = 40$.

To ensure that variable `it` actually represents the additive inverse of variable `t` at each step depth, corresponding predicates are added to the `INIT` and `TRANS` parts:

```
INIT
...
    -- Additive inverse of t.
    it          = -t;
```

```
TRANS
...
    -- Additive inverse of t'.
    it'         = -t';
```

Observe that the SSMT encoding scheme of Subsection 7.3.2 defines the additive inverse of the accumulated cost at step depth k only, while the additive inverse in above SiSAT encoding is available at *each* step depth. This treatment obviously does not destroy correctness in the sense of Proposition 7.1 and is just for the sake of simplicity. That is to say, the input language as well as the front end of SiSAT can be reused without any modifications. Specification of the designated (cost) variable is done via a command-line option. The instance of this variable at step depth k is taken when considering probabilistic transition systems, confer Figure 7.5.

While the `DISTR` section remains unchanged, the predicate of the `TARGET` part is finally modified to be `true` (and, obviously, no self-loop is added to target states). Recall that the SSMT encoding of the k-step minimum cost expectation considers both runs reaching and runs not reaching the target states within k transition steps since such latter runs still yield safe lower bounds on the accumulated costs of their extensions.

```
TARGET
    true;
```

Analysis results. After having encoded the cooling system $(\mathcal{S}_{cool}, cost)$ with cost function into the SiSAT input language, we can now employ the SSMT-based probabilistic bounded model checker SiSAT, enhanced by the feature of computing the maximum conditional expectation of a designated variable, for the MTTF analysis of $(\mathcal{S}_{cool}, cost)$.

All experiments of this section were conducted on a 2.4 GHz AMD Opteron machine with 128 GByte physical memory running Linux. In order to compute lower estimates of the worst-case MTTF and to evaluate the impact of thresholding, we performed several SiSAT runs with different solver settings. In each run, the tool was called upon to solve the corresponding problem for step depths $k = 0$ to 40 with the instance of variable `it` at step depth k being the designated variable, i.e. for which the maximum conditional expectation was computed. The time limit of each SiSAT run was set to 25 hours.

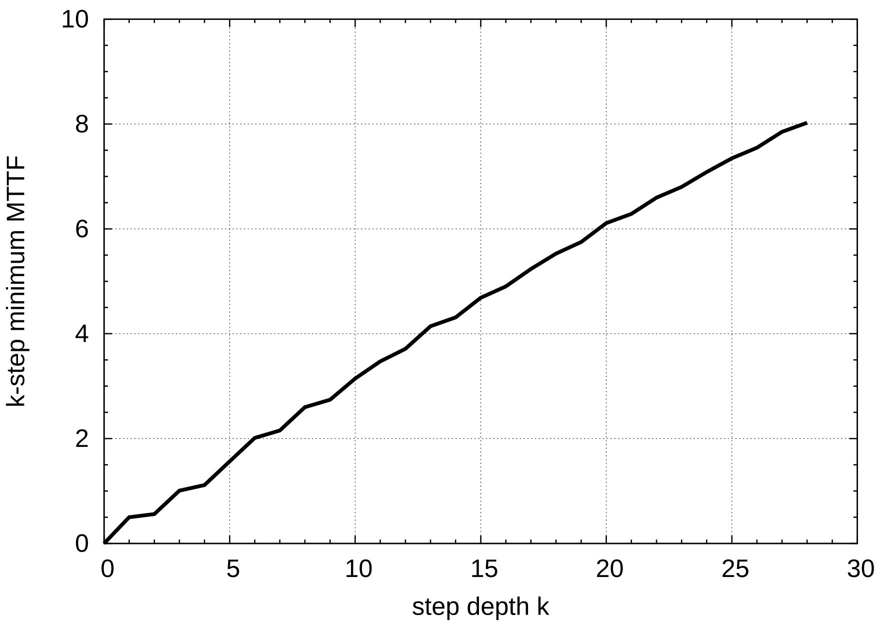


Figure 7.6: Analysis of cooling system ($\mathcal{S}_{cool}$, *cost*) with cost function: evolution of the k-step minimum MTTF over step depth k.

With regard to the computation of lower estimates of the worst-case MTTF, SiSAT was able to determine the k-step minimum cost expectation, i.e. the k-step minimum MTTF, up to step depth 28 within the time limit. Recall that SiSAT returns the additive inverse of the actual result at each depth k, confer Proposition 7.1. The evolution of the k-step minimum MTTF over step depth k is plotted in Figure 7.6. Since the 28-step minimum MTTF is at least 8.02239325 time units, the safety requirement mentioned above, namely that the MTTF of ($\mathcal{S}_{cool}$, *cost*) is always at least 8 time units, is actually verified.

Whenever it suffices to decide whether the k-step minimum MTTF is above some acceptable threshold value θ, we may exploit the algorithmic enhancement of *thresholding*. As explained in Section 7.4, the idea of thresholding is to skip investigation of subproblems if an upper threshold already is exceeded by the processed branches or if a lower threshold can no longer be reached by the remaining branches. With regard to above decision problem, we performed several SiSAT runs where the lower threshold θ_l and the upper threshold θ_u were set to the same value θ, i.e $\theta_l = \theta_u = \theta$. The corresponding experimental results for thresholding with different threshold values $\theta \in \{-20, -18, \ldots, -2, 0\}$ are depicted in Figure 7.7, where "naive" denotes the naive SSMT algorithm from Subsection 7.4 with thresholding being "disabled", i.e. with $\theta_l = -20$ and $\theta_u = 0$.

These experiments show that thresholding can be a very powerful pruning technique. For instance, the naive SiSAT algorithm has detected more than 6 million (approximate) solutions during proof search for the problem at step depth 28. When having enabled thresholding with one of the extreme threshold values $\theta = -20$ and $\theta = 0$, the number of detected (approximate) solutions decreased to 1 and to 0, respectively, while solving time could be improved by five and by four orders of magnitude. Moreover, when thresholding with $\theta = -20$ or $\theta = 0$ was applied, all SSMT problems, i.e. for step depths 0 to 40, were solved within the time limit, more precisely, even in less than 2 or 28 seconds, respectively.

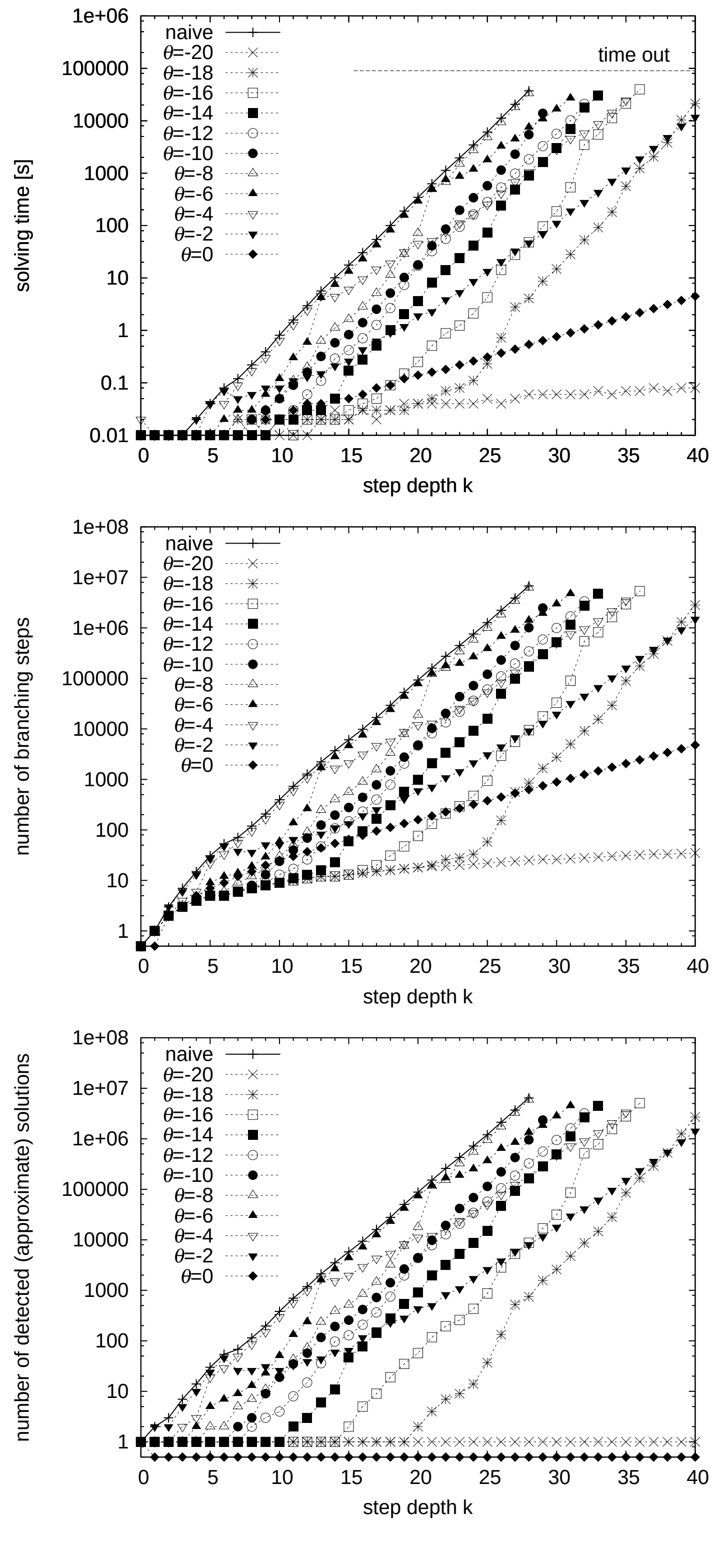


Figure 7.7: Impact of *thresholding*: solving times (top), numbers of branching steps (middle), and numbers of detected (approximate) solutions (bottom) for the corresponding step depths k when thresholding with different threshold values θ was applied.

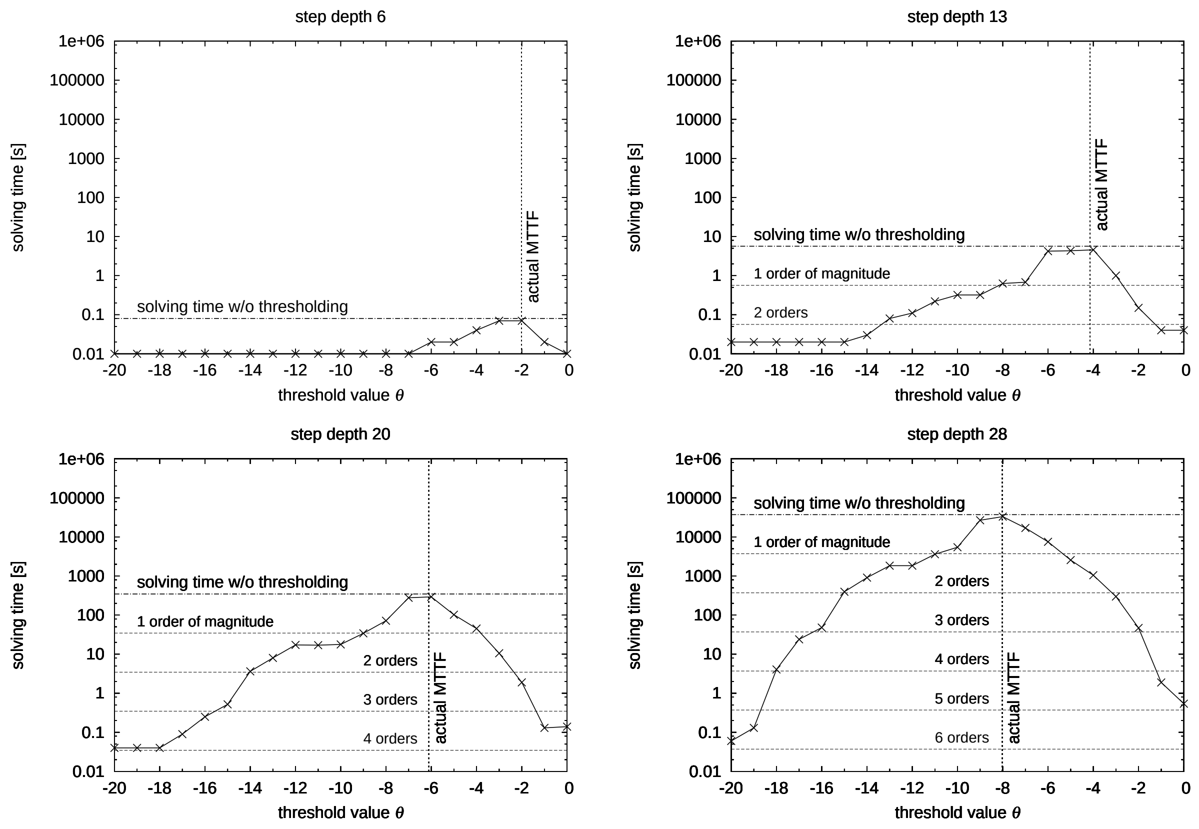


Figure 7.8: Impact of *thresholding*: solving times for the corresponding threshold values θ at step depths 6 (top left), 13 (top right), 20 (bottom left), and 28 (bottom right) when thresholding with different threshold values was applied. If applicable, the corresponding speed-ups in orders of magnitude with respect to the naive algorithm without thresholding are illustrated by dashed lines.

In order to reveal a potential correlation between the effectiveness of thresholding and the distance between the threshold value θ and the actual conditional expectation, Figure 7.8 plots the solving times for threshold values $\theta \in \{-20, -19, -18, \dots, -2, -1, 0\}$ at step depths 6, 13, 20, and 28. Such correlation was observed when dealing with the maximum probability of satisfaction of SSMT formulae, confer Subsections 6.7.2 and 6.7.3. The graphs of Figure 7.8 suggest the same circumstance when addressing the maximum conditional expectation. More precisely, it seems that the larger distance between threshold θ and the actual expectation the more powerful thresholding. The feature of thresholding can be an effective algorithmic optimization in practice, being able to reduce solving time significantly, sometimes by multiple orders of magnitude.

8 Case Study: A Networked Automation System

In this chapter, we apply the symbolic analysis procedures for probabilistic hybrid systems, as introduced in the previous chapters, to a realistic case study from the domain of networked automation systems (NAS) described in Section 3.1 and illustrated in Figure 3.1. For this purpose, we first present a formal model of the NAS in Section 8.1 which is in conformity with the notion of a system of concurrent discrete-time probabilistic hybrid automata from Section 3.3. Thereafter, we devote our attention to the SSMT-based probabilistic reachability as well as expected-value analysis of the NAS in Section 8.2.

We remark that essential parts of this chapter were published in [TEF11] and in [FTE10b] by the author of this thesis together with his co-authors.

8.1 Formel model of the NAS

The NAS case study was modeled as a system of 10 concurrent probabilistic hybrid automata. A graphical representation of this formal NAS model is given in Figures 8.1, 8.2, and 8.3. In order to get an intuition of this rather large model as well as of the interaction between the single automata, we first explain intuitively the basic ideas of each automaton and then exemplify the interconnections in the model by means of a sample system run.

Explanation of the model. The global time variable t and the step duration variable dt are governed by the automaton time progress scheduler from Figure 8.1. The duration of a transition step of the overall system is given by the minimum of the maximum possible step durations of all automata. These latter durations are encoded by variables s_i with corresponding indices i, while each variable s_i is local to some automaton. For instance, the value of variable s_{obj} specifies the maximum possible step duration of automaton object from Figure 8.1.

The automaton object models the workpiece to be transported on the conveyer belt to the drilling position at 0 lu. The automaton updates the position of the object, represented by variable x, depending on the step duration dt, the current speed $\dot{x}$, and the current deceleration $\ddot{x}$. The variables $\dot{x}$ and $\ddot{x}$ are just read by object as their values are computed by automaton transportation unit. Furthermore, object locally determines its maximum possible step duration s_{obj} as well as the time point n_{obj} of its next event. Initially, the next event of object is reaching sensor SA. In this case, n_{obj} is the non-negative value satisfying the equation $X_{\mathrm{sA}} = x-(n_{\mathrm{obj}}-t)\cdot\dot{x}+\frac{1}{2}\cdot(n_{\mathrm{obj}}-t)^2\cdot\ddot{x}$, where X_{sA} is the fixed position of sensor SA and all other variables are assigned their initial values. The automaton consists of the three locations obj_preA, obj_betwAB, and obj_postB meaning that the workpiece has not yet reached sensor SA, is in between both sensors SA and SB, and passed sensor SB, respectively. Reaching sensors SA and SB are indicated by setting the Boolean variables

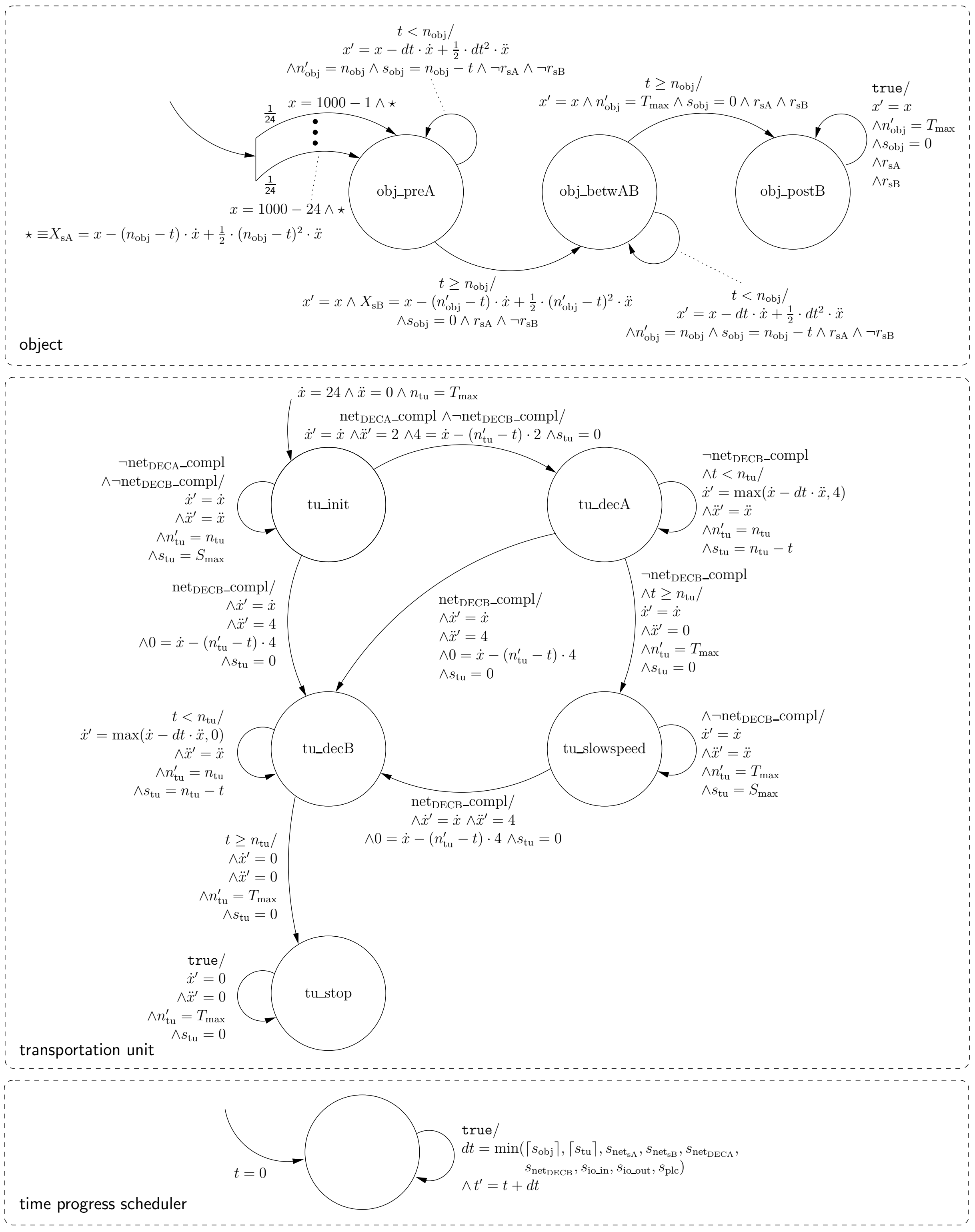

$t < n_{\mathrm{obj}}/$
$x' = x - dt \cdot \dot{x} + \frac{1}{2} \cdot dt^2 \cdot \ddot{x}$
$\wedge n'_{\mathrm{obj}} = n_{\mathrm{obj}} \wedge s_{\mathrm{obj}} = n_{\mathrm{obj}} - t \wedge \neg r_{\mathrm{sA}} \wedge \neg r_{\mathrm{sB}}$
$t \geq n_{\mathrm{obj}}/$
$x' = x \wedge n'_{\mathrm{obj}} = T_{\max} \wedge s_{\mathrm{obj}} = 0 \wedge r_{\mathrm{sA}} \wedge r_{\mathrm{sB}}$
true/
$x' = x$
$\wedge n'_{\mathrm{obj}} = T_{\max}$
$\wedge s_{\mathrm{obj}} = 0$
$\wedge r_{\mathrm{sA}}$
$\wedge r_{\mathrm{sB}}$
$\frac{1}{24}$
$x = 1000 - 1 \wedge \star$
$\frac{1}{24}$
$x = 1000 - 24 \wedge \star$
$\star \equiv X_{\mathrm{sA}} = x - (n_{\mathrm{obj}} - t) \cdot \dot{x} + \frac{1}{2} \cdot (n_{\mathrm{obj}} - t)^2 \cdot \ddot{x}$
obj_preA
obj_betwAB
obj_postB
$t \geq n_{\mathrm{obj}}/$
$x' = x \wedge X_{\mathrm{sB}} = x - (n'_{\mathrm{obj}} - t) \cdot \dot{x} + \frac{1}{2} \cdot (n'_{\mathrm{obj}} - t)^2 \cdot \ddot{x}$
$\wedge s_{\mathrm{obj}} = 0 \wedge r_{\mathrm{sA}} \wedge \neg r_{\mathrm{sB}}$
$t < n_{\mathrm{obj}}/$
$x' = x - dt \cdot \dot{x} + \frac{1}{2} \cdot dt^2 \cdot \ddot{x}$
$\wedge n'_{\mathrm{obj}} = n_{\mathrm{obj}} \wedge s_{\mathrm{obj}} = n_{\mathrm{obj}} - t \wedge r_{\mathrm{sA}} \wedge \neg r_{\mathrm{sB}}$
object
$\dot{x} = 24 \wedge \ddot{x} = 0 \wedge n_{\mathrm{tu}} = T_{\max}$
$\mathrm{net}_{\mathrm{DECA}}\text{\_compl} \wedge \neg \mathrm{net}_{\mathrm{DECB}}\text{\_compl}/$
$\dot{x}' = \dot{x} \wedge \ddot{x}' = 2 \wedge 4 = \dot{x} - (n'_{\mathrm{tu}} - t) \cdot 2 \wedge s_{\mathrm{tu}} = 0$
$\neg \mathrm{net}_{\mathrm{DECA}}\text{\_compl}$
$\wedge \neg \mathrm{net}_{\mathrm{DECB}}\text{\_compl}/$
$\dot{x}' = \dot{x}$
$\wedge \ddot{x}' = \ddot{x}$
$\wedge n'_{\mathrm{tu}} = n_{\mathrm{tu}}$
$\wedge s_{\mathrm{tu}} = S_{\max}$
tu_init
tu_decA
$\neg \mathrm{net}_{\mathrm{DECB}}\text{\_compl}$
$\wedge t < n_{\mathrm{tu}}/$
$\dot{x}' = \max(\dot{x} - dt \cdot \ddot{x}, 4)$
$\wedge \ddot{x}' = \ddot{x}$
$\wedge n'_{\mathrm{tu}} = n_{\mathrm{tu}}$
$\wedge s_{\mathrm{tu}} = n_{\mathrm{tu}} - t$
$\mathrm{net}_{\mathrm{DECB}}\text{\_compl}/$
$\wedge \dot{x}' = \dot{x}$
$\wedge \ddot{x}' = 4$
$\wedge 0 = \dot{x} - (n'_{\mathrm{tu}} - t) \cdot 4$
$\wedge s_{\mathrm{tu}} = 0$
$\mathrm{net}_{\mathrm{DECB}}\text{\_compl}/$
$\wedge \dot{x}' = \dot{x}$
$\wedge \ddot{x}' = 4$
$\wedge 0 = \dot{x} - (n'_{\mathrm{tu}} - t) \cdot 4$
$\wedge s_{\mathrm{tu}} = 0$
$\neg \mathrm{net}_{\mathrm{DECB}}\text{\_compl}$
$\wedge t \geq n_{\mathrm{tu}}/$
$\dot{x}' = \dot{x}$
$\wedge \ddot{x}' = 0$
$\wedge n'_{\mathrm{tu}} = T_{\max}$
$\wedge s_{\mathrm{tu}} = 0$
$t < n_{\mathrm{tu}}/$
$\dot{x}' = \max(\dot{x} - dt \cdot \ddot{x}, 0)$
$\wedge \ddot{x}' = \ddot{x}$
$\wedge n'_{\mathrm{tu}} = n_{\mathrm{tu}}$
$\wedge s_{\mathrm{tu}} = n_{\mathrm{tu}} - t$
tu_decB
tu_slowspeed
$\wedge \neg \mathrm{net}_{\mathrm{DECB}}\text{\_compl}/$
$\dot{x}' = \dot{x}$
$\wedge \ddot{x}' = \ddot{x}$
$\wedge n'_{\mathrm{tu}} = T_{\max}$
$\wedge s_{\mathrm{tu}} = S_{\max}$
$\mathrm{net}_{\mathrm{DECB}}\text{\_compl}/$
$\wedge \dot{x}' = \dot{x} \wedge \ddot{x}' = 4$
$\wedge 0 = \dot{x} - (n'_{\mathrm{tu}} - t) \cdot 4 \wedge s_{\mathrm{tu}} = 0$
$t \geq n_{\mathrm{tu}}/$
$\wedge \dot{x}' = 0$
$\wedge \ddot{x}' = 0$
$\wedge n'_{\mathrm{tu}} = T_{\max}$
$\wedge s_{\mathrm{tu}} = 0$
true/
$\dot{x}' = 0$
$\wedge \ddot{x}' = 0$
$\wedge n'_{\mathrm{tu}} = T_{\max}$
$\wedge s_{\mathrm{tu}} = 0$
tu_stop
transportation unit
true/
$dt = \min(\lceil s_{\mathrm{obj}} \rceil, \lceil s_{\mathrm{tu}} \rceil, s_{\mathrm{net_{sA}}}, s_{\mathrm{net_{sB}}}, s_{\mathrm{net_{DECA}}},$
$s_{\mathrm{net_{DECB}}}, s_{\mathrm{io\_in}}, s_{\mathrm{io\_out}}, s_{\mathrm{plc}})$
$\wedge t' = t + dt$
$t = 0$
time progress scheduler

Figure 8.1: First part of the formal model of the NAS. The upper two automata represent the behavior of the object that is transported and of the transportation unit which controls the speed of the conveyer belt. The lower automaton selects the duration of the current time step dt and performs time progress. (This figure is a slight modification of Figure 7 from [TEF11].)

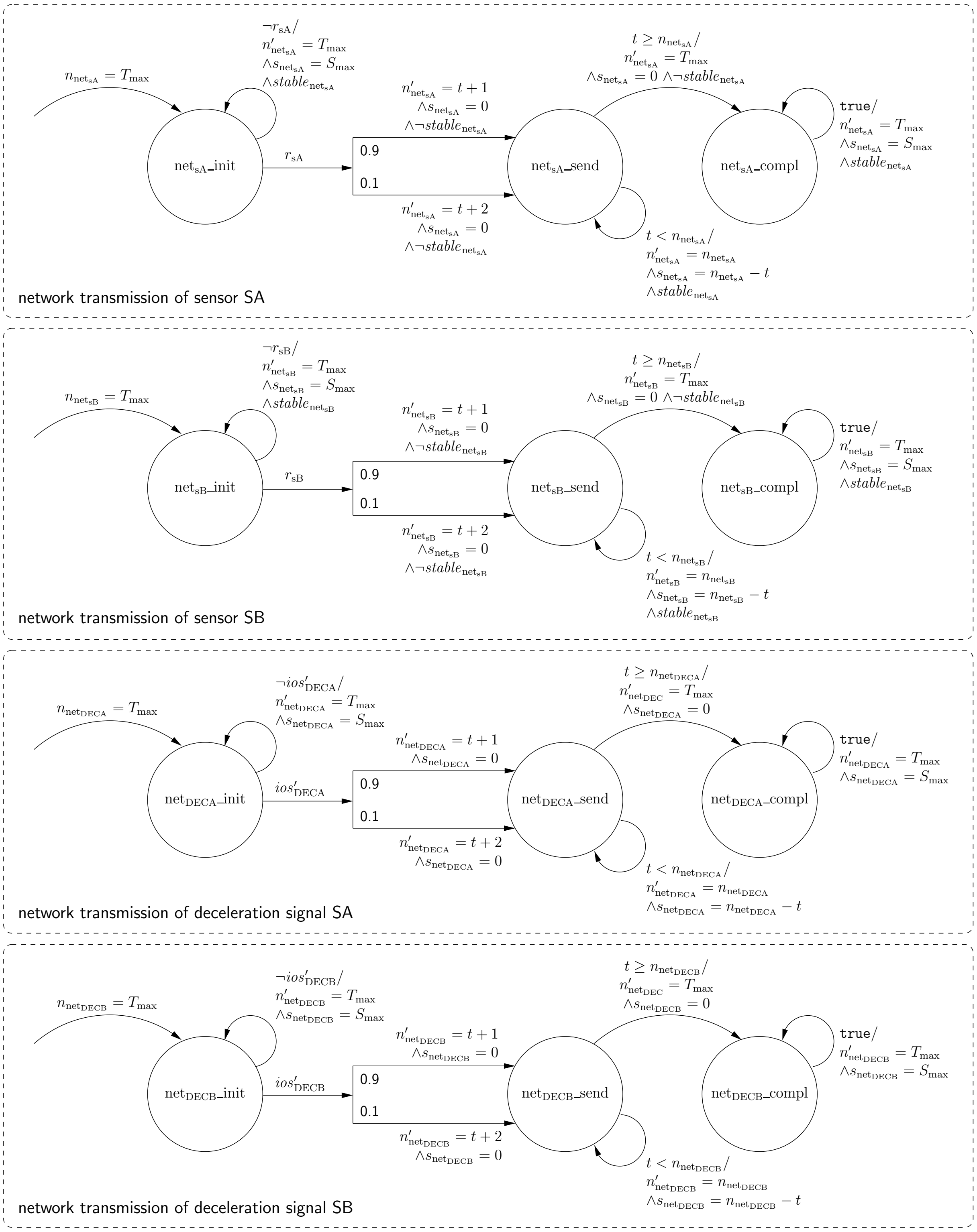

$n_{\mathrm{net_{sA}}} = T_{\max}$
$\neg r_{\mathrm{sA}}/$ $n'_{\mathrm{net_{sA}}} = T_{\max}$ $\wedge s_{\mathrm{net_{sA}}} = S_{\max}$ $\wedge stable_{\mathrm{net_{sA}}}$
net$_{\mathrm{sA}}$_init
r_{sA}
0.9
0.1
$n'_{\mathrm{net_{sA}}} = t+1$ $\wedge s_{\mathrm{net_{sA}}} = 0$ $\wedge\neg stable_{\mathrm{net_{sA}}}$
$n'_{\mathrm{net_{sA}}} = t+2$ $\wedge s_{\mathrm{net_{sA}}} = 0$ $\wedge\neg stable_{\mathrm{net_{sA}}}$
net$_{\mathrm{sA}}$_send
$t \geq n_{\mathrm{net_{sA}}}/$ $n'_{\mathrm{net_{sA}}} = T_{\max}$ $\wedge s_{\mathrm{net_{sA}}} = 0$ $\wedge\neg stable_{\mathrm{net_{sA}}}$
$t < n_{\mathrm{net_{sA}}}/$ $n'_{\mathrm{net_{sA}}} = n_{\mathrm{net_{sA}}}$ $\wedge s_{\mathrm{net_{sA}}} = n_{\mathrm{net_{sA}}} - t$ $\wedge stable_{\mathrm{net_{sA}}}$
net$_{\mathrm{sA}}$_compl
true/ $n'_{\mathrm{net_{sA}}} = T_{\max}$ $\wedge s_{\mathrm{net_{sA}}} = S_{\max}$ $\wedge stable_{\mathrm{net_{sA}}}$
network transmission of sensor SA
$n_{\mathrm{net_{sB}}} = T_{\max}$
$\neg r_{\mathrm{sB}}/$ $n'_{\mathrm{net_{sB}}} = T_{\max}$ $\wedge s_{\mathrm{net_{sB}}} = S_{\max}$ $\wedge stable_{\mathrm{net_{sB}}}$
net$_{\mathrm{sB}}$_init
r_{sB}
0.9
0.1
$n'_{\mathrm{net_{sB}}} = t+1$ $\wedge s_{\mathrm{net_{sB}}} = 0$ $\wedge\neg stable_{\mathrm{net_{sB}}}$
$n'_{\mathrm{net_{sB}}} = t+2$ $\wedge s_{\mathrm{net_{sB}}} = 0$ $\wedge\neg stable_{\mathrm{net_{sB}}}$
net$_{\mathrm{sB}}$_send
$t \geq n_{\mathrm{net_{sB}}}/$ $n'_{\mathrm{net_{sB}}} = T_{\max}$ $\wedge s_{\mathrm{net_{sB}}} = 0$ $\wedge\neg stable_{\mathrm{net_{sB}}}$
$t < n_{\mathrm{net_{sB}}}/$ $n'_{\mathrm{net_{sB}}} = n_{\mathrm{net_{sB}}}$ $\wedge s_{\mathrm{net_{sB}}} = n_{\mathrm{net_{sB}}} - t$ $\wedge stable_{\mathrm{net_{sB}}}$
net$_{\mathrm{sB}}$_compl
true/ $n'_{\mathrm{net_{sB}}} = T_{\max}$ $\wedge s_{\mathrm{net_{sB}}} = S_{\max}$ $\wedge stable_{\mathrm{net_{sB}}}$
network transmission of sensor SB
$n_{\mathrm{net_{DECA}}} = T_{\max}$
$\neg ios'_{\mathrm{DECA}}/$ $n'_{\mathrm{net_{DECA}}} = T_{\max}$ $\wedge s_{\mathrm{net_{DECA}}} = S_{\max}$
net$_{\mathrm{DECA}}$_init
ios'_{DECA}
0.9
0.1
$n'_{\mathrm{net_{DECA}}} = t+1$ $\wedge s_{\mathrm{net_{DECA}}} = 0$
$n'_{\mathrm{net_{DECA}}} = t+2$ $\wedge s_{\mathrm{net_{DECA}}} = 0$
net$_{\mathrm{DECA}}$_send
$t \geq n_{\mathrm{net_{DECA}}}/$ $n'_{\mathrm{net_{DEC}}} = T_{\max}$ $\wedge s_{\mathrm{net_{DECA}}} = 0$
$t < n_{\mathrm{net_{DECA}}}/$ $n'_{\mathrm{net_{DECA}}} = n_{\mathrm{net_{DECA}}}$ $\wedge s_{\mathrm{net_{DECA}}} = n_{\mathrm{net_{DECA}}} - t$
net$_{\mathrm{DECA}}$_compl
true/ $n'_{\mathrm{net_{DECA}}} = T_{\max}$ $\wedge s_{\mathrm{net_{DECA}}} = S_{\max}$
network transmission of deceleration signal SA
$n_{\mathrm{net_{DECB}}} = T_{\max}$
$\neg ios'_{\mathrm{DECB}}/$ $n'_{\mathrm{net_{DECB}}} = T_{\max}$ $\wedge s_{\mathrm{net_{DECB}}} = S_{\max}$
net$_{\mathrm{DECB}}$_init
ios'_{DECB}
0.9
0.1
$n'_{\mathrm{net_{DECB}}} = t+1$ $\wedge s_{\mathrm{net_{DECB}}} = 0$
$n'_{\mathrm{net_{DECB}}} = t+2$ $\wedge s_{\mathrm{net_{DECB}}} = 0$
net$_{\mathrm{DECB}}$_send
$t \geq n_{\mathrm{net_{DECB}}}/$ $n'_{\mathrm{net_{DEC}}} = T_{\max}$ $\wedge s_{\mathrm{net_{DECB}}} = 0$
$t < n_{\mathrm{net_{DECB}}}/$ $n'_{\mathrm{net_{DECB}}} = n_{\mathrm{net_{DECB}}}$ $\wedge s_{\mathrm{net_{DECB}}} = n_{\mathrm{net_{DECB}}} - t$
net$_{\mathrm{DECB}}$_compl
true/ $n'_{\mathrm{net_{DECB}}} = T_{\max}$ $\wedge s_{\mathrm{net_{DECB}}} = S_{\max}$
network transmission of deceleration signal SB

Figure 8.2: Second part of the formal model of the NAS. The automata represent the network transmissions of sensors SA and SB as well as of the corresponding deceleration signals from the PLC-IO card to the transportation unit. (This figure is a slight modification of Figure 8 from [TEF11].)

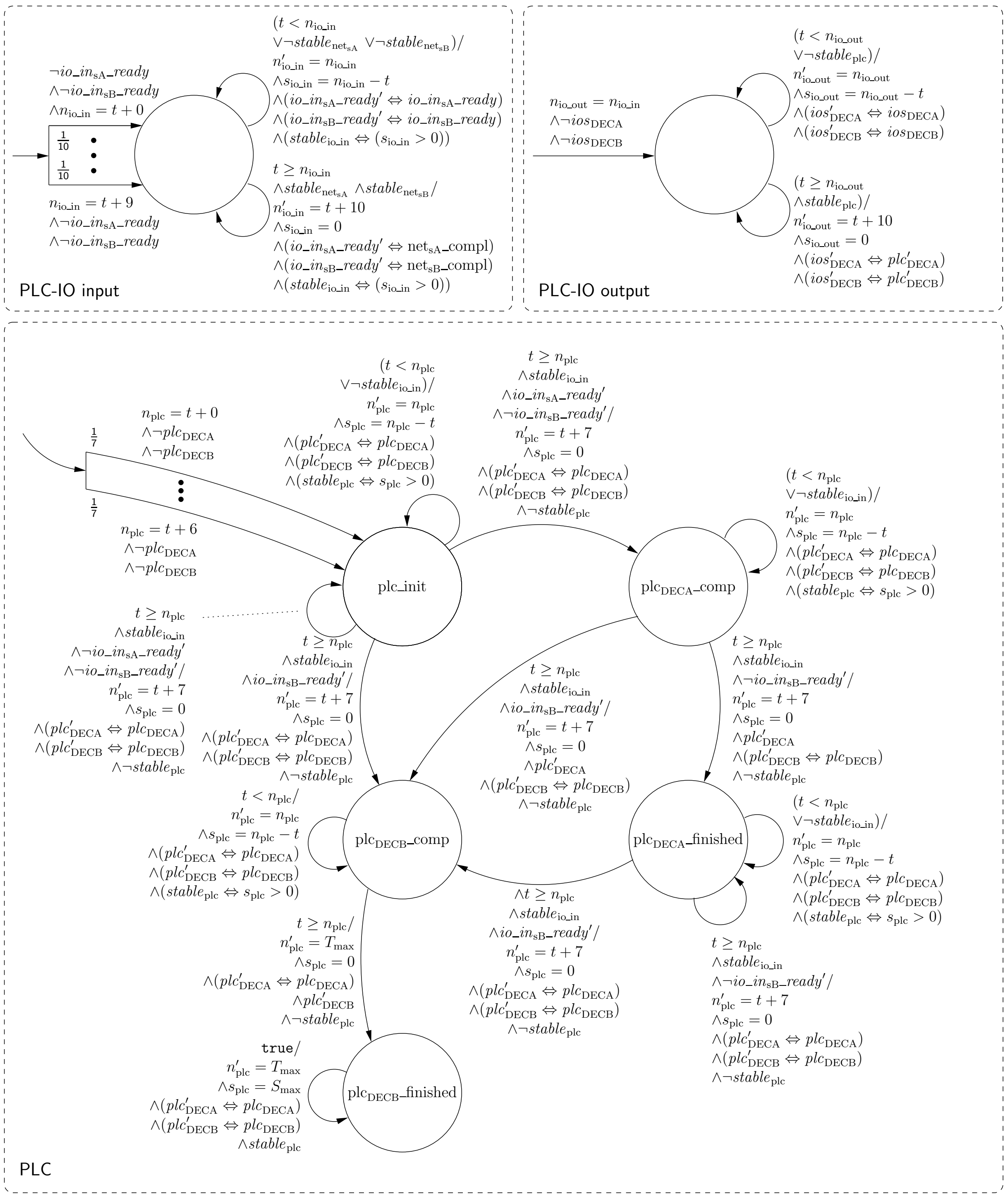

$(t < n_{\text{io\_in}} \vee \neg stable_{\text{net}_{\text{sA}}} \vee \neg stable_{\text{net}_{\text{sB}}})/ n'_{\text{io\_in}} = n_{\text{io\_in}} \wedge s_{\text{io\_in}} = n_{\text{io\_in}} - t \wedge (io\_in_{\text{sA}}\_ready' \Leftrightarrow io\_in_{\text{sA}}\_ready) \wedge (io\_in_{\text{sB}}\_ready' \Leftrightarrow io\_in_{\text{sB}}\_ready) \wedge (stable_{\text{io\_in}} \Leftrightarrow (s_{\text{io\_in}} > 0))$
$\neg io\_in_{\text{sA}}\_ready \wedge \neg io\_in_{\text{sB}}\_ready \wedge n_{\text{io\_in}} = t + 0$
$\frac{1}{10}$
$\frac{1}{10}$
$n_{\text{io\_in}} = t + 9 \wedge \neg io\_in_{\text{sA}}\_ready \wedge \neg io\_in_{\text{sB}}\_ready$
$t \geq n_{\text{io\_in}} \wedge stable_{\text{net}_{\text{sA}}} \wedge stable_{\text{net}_{\text{sB}}}/ n'_{\text{io\_in}} = t + 10 \wedge s_{\text{io\_in}} = 0 \wedge (io\_in_{\text{sA}}\_ready' \Leftrightarrow \text{net}_{\text{sA}}\text{\_compl}) \wedge (io\_in_{\text{sB}}\_ready' \Leftrightarrow \text{net}_{\text{sB}}\text{\_compl}) \wedge (stable_{\text{io\_in}} \Leftrightarrow (s_{\text{io\_in}} > 0))$
PLC-IO input
$n_{\text{io\_out}} = n_{\text{io\_in}} \wedge \neg ios_{\text{DECA}} \wedge \neg ios_{\text{DECB}}$
$(t < n_{\text{io\_out}} \vee \neg stable_{\text{plc}})/ n'_{\text{io\_out}} = n_{\text{io\_out}} \wedge s_{\text{io\_out}} = n_{\text{io\_out}} - t \wedge (ios'_{\text{DECA}} \Leftrightarrow ios_{\text{DECA}}) \wedge (ios'_{\text{DECB}} \Leftrightarrow ios_{\text{DECB}})$
$(t \geq n_{\text{io\_out}} \wedge stable_{\text{plc}})/ n'_{\text{io\_out}} = t + 10 \wedge s_{\text{io\_out}} = 0 \wedge (ios'_{\text{DECA}} \Leftrightarrow plc'_{\text{DECA}}) \wedge (ios'_{\text{DECB}} \Leftrightarrow plc'_{\text{DECB}})$
PLC-IO output
$\frac{1}{7}$
$\frac{1}{7}$
$n_{\text{plc}} = t + 0 \wedge \neg plc_{\text{DECA}} \wedge \neg plc_{\text{DECB}}$
$n_{\text{plc}} = t + 6 \wedge \neg plc_{\text{DECA}} \wedge \neg plc_{\text{DECB}}$
$(t < n_{\text{plc}} \vee \neg stable_{\text{io\_in}})/ n'_{\text{plc}} = n_{\text{plc}} \wedge s_{\text{plc}} = n_{\text{plc}} - t \wedge (plc'_{\text{DECA}} \Leftrightarrow plc_{\text{DECA}}) \wedge (plc'_{\text{DECB}} \Leftrightarrow plc_{\text{DECB}}) \wedge (stable_{\text{plc}} \Leftrightarrow s_{\text{plc}} > 0)$
$t \geq n_{\text{plc}} \wedge stable_{\text{io\_in}} \wedge io\_in_{\text{sA}}\_ready' \wedge \neg io\_in_{\text{sB}}\_ready'/ n'_{\text{plc}} = t + 7 \wedge s_{\text{plc}} = 0 \wedge (plc'_{\text{DECA}} \Leftrightarrow plc_{\text{DECA}}) \wedge (plc'_{\text{DECB}} \Leftrightarrow plc_{\text{DECB}}) \wedge \neg stable_{\text{plc}}$
$(t < n_{\text{plc}} \vee \neg stable_{\text{io\_in}})/ n'_{\text{plc}} = n_{\text{plc}} \wedge s_{\text{plc}} = n_{\text{plc}} - t \wedge (plc'_{\text{DECA}} \Leftrightarrow plc_{\text{DECA}}) \wedge (plc'_{\text{DECB}} \Leftrightarrow plc_{\text{DECB}}) \wedge (stable_{\text{plc}} \Leftrightarrow s_{\text{plc}} > 0)$
plc_init
plc$_{\text{DECA}}$_comp
$t \geq n_{\text{plc}} \wedge stable_{\text{io\_in}} \wedge \neg io\_in_{\text{sA}}\_ready' \wedge \neg io\_in_{\text{sB}}\_ready'/ n'_{\text{plc}} = t + 7 \wedge s_{\text{plc}} = 0 \wedge (plc'_{\text{DECA}} \Leftrightarrow plc_{\text{DECA}}) \wedge (plc'_{\text{DECB}} \Leftrightarrow plc_{\text{DECB}}) \wedge \neg stable_{\text{plc}}$
$t \geq n_{\text{plc}} \wedge stable_{\text{io\_in}} \wedge io\_in_{\text{sB}}\_ready'/ n'_{\text{plc}} = t + 7 \wedge s_{\text{plc}} = 0 \wedge (plc'_{\text{DECA}} \Leftrightarrow plc_{\text{DECA}}) \wedge (plc'_{\text{DECB}} \Leftrightarrow plc_{\text{DECB}}) \wedge \neg stable_{\text{plc}}$
$t \geq n_{\text{plc}} \wedge stable_{\text{io\_in}} \wedge io\_in_{\text{sB}}\_ready'/ n'_{\text{plc}} = t + 7 \wedge s_{\text{plc}} = 0 \wedge plc'_{\text{DECA}} \wedge (plc'_{\text{DECB}} \Leftrightarrow plc_{\text{DECB}}) \wedge \neg stable_{\text{plc}}$
$t \geq n_{\text{plc}} \wedge stable_{\text{io\_in}} \wedge \neg io\_in_{\text{sB}}\_ready'/ n'_{\text{plc}} = t + 7 \wedge s_{\text{plc}} = 0 \wedge plc'_{\text{DECA}} \wedge (plc'_{\text{DECB}} \Leftrightarrow plc_{\text{DECB}}) \wedge \neg stable_{\text{plc}}$
$t < n_{\text{plc}}/ n'_{\text{plc}} = n_{\text{plc}} \wedge s_{\text{plc}} = n_{\text{plc}} - t \wedge (plc'_{\text{DECA}} \Leftrightarrow plc_{\text{DECA}}) \wedge (plc'_{\text{DECB}} \Leftrightarrow plc_{\text{DECB}}) \wedge (stable_{\text{plc}} \Leftrightarrow s_{\text{plc}} > 0)$
plc$_{\text{DECB}}$_comp
plc$_{\text{DECA}}$_finished
$(t < n_{\text{plc}} \vee \neg stable_{\text{io\_in}})/ n'_{\text{plc}} = n_{\text{plc}} \wedge s_{\text{plc}} = n_{\text{plc}} - t \wedge (plc'_{\text{DECA}} \Leftrightarrow plc_{\text{DECA}}) \wedge (plc'_{\text{DECB}} \Leftrightarrow plc_{\text{DECB}}) \wedge (stable_{\text{plc}} \Leftrightarrow s_{\text{plc}} > 0)$
$\wedge t \geq n_{\text{plc}} \wedge stable_{\text{io\_in}} \wedge io\_in_{\text{sB}}\_ready'/ n'_{\text{plc}} = t + 7 \wedge s_{\text{plc}} = 0 \wedge (plc'_{\text{DECA}} \Leftrightarrow plc_{\text{DECA}}) \wedge (plc'_{\text{DECB}} \Leftrightarrow plc_{\text{DECB}}) \wedge \neg stable_{\text{plc}}$
$t \geq n_{\text{plc}} \wedge stable_{\text{io\_in}} \wedge \neg io\_in_{\text{sB}}\_ready'/ n'_{\text{plc}} = t + 7 \wedge s_{\text{plc}} = 0 \wedge (plc'_{\text{DECA}} \Leftrightarrow plc_{\text{DECA}}) \wedge (plc'_{\text{DECB}} \Leftrightarrow plc_{\text{DECB}}) \wedge \neg stable_{\text{plc}}$
$t \geq n_{\text{plc}}/ n'_{\text{plc}} = T_{\max} \wedge s_{\text{plc}} = 0 \wedge (plc'_{\text{DECA}} \Leftrightarrow plc_{\text{DECA}}) \wedge plc'_{\text{DECB}} \wedge \neg stable_{\text{plc}}$
$\texttt{true}/ n'_{\text{plc}} = T_{\max} \wedge s_{\text{plc}} = S_{\max} \wedge (plc'_{\text{DECA}} \Leftrightarrow plc_{\text{DECA}}) \wedge (plc'_{\text{DECB}} \Leftrightarrow plc_{\text{DECB}}) \wedge stable_{\text{plc}}$
plc$_{\text{DECB}}$_finished
PLC

Figure 8.3: Third part of the formal model of the NAS. The upper two automata represent the input and output parts of the PLC-IO card. The lower automaton models the behavior of the PLC which decides when to send deceleration signals depending on the received messages from the sensors. (This figure is a slight modification of Figure 9 from [TEF11].)

r_{sA} and r_{sB} to `true`, respectively. These variables are used for the communication with the automata that model the network. The initial location of automaton object is obj_preA and the initial position x is distributed uniformly over values 999 to 976, the latter fact is modeled by 24 incoming probabilistic transition alternatives. It is important to remark that the concept of a system of concurrent PHAs, as formalized in Definition 3.1, does *not* support such distribution of the initial state. In fact, there must be a *unique* initial system state. However, this feature does not enhance expressiveness and is covered by the standard model as follows: an additional location, say obj_init, is added to object which serves as the new initial location, while variables x and n_{obj} are fixed to some initial values. The actual probabilistic choice of the initial object position is then modeled by means of connecting locations obj_init and obj_preA with the corresponding 24 probabilistic transition alternatives, while the transition guard must be always satisfied, i.e. the guard predicate is `true`. To ensure that the latter action is performed instantaneously, the maximum possible step duration s_{obj} must be set to 0 in each of the above transition alternatives. That is to say, the feature of initial probabilistic transition alternatives is just for the sake of brevity. We remark that this feature is furthermore used in the automata PLC-IO input and PLC from Figure 8.3. The automaton object switches to location obj_betwAB in case the global time t becomes greater than or equal to the time of the next event n_{obj}, i.e. sensor SA was reached. A similar transition to the final location obj_postB is taken if sensor SB was passed.

The automaton transportation unit from Figure 8.1 is responsible for computing the speed $\dot{x}$. It furthermore sets the deceleration $\ddot{x}$ by scanning the current locations of automata network transmission of deceleration signal SA and network transmission of deceleration signal SB. If the first of the latter automata has reached its location net_{DECA}_compl and if the second automaton has not yet entered net_{DECB}_compl then transportation unit switches to location tu_decA thereby setting the deceleration to $2\,\text{lu}/\text{ts}^2$, i.e. the primed variable $\ddot{x}'$ is set to 2. Whenever network transmission of deceleration signal SB has reached its locations net_{DECB}_compl, automaton transportation unit performs a transition step to tu_decB and updates the deceleration to $4\,\text{lu}/\text{ts}^2$. In case the signal to decelerate with $4\,\text{lu}/\text{ts}^2$ has not yet been transmitted by the network but the slow speed of $4\,\text{lu}/\text{ts}$ is reached then the automaton enters location tu_slowspeed and stops braking temporarily by setting the deceleration to zero. When switching to location tu_decB, in which braking is performed with $4\,\text{lu}/\text{ts}^2$, the next event n'_{tu} is determined as the time point where the speed will become zero, i.e. the workpiece will be stopped. This can be computed by the equation $0 = \dot{x} - (n'_{\text{tu}} - t) \cdot 4$ for a fixed speed $\dot{x}$ at time point t. Once the object comes to a standstill, which is detected if the global time t is greater than or equal to the time n_{tu} of the standstill, the automaton transportation unit enters its final location tu_stop.

The network of the system is modeled by the four automata of Figure 8.2. The components network transmission of sensor SA and network transmission of sensor SB are responsible for the transmission of the sensor signals to the IO card of the programmable logic controller. To this end, they first sample the signals r_{sA} and r_{sB} that indicate reaching sensors SA and SB, respectively, by rising edges. If a rising edge occurs then the corresponding automaton switches to its sending location net_{sA}_send or net_{sB}_send. As mentioned in Section 3.1, the network routing time is determined stochastically, needing 1 ts for delivery with probability 0.9 and 2 ts with probability 0.1. This fact is modeled by two probabilistic transition alternatives which set the time of the next event, i.e. $n'_{\text{net}_{\text{sA}}}$ or $n'_{\text{net}_{\text{sB}}}$, to the current time t incremented by the corresponding transmission time 1 or 2.

If these next events $n_{\mathrm{net_{sA}}}$ and $n_{\mathrm{net_{sB}}}$ are reached then automata network transmission of sensor SA and network transmission of sensor SB go to their final locations $\mathrm{net_{sA}\_compl}$ and $\mathrm{net_{sB}\_compl}$, respectively. We remark that the Boolean variables $\mathit{stable}_{\mathrm{net_{sA}}}$ and $\mathit{stable}_{\mathrm{net_{sB}}}$ encode location switches whenever they are set to `false`. Such transitions do not consume any time, being indicated by setting the maximum possible step duration $s_{\mathrm{net_{sA}}}$ or $s_{\mathrm{net_{sB}}}$ to 0. Several location switches may occur at the same physical time t. In order to not overlook any transmitted signal of the network, the model of the PLC-IO card samples its inputs at time t not before both above network automata have performed potential location switches at time t, i.e. not before both $\mathit{stable}_{\mathrm{net_{sA}}}$ and $\mathit{stable}_{\mathrm{net_{sB}}}$ are `true`. The automata network transmission of deceleration signal SA and network transmission of deceleration signal SB are responsible for forwarding the new decelerations to transportation unit, and work very similar to the previous automata. Reaching a final location $\mathrm{net_{DECA}\_compl}$ or $\mathrm{net_{DECB}\_compl}$ immediately triggers a location switch in transportation unit as mentioned above.

The model of the IO card of the PLC is divided into two components, where component PLC-IO input transmits the signals from the network to the PLC and component PLC-IO output delivers the signals for the new deceleration values from the PLC to the network, confer Figure 8.3. Each of both automata consists of only one location and two transitions, one of which is taken every 10 time steps when the corresponding signals are sampled. Sampling points are detected by comparing the current time t with the time of the next event, i.e. with $n_{\mathrm{io\_in}}$ or $n_{\mathrm{io\_out}}$. The meaning of the Boolean variables $\mathit{stable}_{\mathrm{net_{sA}}}$ and $\mathit{stable}_{\mathrm{net_{sB}}}$ was already discussed above, while the same idea holds for variable $\mathit{stable}_{\mathrm{plc}}$.

Finally, the automaton PLC from Figure 8.3 computes the signals for the decelerations of the object depending on the signals from sensors SA and SB. Initially, PLC resides in location plc_init and polls every 7 time steps for new inputs. A new input is detected by accessing the primed Boolean variables $\mathit{io\_in}_{\mathrm{sA}}\mathit{\_ready}'$ and $\mathit{io\_in}_{\mathrm{sB}}\mathit{\_ready}'$ that are set to `true` by PLC-IO input if signals from SA and SB were sampled, respectively. In case only $\mathit{io\_in}_{\mathrm{sA}}\mathit{\_ready}'$ is `true`, the automaton PLC enters location $\mathrm{plc_{DECA}\_comp}$ for computing the deceleration with respect to sensor SA. Completion of this computational process after 7 time steps is indicated by leaving the location and by setting the primed Boolean variable $\mathit{plc}'_{\mathrm{DECA}}$ to `true`. If the primed variable $\mathit{io\_in}_{\mathrm{sB}}\mathit{\_ready}'$ is not yet `true` then location $\mathrm{plc_{DECA}\_finished}$ is visited in order to wait for this signal. Whenever automaton PLC is in one of the three locations plc_init, $\mathrm{plc_{DECA}\_comp}$, and $\mathrm{plc_{DECA}\_finished}$ and if the primed variable $\mathit{io\_in}_{\mathrm{sB}}\mathit{\_ready}'$ is `true` then PLC goes directly to location $\mathrm{plc_{DECB}\_comp}$ in order to compute the deceleration concerning sensor SB. That is, in case both signals from SA and SB arrive at the same time then the signal from SB is prioritized. After having calculated the deceleration corresponding to sensor SB, the automaton PLC enters its final location $\mathrm{plc_{DECB}\_finished}$.

Two important parameters of the overall system are the initial phase shifts of the cycles of the PLC-IO card and of the PLC. A pragmatic, yet idealized, way to handle these phase shifts, for instance, is to synchronize the initial cycles of both the PLC-IO and PLC. These phase shifts however have a fundamental impact on the overall system behavior, as we see in Section 8.2. To cover each possible situation of the initial phase shifts with respect to a minimum sampling interval of 1 time step, the time points of the first cycles of the PLC-IO card and of the PLC are uniformly distributed over values 0 to 9 and over 0 to 6, respectively. That is, there are 10 possible initial cycle times for the PLC-IO card and 7 for the PLC, which is modeled by 10 and by 7 initial probabilistic transition alternatives in automata PLC-IO input and PLC, respectively, confer Figure 8.3.

Sample system run of the model. Exemplifying the executable behavior of the formal NAS model, a sample system run is illustrated in Figure 8.4. We remark that this system run was generated by the iSAT tool which was called on an SMT encoding of a non-probabilistic version of the NAS model, i.e. where probabilistic choices were replaced by non-deterministic ones.

At first, all automata reside in their initial locations while the initial object position x is determined stochastically to be 976. The probabilistic choices of the time points of the first cycles yield 9 for the PLC-IO card and 6 for the PLC. After 12 time steps, the object reaches sensor SA causing automaton object to enter location obj_betwAB. During this discrete state change, the Boolean variable r_{sA} is set to `true` which triggers a synchronization with automaton network transmission of sensor SA. As a consequence, the latter automaton proceeds to location $\mathrm{net}_{\mathrm{sA}}$_send where the probabilistic selection of the network routing time yields 1 ts. That is, network transmission of sensor SA leaves the sending location $\mathrm{net}_{\mathrm{sA}}$_send after 1 ts and enters location $\mathrm{net}_{\mathrm{sA}}$_compl indicating that the signal was successfully transmitted to the PLC-IO card at time point 13.

The next cycle time of PLC-IO input is at time point 19 at which the signal from sensor SA is provided for automaton PLC, the latter processing this signal in its cycle from time point 20 to 27 when residing in location $\mathrm{plc}_{\mathrm{DECA}}$_comp. Automaton PLC-IO output then starts to send the request for the new deceleration of $2\,\mathrm{lu/ts}^2$ at time point 29 over the network. The network routing time for sending this packet, i.e. the duration automaton network transmission of deceleration signal SA stays in location $\mathrm{net}_{\mathrm{DECA}}$_send, is determined probabilistically to be 2 ts. This implies that the latter automaton reaches its final location $\mathrm{net}_{\mathrm{DECA}}$_compl at time point 31. At the same time, transportation unit enters location tu_decA and sets the value of the deceleration $\ddot{x}$ to $2\,\mathrm{lu/ts}^2$.

The speed $\dot{x}$ of the object is then decreasing accordingly until the slow speed of 4 lu/ts is reached at time $t = 41$. In order to keep the speed constant, transportation unit enters location tu_slowspeed and updates the deceleration to 0. The automaton remains in this location until the signal triggered by sensor SB is processed and results in setting the deceleration $\ddot{x}$ to the respective value of $4\,\mathrm{lu/ts}^2$, which happens at time point 51. As a result, the object comes to a standstill at time point 52 with a final position of 50 lu.

8.2 Analysis of the NAS

As mentioned in Section 3.1, the goal of the NAS application is to deliver the workpiece close to the drilling position. The main analysis task thus is to assess quantitatively whether the above goal can be considered as satisfied. More precisely, we are interested in answering questions like

- "*What* is the probability that the workpiece stops close to the drilling position?"

or, phrased as a decision problem,

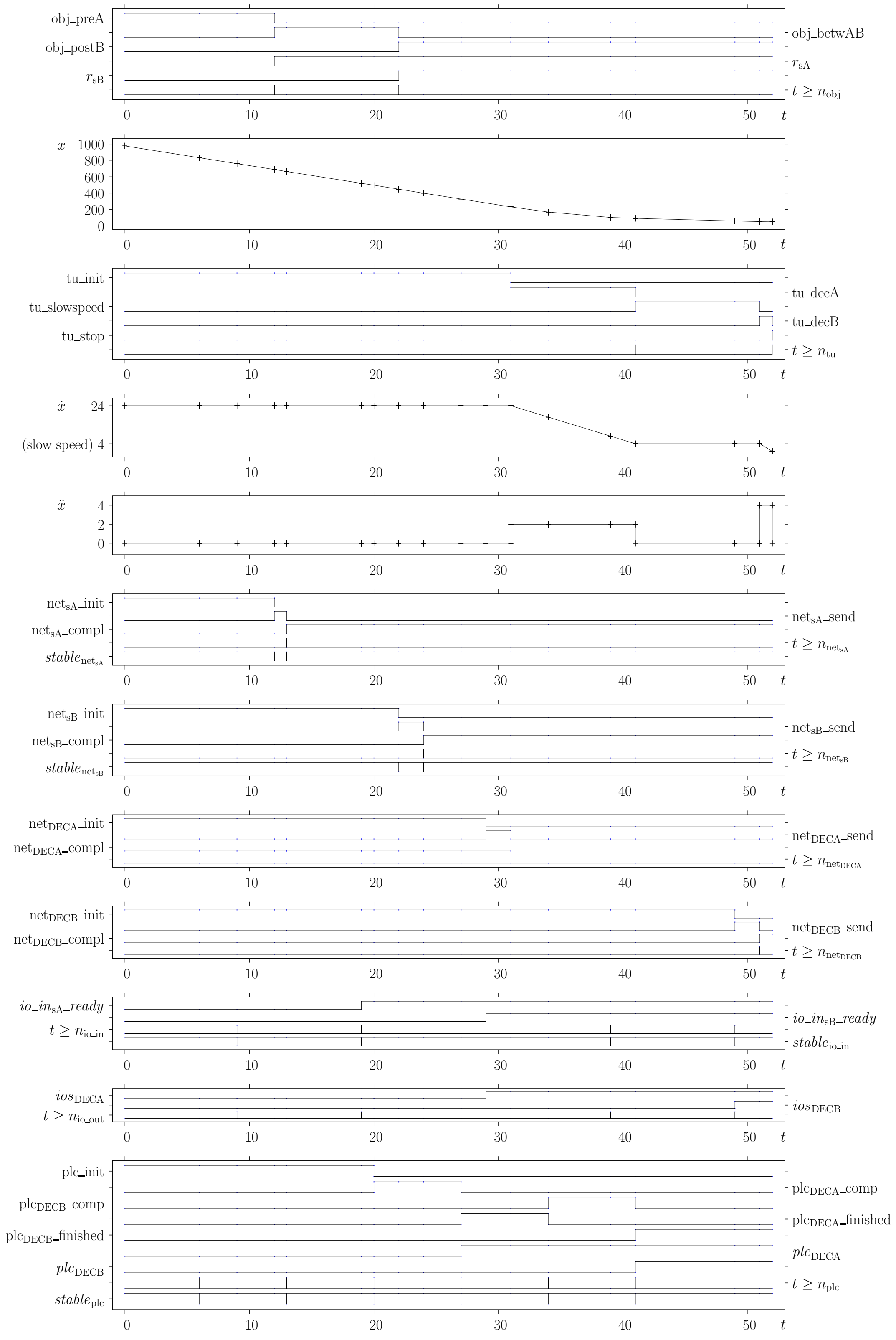


Figure 8.4: Sample system run of the NAS model. (This figure is a slight modification of Figure 10 from [TEF11].)

- "Is the probability that the workpiece stops close to the drilling position *high enough*?"

Apart from such classical reachability probabilities, it frequently is worthwhile to provide information on more expressive quantitative measures like expected values. An additional focus thus is on questions like

- "What is the *mean time to stop* of the workpiece?"

and

- "What is the *expected final position* of the workpiece?"

In the remainder of this section, we present results from the analysis of the NAS case study using the symbolic approaches described in this thesis. The corresponding experiments were conducted on a 2.4 GHz AMD Opteron machine with 128 GByte physical memory running Linux.

SiSAT encoding of the NAS model. In order to enable the automatic and symbolic analysis of the NAS application with the aid of the SSMT-based probabilistic bounded model checker SiSAT, confer Sections 6.6 and 7.5, we encoded the formal model of the NAS from Section 8.1 into the input language of SiSAT. A way of encoding a system of concurrent PHAs into SiSAT according to the SSMT reduction scheme from Section 5.3 is shown in Subsection 6.6.1, while concrete examples are presented in Sections 6.6.3 and 7.5. It is important to remark that our SiSAT encoding of the NAS exploits the idea of "disabling" quantifiers whenever they become irrelevant, as explained in Subsection 6.6.3.

We would like to emphasize that the symbolic SiSAT encoding facilitates a system description of size *linear* in the size of the NAS model, in particular, linear in the size of concurrent components. When unwinding this symbolic system description, i.e. when constructing the SSMT formulae 6.25 for increasing step depths $k \in \mathbb{N}$, confer Subsection 6.6.1, the size of these SSMT formulae grows *linearly* in the number k of transition steps. Due to above facts, the state explosion problem is alleviated, the latter arising from an explicit construction of the product automaton (with respect to the discrete state space) which in general is of size exponential in the number of concurrent subsystems. Several analysis approaches rely on such an explicit construction, confer Section 3.2.

Observe that the product automaton of the formal NAS model consists of more than 24 million discrete states since the discrete state space is spanned by 6075 locations and 12 Boolean variables, resulting in $2^{12} = 4096$ possible truth assignments, used for synchronization, while the continuous state space is given by all possible assignments to the 23 continuous variables present in the model.

Probabilistic reachability analysis of the NAS. We first consider the former two questions above concerning the probability of stopping close to the drilling position. Recall that SSMT-based probabilistic bounded reachability analysis is just able to address *step-bounded* system behavior, confer Chapter 5. That is, only *lower* bounds on the probability of stopping close to the drilling position can be provided in general. From the description of the NAS, confer Section 3.1, it is not hard to see that the workpiece finally stops under all possible probabilistic dynamics, which means that the behavior of the NAS is *bounded* in a way. This gives rise to a technique coping with the full system behavior and answering both probabilistic reachability questions above.

To achieve such complete analysis, we are first interested in the number k of transition steps of the formal NAS model such that the workpiece stops in all anchored system runs of length k. Observe that the workpiece stops in a run r if and only if r reaches location tu_stop in automaton transportation unit from Figure 8.1. Since the NAS does not comprise any non-determinism, it holds that the workpiece stops in all anchored runs of length k if and only if the (maximum) probability of reaching location tu_stop within k transition steps is 1. Let $PBMC_{\text{NAS},\mathit{Target}}(k)$ be the SSMT encoding of the NAS for step depth k, confer Section 5.3. It then follows that $Pr(PBMC_{\text{NAS,tu\_stop}}(k)) = 1$ if and only if the workpiece stops in all anchored runs of length k.

In order to determine the desired step depth k, the SiSAT tool was called to successively solve the SSMT formulae $PBMC_{\text{NAS,tu\_stop}}(k)$ for increasing step depths $k \in \mathbb{N}$ until $Pr(PBMC_{\text{NAS,tu\_stop}}(k)) = 1$ holds. The latter is true for step depth $k = 44$. SiSAT used about 10.41 hours to solve all these 45 SSMT formulae when the algorithmic enhancement of *caching solutions*, as described in Subsection 6.5.6, was enabled.

We remark that SiSAT actually returned the interval $[0.99999982, 1]$ as the probability result for step depth 44. Recall that SiSAT employs floating-point intervals for representing probabilities, both probabilities of randomized quantifiers and (intermediate) probability results, in order to cope with numerical issues, confer Subsection 6.6.2. It thus is not ensured in general that the exact probability result pr is represented by a point interval $[pr, pr]$ but rather by a small interval $[\underline{pr}, \overline{pr}]$ with $\underline{pr} \leq pr \leq \overline{pr}$.

When having exploited the optimization of *thresholding* additionally, confer Section 6.4, the overall solving time for all the above 45 SSMT formulae could be reduced by a factor of about 4.6 to about 2.23 hours. With regard to this tool feature, one would expect to set both the lower and upper thresholds to value 1 in order to decide whether $Pr(PBMC_{\text{NAS,tu\_stop}}(k)) = 1$ holds. As mentioned above, SiSAT returns small intervals enclosing the actual probability result. Taking this into account and being a bit pragmatic, we used a value slightly less than 1, namely $1 - 10^{-5} = 0.99999$, as the lower and the upper threshold value, i.e. $\theta_l = \theta_u = 0.99999$.

After having discovered that the workpiece always stops within 44 transition steps, we now focus on the former two questions by means of computing the probability of stopping within some target region. For this purpose, the *Target* predicate is given by $(L \geq x) \wedge (x \geq R)$ with rational constants L and R defining the region. For instance, for the target region of interest being specified by $L = 100$ and $R = 0$, i.e. the workpiece should halt left to and at most 1 mm away from the drilling position,[1] SiSAT solved the corresponding SSMT formula of step depth 44, i.e. $PBMC_{\text{NAS},\mathit{Target}}(44)$, within about 71 minutes returning the hit probability interval $[0.39734516, 0.39734529]$.

Whenever the exact probability is not of interest but the problem is to decide whether the exact probability is below or above some given threshold value θ, as indicated by the second analysis question above, the algorithmic enhancement of thresholding becomes applicable, the latter frequently improving performance of the SSMT proof search. For instance, when having enabled thresholding with $\theta_l = \theta_u = \theta$ for threshold values $\theta = 0.05$, $\theta = 0.1$, $\theta = 0.9$, and $\theta = 0.95$, solving time for the above problem, i.e. for the target region

[1]As defined in Section 3.1, one length unit corresponds to 0.01 mm.

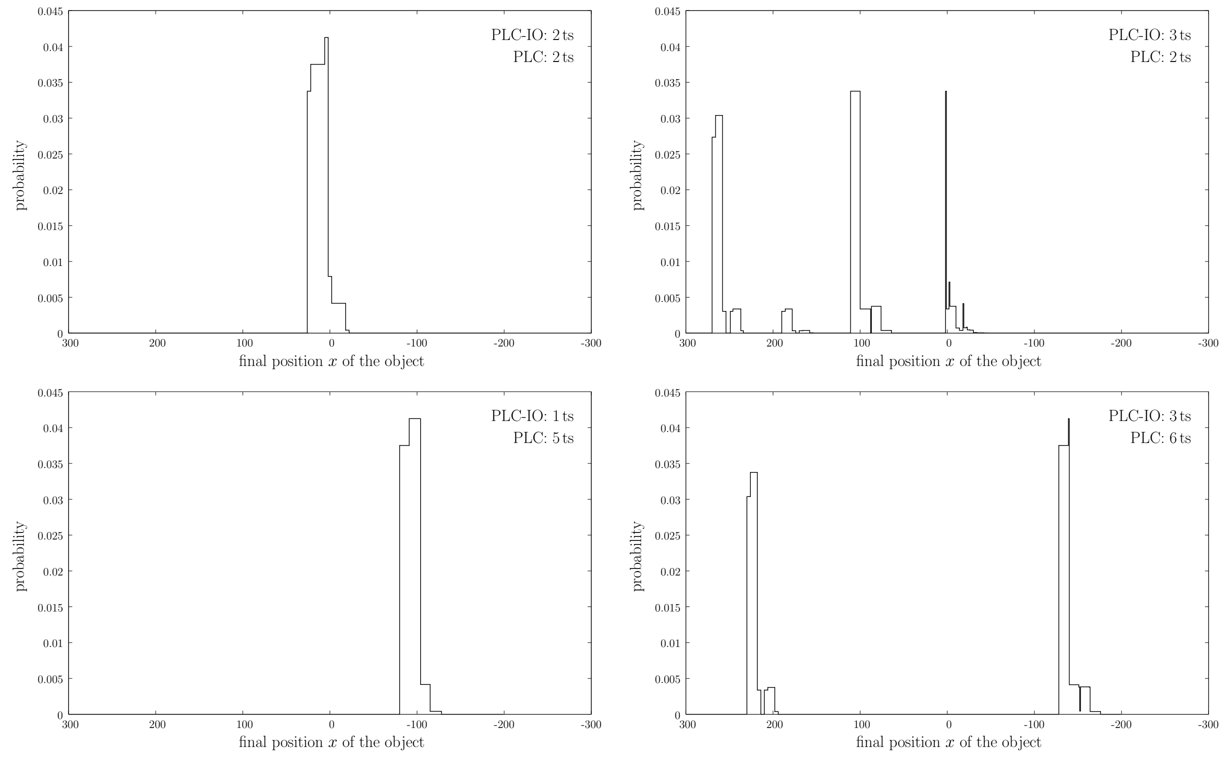


Figure 8.5: Analysis of the NAS: different distributions of the final object position x for fixed initial phase shifts of the cycles of the PLC-IO card and of the PLC. The corresponding time points of the first cycles are given in the top-right corner of each subfigure. The final object position is given on the x-axis, while the probabilities of stopping within regions $[v-1, v)$ for the integers $v = 300$ down to -299 are given on the y-axis. (Source of figure: [TEF11])

given by $L = 100$ and $R = 0$, could be reduced to about 11 minutes, about 20 minutes, about 13 minutes, and about 11 minutes, respectively.

As anticipated in Section 8.1, the initial phase shifts of the cycles of the PLC-IO card and of the PLC play an important role for the distribution of the final object position. Though this is not the intended use case of SiSAT, we were able to calculate the distribution of the final position x by solving a set of SSMT formulae $PBMC_{\text{NAS},\mathit{Target}}(44)$ where the *Target* predicates are given by $(v > x) \wedge (x \geq v - 1)$ for the integers $v = 300$ down to -299. For each of these SSMT formulae, we obtained the probability of stopping within a target region of length 1. Recall that the NAS does *not* involve any non-determinism. Consequently, the distribution of the final object position could be simply derived from the latter probability results as all these regions are pairwise disjoint and together cover the region $\mathcal{R}$ with $300 > \mathcal{R} \geq -300$ in which the object always halts. The latter statement is true since $Pr(PBMC_{\text{NAS},(x \geq 300 \vee x < -300)}(44)) = 0$, which was proven by SiSAT within about 3.22 hours.

Four possible scenarios for *fixed* initial phase shifts of the cycles of the PLC-IO card and of the PLC are depicted in Figure 8.5. To achieve valid distributions, the SSMT encodings of the latter scenarios were modified such that the randomized variables encoding the initial phase shifts take the corresponding fixed time points of the first cycles with probability 1. From the case where the first cycles of the PLC-IO card and of the PLC are

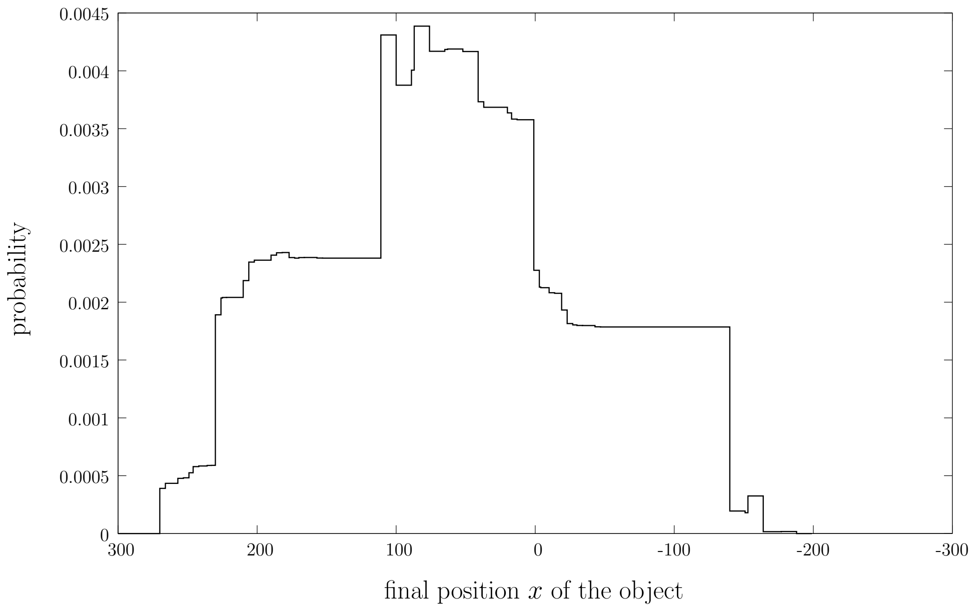


Figure 8.6: Analysis of the NAS: actual distribution of the final object position x, i.e. the initial phase shifts of the cycles of the PLC-IO card and of the PLC are distributed uniformly. The final object position is given on the x-axis, while the probabilities of stopping within regions $[v-1, v)$ for the integers $v = 300$ down to -299 are given on the y-axis. (Source of figure: [TEF11])

both at time point 2, one may wrongly conclude that the model works as desired since the workpiece always stops in a very small region around the drilling position. If the PLC-IO and the PLC phase shifts however are 3 and 6, respectively, there is a large region around position 0 in which the object does never halt.

To get an authentic picture about the distribution of the final object position, one has to incorporate the initial phase shifts in a realistic way. Since there are no semantic conditions on the PLC and its IO card, like periodic resets or other synchronizations, we realistically assume that any combination of the phase shifts may arise when the object enters. In the formal NAS model, the possible initial phase shifts of the PLC-IO card and of the PLC are therefore distributed uniformly, as explained in Section 8.1. For this more authentic NAS model, we obtained the distribution of the final object position presented in Figure 8.6.

The CPU times for computing the distributions are about 5.7 hours for the initial phase shifts 2 (PLC-IO) and 2 (PLC), about 7.5 hours for 3 (PLC-IO) and 2 (PLC), about 5 hours for 1 (PLC-IO) and 5 (PLC), and about 6.9 hours for 3 (PLC-IO) and 6 (PLC). For the distribution where the initial phase shifts are distributed uniformly, the CPU time is much higher, namely almost 32 days, due to the enormous growth of the probabilistic system behavior. However, due to the possibility of running multiple SiSAT calls in parallel, the distribution was actually calculated in roughly a day on two 16-core machines.

Expected-value analysis of the NAS. We finally respond to the latter two questions from the beginning of this section concerning the expected values of the time to stop and of the final object position, namely by means of the symbolic approach from Chapter 7.

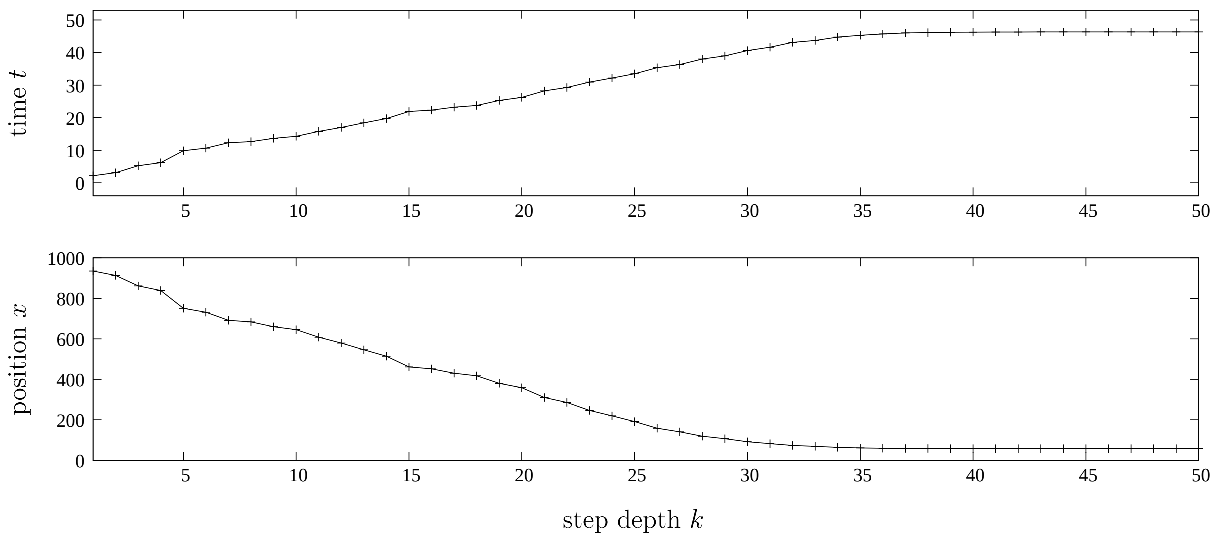


Figure 8.7: Analysis of the NAS: evolution of the expected values of the time to stop t and of the object position x over step depth k. (This figure is based on Figure 5 from [FTE10b].)

To this end, we slightly modified the SiSAT encoding of the NAS as explained in Subsection 7.3.2, where the cost of a transition step is the time step $dt \geq 0$ or the decrease of the position $x' - x$ when aiming at the mean time to stop or at the expected final object position, respectively. Observe that x' can be represented by non-primed variables and that the decrease $x' - x$ is at most 0, i.e. the object never moves backwards. Thus, the position is monotonically decreasing, while the time clearly is monotonically increasing. Furthermore note that time t is stopped whenever location tu_stop is reached since then $s_{\text{tu}} = 0$ which implies that $dt = 0$ and thus $t' = t$. Recall that negative costs are not permissible in the notion of a system of concurrent PHAs with cost function, confer Definition 7.1. The latter restriction is due to the computation of lower estimates of the infimal cost expectation for the full system behavior by means of a step-bounded variant thereof, confer Lemma 7.1, giving rise to a verification procedure for the CEMC problem from Definition 7.4.

Within the analysis of the NAS, we are just interested in the evolution of the expected object position over a bounded number of transition steps. In such scenarios, we do not need to restrict ourselves to non-negative cost but we may permit arbitrary cost functions which values can be both non-negative and negative. Observe that the SSMT algorithm from Section 7.4 is able to compute the conditional expectation of each free variable in an SSMT formula. As a consequence, the SiSAT tool can be employed to compute expected values of the object position.

The evolution of the expectations of the time to stop t and of the object position x over step depth k is depicted in Figure 8.7. Recall that the workpiece always stops within 44 transition steps. These expected values thus stabilize at step depth 44, yielding a mean time to stop of about 46.32 ts and an expected final object position of about 57.8 lu. Each of the latter expected values could be computed within about 1.45 hours by solving the corresponding SSMT formula for step depth 44.

9 Beyond Probabilistic Bounded Reachability by Means of Generalized Craig Interpolation

In the previous chapters, we presented SSMT-based approaches to probabilistic bounded state reachability as well as to expected-value analysis of concurrent discrete-time probabilistic hybrid automata. Both approaches are inherently confined to analysis of *bounded* system behavior. In this chapter, we pioneer symbolic procedures that go beyond probabilistic bounded state reachability. They are currently based on SSAT and thus restricted to probabilistic finite-state models like Markov decision processes. We remark that essential parts of this chapter were published in [TF11] and in [TF12] by the author of this thesis together with his co-author.

The development of these procedures is motivated by symbolic model checking techniques for non-probabilistic systems based on Craig interpolation. Given two formulae φ and ψ for which $\varphi \Rightarrow \psi$ is true, a *Craig interpolant* $\mathcal{I}$ for φ and ψ is a formula over variables common to φ and ψ that "lies in between" φ and ψ in the sense that $\varphi \Rightarrow \mathcal{I}$ and $\mathcal{I} \Rightarrow \psi$ [Cra57], confer Figure 9.1. In the automatic hardware and software verification communities, Craig interpolation has found widespread use in model checking algorithms, both as a means of extracting reasons for non-concretizability of a counterexample obtained on an abstraction as well as for computing a symbolic description of the reachable state set. In McMillan's approach [McM03, McM05a, McM05b] targeting at the verification of safety properties of the shape "the unsafe states are never reachable", Craig interpolants are used for describing symbolically an overapproximation of the step-bounded reachable state set. If the sequence of interpolants thus obtained stabilizes eventually, i.e. no additional state is found to be reachable, then the corresponding state-set predicate *Reach* has all reachable system states as models. Let *Unsafe* be a predicate that encodes the unsafe states. Then, the above safety property is verified whenever the formula *Reach* $\wedge$ *Unsafe* is unsatisfiable which means that the set of all reachable states and the set of unsafe states are disjoint. From this fact it clearly follows that the unsafe states are never reachable.

Given McMillan's symbolic approach to state reachability analysis of non-probabilistic finite-state systems based on Craig interpolation for SAT [McM03], it is natural to ask whether a corresponding probabilistic counterpart could be developed, i.e. a symbolic approach to *probabilistic state reachability analysis of probabilistic finite-state systems based on Craig interpolation for SSAT*. In this chapter, we suggest such an approach being able to *verify* probabilistic safety properties of the shape "the worst-case probability of reaching the unsafe states is at most 1‰" and thus complementing the symbolic falsification procedure from Chapter 5, though the latter being applicable to probabilistic hybrid systems.

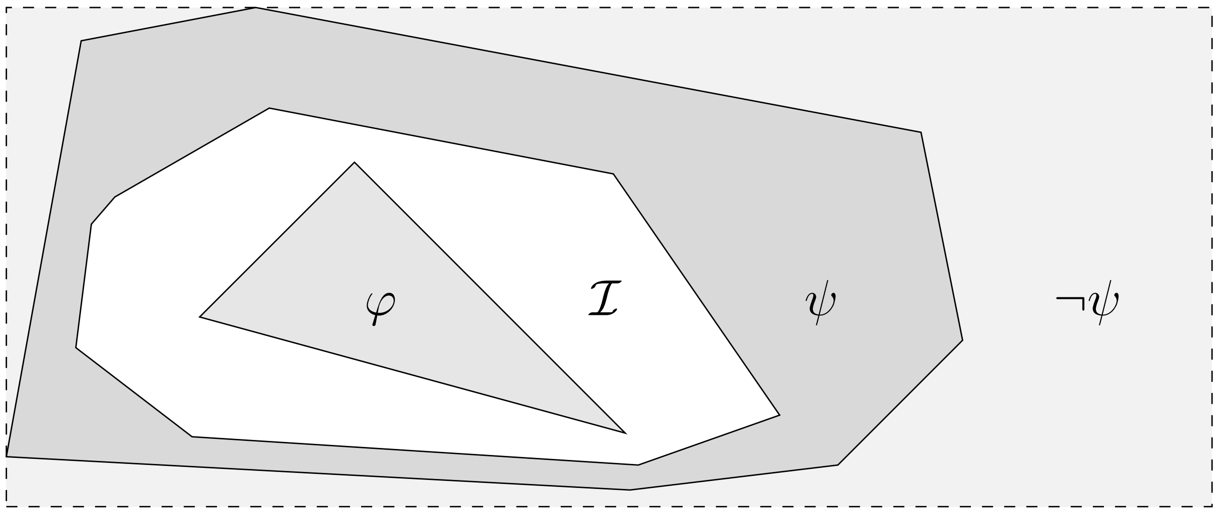


Figure 9.1: Graphical representation of a Craig interpolant $\mathcal{I}$ for formulae φ and ψ.

In addition to probabilistic state reachability, we further address the problem of *probabilistic region stability*. The latter problem is motivated by the notion of region stability for non-probabilistic hybrid systems [PW07a, PW07b], where a system is called stable with respect to some region *Region* if and only if all system runs eventually reach *Region* and finally stay in *Region* forever. We suggest an adaptation of region stability to the probabilistic case along with a symbolic, Craig interpolation-based procedure for the *verification* of probabilistic stability properties like "the probability that the system stabilizes within *Region* always is at least 99.9%".

This chapter is organized as follows. In Section 9.1, we introduce a generalization of the notion of Craig interpolants suitable for SSAT, while Section 9.2 deals with an algorithm for computing such generalized Craig interpolants based on a resolution calculus for SSAT, more precisely, on strong S-resolution. The application of generalized Craig interpolation to the symbolic analysis of probabilistic finite-state systems, namely to probabilistic state reachability and to probabilistic region stability, is then addressed in Section 9.3. The latter section also provides proofs of concept of these novel techniques using small examples.

9.1 Generalized Craig interpolants

Craig interpolation [Cra57] is a well-studied notion in formal logics which has several applications in Computer Science, among them model checking [McM03, McM05a, McM05b]. Given two formulae φ and ψ such that $\varphi \Rightarrow \psi$ is valid, a *Craig interpolant* for φ and ψ is a formula $\mathcal{I}$ which refers only to common variables of φ and ψ, and $\mathcal{I}$ is "intermediate" in the sense that $\varphi \Rightarrow \mathcal{I}$ and $\mathcal{I} \Rightarrow \psi$, confer Figure 9.1. Such interpolants do trivially exist in all logics permitting quantifier elimination, for instance, in propositional logic.

The observation that $\varphi \Rightarrow \psi$ holds if and only if $\neg(\varphi \Rightarrow \psi)$, or rather $\varphi \wedge \neg\psi$, is unsatisfiable gives rise to an equivalent definition: for an unsatisfiable formula $A \wedge B$, a formula $\mathcal{I}$ is a Craig interpolant for A and $\neg B$ if and only if both $A \wedge \neg\mathcal{I}$ and $\mathcal{I} \wedge B$ are unsatisfiable and $\mathcal{I}$ mentions only common variables. In order to avoid confusion in the latter case concerning the negation of B, we slightly abuse notation and simply say that $\mathcal{I}$ is a Craig interpolant for (A, B). For technical reasons, we refer to this alternative definition in the rest of the chapter. Recall that SSAT formulae are interpreted by the maximum probability of satisfaction, confer Definition 4.2. As the *maximum* probability that an implication $\varphi \Rightarrow \psi$ holds is inappropriate for our purpose, we reason about the maximum satisfaction probability p of the negated implication, i.e. of $\varphi \wedge \neg\psi$, instead. The latter relates to the *minimum* probability $1 - p$ that $\varphi \Rightarrow \psi$ holds, which is the desired notion.

In the remainder of this section, we investigate the issue of Craig interpolation for the SSAT framework. When approaching a reasonable definition of a Craig interpolant for SSAT, the semantics of the non-classical quantifier prefix poses problems. Let $\mathcal{Q} : (A \wedge B)$ be an SSAT formula. Each variable in $A \wedge B$ is bound by prefix $\mathcal{Q}$, providing the probabilistic interpretation of the variables which is lacking without the quantifier prefix. This issue can be addressed by considering the quantifier prefix $\mathcal{Q}$ as the global setting that serves to interpret the quantifier-free part, and consequently by interpreting the interpolant also within the scope of $\mathcal{Q}$, thus reasoning about the SSAT formulae $\mathcal{Q} : (A \wedge \neg\mathcal{I})$ and $\mathcal{Q} : (\mathcal{I} \wedge B)$.

A more fundamental problem is that a classical Craig interpolant for (A, B) only exists if $Pr(\mathcal{Q} : (A \wedge B)) = 0$, since $A \wedge B$ has to be unsatisfiable by definition of a Craig interpolant which applies if and only if $Pr(\mathcal{Q} : (A \wedge B)) = 0$. The precondition that $Pr(\mathcal{Q} : (A \wedge B)) = 0$ would be far too restrictive for application of interpolation in the stochastic setting, as the notion of unsatisfiability of $A \wedge B$ is naturally generalized to *satisfiability with insufficient probability*, i.e. $Pr(\mathcal{Q} : (A \wedge B))$ being "sufficiently small". Such relaxed requirements actually appear in practice, for instance, in probabilistic verification where safety properties like "a fatal system error is *never* reachable" are frequently replaced by probabilistic ones like "a fatal system error is reachable only with *sufficiently small probability* of at most 0.1‰". Motivated by above facts, interpolants for SSAT should also exist if $A \wedge B$ is satisfiable with reasonably low probability.

The resulting notion of interpolation, being made precise in Definition 9.1, matches the following intuition. In classical Craig interpolation, when performed in logics permitting quantifier elimination, the Craig interpolants for (A, B) with $A \wedge B$ being unsatisfiable form a lattice with implication as its ordering and with

$$A^{\exists} \;=\; \exists a_1, \ldots, a_\alpha : A$$

as its bottom element and

$$\overline{B}^{\forall} \;=\; \neg\exists b_1, \ldots, b_\beta : B$$

as its top element, where the $a_i \in Var(A) \setminus Var(B)$ and $b_i \in Var(B) \setminus Var(A)$ are the local variables of A and of B, respectively. In the generalized setting required for SSAT, $A \Rightarrow \neg B$ and thus $A^{\exists} \Rightarrow \overline{B}^{\forall}$ may no longer hold such that the above lattice can collapse to the empty set. To preserve the overall structure, it is however natural to use the lattice of propositional formulae over common variables of A and B "in between"

$$A^{\exists} \wedge \overline{B}^{\forall}$$

as bottom element and

$$A^{\exists} \vee \overline{B}^{\forall}$$

as top element instead. This lattice is non-empty and coincides with the classical one whenever $A \wedge B$ is unsatisfiable.

Definition 9.1 (Generalized Craig interpolant)
Let A and B be propositional formulae and $V_A := Var(A) \setminus Var(B) = \{a_1, \ldots, a_\alpha\}$, $V_B := Var(B) \setminus Var(A) = \{b_1, \ldots, b_\beta\}$, $V_{A,B} := Var(A) \cap Var(B)$, $A^\exists = \exists a_1, \ldots, a_\alpha : A$, and $\overline{B}^\forall = \neg\exists b_1, \ldots, b_\beta : B$. A propositional formula $\mathcal{I}$ is called generalized Craig interpolant for (A, B) *if and only if the following properties are satisfied.*

1. $Var(\mathcal{I}) \subseteq V_{A,B}$
2. $\left(A^\exists \wedge \overline{B}^\forall\right) \Rightarrow \mathcal{I}$
3. $\mathcal{I} \Rightarrow \left(A^\exists \vee \overline{B}^\forall\right)$

For two propositional formulae A and B, the four quantifier-free propositional formulae equivalent to $A^\exists \wedge \overline{B}^\forall$, to $A^\exists$, to $\overline{B}^\forall$, and to $A^\exists \vee \overline{B}^\forall$ are generalized Craig interpolants for (A, B). These generalized interpolants always exist since propositional logic has quantifier elimination.

While Definition 9.1 motivates the generalized notion of Craig interpolant from a model-theoretic perspective, we state an equivalent definition of generalized Craig interpolants in Lemma 9.1 that substantiates the intuition of generalized interpolants and allows for an illustration of their geometric shape. Given two formulae A and B, the idea of a generalized Craig interpolant is depicted in Figure 9.2. The set of solutions of A is defined by the rectangle on the $V_A, V_{A,B}$-plane with a cylindrical extension in V_B-direction as A does not contain variables in V_B. Similarly, the solution set of B is given by the triangle on the $V_B, V_{A,B}$-plane and its cylinder in V_A-direction. The solution set of $A \wedge B$ is then determined by the intersection of both cylinders. Since $A \wedge B \wedge \neg(A \wedge B)$ is unsatisfiable, the sets $A \wedge \neg(A \wedge B)$ and $B \wedge \neg(A \wedge B)$ are disjoint. This gives the opportunity to talk about interpolants with respect to these sets. However, a formula $\mathcal{I}$ over only common variables in $V_{A,B}$ need not exist when demanding unsatisfiability of $A \wedge \neg(A \wedge B) \wedge \neg\mathcal{I}$ and of $\mathcal{I} \wedge B \wedge \neg(A \wedge B)$. This is indicated by Figure 9.2 and proven by the simple example where $A = a$ and $B = b$. As $V_{A,B} = \emptyset$, formula $\mathcal{I}$ is either `true` or `false`. In the first case,

$$\mathcal{I} \wedge B \wedge \neg(A \wedge B) \quad = \quad \mathtt{true} \wedge b \wedge \neg(a \wedge b)$$

is satisfiable, while

$$A \wedge \neg(A \wedge B) \wedge \neg\mathcal{I} \quad = \quad a \wedge \neg(a \wedge b) \wedge \neg\mathtt{false}$$

is in the second case. If we however project the solution set of $A \wedge B$ onto the $V_{A,B}$-axis and subtract the resulting hyperplane $\mathcal{S}_{A,B}$ from the solution sets of A and of B then such a formula $\mathcal{I}$ over $V_{A,B}$-variables exists, as illustrated in Figure 9.2. In the simple example above, this hyperplane $\mathcal{S}_{A,B}$ covers all possible assignments since $V_{A,B} = \emptyset$. As a consequence, subtraction of $\mathcal{S}_{A,B}$ from the solution sets of A and of B leads to the empty set in both cases, which trivially allows for the greatest and smallest possible generalized interpolants `true` and `false`, respectively.

The next lemma formalizes such generalized Craig interpolants $\mathcal{I}$ and shows their equivalence to the ones from Definition 9.1.

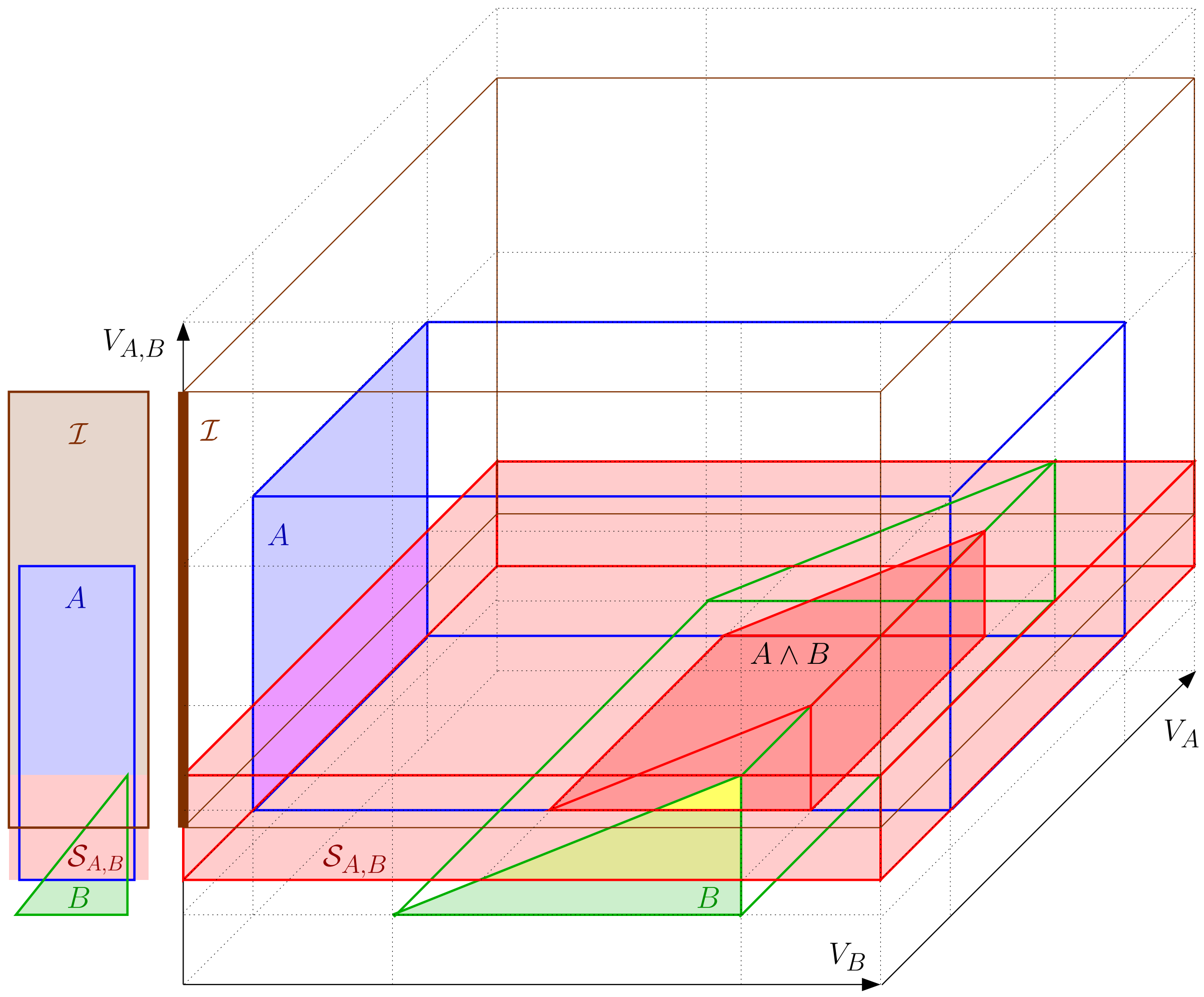


Figure 9.2: Geometric interpretation of a generalized Craig interpolant $\mathcal{I}$ for (A, B). V_A-, V_B-, and $V_{A,B}$-axes denote assignments to variables occurring only in A, only in B, and in both A and B, respectively. (Source of figure: [TF11])

Lemma 9.1 (Generalized Craig interpolant for SSAT)
Let $\Phi = \mathcal{Q} : (A \wedge B)$ be some SSAT formula, V_A, V_B, and $V_{A,B}$ be defined as in Definition 9.1, and $\mathcal{S}_{A,B}$ be a propositional formula with $Var(\mathcal{S}_{A,B}) \subseteq V_{A,B}$ such that $\mathcal{S}_{A,B} \equiv \exists a_1, \ldots, a_\alpha, b_1, \ldots, b_\beta : (A \wedge B)$. Then, a propositional formula $\mathcal{I}$ is a generalized Craig interpolant for (A, B) if and only if the following properties are satisfied.

1. $Var(\mathcal{I}) \subseteq V_{A,B}$
2. $Pr(\mathcal{Q} : (A \wedge \neg\mathcal{S}_{A,B} \wedge \neg\mathcal{I})) = 0$
3. $Pr(\mathcal{Q} : (\mathcal{I} \wedge B \wedge \neg\mathcal{S}_{A,B})) = 0$

Proof. We prove the lemma by showing that above items hold if and only if the items of Definition 9.1 hold. Trivially, above item 1 is true if and only if item 1 of Definition 9.1 is true.

To prove that above item 2 is satisfied if and only if item 2 of Definition 9.1 is satisfied, we conclude the following: $\models (A^{\exists} \wedge \overline{B}^{\forall}) \Rightarrow \mathcal{I}$ if and only if $\models (\exists a_1, \ldots, a_\alpha : A \wedge \overline{B}^{\forall}) \Rightarrow \mathcal{I}$

if and only if $\models \forall a_1, \ldots, a_\alpha : ((A \wedge \overline{B}^{\forall}) \Rightarrow \mathcal{I})$ if and only if $\models (A \wedge \overline{B}^{\forall}) \Rightarrow \mathcal{I}$ if and only if $\models (A \wedge (\neg A^{\exists} \vee \overline{B}^{\forall})) \Rightarrow \mathcal{I}$ if and only if $\models (A \wedge (\neg \exists a_1, \ldots, a_\alpha : A \vee \neg \exists b_1, \ldots, b_\beta : B)) \Rightarrow \mathcal{I}$ if and only if $\models (A \wedge (\forall a_1, \ldots, a_\alpha : \neg A \vee \forall b_1, \ldots, b_\beta : \neg B)) \Rightarrow \mathcal{I}$ if and only if $\models (A \wedge \forall a_1, \ldots, a_\alpha, b_1, \ldots, b_\beta : (\neg A \vee \neg B)) \Rightarrow \mathcal{I}$ if and only if $\models (A \wedge \neg \exists a_1, \ldots, a_\alpha, b_1, \ldots, b_\beta : (A \wedge B)) \Rightarrow \mathcal{I}$ if and only if $\models (A \wedge \neg \mathcal{S}_{A,B}) \Rightarrow \mathcal{I}$ if and only if $A \wedge \neg \mathcal{S}_{A,B} \wedge \neg \mathcal{I}$ is unsatisfiable if and only if $Pr(\mathcal{Q} : (A \wedge \neg \mathcal{S}_{A,B} \wedge \neg \mathcal{I})) = 0$.

An analogous reasoning is applied for above item 3 and item 3 of Definition 9.1: $\models \mathcal{I} \Rightarrow (A^{\exists} \vee \overline{B}^{\forall})$ if and only if $\models \mathcal{I} \Rightarrow (A^{\exists} \vee \neg \exists b_1, \ldots, b_\beta : B)$ if and only if $\models \mathcal{I} \Rightarrow (A^{\exists} \vee \forall b_1, \ldots, b_\beta : \neg B)$ if and only if $\models \forall b_1, \ldots, b_\beta : (\mathcal{I} \Rightarrow (A^{\exists} \vee \neg B))$ if and only if $\models \mathcal{I} \Rightarrow (A^{\exists} \vee \neg B)$ if and only if $\models \mathcal{I} \Rightarrow ((A^{\exists} \wedge \neg \overline{B}^{\forall}) \vee \neg B)$ if and only if $\models \mathcal{I} \Rightarrow ((\exists a_1, \ldots, a_\alpha : A \wedge \neg\neg \exists b_1, \ldots, b_\beta : B) \vee \neg B)$ if and only if $\models \mathcal{I} \Rightarrow ((\exists a_1, \ldots, a_\alpha, b_1, \ldots, b_\beta : (A \wedge B)) \vee \neg B)$ if and only if $\models \mathcal{I} \Rightarrow (\mathcal{S}_{A,B} \vee \neg B)$ if and only if $\mathcal{I} \wedge \neg \mathcal{S}_{A,B} \wedge B$ is unsatisfiable if and only if $Pr(\mathcal{Q} : (\mathcal{I} \wedge B \wedge \neg \mathcal{S}_{A,B})) = 0$. □

Observe that the concept of generalized Craig interpolants is a generalization of Craig interpolants in the sense that whenever $A \wedge B$ is unsatisfiable, i.e. if $Pr(\mathcal{Q} : (A \wedge B)) = 0$, then each generalized Craig interpolant $\mathcal{I}$ for (A, B) actually is a Craig interpolant for (A, B) since $\mathcal{S}_{A,B} \equiv \texttt{false}$.

9.2 Computing generalized Craig interpolants

In this section, we proceed to the efficient computation of generalized Craig interpolants. The remark following Definition 9.1 indicates that generalized Craig interpolants for SSAT can in principle be computed by *explicit* quantifier elimination methods, like Shannon's expansion [Sha49] or binary decision diagrams (BDDs) [Lee59, Bry86]. We however aim at a more efficient method based on *strong S-resolution* introduced in Subsection 6.2.2, akin to resolution-based Craig interpolation for propositional SAT by Pudlák [Pud97]. The latter approach has been integrated into DPLL-based SAT solvers featuring conflict analysis and successfully applied to symbolic model checking [McM03, McM05a, McM05b].

We remark that on SSAT formulae $\mathcal{Q} : (A \wedge B)$, Pudlák's algorithm, which has unsatisfiability of $A \wedge B$ as precondition, does not work in general. When instead considering the unsatisfiable formula $A \wedge B \wedge \neg \mathcal{S}_{A,B}$, being in CNF, then Pudlák's method is applicable and actually produces a generalized Craig interpolant. The main drawback of this approach however is the explicit construction of the quantifier-free formula $\neg \mathcal{S}_{A,B}$, again calling for explicit quantifier elimination.

We instead propose an algorithm based on strong S-resolution for computing generalized Craig interpolants, which operates directly on SSAT formulae $\mathcal{Q} : (A \wedge B)$ without adding $\neg \mathcal{S}_{A,B}$, and thus does not comprise any preprocessing involving quantifier elimination. In what follows, we assume that the matrix φ of an SSAT formula $\mathcal{Q} : \varphi$ is in CNF. Recall that strong S-resolution, consisting of rules R.1, R.2s, and R.3, is a sound and complete procedure for solving SSAT formulae in CNF, confer Corollary 6.1 and Theorem 6.1. Moreover, the clauses in all pairs $c^{(pl,pu)} | \mathcal{Q} : \varphi$ derived by strong S-resolution carry tight bounds, i.e. $pl = pu$, according to Lemma 6.1, and the SSAT formula $\mathcal{Q} : \varphi$ remains unchanged after execution of any rule of strong S-resolution. Due to the latter facts, we denote by $\vdash_{\mathsf{R.1}} c^p$, $\vdash_{\mathsf{R.2s}} c^p$, and $(c_1^{p_1}, c_2^{p_2}) \vdash_{\mathsf{R.3}} c^p$ the application of rules R.1, R.2s, and R.3, respectively.

For the purpose of computing generalized Craig interpolants by means of strong S-resolution, the above rules are enhanced to deal with pairs (c^p, I) of annotated clauses c^p and propositional formulae I. Such formulae I are in a certain sense *intermediate* generalized Craig interpolants, i.e. generalized interpolants for subformulae arising from instantiating some variables by partial assignments that falsify clauses c. The precise interpretation is formalized by Lemma 9.2 later on. Once a pair $(\emptyset^p, I)$ comprising the empty clause is derived, I thus is a generalized Craig interpolant for the given SSAT formula. This augmented S-resolution, which we call *interpolating strong S-resolution* or simply *interpolating S-resolution*, is defined by the rules RI.1, RI.2, and RI.3 introduced below. The construction of intermediate interpolants I in RI.1 and RI.3 coincides with the classical rules by Pudlák [Pud97], while RI.2 misses a corresponding counterpart. The rationale is that RI.2, or rather R.2s, refers to satisfying assignments τ of $A \wedge B$, which do not exist in classical Craig interpolation. As formula $A \wedge B$ becomes a tautology after substituting the partial assignment τ into it, confer rule R.2s, its quantified variant $\mathcal{S}_{A,B} = \exists a_1, \ldots, b_1, \ldots : (A \wedge B)$ also becomes tautological under the same substitution, i.e. $\models \mathcal{S}_{A,B}[\tau(x_1)/x_1, \ldots, \tau(x_i)/x_i]$. As a consequence, $\neg\mathcal{S}_{A,B}[\tau(x_1)/x_1, \ldots, \tau(x_i)/x_i]$ is unsatisfiable, and so are $(A \wedge \neg\mathcal{S}_{A,B})[\tau(x_1)/x_1, \ldots, \tau(x_i)/x_i]$ and $(B \wedge \neg\mathcal{S}_{A,B})[\tau(x_1)/x_1, \ldots, \tau(x_i)/x_i]$. This immediately implies that the actual intermediate interpolant in rule RI.2 can be chosen arbitrarily over common variables in $V_{A,B}$. This freedom allows us to control the geometric extent of generalized Craig interpolants within the "don't care"-region provided by the models of $\mathcal{S}_{A,B}$, as formalized by Corollary 9.2 later on.

Let $\mathcal{Q} : (A \wedge B)$ be an SSAT formula where $A \wedge B$ is in CNF. The rules of interpolating S-resolution are then as follows.

$$\text{(RI.1)} \qquad \frac{c \vdash_{\mathsf{R.1}} c^p, \quad I = \begin{cases} \texttt{false} \ ; c \in A \\ \texttt{true} \ \ ; c \in B \end{cases}}{(c^p, I)}$$

$$\text{(RI.2)} \qquad \frac{\vdash_{\mathsf{R.2s}} c^p, \quad I \text{ is any formula over } V_{A,B}}{(c^p, I)}$$

$$\text{(RI.3)} \qquad \frac{\begin{array}{c} ((c_1 \vee \neg x)^{p_1}, I_1), \ ((c_2 \vee x)^{p_2}, I_2), \\ ((c_1 \vee \neg x)^{p_1}, (c_2 \vee x)^{p_2}) \vdash_{\mathsf{R.3}} (c_1 \vee c_2)^p, \\ I = \begin{cases} I_1 \vee I_2 & \text{if } x \in V_A \\ I_1 \wedge I_2 & \text{if } x \in V_B \\ (\neg x \vee I_1) \wedge (x \vee I_2) & \text{if } x \in V_{A,B} \end{cases} \end{array}}{((c_1 \vee c_2)^p, I)}$$

The following lemma establishes the theoretical foundation of computing generalized Craig interpolants by interpreting the derived pairs (c^p, I).

Lemma 9.2 (Interpretation of derived pairs (c^p, I))
Let $\Phi = \mathcal{Q} : (A \wedge B)$ be some SSAT formula with $\mathcal{Q} = Q_1 x_1 \ldots Q_n x_n$ and with $A \wedge B$ being in CNF, and let the pair (c^p, I) be derivable from Φ by interpolating S-resolution with $\mathcal{Q}(c) = Q_1 x_1 \ldots Q_i x_i$. Then, for each truth assignment $\tau : Var(\mathcal{Q}(c)) \to \mathbb{B}$ with $\forall x \in Var(c) : \tau(x) = ff_c(x)$ it holds that

1. *$Var(I) \subseteq V_{A,B}$,*
2. *$Pr(Q_{i+1}x_{i+1} \ldots Q_n x_n : (A \wedge \neg\mathcal{S}_{A,B} \wedge \neg I)[\tau(x_1)/x_1] \ldots [\tau(x_i)/x_i]) = 0$, and*
3. *$Pr(Q_{i+1}x_{i+1} \ldots Q_n x_n : (I \wedge B \wedge \neg\mathcal{S}_{A,B})[\tau(x_1)/x_1] \ldots [\tau(x_i)/x_i]) = 0$.*

Proof. We prove the lemma by induction over application of the interpolating S-resolution rules RI.1, RI.2, and RI.3.

In the base case, we can just apply RI.1 and RI.2. Item 1 clearly holds for both rules since I contains only variables in $V_{A,B}$. Let us consider RI.1 first. If $c \in A$ then $I = \texttt{false}$. By construction of τ, i.e. c evaluates to `false` under τ, it follows that $A[\tau(x_1)/x_1] \ldots [\tau(x_i)/x_i]$ is unsatisfiable and thus

$$Pr(\mathcal{Q}' : (A \wedge \neg\mathcal{S}_{A,B} \wedge \neg I)[\tau(x_1)/x_1] \ldots [\tau(x_i)/x_i]) = 0 \quad .$$

As $I = \texttt{false}$, immediately

$$Pr(\mathcal{Q}' : (I \wedge B \wedge \neg\mathcal{S}_{A,B})[\tau(x_1)/x_1] \ldots [\tau(x_i)/x_i]) = 0 \quad .$$

If $c \in B$ then $I = \texttt{true}$. Obviously,

$$Pr(\mathcal{Q}' : (A \wedge \neg\mathcal{S}_{A,B} \wedge \neg I)[\tau(x_1)/x_1] \ldots [\tau(x_i)/x_i]) = 0$$

and by construction of τ,

$$Pr(\mathcal{Q}' : (I \wedge B \wedge \neg\mathcal{S}_{A,B})[\tau(x_1)/x_1] \ldots [\tau(x_i)/x_i]) = 0 \quad .$$

For rule RI.2, we have $\models (A \wedge B)[\tau(x_1)/x_1] \ldots [\tau(x_i)/x_i]$ which immediately implies that $\models (\exists a_1, \ldots, a_\alpha, b_1, \ldots, b_\beta : (A \wedge B))[\tau(x_1)/x_1] \ldots [\tau(x_i)/x_i]$, i.e. $\models \mathcal{S}_{A,B}[\tau(x_1)/x_1] \ldots [\tau(x_i)/x_i]$ by definition of $\mathcal{S}_{A,B}$. Rephrasing the latter, $\neg\mathcal{S}_{A,B}[\tau(x_1)/x_1] \ldots [\tau(x_i)/x_i]$ is unsatisfiable. Consequently, for any propositional formula I

$$\begin{aligned} Pr(\mathcal{Q}' : (A \wedge \neg\mathcal{S}_{A,B} \wedge \neg I)[\tau(x_1)/x_1] \ldots [\tau(x_i)/x_i]) &= 0 \quad , \\ Pr(\mathcal{Q}' : (I \wedge B \wedge \neg\mathcal{S}_{A,B})[\tau(x_1)/x_1] \ldots [\tau(x_i)/x_i]) &= 0 \quad . \end{aligned}$$

This proves items 2 and 3 for the base case.

In the induction step, we now assume that the lemma holds for all clauses in the premises of rule RI.3. Then, by construction of I, item 1 clearly holds for I, i.e. $Var(I) \subseteq V_{A,B}$. Induction hypothesis assumes that

$$\begin{aligned} Pr(\mathcal{Q}' : (A \wedge \neg\mathcal{S}_{A,B} \wedge \neg I_1)[\tau_1(x_1)/x_1] \ldots [\tau_1(x_{j-1})/x_{j-1}][\texttt{true}/x_j]) &= 0 \quad , \\ Pr(\mathcal{Q}' : (I_1 \wedge B \wedge \neg\mathcal{S}_{A,B})[\tau_1(x_1)/x_1] \ldots [\tau_1(x_{j-1})/x_{j-1}][\texttt{true}/x_j]) &= 0 \end{aligned}$$

holds for $((c_1 \vee \neg x_j)^{p_1}, I_1)$ and for each $\tau_1 : \{x_1, \ldots, x_{j-1}\} \to \mathbb{B}$ with $\forall x \in \mathit{Var}(c_1) : \tau_1(x) = \mathit{ff}_{c_1}(x)$, and that

$$\begin{aligned} Pr(\mathcal{Q}' : (A \wedge \neg\mathcal{S}_{A,B} \wedge \neg I_2)[\tau_2(x_1)/x_1] \ldots [\tau_2(x_{j-1})/x_{j-1}][\mathtt{false}/x_j]) = 0 \ , \\ Pr(\mathcal{Q}' : (I_2 \wedge B \wedge \neg\mathcal{S}_{A,B})[\tau_2(x_1)/x_1] \ldots [\tau_2(x_{j-1})/x_{j-1}][\mathtt{false}/x_j]) = 0 \end{aligned}$$

holds for $((c_2 \vee x_j)^{p_2}, I_2)$ and for each $\tau_2 : \{x_1, \ldots, x_{j-1}\} \to \mathbb{B}$ with $\forall x \in \mathit{Var}(c_2) : \tau_2(x) = \mathit{ff}_{c_2}(x)$, where $j \geq i+1$ and $\mathcal{Q}' = Q_{j+1}x_{j+1} \ldots Q_n x_n$. Let $\tau : \{x_1, \ldots, x_{j-1}\} \to \mathbb{B}$ be any assignment with $\tau(x) = \tau_1(x)$ if $x \in \mathit{Var}(c_1)$ and $\tau(x) = \tau_2(x)$ if $x \in \mathit{Var}(c_2)$. Note that τ is well-defined as $\not\models (c_1 \vee c_2)$, i.e. for each $x \in \mathit{Var}(c_1) \cap \mathit{Var}(c_2) : \tau_1(x) = \tau_2(x)$. We now show that

$$\begin{aligned} Pr_A &:= Pr(Q_j x_j \mathcal{Q}' : (A \wedge \neg\mathcal{S}_{A,B} \wedge \neg I)[\tau(x_1)/x_1] \ldots [\tau(x_{j-1})/x_{j-1}]) &&= 0 \ , \\ Pr_B &:= Pr(Q_j x_j \mathcal{Q}' : (I \wedge B \wedge \neg\mathcal{S}_{A,B})[\tau(x_1)/x_1] \ldots [\tau(x_{j-1})/x_{j-1}]) &&= 0 \end{aligned}$$

by proving that

$$\begin{aligned} Pr_{A,x} &:= Pr(\mathcal{Q}' : (A \wedge \neg\mathcal{S}_{A,B} \wedge \neg I)[\tau(x_1)/x_1] \ldots [\tau(x_{j-1})/x_{j-1}][\mathtt{true}/x_j]) &&= 0 \ , \\ Pr_{A,\neg x} &:= Pr(\mathcal{Q}' : (A \wedge \neg\mathcal{S}_{A,B} \wedge \neg I)[\tau(x_1)/x_1] \ldots [\tau(x_{j-1})/x_{j-1}][\mathtt{false}/x_j]) &&= 0 \ , \\ Pr_{B,x} &:= Pr(\mathcal{Q}' : (I \wedge B \wedge \neg\mathcal{S}_{A,B})[\tau(x_1)/x_1] \ldots [\tau(x_{j-1})/x_{j-1}][\mathtt{true}/x_j]) &&= 0 \ , \\ Pr_{B,\neg x} &:= Pr(\mathcal{Q}' : (I \wedge B \wedge \neg\mathcal{S}_{A,B})[\tau(x_1)/x_1] \ldots [\tau(x_{j-1})/x_{j-1}][\mathtt{false}/x_j]) &&= 0 \ . \end{aligned}$$

We therefore distinguish the three cases $x_j \in V_A$, $x_j \in V_B$, and $x_j \in V_{A,B}$.

First, let be $x_j \in V_A$. Then, $I = I_1 \vee I_2$. By induction hypothesis and by construction of I,

$$\begin{aligned} 0 &= Pr(\mathcal{Q}' : (A \wedge \neg\mathcal{S}_{A,B} \wedge \neg I_1)[\tau_1(x_1)/x_1] \ldots [\tau_1(x_{j-1})/x_{j-1}][\mathtt{true}/x_j]) \\ &\geq Pr(\mathcal{Q}' : (A \wedge \neg\mathcal{S}_{A,B} \wedge \neg I_1 \wedge \neg I_2)[\tau_1(x_1)/x_1] \ldots [\tau_1(x_{j-1})/x_{j-1}][\mathtt{true}/x_j]) \\ &= Pr(\mathcal{Q}' : (A \wedge \neg\mathcal{S}_{A,B} \wedge \neg I)[\tau_1(x_1)/x_1] \ldots [\tau_1(x_{j-1})/x_{j-1}][\mathtt{true}/x_j]) \ . \end{aligned}$$

Due to construction of τ, it holds in particular that

$$0 = Pr_{A,x} \ .$$

Analogously,

$$\begin{aligned} 0 &= Pr(\mathcal{Q}' : (A \wedge \neg\mathcal{S}_{A,B} \wedge \neg I_2)[\tau_2(x_1)/x_1] \ldots [\tau_2(x_{j-1})/x_{j-1}][\mathtt{false}/x_j]) \\ &\geq Pr(\mathcal{Q}' : (A \wedge \neg\mathcal{S}_{A,B} \wedge \neg I_1 \wedge \neg I_2)[\tau_2(x_1)/x_1] \ldots [\tau_2(x_{j-1})/x_{j-1}][\mathtt{false}/x_j]) \\ &= Pr(\mathcal{Q}' : (A \wedge \neg\mathcal{S}_{A,B} \wedge \neg I)[\tau_2(x_1)/x_1] \ldots [\tau_2(x_{j-1})/x_{j-1}][\mathtt{false}/x_j]) \end{aligned}$$

and thus

$$0 = Pr_{A,\neg x} \ .$$

As $x_j \notin \mathit{Var}(I) \cup \mathit{Var}(B) \cup \mathit{Var}(\neg\mathcal{S}_{A,B})$, for each $v \in \mathbb{B}$ it holds that

$$Pr(\mathcal{Q}' : (I \wedge B \wedge \neg\mathcal{S}_{A,B})[\tau(x_1)/x_1] \ldots [\tau(x_{j-1})/x_{j-1}][v/x_j])$$

$$= Pr(\mathcal{Q}' : (I \wedge B \wedge \neg\mathcal{S}_{A,B})[\tau(x_1)/x_1] \dots [\tau(x_{j-1})/x_{j-1}])$$

which implies $Pr_{B,x} = Pr_{B,\neg x}$. We conclude from induction hypothesis that

$$\begin{aligned} Pr(\mathcal{Q}' : (I_1 \wedge B \wedge \neg\mathcal{S}_{A,B})[\tau_1(x_1)/x_1] \dots [\tau_1(x_{j-1})/x_{j-1}]) = 0 \quad , \\ Pr(\mathcal{Q}' : (I_2 \wedge B \wedge \neg\mathcal{S}_{A,B})[\tau_2(x_1)/x_1] \dots [\tau_2(x_{j-1})/x_{j-1}]) = 0 \end{aligned}$$

again by virtue of $x_j \notin Var(I) \cup Var(B) \cup Var(\neg\mathcal{S}_{A,B})$. Moreover,

$$\begin{aligned} Pr(\mathcal{Q}' : (I_1 \wedge B \wedge \neg\mathcal{S}_{A,B})[\tau(x_1)/x_1] \dots [\tau(x_{j-1})/x_{j-1}]) = 0 \quad , \\ Pr(\mathcal{Q}' : (I_2 \wedge B \wedge \neg\mathcal{S}_{A,B})[\tau(x_1)/x_1] \dots [\tau(x_{j-1})/x_{j-1}]) = 0 \end{aligned}$$

due to construction of τ. Note that if $Pr(\mathcal{Q} : \varphi_1) = 0$ and $Pr(\mathcal{Q} : \varphi_2) = 0$ then $Pr(\mathcal{Q} : (\varphi_1 \vee \varphi_2)) = 0$ since $Pr(\mathcal{Q} : \varphi) = 0$ if and only if φ is unsatisfiable.[1] As a consequence,

$$\begin{aligned} 0 &= Pr\left(\mathcal{Q}' : \left(\begin{array}{rl} & (I_1 \wedge B \wedge \neg\mathcal{S}_{A,B})[\tau(x_1)/x_1] \dots [\tau(x_{j-1})/x_{j-1}] \\ \vee & (I_2 \wedge B \wedge \neg\mathcal{S}_{A,B})[\tau(x_1)/x_1] \dots [\tau(x_{j-1})/x_{j-1}] \end{array}\right)\right) \\ &= Pr(\mathcal{Q}' : ((I_1 \vee I_2) \wedge B \wedge \neg\mathcal{S}_{A,B})[\tau(x_1)/x_1] \dots [\tau(x_{j-1})/x_{j-1}]) \\ &= Pr(\mathcal{Q}' : (I \wedge B \wedge \neg\mathcal{S}_{A,B})[\tau(x_1)/x_1] \dots [\tau(x_{j-1})/x_{j-1}]) \\ &= Pr_{B,x} = Pr_{B,\neg x} \quad . \end{aligned}$$

Second, let be $x_j \in V_B$. Then, $I = I_1 \wedge I_2$. As $x_j \notin Var(A) \cup Var(\neg\mathcal{S}_{A,B}) \cup Var(\neg I)$, with the same argument as above,

$$\begin{aligned} 0 &= Pr\left(\mathcal{Q}' : \left(\begin{array}{rl} & (A \wedge \neg\mathcal{S}_{A,B} \wedge \neg I_1)[\tau(x_1)/x_1] \dots [\tau(x_{j-1})/x_{j-1}] \\ \vee & (A \wedge \neg\mathcal{S}_{A,B} \wedge \neg I_2)[\tau(x_1)/x_1] \dots [\tau(x_{j-1})/x_{j-1}] \end{array}\right)\right) \\ &= Pr(\mathcal{Q}' : (A \wedge \neg\mathcal{S}_{A,B} \wedge (\neg I_1 \vee \neg I_2))[\tau(x_1)/x_1] \dots [\tau(x_{j-1})/x_{j-1}]) \\ &= Pr(\mathcal{Q}' : (A \wedge \neg\mathcal{S}_{A,B} \wedge \neg I)[\tau(x_1)/x_1] \dots [\tau(x_{j-1})/x_{j-1}]) \\ &= Pr_{A,x} = Pr_{A,\neg x} \quad . \end{aligned}$$

Again following the reasoning above, we have

$$\begin{aligned} 0 &= Pr(\mathcal{Q}' : (I_1 \wedge B \wedge \neg\mathcal{S}_{A,B})[\tau_1(x_1)/x_1] \dots [\tau_1(x_{j-1})/x_{j-1}][\texttt{true}/x_j]) \\ &\geq Pr(\mathcal{Q}' : (I_1 \wedge I_2 \wedge B \wedge \neg\mathcal{S}_{A,B})[\tau_1(x_1)/x_1] \dots [\tau_1(x_{j-1})/x_{j-1}][\texttt{true}/x_j]) \\ &= Pr(\mathcal{Q}' : (I \wedge B \wedge \neg\mathcal{S}_{A,B})[\tau_1(x_1)/x_1] \dots [\tau_1(x_{j-1})/x_{j-1}][\texttt{true}/x_j]) \end{aligned}$$

and thus

$$0 = Pr_{B,x}$$

as well as

$$0 = Pr(\mathcal{Q}' : (I_2 \wedge B \wedge \neg\mathcal{S}_{A,B})[\tau_2(x_1)/x_1] \dots [\tau_2(x_{j-1})/x_{j-1}][\texttt{false}/x_j])$$

[1] This statement is not true in general if $\mathcal{Q}$ also contains *universal* quantifiers, which however is not the case in this thesis.

$$\begin{aligned}
&\geq Pr(\mathcal{Q}' : (I_1 \wedge I_2 \wedge B \wedge \neg\mathcal{S}_{A,B})[\tau_2(x_1)/x_1]\dots[\tau_2(x_{j-1})/x_{j-1}][\mathtt{false}/x_j]) \\
&= Pr(\mathcal{Q}' : (I \wedge B \wedge \neg\mathcal{S}_{A,B})[\tau_2(x_1)/x_1]\dots[\tau_2(x_{j-1})/x_{j-1}][\mathtt{false}/x_j]) \quad ,
\end{aligned}$$

and thus

$$0 = Pr_{B,\neg x} \quad .$$

Third, let be $x_j \in V_{A,B}$. Then, $I = (\neg x_j \vee I_1) \wedge (x_j \vee I_2)$, and we deduce

$$\begin{aligned}
0 &= Pr(\mathcal{Q}' : (A \wedge \neg\mathcal{S}_{A,B} \wedge \neg I_1)[\tau_1(x_1)/x_1]\dots[\tau_1(x_{j-1})/x_{j-1}][\mathtt{true}/x_j]) \\
&= Pr(\mathcal{Q}' : (A \wedge \neg\mathcal{S}_{A,B} \\
&\qquad \wedge ((x_j \wedge \neg I_1) \vee (\neg x_j \wedge \neg I_2)))[\tau_1(x_1)/x_1]\dots[\tau_1(x_{j-1})/x_{j-1}][\mathtt{true}/x_j]) \\
&= Pr(\mathcal{Q}' : (A \wedge \neg\mathcal{S}_{A,B} \wedge \neg I)[\tau_1(x_1)/x_1]\dots[\tau_1(x_{j-1})/x_{j-1}][\mathtt{true}/x_j])
\end{aligned}$$

and, in particular,

$$0 = Pr_{A,x} \quad .$$

Analogously,

$$\begin{aligned}
0 &= Pr(\mathcal{Q}' : (A \wedge \neg\mathcal{S}_{A,B} \wedge \neg I_2)[\tau_2(x_1)/x_1]\dots[\tau_2(x_{j-1})/x_{j-1}][\mathtt{false}/x_j]) \\
&= Pr(\mathcal{Q}' : (A \wedge \neg\mathcal{S}_{A,B} \\
&\qquad \wedge ((x_j \wedge \neg I_1) \vee (\neg x_j \wedge \neg I_2)))[\tau_2(x_1)/x_1]\dots[\tau_2(x_{j-1})/x_{j-1}][\mathtt{false}/x_j]) \\
&= Pr(\mathcal{Q}' : (A \wedge \neg\mathcal{S}_{A,B} \wedge \neg I)[\tau_2(x_1)/x_1]\dots[\tau_2(x_{j-1})/x_{j-1}][\mathtt{false}/x_j])
\end{aligned}$$

and, in particular,

$$0 = Pr_{A,\neg x} \quad .$$

Furthermore,

$$\begin{aligned}
0 &= Pr(\mathcal{Q}' : (I_1 \wedge B \wedge \neg\mathcal{S}_{A,B})[\tau_1(x_1)/x_1]\dots[\tau_1(x_{j-1})/x_{j-1}][\mathtt{true}/x_j]) \\
&= Pr(\mathcal{Q}' : ((\neg x_j \vee I_1) \wedge (x_j \vee I_2) \wedge B \\
&\qquad \wedge \neg\mathcal{S}_{A,B})[\tau_1(x_1)/x_1]\dots[\tau_1(x_{j-1})/x_{j-1}][\mathtt{true}/x_j]) \\
&= Pr(\mathcal{Q}' : (I \wedge B \wedge \neg\mathcal{S}_{A,B})[\tau_1(x_1)/x_1]\dots[\tau_1(x_{j-1})/x_{j-1}][\mathtt{true}/x_j])
\end{aligned}$$

and, in particular,

$$0 = Pr_{B,x} \quad .$$

Finally,

$$\begin{aligned}
0 &= Pr(\mathcal{Q}' : (I_2 \wedge B \wedge \neg\mathcal{S}_{A,B})[\tau_2(x_1)/x_1]\dots[\tau_2(x_{j-1})/x_{j-1}][\mathtt{false}/x_j]) \\
&= Pr(\mathcal{Q}' : ((\neg x_j \vee I_1) \wedge (x_j \vee I_2) \wedge B
\end{aligned}$$

$$\begin{aligned}&\wedge \neg\mathcal{S}_{A,B})[\tau_2(x_1)/x_1]\dots[\tau_2(x_{j-1})/x_{j-1}][\texttt{false}/x_j])\\ &= Pr(\mathcal{Q}' : (I \wedge B \wedge \neg\mathcal{S}_{A,B})[\tau_2(x_1)/x_1]\dots[\tau_2(x_{j-1})/x_{j-1}][\texttt{false}/x_j])\end{aligned}$$

and, in particular,

$$0 = Pr_{B,\neg x} \quad .$$

Having shown that $Pr_{A,x} = Pr_{A,\neg x} = Pr_{B,x} = Pr_{B,\neg x} = 0$, we can now prove the intermediate result above, i.e. $Pr_A = Pr_B = 0$. If $Q_j = \exists$ then $Pr_A = \max(Pr_{A,x}, Pr_{A,\neg x}) = 0$ and $Pr_B = \max(Pr_{B,x}, Pr_{B,\neg x}) = 0$, and if $Q_j = \reflectbox{R}^{p_x}$ then $Pr_A = p_x \cdot Pr_{A,x} + (1-p_x) \cdot Pr_{A,\neg x} = 0$ and $Pr_B = p_x \cdot Pr_{B,x} + (1-p_x) \cdot Pr_{B,\neg x} = 0$.

To finish the proof, we finally need to show that items 2 and 3, i.e.

$$\begin{aligned}&Pr(Q_{i+1}x_{i+1}\dots Q_n x_n : (A \wedge \neg\mathcal{S}_{A,B} \wedge \neg I)[\tau(x_1)/x_1]\dots[\tau(x_i)/x_i]) &= 0 \quad ,\\ &Pr(Q_{i+1}x_{i+1}\dots Q_n x_n : (I \wedge B \wedge \neg\mathcal{S}_{A,B})[\tau(x_1)/x_1]\dots[\tau(x_i)/x_i]) &= 0 \quad ,\end{aligned}$$

follow from $Pr_A = Pr_B = 0$, i.e. from

$$\begin{aligned}&Pr(Q_j x_j\dots Q_n x_n : (A \wedge \neg\mathcal{S}_{A,B} \wedge \neg I)[\tau(x_1)/x_1]\dots[\tau(x_{j-1})/x_{j-1}]) &= 0 \quad ,\\ &Pr(Q_j x_j\dots Q_n x_n : (I \wedge B \wedge \neg\mathcal{S}_{A,B})[\tau(x_1)/x_1]\dots[\tau(x_{j-1})/x_{j-1}]) &= 0 \quad .\end{aligned}$$

If $j = i+1$ then the result is obvious. Otherwise, i.e. if $j > i+1$, the variables $x_{i+1}, \dots, x_{j-1}$ do not occur in the derived clause $(c_1 \vee c_2)$ since $\mathcal{Q}(c_1 \vee c_2) = Q_1 x_1 \dots Q_i x_i$. By definition of assignment τ, for $k = j-1$ down to $i+1$ we may therefore successively conclude that

$$\begin{aligned}&Pr(Q_{k+1}x_{k+1}\dots Q_n x_n :\\ &\qquad (A \wedge \neg\mathcal{S}_{A,B} \wedge \neg I)[\tau(x_1)/x_1]\dots[\tau(x_{k-1})/x_{k-1}][\texttt{true}/x_k]) &= 0 \quad ,\\ &Pr(Q_{k+1}x_{k+1}\dots Q_n x_n :\\ &\qquad (A \wedge \neg\mathcal{S}_{A,B} \wedge \neg I)[\tau(x_1)/x_1]\dots[\tau(x_{k-1})/x_{k-1}][\texttt{false}/x_k]) &= 0 \quad ,\\ &Pr(Q_{k+1}x_{k+1}\dots Q_n x_n :\\ &\qquad (I \wedge B \wedge \neg\mathcal{S}_{A,B})[\tau(x_1)/x_1]\dots[\tau(x_{k-1})/x_{k-1}][\texttt{true}/x_k]) &= 0 \quad ,\\ &Pr(Q_{k+1}x_{k+1}\dots Q_n x_n :\\ &\qquad (I \wedge B \wedge \neg\mathcal{S}_{A,B})[\tau(x_1)/x_1]\dots[\tau(x_{k-1})/x_{k-1}][\texttt{false}/x_k]) &= 0 \quad .\end{aligned}$$

From case $k = i+1$ the result immediately follows. □

Completeness of (strong) S-resolution, as stated in Theorem 6.1, together with above Lemma 9.2, applied to the derived pair $(\emptyset^p, I)$, yields

Corollary 9.1 (Generalized Craig interpolants computation)
If $\mathcal{Q} : (A \wedge B)$ is an SSAT formula with $A \wedge B$ being in CNF then a generalized Craig interpolant for (A, B) can be computed by interpolating S-resolution.

Note that computation of generalized interpolants does not depend on the actual truth state of $A \wedge B$. The next observation facilitates to effectively control the geometric extent of generalized Craig interpolants within the "don't care"-region provided by the models of $\mathcal{S}_{A,B}$. This result is useful within applications of generalized Craig interpolation to the symbolic analysis of probabilistic (finite-state) systems being investigated in Section 9.3.

Corollary 9.2 (Controlling generalized Craig interpolants computation) *If $I = \texttt{true}$ is used within each application of rule* $\mathsf{RI.2}$ *then $Pr(\mathcal{Q} : (A \wedge \neg\mathcal{I})) = 0$. Likewise, if $I = \texttt{false}$ is used in rule* $\mathsf{RI.2}$ *then $Pr(\mathcal{Q} : (\mathcal{I} \wedge B)) = 0$.*

Proof. The proof works analogously to the one of Lemma 9.2. For the base case, it is clear that the desired property for $\mathsf{RI.1}$ is independent of $\neg\mathcal{S}_{A,B}$. For $\mathsf{RI.2}$, if $I = \texttt{true}$ then clearly $Pr(\mathcal{Q}' : (A \wedge \neg I)[\tau(x_1)/x_1] \ldots [\tau(x_i)/x_i]) = 0$, and if $I = \texttt{false}$ then $Pr(\mathcal{Q}' : (I \wedge B)[\tau(x_1)/x_1] \ldots [\tau(x_i)/x_i]) = 0$. Then, we can modify the induction hypothesis: for case "$I = \texttt{true}$ in $\mathsf{RI.2}$", we assume that $Pr(\mathcal{Q}' : (A \wedge \neg I_1)[\tau_1(x_1)/x_1] \ldots [\texttt{true}/x_j]) = 0$, $Pr(\mathcal{Q}' : (A \wedge \neg I_2)[\tau_2(x_1)/x_1] \ldots [\texttt{false}/x_j]) = 0$, and for "$I = \texttt{false}$ in $\mathsf{RI.2}$" that $Pr(\mathcal{Q}' : (I_1 \wedge B)[\tau_1(x_1)/x_1] \ldots [\texttt{true}/x_j]) = 0$, $Pr(\mathcal{Q}' : (I_2 \wedge B)[\tau_2(x_1)/x_1] \ldots [\texttt{false}/x_j]) = 0$. The induction step then follows the same reasoning as in the remaining proof of Lemma 9.2. □

Observe that the special interpolants $\mathcal{I}$ from Corollary 9.2 relate to the classical strongest and weakest Craig interpolants $A^{\exists}$ and $\overline{B}^{\forall}$, respectively, in the following sense: $Pr(\mathcal{Q} : (A \wedge \neg\mathcal{I})) = 0$ if and only if $\models A \Rightarrow \mathcal{I}$ if and only if $\models \forall a_1, \ldots, a_\alpha : (A \Rightarrow \mathcal{I})$ if and only if $\models (A^{\exists} \Rightarrow \mathcal{I})$, as $a_1, \ldots, a_\alpha$ do not occur in $\mathcal{I}$. Analogously, $Pr(\mathcal{Q} : (\mathcal{I} \wedge B)) = 0$ if and only if $\models \mathcal{I} \Rightarrow \neg B$ if and only if $\models \forall b_1, \ldots, b_\beta : (\mathcal{I} \Rightarrow \neg B)$ if and only if $\models \mathcal{I} \Rightarrow \overline{B}^{\forall}$.

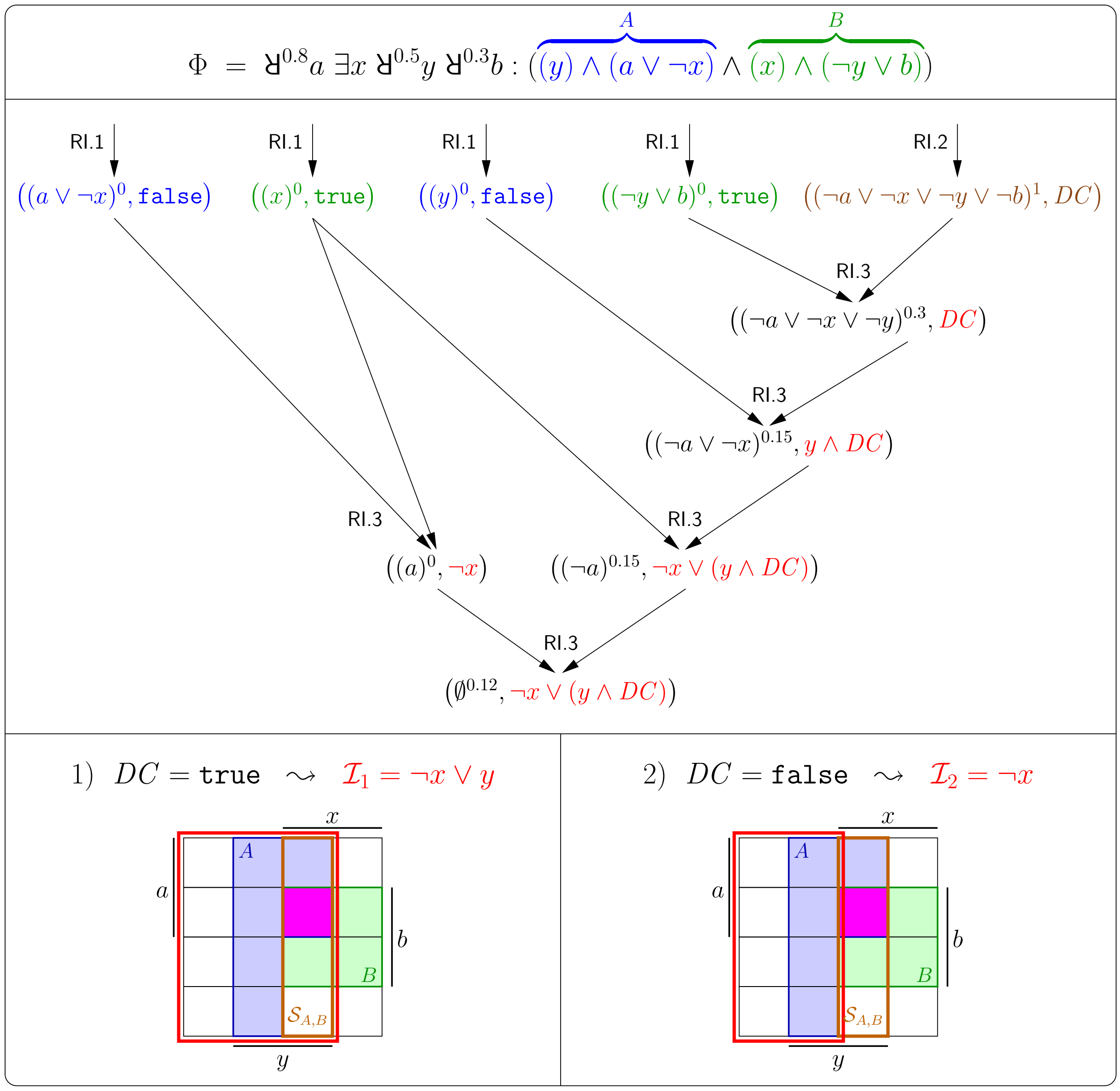
Example of interpolating S-resolution. Consider the SSAT formula

$$\Phi \;=\; \text{Я}^{0.8} a \; \exists x \; \text{Я}^{0.5} y \; \text{Я}^{0.3} b : (A \wedge B)$$

with $A = ((y) \wedge (a \vee \neg x))$ and $B = ((x) \wedge (\neg y \vee b))$. Then, $V_A = \{a\}$, $V_B = \{b\}$, and $V_{A,B} = \{x, y\}$. It is not hard to see that the only satisfying assignment τ of the propositional formula $A \wedge B$ is given by $\tau(a) = \texttt{true}$, $\tau(x) = \texttt{true}$, $\tau(y) = \texttt{true}$, and $\tau(b) = \texttt{true}$. Hence, $Pr(\Phi) = 0.12$. A derivation of the empty clause $\emptyset^{0.12}$ together with its associated generalized Craig interpolant $\neg x \vee (y \wedge DC)$ is shown in Figure 9.3, where DC stands for any formula over variables in $V_{A,B}$ as in rule $\mathsf{RI.2}$. Note that the pair $((\neg a \vee \neg x \vee \neg y \vee \neg b)^1, DC)$ is derivable by rule $\mathsf{RI.2}$ since $\models (A \wedge B)[\tau(a)/a][\tau(x)/x][\tau(y)/y][\tau(b)/b]$. Applying $DC = \texttt{true}$ or $DC = \texttt{false}$, we obtain the generalized Craig interpolants $\mathcal{I}_1 = \neg x \vee y$ or $\mathcal{I}_2 = \neg x$, respectively, such that $Pr(\mathcal{Q} : (A \wedge \neg\mathcal{I}_1)) = 0$ or $Pr(\mathcal{Q} : (\mathcal{I}_2 \wedge B)) = 0$ by Corollary 9.2. In other words, $A \Rightarrow \mathcal{I}_1$ and $\mathcal{I}_2 \Rightarrow \neg B$, as illustrated by the Karnaugh-Veitch diagrams in Figure 9.3.

9.3 Applications to symbolic analysis of probabilistic systems

In this section, we investigate the application of generalized Craig interpolation to the symbolic analysis of probabilistic systems. We direct our attention to two analysis goals, namely to *probabilistic state reachability* in Subsection 9.3.1 as well as to *probabilistic region stability* in Subection 9.3.2. All experiments of this section were performed on a 1.83 GHz Intel Core 2 Duo machine with 1 GByte physical memory running Linux.



$\Phi \;=\; \text{Я}^{0.8}a\ \exists x\ \text{Я}^{0.5}y\ \text{Я}^{0.3}b : (\overbrace{(y) \wedge (a \vee \neg x)}^{A} \wedge \overbrace{(x) \wedge (\neg y \vee b)}^{B})$
RI.1
RI.1
RI.1
RI.1
RI.2
$((a \vee \neg x)^0, \texttt{false})$
$((x)^0, \texttt{true})$
$((y)^0, \texttt{false})$
$((\neg y \vee b)^0, \texttt{true})$
$((\neg a \vee \neg x \vee \neg y \vee \neg b)^1, DC)$
RI.3
$((\neg a \vee \neg x \vee \neg y)^{0.3}, DC)$
RI.3
$((\neg a \vee \neg x)^{0.15}, y \wedge DC)$
RI.3
RI.3
$((a)^0, \neg x)$
$((\neg a)^{0.15}, \neg x \vee (y \wedge DC))$
RI.3
$(\emptyset^{0.12}, \neg x \vee (y \wedge DC))$
1) $DC = \texttt{true} \rightsquigarrow \mathcal{I}_1 = \neg x \vee y$
2) $DC = \texttt{false} \rightsquigarrow \mathcal{I}_2 = \neg x$
x
a
A
b
B
$\mathcal{S}_{A,B}$
y
x
a
A
b
B
$\mathcal{S}_{A,B}$
y

Figure 9.3: Example of interpolating S-resolution and illustration of the resulting generalized Craig interpolants by means of Karnaugh-Veitch diagrams. Arrows denote applications of the specified interpolating S-resolution rules, while DC stands for any formula over $V_{A,B}$ as in rule RI.2. (Source of figure: [TF12])

Markov decision processes. As a system model, we consider finite-state Markov decision processes (MDPs) [Bel57]. An MDP $\mathcal{M} = (\imath, S, Act, ps(\cdot,\cdot,\cdot))$ is a finite-state system in which state changes are subject to *non-deterministic* selection among available actions followed by a *probabilistic* choice among potential successor states, with the probability distribution of the latter choice depending on the selected action. More precisely, S is a finite set of states, $\imath \in S$ is the initial state, Act is a finite set of actions, and $ps(s, a, s')$ gives the probability that $\mathcal{M}$ performs a transition step from $s \in S$ to $s' \in S$ under action $a \in Act$. For an example, consider the simple MDP $\mathcal{M}$ from Figure 9.4 where $\imath = i$, $S = \{i, f, e, s\}$, and $Act = \{a, b\}$. A transition $ps(z, act, z') = p > 0$ is indicated by an

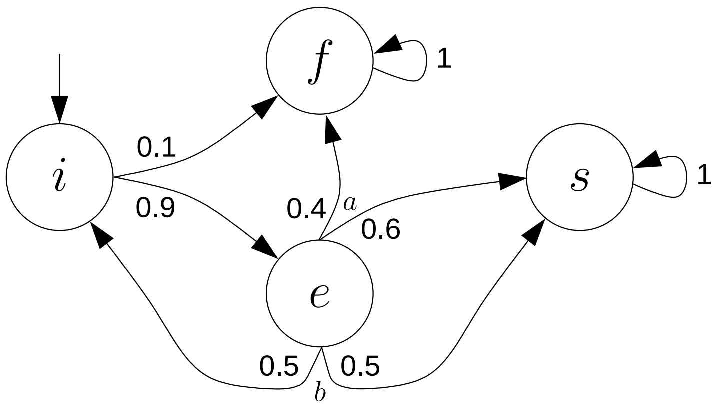


Figure 9.4: A simple MDP $\mathcal{M}$. (Source of figure: [TF12])

arrow from z to z' accompanied by action act and by the corresponding transition probability p. If two states are not connected by an arrow then the corresponding transition probability is 0, and if no action is specified then that transition is feasible for all actions.

A probability measure of an MDP is well-defined only if considering a particular scheduler σ resolving the non-determinism. That is, σ schedules the action for the current state. Different such schedulers σ have been investigated in the literature, confer, for instance, [BHKH05]: σ may select the next action either in a deterministic or randomized fashion. In both cases, σ may have access to and thus base its selection on either the current state only or the full system history.

In the scenarios of this section, we do not manipulate schedulers explicitly but define the probability measures obtained by worst-case deterministic schedulers achieving maximum or minimum, depending on how the worst case is understood, probability of reaching target states directly as the limit of a recursive function over $\mathbb{N}$. For each $k \in \mathbb{N}$, the recursive function determines the maximum or minimum probability of reaching the target states within k steps, as achieved by a worst-case history-dependent scheduler. As such a latter scheduler will always maximize or minimize the probability of reaching the target within the remaining number of steps, its performance coincides with the probabilities computed by a backward induction resolving non-deterministic choices by taking the maximum or minimum, respectively, of the probability values obtained from the next-lower recursion depth.

SSAT encoding scheme for MDPs. The generalized Craig interpolation-based analysis approaches rest upon an SSAT encoding of the MDP to be analyzed, which is similar in nature to the SSMT encoding of a probabilistic hybrid system from Section 5.3. More precisely, the initial state and the transition relation of the given MDP $\mathcal{M}$ are described by propositional formulae in CNF, namely by $INIT_{\mathcal{M}}(\boldsymbol{s})$ and $TRANS_{\mathcal{M}}(\boldsymbol{s}, \boldsymbol{nc}, \boldsymbol{pc}, \boldsymbol{s}')$, respectively, where the propositional variable vector $\boldsymbol{s}$ represents the system state before and $\boldsymbol{s}'$ after a transition step, and the propositional variable vectors $\boldsymbol{nc}$ and $\boldsymbol{pc}$ encode non-deterministic selection among available actions and probabilistic choice of the successor state, respectively. Assignments to these variables determine which of possibly multiple available transitions departing from $\boldsymbol{s}$ is taken. Akin to the SSMT encoding scheme for probabilistic hybrid systems from Section 5.3, the variables $\boldsymbol{nc}$ and $\boldsymbol{pc}$, encoding non-deterministic and probabilistic choices, are bound by existential and randomized quantifiers in the prefixes $\mathcal{Q}_{\boldsymbol{nc}}$ and $\mathcal{Q}_{\boldsymbol{pc}}$, respectively, while the state variables $\boldsymbol{s}$ and $\boldsymbol{s}'$ are existentially quantified by prefixes $\mathcal{Q}_{\boldsymbol{s}}$ and $\mathcal{Q}_{\boldsymbol{s}'}$, respectively. Note that the state variables $\boldsymbol{s}$ and $\boldsymbol{s}'$ are quantified explicitly. This is in contrast to the hybrid-state case where the continuous-domain state variables in the SSMT encoding are interpreted as innermost existentially quantified due to the lack of existential quantification over continuous variables. As a consequence, the explicit (existential) quantification of state variables renders unnecessary the restriction of post-states being uniquely determined by the non-deterministic and probabilistic choices, as in Definition 3.1 for probabilistic hybrid automata. This particularly shows an advantage for probabilistic finite-state models featuring (finite-domain) data variables. For the sake of clarity, let be $\boldsymbol{t} := \boldsymbol{nc} \cup \boldsymbol{pc}$ and $\mathcal{Q}_{\boldsymbol{t}} := \mathcal{Q}_{\boldsymbol{nc}}\mathcal{Q}_{\boldsymbol{pc}}$. The branching structure of the MDP $\mathcal{M}$ for k transition steps is then reflected by the quantifier prefix

$$\mathit{CHOICE}_{\mathcal{M}}(k) \quad := \quad \mathcal{Q}_{\boldsymbol{s}_0}\mathcal{Q}_{\boldsymbol{t}_1}\mathcal{Q}_{\boldsymbol{s}_1}\dots\mathcal{Q}_{\boldsymbol{s}_{k-1}}\mathcal{Q}_{\boldsymbol{t}_k}\mathcal{Q}_{\boldsymbol{s}_k}$$

where the variable vectors $\boldsymbol{s}_i$ and $\boldsymbol{t}_i$ encode the system state at step depth i and transition selection of step i, respectively.

A technicality worth mentioning is the representation of branching over n alternatives by quantified propositional variables. For non-deterministic branching, the n alternatives can be encoded by a binary tree of depth $\lceil \log_2 n \rceil$ and thus by $\lceil \log_2 n \rceil$ existential quantifiers. For instance, the choice between the six actions $act_1, \dots, act_6$ can be represented by three existential variables nc_1, nc_2, and nc_3, the latter permitting to distinguish between even eight different cases. For probabilistic branching, a sequence of at most $n-1$ binary branches is necessary, which results in at most $n-1$ randomized quantifiers. For instance, the probabilistic choice between the three alternatives pa_1, pa_2, and pa_3 associated with probabilities 0.2, 0.6, and 0.2, respectively, can be described by two randomized variables $\text{Я}^{0.2} pc_1$ and $\text{Я}^{0.75} pc_2$. Then, probabilistic alternative pa_1 is encoded by the assignments τ_1 to the variables pc_1 and pc_2 such that $\tau_1(pc_1) = \texttt{true}$. The probability of such assignments clearly is 0.2. The alternatives pa_2 and pa_3 are represented by the assignments τ_2 and τ_3, respectively, such that $\tau_2(pc_1) = \tau_3(pc_1) = \texttt{false}$, $\tau_2(pc_2) = \texttt{true}$, and $\tau_3(pc_2) = \texttt{false}$. The probabilities of τ_2 and τ_3 are $0.8 \cdot 0.75 = 0.6$ and $0.8 \cdot 0.25 = 0.2$, respectively.

Example of the symbolic encoding. We illustrate the SSAT encoding scheme by means of the simple MDP $\mathcal{M}$ from Figure 9.4. The state space of $\mathcal{M}$, consisting of the four states i, f, e, and s, is encoded by four Boolean variables i, f, e, and s, sharing their names with the systems states, as follows: if variable i carries truth value `true` then $\mathcal{M}$ is in state i and otherwise, i.e. if variable i is assigned value `false`, $\mathcal{M}$ is not in state i. The same holds analogously for the other states. In order to encode valid system states, note that we have to ensure that *exactly one* of the variables i, f, e, and s is `true` at each instant of time. The encoding of this constraint is explained later on. We remark that the assignments of truth values `true` and `false` to a propositional variable x are encoded symbolically by the positive literal x and the negative literal $\neg x$, respectively.

The non-deterministic choice between actions a and b is encoded by a Boolean variable act while action a is represented by the positive literal act and action b by the negative literal $\neg act$. For the three probabilistic choices present in $\mathcal{M}$, we introduce three Boolean variables, namely pi for the choice from state i, pea for the choice from state e under action a, and peb for the choice from state e under action b. The positive literal pi stands for successor state e, while the negative literal $\neg pi$ means a state change to f. Furthermore, literals pea, $\neg pea$, peb, and $\neg peb$ encode the transitions to states s, f, i, and s, respectively. We thus obtain the following quantifier prefixes

$$\begin{aligned}
\mathcal{Q}_{\boldsymbol{s}} &= \exists i \; \exists f \; \exists e \; \exists s \quad , \\
\mathcal{Q}_{\boldsymbol{t}} &= \exists act \; \text{ꓤ}^{0.9} pi \; \text{ꓤ}^{0.6} pea \; \text{ꓤ}^{0.5} peb \quad , \\
\mathcal{Q}_{\boldsymbol{s}'} &= \exists i' \; \exists f' \; \exists e' \; \exists s' \quad .
\end{aligned}$$

The formula in CNF representing the initial state i of $\mathcal{M}$ is specified by

$$INIT_{\mathcal{M}}(\boldsymbol{s}) \;=\; (i) \wedge (\neg f) \wedge (\neg e) \wedge (\neg s) \quad .$$

To obtain the transition relation predicate, we encode each single transition step. For instance, a step from state e to state f under action a can be encoded by the implication $(e \wedge act \wedge \neg pea) \Rightarrow f'$, which is equivalent to the clause $(\neg e \vee \neg act \vee pea \vee f')$. The conjunction of all these clauses then reflects the full system behavior in a symbolic manner. Since we represent each system state by an own Boolean variable, as mentioned above, we need to enforce that exactly one of the primed state variables, constituting the system state after the transition step has taken place, carries value `true`. This is simply achieved by the formula $exactly\_one(i', f', e', s') = (i' \vee f' \vee e' \vee s') \wedge (\neg i' \vee \neg f') \wedge (\neg i' \vee \neg e') \wedge (\neg i' \vee \neg s') \wedge (\neg f' \vee \neg e') \wedge (\neg f' \vee \neg s') \wedge (\neg e' \vee \neg s')$ in CNF. The transition relation predicate in CNF then is

$$\begin{aligned}
TRANS_{\mathcal{M}}(\boldsymbol{s}, \boldsymbol{t}, \boldsymbol{s}') \;=\; & \quad (\neg i \vee pi \vee f') \;\wedge\; (\neg i \vee \neg pi \vee e') \\
& \wedge \; (\neg e \vee \neg act \vee pea \vee f') \;\wedge\; (\neg e \vee \neg act \vee \neg pea \vee s') \\
& \wedge \; (\neg e \vee act \vee peb \vee s') \;\wedge\; (\neg e \vee act \vee \neg peb \vee i') \\
& \wedge \; (\neg f \vee f') \;\wedge\; (\neg s \vee s') \;\wedge\; exactly\_one(i', f', e', s') \quad .
\end{aligned}$$

9.3.1 Interpolation-based probabilistic state reachability

As an application of generalized Craig interpolation, we first devote our attention to *probabilistic state reachability*. Let be given an MDP $\mathcal{M}$ and a set of target states $Target \subseteq S$ in $\mathcal{M}$. The goal then is to compute the probability of reaching the target states $Target$ from the initial state $\imath$ under some explicitly or implicitly (for instance, by an optimality condition) given scheduler σ. In most scenarios, the target states are considered to be *bad*, for instance, to be fatal system errors, such that one has to deal with computing the *worst-case* probability of reaching the bad states, i.e. *maximizing* the reachability probability under each possible scheduler. This maximum probability $MaxReach(\mathcal{M}, Target)$ can be defined directly as the limit of the maximum step-bounded probability of reaching the target states from the initial state $\imath$, as similarly shown in [FHH+11, Lemma 1], i.e.

$$MaxReach(\mathcal{M}, Target) \;:=\; \lim_{k \to \infty} MaxReach^{k}_{\mathcal{M}, Target}(\imath)$$

where

$$MaxReach^{k}_{\mathcal{M},\,Target}(s) := \begin{cases} 1 & \text{if } s \in Target \;\;, \\ 0 & \text{if } s \notin Target \text{ and } k = 0 \;\;, \\ \max\limits_{a \in Act} \sum\limits_{s' \in S} ps(s,a,s') \cdot MaxReach^{k-1}_{\mathcal{M},\,Target}(s') & \\ & \text{if } s \notin Target \text{ and } k > 0 \end{cases}$$

gives the maximum probability of reaching the target states from state $s \in S$ within $k \in \mathbb{N}$ steps under each possible scheduler. For some threshold value $\theta \in [0,1]$, the *safety verification problem* is to decide whether the worst-case probability of reaching the bad states is at most θ, i.e. to decide whether

$$MaxReach(\mathcal{M},\, Target) \leq \theta \tag{9.1}$$

holds.

The approach of Chapter 5 establishes a symbolic *falsification* procedure for above problem 9.1. Though this approach is based on SSMT, i.e. the arithmetic extension of SSAT, and works for the more general class of discrete-time probabilistic hybrid systems, which roughly are MDPs with arithmetic-logical transition guards and assignments, the same procedure restricted to SSAT is applicable to finite-state MDPs. More precisely, the values $lb_k = MaxReach^{k}_{\mathcal{M},\,Target}(\imath)$ can be determined, first, by exploiting the above SSAT encoding scheme for MDPs to obtain SSAT formulae akin to the SSMT formulae from reduction step 14 of Section 5.3 and, second, by solving these SSAT formulae using an SSAT algorithm. Observe that each value lb_k constitutes a lower bound of the maximum reachability probability $MaxReach(\mathcal{M},\, Target)$ due to monotonicity of the chain $\big(MaxReach^{k}_{\mathcal{M},\,Target}(\imath)\big)_{k \in \mathbb{N}}$. This probabilistic bounded model checking (PBMC) approach is then able to falsify safety properties of shape 9.1 once a value $lb_k > \theta$ is computed for some step depth k.

In the remainder of this subsection, we develop a corresponding counterpart based on generalized Craig interpolation for SSAT that is able to compute *upper* bounds ub_k of the maximum reachability probability $MaxReach(\mathcal{M},\, Target)$. This symbolic *verification* procedure, permitting to *verify* safety properties of shape 9.1 once an upper bound $ub_k \leq \theta$ is computed for some k, proceeds in two phases. Phase 1 determines a symbolic representation of an *overapproximation of the backward reachable state set*, where a state is backward reachable if it is the origin of a transition sequence leading to the target states *Target*. Phase 1 can be integrated into PBMC, as used to falsify the probabilistic safety property. Whenever such falsification fails for a given step depth k starting at depth 0, we apply generalized Craig interpolation to the just failed PBMC proof in order to compute a symbolic overapproximation of the backward reachable state set at depth k and then proceed to PBMC at depth $k+1$. As an alternative to the integration into PBMC, interpolants describing the backward reachable state sets can be successively extended by means of "stepping" them by prepending another transition, as explained below. In either case, phase 1 ends when the backward reachable state set becomes stable, i.e. no new backward reachable state is found, in which case we have computed a symbolic overapproximation of the whole backward reachable state set. In phase 2, we then construct a family of SSAT formulae with parameter k that forces the system to *stay within the backward reachable state set* for k steps. The maximum probabilities of satisfaction of these SSAT formulae then give upper bounds on the maximum probability of reaching the target states. The rationale is that system runs leaving the backward reachable state set will never reach the target states.

Phase 1 (Symbolic overapproximation of backward reachable states). Let be given an SSAT encoding of the MDP $\mathcal{M}$ as above and let $\mathit{Target}(\boldsymbol{s})$ be a predicate in CNF that encodes the target states *Target*. Then, the state-set predicate $\mathcal{B}^k(\boldsymbol{s})$ for $k \in \mathbb{N}$ over state variables $\boldsymbol{s}$ is inductively defined as

- $\mathcal{B}^0(\boldsymbol{s}) \;:=\; \mathit{Target}(\boldsymbol{s})$, and
- $\mathcal{B}^{k+1}(\boldsymbol{s}) \;:=\; \mathcal{B}^k(\boldsymbol{s}) \;\vee\; \mathcal{I}^{k+1}(\boldsymbol{s})$

where $\mathcal{I}^{k+1}(\boldsymbol{s}_{j-1})$ is a generalized Craig interpolant for

$$\left(\overbrace{\mathit{TRANS}_{\mathcal{M}}(\boldsymbol{s}_{j-1}, \boldsymbol{t}_j, \boldsymbol{s}_j) \wedge \mathcal{B}^k(\boldsymbol{s}_j)}^{=A},\; \overbrace{\mathit{INIT}_{\mathcal{M}}(\boldsymbol{s}_0) \wedge \bigwedge_{i=1}^{j-1} \mathit{TRANS}_{\mathcal{M}}(\boldsymbol{s}_{i-1}, \boldsymbol{t}_i, \boldsymbol{s}_i)}^{=B} \right)$$

with $j \geq 1$ with respect to SSAT formula

$$(9.2) \qquad \mathit{CHOICE}_{\mathcal{M}}(j) : \left(\begin{array}{l} \overbrace{\mathit{INIT}_{\mathcal{M}}(\boldsymbol{s}_0) \wedge \bigwedge_{i=1}^{j-1} \mathit{TRANS}_{\mathcal{M}}(\boldsymbol{s}_{i-1}, \boldsymbol{t}_i, \boldsymbol{s}_i)}^{j-1 \text{ steps “forward” } (=B)} \\ \wedge \underbrace{\mathit{TRANS}_{\mathcal{M}}(\boldsymbol{s}_{j-1}, \boldsymbol{t}_j, \boldsymbol{s}_j) \wedge \mathcal{B}^k(\boldsymbol{s}_j)}_{\text{one step “backward” } (=A)} \end{array} \right) .$$

Observe that each generalized Craig interpolant $\mathcal{I}^{k+1}(\boldsymbol{s}_{j-1})$ can be computed by interpolating S-resolution if we rewrite $\mathcal{B}^k(\boldsymbol{s}_j)$ into CNF, the latter being always possible in linear time using the Tseitin transformation [Tse68], which potentially adds auxiliary V_A-variables. During computation of $\mathcal{I}^{k+1}(\boldsymbol{s}_{j-1})$, we take $I = \texttt{true}$ in every application of rule RI.2 such that $\mathcal{B}^k(\boldsymbol{s})$ overapproximates all system states which are backward reachable from the target states within k steps as per Corollary 9.2. Whenever the computation of $\mathcal{B}^k(\boldsymbol{s})$ has reached a fixed point, i.e. if

$$\mathcal{B}^{k+1}(\boldsymbol{s}) \;\Rightarrow\; \mathcal{B}^k(\boldsymbol{s})$$

holds for some k, it follows that $\mathcal{B}(\boldsymbol{s}) := \mathcal{B}^k(\boldsymbol{s})$ overapproximates all backward reachable states. It is obvious that such a fixed point is finally reached in the finite-state case.

Note that parameter $j \geq 1$ can be chosen arbitrarily, i.e. the system may execute any number of transitions until state $\boldsymbol{s}_{j-1}$ is reached since this does not destroy the “backward-overapproximating” property of $\mathcal{B}^{k+1}(\boldsymbol{s})$. The rationale of having parameter j is the additional freedom in constructing generalized interpolants since j may influence the shape of $\mathcal{I}^{k+1}(\boldsymbol{s})$, as we see in the proof of concept below.

Phase 2 (Upper bounds on maximum reachability probability). After having described symbolically all backward reachable states by the predicate $\mathcal{B}(\boldsymbol{s})$, *upper* bounds ub_k on the maximum probability $\mathit{MaxReach}(\mathcal{M}, \mathit{Target})$ of reaching the target states *Target* can now be determined by SSAT solving, more precisely, by computing

$$(9.3)\quad ub_k \;:=\; Pr\left(\mathit{CHOICE}_{\mathcal{M}}(k) : \left(\begin{array}{ll} \multicolumn{2}{c}{\overbrace{\mathit{INIT}_{\mathcal{M}}(\boldsymbol{s}_0) \wedge \bigwedge_{i=1}^{k} \mathit{TRANS}_{\mathcal{M}}(\boldsymbol{s}_{i-1}, \boldsymbol{t}_i, \boldsymbol{s}_i)}^{\text{states reachable within } k \text{ steps}}} \\ \wedge & \underbrace{\bigwedge_{i=0}^{k} \mathcal{B}(\boldsymbol{s}_i)}_{\text{stay within backward reachable state set}} \end{array} \right)\right).$$

First observe that the formula above excludes all system runs that leave the set of backward reachable states. This is sound since leaving $\mathcal{B}(\boldsymbol{s})$ means to never reach the target states. Second, the system behavior becomes more and more constrained for increasing k, i.e. the ub_k's are monotonically decreasing. With regard to solving the safety verification problem 9.1, the safety property $\mathit{MaxReach}(\mathcal{M}, \mathit{Target}) \leq \theta$ is *verified* by the procedure above once an upper bound $ub_k \leq \theta$ is computed for some k.

Proof of concept. To demonstrate feasibility of the symbolic approach to probabilistic safety verification based on generalized Craig interpolation, consider the simple MDP $\mathcal{M}$ from Figure 9.4 and let state s be the only target state, i.e. $\mathit{Target}(\boldsymbol{s}) = (\neg i) \wedge (\neg f) \wedge (\neg e) \wedge (s)$. In the following, we refer to the symbolic SSAT encoding of $\mathcal{M}$ as explained above.

Application of probabilistic bounded model checking, as introduced in Chapter 5, yields lower bounds lb_k on the maximum probability of reaching target state s, for instance, $lb_0 = lb_1 = 0$, $lb_2 = lb_3 = 0.54$, $lb_4 = lb_5 = 0.693$, …, $lb_{20} = 0.817971$, …, and $lb_{100} = 0.81818181818181803208$. These results were achieved by employing the SSMT solver SiSAT, confer Section 6.6. Concerning solving time, all 100 SSAT formulae were solved within 37.05 seconds, while computation of the first 20 lower bounds lb_0 to lb_{20} just needed 370 milliseconds. The highest computation time for a single SSAT problem was obtained for lb_{100}, namely 1.14 seconds. The evolution of the lb_k's up to step depth $k = 20$ is presented graphically on the right of Figure 9.6. Given these results, one might suppose that the lower bounds converge to and never exceed value $9/11 = 0.\overline{81}$.

In order to substantiate the above guess, we apply the generalized Craig interpolation-based approach suggested in this subsection. We therefore first compute an overapproximation of the backward reachable state set using the generalized Craig interpolation scheme 9.2. Thereafter, upper bounds ub_k on the maximum reachability probability are determined by means of scheme 9.3. In order to compute the generalized Craig interpolants $\mathcal{I}^{k+1}(\boldsymbol{s}_{j-1})$ automatically during solving the SSAT formulae 9.2, we implemented a simple DPLL-based SSAT solver that integrates interpolating S-resolution. As mentioned earlier, scheme 9.2 leaves freedom in choosing parameter $j \geq 1$. This parameter permits to specify the number $j-1$ of transition steps until system state $\boldsymbol{s}_{j-1}$ is reached, which is the common state of formula parts A and B. The experimental results of applying the generalized Craig interpolation scheme 9.2 on the MDP $\mathcal{M}$ for different values

j	$\mathcal{I}^1$	$\mathcal{B}^1$	$\mathcal{I}^2$	$\mathcal{B}^2$	$\mathcal{I}^3$	$\mathcal{B}^3$	$\mathcal{B}$
1	$\neg i$	$\neg i \vee s$	`true`	`true`	`true`	`true`	`true`
	$\{f, e, s\}$	$\{f, e, s\}$	$\{i, f, e, s\}$	$\{i, f, e, s\}$	$\{i, f, e, s\}$	$\{i, f, e, s\}$	$\{i, f, e, s\}$
2	$\neg f$	$\neg f \vee s$	$\neg f$	$\neg f \vee s$	—	—	$\neg f \vee s$
	$\{i, e, s\}$	$\{i, e, s\}$	$\{i, e, s\}$	$\{i, e, s\}$	—	—	$\{i, e, s\}$
3	$\neg i \wedge \neg f$	$\neg i \wedge \neg f \vee s$	$\neg f$	$\neg f \vee s$	$\neg f$	$\neg f \vee s$	$\neg f \vee s$
	$\{e, s\}$	$\{e, s\}$	$\{i, e, s\}$	$\{i, e, s\}$	$\{i, e, s\}$	$\{i, e, s\}$	$\{i, e, s\}$

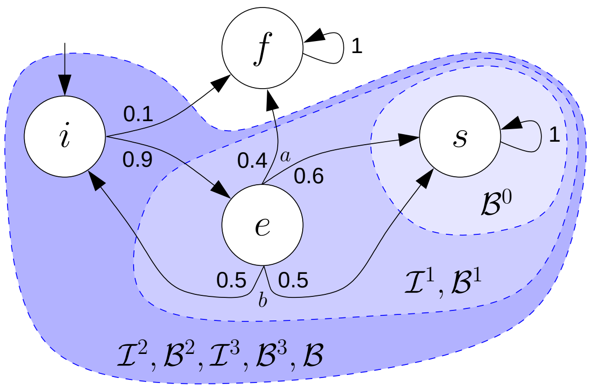
Figure 9.5: Probabilistic state reachability analysis of MDP $\mathcal{M}$: experimental results of applying the generalized Craig interpolation scheme 9.2 for $\mathcal{M}$ from Figure 9.4 for different values of parameter j. In addition to the formal presentation of the predicates, the concrete state sets are given explicitly. (Source of figure: [TF12])

of j are shown in Figure 9.5. With regard to the obtained state-set predicates P, it is important to remark that a state z lies in the state set encoded by P if and only if there exists a solution τ of P such that $\tau(z) = \texttt{true}$ and $\tau(z') = \texttt{false}$ for all $z' \neq z$.

From the results of Figure 9.5, we observe that the value of j actually has an impact on the shape of the resulting interpolants. Let us consider the first interpolants $\mathcal{I}^1$ which overapproximate all states backward reachable in one step. Clearly, the exact set of states backward reachable in one step is $\{e, s\}$. For $j = 1$, the overapproximated set $\{f, e, s\}$ computed by the procedure is too coarse and actually contains a state which is *not* backward reachable at all, namely f. Though the set $\{i, e, s\}$ for $j = 2$ actually consists of backward reachable states only, it is not tight enough as the initial state i is backward reachable only after two steps. For $j = 3$, we achieved the precise set $\{e, s\}$. Continuing the scheme for $j = 1$, $\mathcal{I}^2$ and then $\mathcal{I}^3$ became `true` meaning that the overapproximated set $\mathcal{B}$ of backward reachable states covers the whole state space. Using this inconclusive result in scheme 9.3 would only yield trivial upper bounds $ub_k = 1$ for all k. With regard to $j = 2$, the interpolation process reached a fixed point after computation of $\mathcal{I}^2$. The resulting state set $\{i, e, s\}$ encoded by $\mathcal{B}$ actually is the precise set of all backward reachable states. Though $\mathcal{I}^1$ was too coarse, this could be compensated in the computation of $\mathcal{I}^2$. For $j = 3$, we observe that all generalized Craig interpolants $\mathcal{I}^1$, $\mathcal{I}^2$, and $\mathcal{I}^3$ describe the corresponding backward reachable states accurately, thus leading to the precise set of all backward reachable states. The computed state sets for $j = 3$ are illustrated on the left of Figure 9.6. After having examined the results above, it seems that the greater value of j, i.e. the more transition steps performed, the more accurate the overapproximation of the backward reachable state set.

Concerning runtime, each generalized Craig interpolant was computed by the interpolating DPLL-based SSAT solver within fractions of a second, where the highest runtime of 36 milliseconds was observed when computing $\mathcal{I}^3$ for $j = 3$.

After having determined a symbolic representation $\mathcal{B}(\boldsymbol{s})$ of an overapproximation of all backward reachable states, we computed upper bounds ub_k on the maximum reachability probability by means of scheme 9.3 for $\mathcal{B}(\boldsymbol{s}) = \neg f \vee s$, as obtained for $j = 3$ as well as for $j = 2$, again having employed the SSMT tool SiSAT. Some of the results are $ub_0 = 1$, $ub_1 = ub_2 = 0.9$, $ub_3 = ub_4 = 0.855$, $ub_5 = ub_6 = 0.83475$, ..., $ub_{20} = 0.818243$, ...,



f
1
0.1
i
0.9
0.4
a
0.6
s
1
e
$\mathcal{B}^0$
0.5
0.5
b
$\mathcal{I}^1, \mathcal{B}^1$
$\mathcal{I}^2, \mathcal{B}^2, \mathcal{I}^3, \mathcal{B}^3, \mathcal{B}$

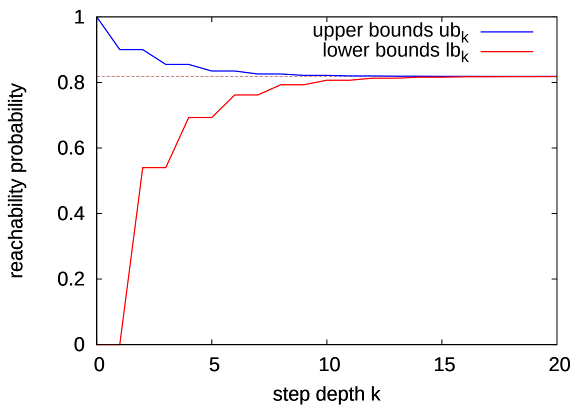


Figure 9.6: Probabilistic state reachability analysis of MDP $\mathcal{M}$: illustration of the computed state sets for $\mathcal{M}$ by the generalized Craig interpolation scheme 9.2 with $j = 3$ (left), and lower bounds lb_k and upper bounds ub_k on the maximum probability of reaching target state s over step depth k computed by PBMC and by scheme 9.3, respectively (right). (Source of figure: [TF12])

and $ub_{100} = 0.8181818181818181821948$. Concerning runtime, all 100 SSAT formulae were solved within 54.76 seconds, while computation of the first 20 upper bounds ub_0 to ub_{20} just needed 400 milliseconds. The highest computation time for a single SSAT problem was obtained for ub_{100}, namely 1.77 seconds. The evolution of the ub_k's up to depth $k = 20$ is presented graphically on the right of Figure 9.6.

In addition to estimating the maximum reachability probability from below using PBMC, generalized Craig interpolation-based probabilistic reachability analysis facilitates to estimate the probability also from above. In our example, we can safely conclude that

$$\begin{array}{rcll} 0.8181818181818181803208 & = & lb_{100} & \\ & \leq & \mathit{MaxReach}(\mathcal{M}, \{s\}) & \\ & \leq & ub_{100} & = \ 0.8181818181818181821948 \end{array}$$

holds where the difference $ub_{100} - lb_{100}$ is below 10^{-15}. The total computational effort for obtaining this precise result is about 92 seconds. If reduced accuracy suffices then runtime obviously improves. For instance, the fact that

$$0.817971 \ = \ lb_{20} \ \leq \ \mathit{MaxReach}(\mathcal{M}, \{s\}) \ \leq \ ub_{20} \ = \ 0.818243$$

is satisfied where $ub_{20} - lb_{20} < 10^{-3}$ was deduced within one second. With regard to the safety verification problem 9.1, system safety for each threshold value θ with $\theta < 0.817971$ or $\theta \geq 0.818243$ is *falsified* or *verified*, respectively, within a second.

With respect to competitive and more established methods based on value or policy iteration, we observed that the runtime of our prototypic tool chain does not compare favorably on the simple probabilistic reachability problem above. For instance, the version 4.0.1 of the PRISM model checker[2] [KNP11] solved the problem in about 600 milliseconds with a precision of 10^{-15}, returning the result 0.8181818181818175.

[2]More information can be found on `http://www.prismmodelchecker.org`.

In spite of the above fact, we have identified two promising directions for future research where probabilistic reachability analysis based on generalized Craig interpolation may pay off:

1. Embedding the same interpolation process into SSMT, i.e. the arithmetic extension of SSAT, renders the generalized Craig interpolation scheme 9.2 *directly* applicable to probabilistic *hybrid* automata, as introduced in Section 3.3, yielding a symbolic overapproximation of the backward reachable state set. As for the finite-state case, scheme 9.3 then facilitates computing upper bounds on the maximum reachability probability for probabilistic hybrid systems by means of SSMT solving. This would establish a symbolic *verification* procedure for probabilistic hybrid systems.

   It is important to remark that classical value or policy iteration procedures are *not* directly applicable in the hybrid state case, but require a finite-state abstraction, confer, for instance, [ZSR^{+}10, FHH^{+}11].

   The concept of a generalized Craig interpolant is smoothly adaptable to the more general, non-linear arithmetic SSMT case, while such generalized interpolants then need not exist in general. One approach to such an extension to SSMT might build upon the work by Kupferschmid et al. [KB11a, KB11b], which has enhanced a resolution calculus for non-linear arithmetic SMT formulae over the reals and integers [KTBF09, KBTF11] by construction rules to compute (classical) Craig interpolants for such SMT formulae.

2. Due to its *symbolic* nature, the analysis procedures based on SSAT and SSMT support *compact* representations of *concurrent* probabilistic (finite-state and hybrid) systems without an explicit construction of the product automaton, the latter being of size exponential in the number of parallel components. This fact constitutes a strong argument that these symbolic procedures may be able to alleviate the state explosion problem, which arises necessarily when applying explicit-state algorithms or methods based on finite-state abstraction refinement, and thus provide a better scalability.

9.3.2 Interpolation-based probabilistic region stability

In addition to probabilistic state reachability being investigated in the previous subsection, we now address the problem of *probabilistic region stability*. For that purpose, we take into account the notion of *region stability*, as introduced for non-probabilistic hybrid systems by Podelski and Wagner in [PW07a, PW07b]. According to their definition, given some set R of states called *region*, a (non-probabilistic) system is *stable with respect to region R* if and only if for every infinite run $\langle s_0, s_1, \ldots, s_i, \ldots \rangle$ of the system, i.e. for every infinite sequence of states that follows the transition relation, there is some point in time $i \geq 0$ such that from i on the system remains in R forever, i.e. $\exists i \geq 0\ \forall j \geq i : s_j \in R$.

Concerning the probabilistic case, several adaptations of region stability seem feasible, some of which pose measurability problems. Our main concern in this section is to identify potential application areas of generalized Craig interpolation rather than to discuss semantic issues of probabilistic stability. We therefore study a simple notion of probabilistic

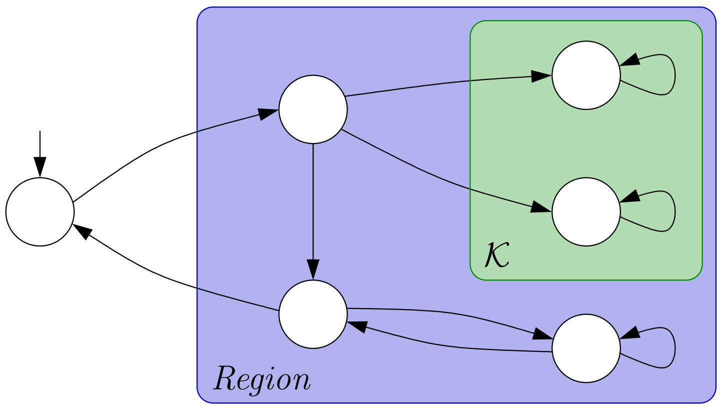


Figure 9.7: Illustration of the maximal invariance kernel $\mathcal{K}$ within the stabilization region *Region*.

region stability in the sequel which circumvents measure-theoretic issues. As for probabilistic state reachability, we aim at defining a reasonable probability measure as the limit of the value of a recursive function defining the corresponding step-bounded measures. Intuitively, we consider finite run prefixes $\langle s_0, s_1, \ldots, s_i\rangle$ such that from time point i on the probabilistic system remains in the given region forever under each possible future behavior, i.e. independent of the non-deterministic and probabilistic choices the system will take. The latter fact is guaranteed whenever the system has reached an invariance kernel of the given region that can never be left. The probability measure is then defined by the minimum probability of reaching the maximal invariance kernel.

Formally, given an MDP $\mathcal{M}$ and a set of states $\mathit{Region} \subseteq S$ called the *stabilization region* or the *region* for short, an *invariance kernel* $\mathcal{K} \subseteq \mathit{Region}$ with respect to $\mathcal{M}$ is a set of states from *Region* such that there is no transition from a state in $\mathcal{K}$ to a state outside $\mathcal{K}$, i.e. there does not exist a tuple $(z, \mathit{act}, z') \in \mathcal{K} \times \mathit{Act} \times (S \setminus \mathcal{K})$ such that $\mathit{ps}(z, \mathit{act}, z') > 0$. An invariance kernel $\mathcal{K}$ is called *maximal* if adding any new states to $\mathcal{K}$ does not lead to an invariance kernel, i.e. each $\mathcal{K} \cup Z$ with $Z \subseteq \mathit{Region} \setminus \mathcal{K}$ and $Z \neq \emptyset$ is not an invariance kernel. An example of a maximal invariance kernel is illustrated in Figure 9.7. Note that the maximal invariance kernel is unique. The latter fact can be simply shown using the observation that the set of all invariance kernels $\mathcal{K} \subseteq \mathit{Region}$ with respect to $\mathcal{M}$ is closed under union. Let $\mathcal{K}^* \subseteq \mathit{Region}$ be the (unique) maximal invariance kernel with respect to $\mathcal{M}$. Then, the minimum probability $\mathit{MinStable}(\mathcal{M}, \mathit{Region})$ that $\mathcal{M}$ is *stable with respect to Region* is defined as the limit of the minimum step-bounded probability of reaching the maximal invariance kernel $\mathcal{K}^*$ from the initial state $\imath$, i.e.

$$\mathit{MinStable}(\mathcal{M}, \mathit{Region}) \;:=\; \lim_{k\to\infty} \mathit{MinReach}^k_{\mathcal{M},\mathcal{K}^*}(\imath)$$

where

$$\mathit{MinReach}^k_{\mathcal{M},\mathcal{K}^*}(s) := \begin{cases} 1 & \text{if } s \in \mathcal{K}^* \quad , \\ 0 & \text{if } s \notin \mathcal{K}^* \text{ and } k = 0 \quad , \\ \min\limits_{a\in \mathit{Act}} \sum\limits_{s'\in S} \mathit{ps}(s, a, s') \cdot \mathit{MinReach}^{k-1}_{\mathcal{M},\mathcal{K}^*}(s') & \\ & \text{if } s \notin \mathcal{K}^* \text{ and } k > 0 \end{cases}$$

gives the minimum probability of reaching $\mathcal{K}^*$ from state $s \in S$ within $k \in \mathbb{N}$ steps under each possible scheduler.

When considering stabilization within *Region* as the *desired property* then the value of *MinStable*$(\mathcal{M}, \mathit{Region})$ establishes the probability of stabilizing in *worst case*, i.e. under an optimal adversarial scheduler. For some threshold value $\theta \in [0,1]$, the *stability verification problem* then is to decide whether this worst-case probability is at least θ, i.e. to decide whether

$$\mathit{MinStable}(\mathcal{M}, \mathit{Region}) \geq \theta \tag{9.4}$$

holds.

In what follows, we propose a symbolic *verification* procedure for above problem 9.4 which also proceeds in two phases. In phase 1, we compute a *symbolic representation of an invariance kernel* by means of generalized Craig interpolation. The main idea here is to iteratively eliminate states z not belonging to an invariance kernel from *Region* until a fixed point is reached. Due to the use of interpolation, the set of such states z is overapproximated in each iteration, meaning that potentially too many states are removed. This implies that the resulting invariance kernel is *not* necessarily maximal. However, each invariance kernel can be exploited for computing *lower bounds lb* of *MinStable*$(\mathcal{M}, \mathit{Region})$. The latter computation then is performed in phase 2 by means of SSAT-based bounded reachability analysis. Once a lower bound $lb \geq \theta$ is computed, property 9.4 is verified.

Phase 1 (Symbolic representation of an invariance kernel). Let be given an SSAT encoding of an MDP $\mathcal{M}$ as above and a propositional formula $\mathit{Region}(\boldsymbol{s})$ encoding the stabilization region *Region*. The state-set predicate $\mathcal{R}^k(\boldsymbol{s})$ for $k \in \mathbb{N}$ over state variables $\boldsymbol{s}$ is inductively defined as

- $\mathcal{R}^0(\boldsymbol{s}) \;:=\; \mathit{Region}(\boldsymbol{s})$, and
- $\mathcal{R}^{k+1}(\boldsymbol{s}) \;:=\; \mathcal{R}^k(\boldsymbol{s}) \;\wedge\; \neg\mathcal{I}^{k+1}(\boldsymbol{s})$

where $\mathcal{I}^{k+1}(\boldsymbol{s}_{j-1})$ is a generalized Craig interpolant for

$$\left(\overbrace{\mathit{TRANS}_{\mathcal{M}}(\boldsymbol{s}_{j-1}, \boldsymbol{t}_j, \boldsymbol{s}_j) \wedge \neg\mathcal{R}^k(\boldsymbol{s}_j)}^{=A},\; \overbrace{\mathit{INIT}_{\mathcal{M}}(\boldsymbol{s}_0) \wedge \bigwedge_{i=1}^{j-1} \mathit{TRANS}_{\mathcal{M}}(\boldsymbol{s}_{i-1}, \boldsymbol{t}_i, \boldsymbol{s}_i)}^{=B} \right)$$

with $j \geq 1$ with respect to SSAT formula

$$\mathit{CHOICE}_{\mathcal{M}}(j) : \left(\begin{array}{l} \overbrace{\mathit{INIT}_{\mathcal{M}}(\boldsymbol{s}_0) \wedge \bigwedge_{i=1}^{j-1} \mathit{TRANS}_{\mathcal{M}}(\boldsymbol{s}_{i-1}, \boldsymbol{t}_i, \boldsymbol{s}_i)}^{j-1 \text{ steps “forward” } (=B)} \\ \wedge\, \underbrace{\mathit{TRANS}_{\mathcal{M}}(\boldsymbol{s}_{j-1}, \boldsymbol{t}_j, \boldsymbol{s}_j) \wedge \neg\mathcal{R}^k(\boldsymbol{s}_j)}_{\text{one step “backward” from } \neg\mathcal{R}^k \; (=A)} \end{array} \right). \tag{9.5}$$

Observe that each $\mathcal{I}^{k+1}(\boldsymbol{s}_{j-1})$ can be computed by interpolating S-resolution if we rewrite $\neg\mathcal{R}^k(\boldsymbol{s}_j)$ into CNF, the latter being always possible in linear time using the Tseitin transformation [Tse68], which potentially adds auxiliary V_A-variables. During computation of

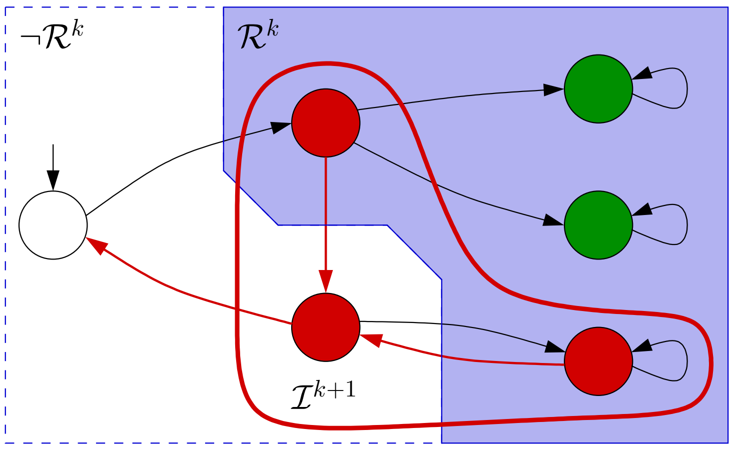


Figure 9.8: Illustration of the generalized Craig interpolation scheme 9.5: in this example, each generalized Craig interpolant $\mathcal{I}^{k+1}(\boldsymbol{s})$ describes at least all red states as these directly lead to the state set $\neg\mathcal{R}^k(\boldsymbol{s})$, but $\mathcal{I}^{k+1}(\boldsymbol{s})$ may also comprise some of the other states. Observe that the maximal invariance kernel consists of the both green states.

each $\mathcal{I}^{k+1}(\boldsymbol{s}_{j-1})$, we take $I = \texttt{true}$ in every application of rule RI.2 such that $\mathcal{I}^{k+1}(\boldsymbol{s})$ overapproximates all system states directly leading to the state set $\neg\mathcal{R}^k(\boldsymbol{s})$ due to Corollary 9.2. As a consequence, from each state in $\mathcal{R}^{k+1}(\boldsymbol{s}) = \mathcal{R}^k(\boldsymbol{s}) \wedge \neg\mathcal{I}^{k+1}(\boldsymbol{s})$ it is infeasible to leave the set $\mathcal{R}^k(\boldsymbol{s})$ in one step. For an illustration of the generalized Craig interpolation scheme 9.5, confer Figure 9.8. Whenever the chain $\mathcal{R}^k(\boldsymbol{s})$ has reached a fixed point, i.e. if

$$\mathcal{R}^k(\boldsymbol{s}) \;\Rightarrow\; \mathcal{R}^{k+1}(\boldsymbol{s})$$

holds for some k, it follows that $\mathcal{K}(\boldsymbol{s}) := \mathcal{R}^k(\boldsymbol{s})$ is a not necessarily maximal invariance kernel of *Region* with respect to $\mathcal{M}$, i.e. once entered, the system cannot leave the set $\mathcal{K}(\boldsymbol{s})$. Obviously, the chain $\mathcal{R}^k(\boldsymbol{s})$ eventually reaches a fixed point in the finite-state case (which, however, may be trivial).

Similar to scheme 9.2, parameter $j \geq 1$ can be chosen arbitrarily, i.e. the system may execute any number of transitions until state $\boldsymbol{s}_{j-1}$ is reached since this does not destroy the overapproximation property of $\mathcal{I}^{k+1}(\boldsymbol{s})$. The presence of parameter j gives us additional freedom in constructing generalized Craig interpolants, as j may influence the shape of $\mathcal{I}^{k+1}(\boldsymbol{s})$ like in the proof of concept below.

Phase 2 (Lower bounds on minimum probability of reaching the kernel). After having determined a symbolic representation $\mathcal{K}(\boldsymbol{s})$ of a not necessarily maximal invariance kernel $\mathcal{K}$ with respect to MDP $\mathcal{M}$, we now compute lower bounds on the minimum probability $\mathit{MinStable}(\mathcal{M}, \mathit{Region})$ of stabilizing within *Region* by means of SSAT solving. To this end, first observe that $\mathit{MinReach}^k_{\mathcal{M},\mathcal{K}^*}(\imath)$ is monotonically increasing in k which implies that

$$\mathit{MinReach}^k_{\mathcal{M},\mathcal{K}^*}(\imath) \;\leq\; \mathit{MinStable}(\mathcal{M}, \mathit{Region})$$

for each $k \in \mathbb{N}$. Let $\mathcal{K}^*$ be the unique maximal invariance kernel with respect to $\mathcal{M}$. Then, $\mathcal{K} \subseteq \mathcal{K}^*$ since $\mathcal{K}$ is an invariance kernel and the maximal invariance kernel $\mathcal{K}^*$ is unique. As a consequence,

$$\mathit{MinReach}^k_{\mathcal{M},\mathcal{K}}(\imath) \;\leq\; \mathit{MinReach}^k_{\mathcal{M},\mathcal{K}^*}(\imath)$$

for each $k \in \mathbb{N}$. Summing up, each value of $\mathit{MinReach}^k_{\mathcal{M},\mathcal{K}}(\imath)$ establishes a lower bound of $\mathit{MinStable}(\mathcal{M}, \mathit{Region})$.

In principle, $\mathit{MinReach}^k_{\mathcal{M},\mathcal{K}}(\imath)$ can be reduced to an SSAT formula similar to the SSMT encoding scheme of Section 5.3. The difference however is that we need to *minimize* the satisfaction probability. The latter can be achieved by an SSAT reduction scheme that exploits *universal* quantifiers to resolve non-deterministic choices of actions. Universal quantifiers then aim at minimizing the satisfaction probability, confer Section 4.2. Though the SSMT solver SiSAT actually supports universal quantification, we instead stay within the scope of the logic exposed in this thesis and rephrase minimum probabilistic state reachability as a *maximum probabilistic state avoidance problem* as follows:

$$\mathit{MaxAvoid}^k_{\mathcal{M},\mathcal{K}}(s) := \begin{cases} 0 & \text{if } s \in \mathcal{K} \;\;, \\ 1 & \text{if } s \notin \mathcal{K} \text{ and } k = 0 \;\;, \\ \max\limits_{a \in Act} \sum\limits_{s' \in S} ps(s,a,s') \cdot \mathit{MaxAvoid}^{k-1}_{\mathcal{M},\mathcal{K}}(s') & \\ & \text{if } s \notin \mathcal{K} \text{ and } k > 0 \;\;. \end{cases}$$

It then holds that

$$\mathit{MinReach}^k_{\mathcal{M},\mathcal{K}}(\imath) \;=\; 1 - \mathit{MaxAvoid}^k_{\mathcal{M},\mathcal{K}}(\imath)$$

which can be proven by straightforward induction on the step bound k. In the base cases, i.e. if $k = 0$ and $s \in \mathcal{K}$ or $s \notin \mathcal{K}$, the statement is obvious. Within the induction step, we exploit the property that

$$\min_i \sum\nolimits_j p_{i,j} \cdot P_{i,j} = 1 - \max_i \sum\nolimits_j p_{i,j} \cdot (1 - P_{i,j})$$

is true for $0 \leq P_{i,j} \leq 1$ and $\sum_j p_{i,j} = 1$.

The problem of computing the value of $\mathit{MaxAvoid}^k_{\mathcal{M},\mathcal{K}}(\imath)$ can be reduced to computing the maximum probability of satisfaction of the SSAT formula

$$\Phi^k_{\mathcal{M},\mathcal{K}} \;:=\; \mathit{CHOICE}_{\mathcal{M}}(k) : \left(\begin{array}{l} \overbrace{\mathit{INIT}_{\mathcal{M}}(\boldsymbol{s}_0) \wedge \bigwedge_{i=1}^{k} \mathit{TRANS}_{\mathcal{M}}(\boldsymbol{s}_{i-1}, \boldsymbol{t}_i, \boldsymbol{s}_i)}^{\text{states reachable within } k \text{ steps}} \\ \wedge \quad \underbrace{\bigwedge_{i=0}^{k} \neg\mathcal{K}(\boldsymbol{s}_i)}_{\text{avoid invariance kernel } \mathcal{K}} \end{array} \right) \;.$$

According to the definition of $\mathit{MaxAvoid}^k_{\mathcal{M},\mathcal{K}}(\imath)$, the propositional formula of $\Phi^k_{\mathcal{M},\mathcal{K}}$ describes all system runs avoiding the invariance kernel $\mathcal{K}$ for at least k transition steps. That is, all assignments encoding such latter runs yield satisfaction probability 1, while assignments encoding runs that visit $\mathcal{K}$ within the first k steps do not satisfy the propositional formula, thus leading to satisfaction probability 0. As a consequence,

$$\mathit{MaxAvoid}^k_{\mathcal{M},\mathcal{K}}(\imath) \;=\; \mathit{Pr}\left(\Phi^k_{\mathcal{M},\mathcal{K}}\right) \;\;.$$

j	$\mathcal{I}^1$	$\mathcal{R}^1$	$\mathcal{I}^2$	$\mathcal{R}^2$	$\mathcal{K}$
1	`true`	`false`	`true`	`false`	`false`
	$\{i, f, e, s\}$	$\emptyset$	$\{i, f, e, s\}$	$\emptyset$	$\emptyset$
2	`true`	`false`	`true`	`false`	`false`
	$\{i, f, e, s\}$	$\emptyset$	$\{i, f, e, s\}$	$\emptyset$	$\emptyset$
3	$\neg s$	$\neg f \wedge s$	$\neg s$	$\neg f \wedge s$	$\neg f \wedge s$
	$\{i, f, e\}$	$\{s\}$	$\{i, f, e\}$	$\{s\}$	$\{s\}$
4	$\neg s$	$\neg f \wedge s$	$\neg s$	$\neg f \wedge s$	$\neg f \wedge s$
	$\{i, f, e\}$	$\{s\}$	$\{i, f, e\}$	$\{s\}$	$\{s\}$

Figure 9.9: Probabilistic region stability analysis of MDP $\mathcal{M}$: experimental results of applying the generalized Craig interpolation scheme 9.5 for $\mathcal{M}$ from Figure 9.4 for different values of parameter j, with the stabilization region consisting of the states i, e, and s. In addition to the symbolic representations computed by interpolation, the concrete state sets represented by these predicates are stated explicitly. (Source of figure: [TF12])

Using above facts, we deduce the following relation

$$\begin{aligned}1 - Pr\left(\Phi^k_{\mathcal{M},\mathcal{K}}\right) &= 1 - \mathit{MaxAvoid}^k_{\mathcal{M},\mathcal{K}}(\imath)\\ &= \mathit{MinReach}^k_{\mathcal{M},\mathcal{K}}(\imath)\\ &\leq \mathit{MinReach}^k_{\mathcal{M},\mathcal{K}^*}(\imath)\\ &\leq \mathit{MinStable}(\mathcal{M}, \mathit{Region}) \quad .\end{aligned}$$

This finally enables us to compute lower bounds lb_k of $\mathit{MinStable}(\mathcal{M}, \mathit{Region})$ using the scheme

$$lb_k := 1 - Pr\left(\Phi^k_{\mathcal{M},\mathcal{K}}\right) \quad , \tag{9.6}$$

where $Pr(\Phi^k_{\mathcal{M},\mathcal{K}})$ is determined by SSAT solving. Note that the system behavior encoded by $\Phi^k_{\mathcal{M},\mathcal{K}}$ becomes more and more constrained for increasing k such that the satisfaction probabilities $Pr\left(\Phi^k_{\mathcal{M},\mathcal{K}}\right)$ are monotonically decreasing. This in turn means that the lb_k's are monotonically increasing. With regard to solving the stability verification problem 9.4, the desired property $\mathit{MinStable}(\mathcal{M}, \mathit{Region}) \geq \theta$ is *verified* by the procedure above once a lower bound $lb_k \geq \theta$ is computed for some k.

Proof of concept. To demonstrate feasibility of the symbolic approach to probabilistic region stability based on generalized Craig interpolation, again consider the simple MDP $\mathcal{M}$ from Figure 9.4 and let the symbolic representation of the stabilization region be given by $\mathit{Region}(\boldsymbol{s}) = \neg f$. Accordingly, the region in which $\mathcal{M}$ should stabilize consists of the states i, e, and s. We again use the symbolic SSAT encoding of $\mathcal{M}$ as introduced before.

We are first interested in computing an invariance kernel $\mathcal{K} \subseteq \mathit{Region}(\boldsymbol{s})$ with respect to $\mathcal{M}$ by means of the generalized Craig interpolation scheme 9.5. To cope with the latter scheme automatically, we again employ the simple interpolating DPLL-based SSAT solver

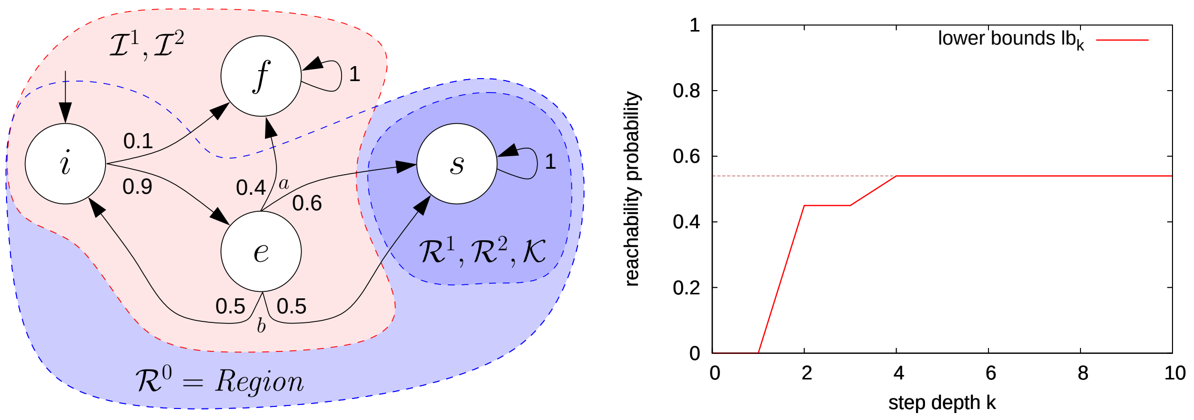


Figure 9.10: Probabilistic region stability analysis of MDP $\mathcal{M}$: illustration of the computed state sets for $\mathcal{M}$ by the generalized Craig interpolation scheme 9.5 with $j \in \{3, 4\}$ (left), and lower bounds lb_k on the minimum probability of reaching the invariance kernel $\mathcal{K} = \{s\}$ over step depth k computed by scheme 9.6 (right). (Source of figure: [TF12])

mentioned in the previous subsection. The results of these experiments for different values of j are shown in Figure 9.9. It is not hard to see that the unique maximal invariance kernel $\mathcal{K}^*$ consists of the state s only. Recall that each interpolant $\mathcal{I}^{k+1}$ overapproximates all system states directly leading to the state set $\neg\mathcal{R}^k$. When setting parameter j to value 1 or 2, we observe that interpolant $\mathcal{I}^1 = \texttt{true}$ is too coarse since it includes the whole state space. This causes the trivial invariance kernel $\mathcal{K} = \texttt{false}$ to be computed representing the empty set. For choices $j = 3$ and $j = 4$, however, $\mathcal{I}^1 = \neg s$ describes the exact set of states which lead to $\neg\mathcal{R}^0 = \neg Region$. Finally, the non-trivial invariance kernel $\mathcal{K} = \neg f \wedge s$ consisting of state s only is computed. Note that $\mathcal{K}$ actually is the maximal invariance kernel. The computed state sets for $j \in \{3, 4\}$ are illustrated on the left of Figure 9.10.

These results confirm the observation obtained from the experiments of Subsection 9.3.1, namely that the greater value of j, i.e. the more transition steps are performed, the more accurate the overapproximation. Concerning runtime, each generalized Craig interpolant was computed by the interpolating DPLL-based SSAT solver within fractions of a second, where the highest runtime of 88 milliseconds was observed when computing $\mathcal{I}^2$ for $j = 4$.

After having determined a symbolic representation of an invariance kernel $\mathcal{K} \subseteq Region$ with respect to $\mathcal{M}$, we computed lower bounds lb_k on the minimum probability that $\mathcal{M}$ is stable with respect to $Region$ by means of scheme 9.6 for $\mathcal{K}(\boldsymbol{s}) = \neg f \wedge s$, as obtained for $j \in \{3, 4\}$, again having employed the SSMT tool SiSAT. Some of the results are $lb_0 = lb_1 = 0$, $lb_2 = lb_3 = 0.45$, $lb_4 = lb_5 = 0.54$, ..., and $lb_{100} = 0.54$. Concerning runtime, all 100 SSAT formulae were solved within 88.16 seconds, while computation of the first 20 lower bounds lb_0 to lb_{20} consumed just 600 milliseconds. The highest computation time for a single SSAT problem was obtained on lb_{100}, namely 2.91 seconds. The evolution of the lb_k's up to depth $k = 10$ is presented graphically on the right of Figure 9.10. With regard to the stability verification problem 9.4, the desired property $MinStable(\mathcal{M}, Region) \geq \theta$ is *verified* for each threshold value $\theta \leq 0.54$ within a second.

Concerning competitive approaches, we remark that the probabilistic model checking tool PRISM 4.0.1 [KNP11] is also able to deal with probabilistic region stability of MDPs by means of path operators.[3] To determine the value of $\mathit{MinStable}(\mathcal{M}, \mathit{Region})$ for the example above, we used the specification `Pmin=? [F P>=1 [G (!f)]]` meaning that we are interested in the minimum probability (`Pmin=?`) that finally (`F`) the system satisfies almost surely (`P>=1`) the property that globally (`G`) state f is never visited (`!f`). PRISM solved the problem in 644 milliseconds returning the result 0.54.

As discussed for the case of probabilistic state reachability in Subsection 9.3.1, we are also confident that the presented approach to probabilistic region stability based on generalized Craig interpolation becomes beneficial when adapted to probabilistic *hybrid* systems, where the classical procedures are not directly applicable. Furthermore, a particular pay-off is expected when dealing with *concurrent* probabilistic systems owing to the *symbolic* nature of the interpolation-based technique.

[3]Confer `http://www.prismmodelchecker.org/manual/PropertySpecification/ThePOperator` for more detailed information.

10 Conclusion

In this chapter, we first summarize the achievements in Section 10.1, then elaborate on potential ideas for future research in Section 10.2, and give closing words in Section 10.3.

10.1 Summary of achievements

In this thesis, we made three contributions to symbolic model checking of probabilistic hybrid and finite-state systems.

At first, we elaborated on the main contribution, namely a symbolic falsification procedure for probabilistic safety properties of probabilistic hybrid automata based on SSMT solving. As a formal model, we considered probabilistic hybrid automata (PHAs), which are, on the one hand, restricted to discrete time, to finite non-determinism, and to finite probabilistic choices but, on the other hand, allow for a very general concept of concurrency. Recall that the system class of PHAs does not support ordinary differential equations (ODEs) directly. It is however always possible to safely approximate ODEs by, for instance, Taylor series. In spite of these limitations, interesting applications can be covered as shown by the realistic case study from the networked automation systems (NAS) domain, confer Section 3.1 and Chapter 8. We again remark that these restrictions are not essential for the approach and that we discuss a much more expressive automata model which incorporates ODEs and is interpreted over continuous time in Section 10.2.

With regard to the analysis of PHAs, we were interested in probabilistic bounded state reachability, more precisely, in the worst-case probability of reaching a set of (unsafe) target states within a bounded number of system steps. In order to address the latter analysis goal fully symbolically, probabilistic bounded reachability for concurrent PHAs is encoded as a stochastic satisfiability modulo theories (SSMT) formula involving rich arithmetic constraints. The logical framework of SSMT extends the notion of satisfiability modulo theories (SMT) with alternating existential and randomized quantifiers, leading to a quantitative semantics of SSMT formulae, namely to the maximum probability of satisfaction. These existential and randomized quantifiers are utilized to describe the non-deterministic and probabilistic choices of PHAs, respectively. The satisfaction probabilities of the SSMT formulae reflecting the probabilistic reachability problem up to some specified step depths yield lower bounds on the worst-case probability of reaching the target states. Recall that the symbolic encoding of a system of concurrent PHAs is of size linear in the number of concurrent components. The state explosion problem, arising in several other analysis approaches from an explicit construction of the product automaton with respect to the discrete state space, is thus mitigated which contributes to a better scalability.

In order to complete the symbolic analysis procedure, we explained a DPLL-style backtracking algorithm for solving SSMT formulae. Basically, this algorithm traverses the tree spanned by the domains of the existential and randomized variables. Whenever a leaf of this tree, representing a quantifier-free SMT subproblem, is visited, the SMT solver iSAT is called to address this subproblem. We discussed that such naive approach is far from scalable as it has to explore the entire quantifier tree which is of size exponential in the number of quantified variables. To overcome this problem, we proposed a number of algorithmic enhancements based on semantic inferences pruning major parts of the quantifier tree. By means of experimental results, it was shown that these optimizations can lead to tremendous performance gains, sometimes by multiple orders of magnitude. Concerning probabilistic safety properties of the shape "the probability of reaching the unsafe states is at most 1‰ in worst case", this probabilistic bounded model checking approach establishes a falsification procedure being able to refute such requirements on PHAs whenever a lower bound on the worst-case probability of reaching the unsafe states is computed which exceeds the acceptable threshold value 1‰. Practical applicability of this symbolic analysis procedure was demonstrated on a realistic case study from the networked automation systems (NAS) domain.

Motivated by the fact that industrial applications often call for quantitative measures distinct from classical reachability probabilities, a second achievement of this thesis is a symbolic verification procedure for safety requirements on expected values of probabilistic hybrid automata. To this end, we suggested an SSMT-based method for computing expected values of concurrent discrete-time PHAs like, for instance, mean time to failure (MTTF). We extended the concept of concurrent discrete-time PHAs by a notion of costs and then defined the cost expectation for such systems as well as the corresponding cost-expectation model-checking problem. In order to address a step-bounded variant of the latter problem symbolically, we extended the semantics of SSMT to deal with the maximum conditional expectation of a designated free variable in an SSMT formula, and presented a reduction from the step-bounded cost-expectation problem to the enhanced version of SSMT. The extended semantics of the resulting SSMT formulae, i.e. the maximum conditional expectation of the designated variable, yields lower bounds on the cost expectation for the given system. The presentation of an algorithm dealing with the extended semantics of SSMT completed the symbolic verification procedure being able to validate probabilistic safety requirements of the shape "the MTTF is always at least 20 minutes" whenever a lower bound on the worst-case MTTF is computed which is at least 20 minutes. Applicability of this SSMT-based expected-value analysis approach was also demonstrated on the NAS case study.

A third contribution dealt with approaches that go beyond probabilistic bounded state reachability but are at the moment restricted to probabilistic finite-state models like Markov decision processes. For this purpose, the novel concept of generalized Craig interpolation for SSAT formulae was suggested. In order to determine generalized Craig interpolants algorithmically, we devised a resolution calculus for SSAT formulae called S-resolution and then augmented it with a Pudlák-style interpolant generation. We further showed how S-resolution can be integrated into a DPLL-based SSAT procedure, which is the core algorithm of state-of-the-art SSAT solvers, with the objective of computing generalized interpolants more efficiently in practice, as known from the non-probabilistic case.

As an application of generalized Craig interpolation, we first identified probabilistic state reachability and developed a symbolic verification procedure for probabilistic safety properties of probabilistic finite-state systems. Though currently being confined to the finite-state case, this approach can be considered as complementary to the symbolic falsification procedure mentioned above. Akin to symbolic methods for non-probabilistic systems, generalized Craig interpolation provides a technique for computing a symbolic overapproximation of the (backward) reachable state set of probabilistic systems. While Craig interpolation-based model checking for non-probabilistic systems is able to verify safety properties of the shape "the unsafe states are unreachable", namely if the overapproximated set of all reachable states has an empty intersection with the set of unsafe states, reaching the unsafe states is frequently unavoidable in probabilistic scenarios. A simple check for empty intersection thus does not suffice in general to verify probabilistic safety properties like "the probability of reaching the unsafe states is at most 1‰ in worst case". The verification procedure proposed in this thesis exploits the symbolic overapproximation of the backward reachable state set, being the fixed point of an iterative computation of generalized Craig interpolants, as well as a predicative description of the system in order to construct SSAT formulae whose quantitative interpretations yield upper bounds on the worst-case probability of reaching the unsafe states. Whenever an upper bound of at most 1‰ is computed using an SSAT solver then above probabilistic safety property is verified.

As another application area of generalized Craig interpolation, we gave attention to probabilistic region stability of probabilistic finite-state systems. We devised a symbolic procedure for the verification of probabilistic stability properties of the shape "the probability that the system stabilizes within a given region is always at least 99.9%". More precisely, we were interested in the minimum probability of reaching the maximal invariance kernel of the given region which can never be left. The proposed approach first computes a symbolic representation of a not necessarily maximal invariance kernel by means of generalized Craig interpolation. The main idea here is to iteratively eliminate states from the given region not belonging to an invariance kernel until a fixed point is reached. Recall that the set of such states is overapproximated due to the use of interpolation which implies that potentially too many states are removed. However, the resulting invariance kernel always is a subset of the maximal invariance kernel and thus can be used for computing lower bounds on the minimum probability of reaching the maximal invariance kernel, namely by means of SSAT-based probabilistic bounded reachability analysis. In case a lower bound of at least 99.9% is computed, the above probabilistic stability property is verified.

10.2 Future directions

In this section, we elaborate on interesting directions for future research.

SSMT-based analysis of continuous-time PHAs involving ODEs. One of the central points of future work is to generalize the symbolic methods presented in this thesis to a more expressive model of probabilistic hybrid automata which is interpreted over *continuous time* and which supports *uncountable non-determinism* in transition assign-

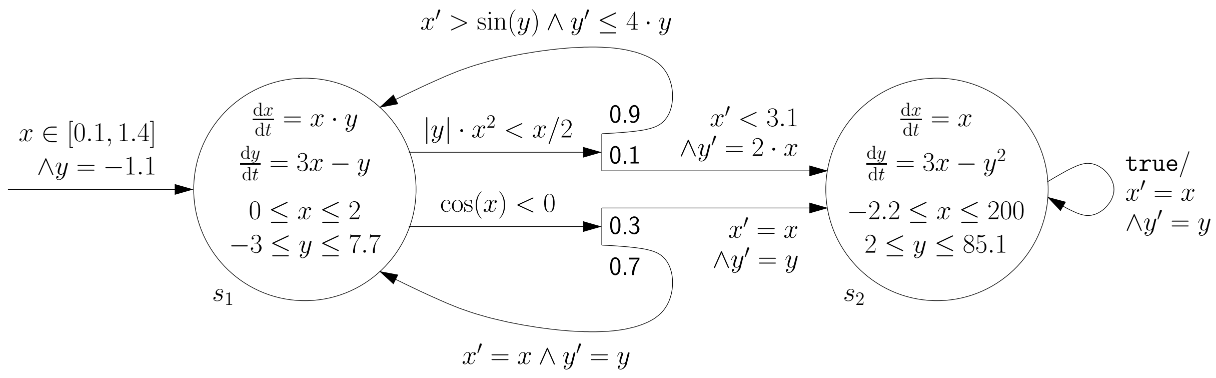


Figure 10.1: Graphical representation of a continuous-time probabilistic hybrid automaton involving ODEs. (This figure is a slight modification of Figure 1 from [FTE10a].)

ments as well as *ordinary differential equations* (ODEs) to describe continuous evolutions. In recent work [FTE10a], the author of this thesis and his co-authors proposed an SSMT-based approach to probabilistic bounded state reachability analysis of such continuous-time PHAs. An example of the more general system class is illustrated in Figure 10.1. While the conceptual idea of the generalized approach remains the same, the expressiveness of SSMT is enhanced considerably in order to cope with the richer semantics of continuous-time PHAs.

One extension is the direct *integration of the theory of ODEs into SSMT* in addition to the theory of non-linear arithmetic over the reals and integers, as pursued in the non-probabilistic setting by Eggers et al. [EFH08, EFH09]. That is, the extended SSMT framework can handle constraints involving ODEs. More precisely, an ODE constraint is represented by a tuple $(ODE(x), Inv(x), s, e, \tau)$, where

- $ODE(x)$ is an ODE of the shape

$$\frac{dx(t)}{dt} = f(x(t), y_1(t), \ldots, y_n(t))$$

  describing the trajectory of variable x defined by this ODE constraint with $y_1, \ldots, y_n$ being the variables defined by other ODE constraints,

- $Inv(x)$ is an invariant $l \leq x(t) \leq u$ describing the interval that must not be left during continuous evolution,

- a variable s called the start point of the trajectory in dimension x,

- a variable e called the end point of the trajectory in dimension x, and

- a variable τ called the length of the trajectory.

That is to say, an ODE constraint encodes a trajectory of variable x which starts in s, evolves according to $ODE(x)$, ends in e, and is of length τ, thereby never violating the invariant $Inv(x)$, as depicted in Figure 10.2.

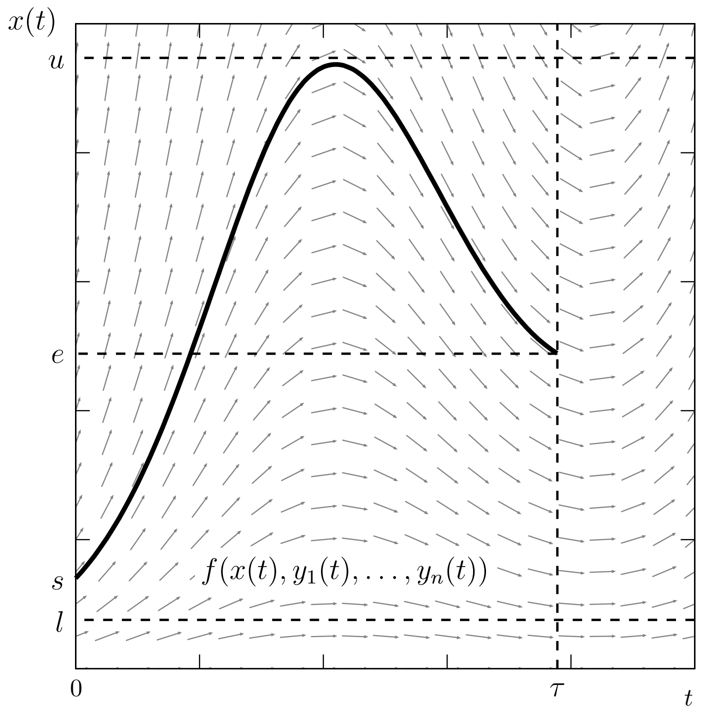


Figure 10.2: Graphical representation of a trajectory defined by an ODE constraint. (This figure is a slight modification of Figure 2 from [FTE10a].)

Another extension is to permit *existential quantification over real-valued variables in SSMT* in order to cope with uncountable non-determinism. Sources of such uncountable non-determinism are, on the one hand, non-deterministic assignments like $x' > x \cdot (1 + x^2 \cdot y^3)$ in discrete transitions and, on the other hand, the lengths τ of the ODE trajectories. When encoding the maximum probability of reaching the target states within a bounded number of system steps, where a system step is either a continuous flow or a discrete transition, as an SSMT formula, one is faced with maximizing the step-bounded reachability probability over uncountably many values of the continuous state components like x' above and of the lengths τ of the ODE trajectories. These latter non-deterministic choices in continuous-time PHAs are mapped to existential quantification in SSMT as in the discrete-time case, but the domains of these existential variables are now given by real-valued intervals instead of finite sets.

Algorithmically, computing the maximum probability of satisfaction for the extended notion of SSMT is not straightforward in general. This is due to maximizing over uncountably many alternatives and due to dealing with the undecidable theory of ODEs in addition to non-linear arithmetic. To nevertheless find safe solutions to this important and challenging problem, we presented an algorithmic approach to effectively compute safe upper bounds of satisfaction probabilities in [FTE10a]. The main idea of this algorithm is as follows. To cope with ODE constraints, we exploited overapproximation techniques based on safe interval calculations which permit safe reasoning by "enclosing" ODE solutions in interval-valued functions. This integration of ODE enclosure methods into SSMT solving is based on the work described in [EFH08, EFH09], but more sophisticated approaches as in [ERNF11, ERNF12] are conceivable. To exhaustively explore the uncountable domains of existential variables, they are partitioned into small and finitely many subintervals. Instead of values, these small intervals are assigned to existential variables during SSMT proof search. This leads to finite branching for each existential continuous-domain variable. The resulting algorithm always terminates and is sound in the sense that it delivers safe upper bounds of the maximum probability of satisfaction. In [FTE10a], we applied a prototypical implementation of the procedure sketched above to the quantitative analysis of a networked control system. This prototype already incorporates ODE enclosure methods but still lacks existential quantification over continuous domains.

The above approach establishes a fully symbolic analysis procedure for the very expressive system model of continuous-time PHAs involving ODEs. Extending the implementation to support existential quantification over continuous domains as well as improving the practical applicability and performance of the resulting tool are thus important and challenging aspects of future work. Moreover, generalizing the SSMT-based expected-value analysis procedure to the continuous-time case is another well-motivated future direction. A further issue is to adapt the generalized Craig interpolation-based approaches to probabilistic state reachability and probabilistic region stability to continuous-time PHAs involving ODEs. The latter idea however is a major challenge even for the discrete-time model considered in this thesis, as mentioned next.

Generalized Craig interpolation for SSMT. The novel concept of a generalized Craig interpolant can be smoothly adapted to the more general SSMT case, while then such generalized Craig interpolants need not exist in general. The computation of these interpolants however is not straightforward. Recall that the procedure for determining generalized interpolants is based on a resolution calculus for SSAT called S-resolution. Since SSMT formulae involve quantified variables with arbitrary finite domains as well as non-quantified variables with continuous domains, S-resolution is not directly applicable to SSMT. A potential generalization of S-resolution to cope with SSMT formulae comprising non-linear real arithmetic constraints might incorporate the work on a resolution calculus for SMT with respect to the undecidable theory of non-linear arithmetic over the reals and integers [KTBF09, KBTF11]. The latter resolution calculus was enhanced by construction rules to compute Craig interpolants for such SMT formulae in [KB11a, KB11b].

An interesting as well as challenging point for future research thus is, first, to develop a resolution calculus for SSMT and, second, to augment this by rules for the construction of generalized Craig interpolants. Such an interpolating SSMT resolution calculus would render the generalized Craig interpolation-based approaches to probabilistic state reachability and probabilistic region stability directly applicable to concurrent discrete-time probabilistic hybrid automata.

More expressive specification logics. The main approach of this thesis aims at probabilistic safety properties of the shape “an unsafe state is reachable only with very low probability”. Such properties just consider (un)reachability of states. It however is oftentimes necessary to specify more expressive requirements like “each message will be finally delivered with probability 1 and whenever a message is tried to be sent then this message will be delivered in five system steps with very high probability of at least 0.995”.

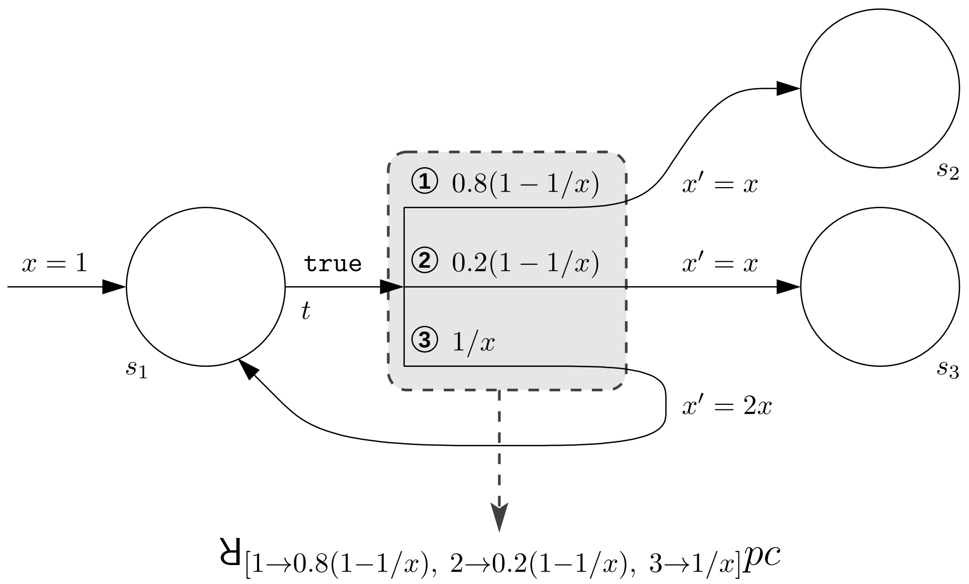


Figure 10.3: A probabilistic hybrid automaton that incorporates state-dependent probability distributions within state transitions, and a potential generalization of randomized quantification to cope with such distributions. The encircled numbers enumerate the probabilistic transition alternatives.

For this purpose, several temporal logics like probabilistic versions of CTL and LTL have been proposed in the literature.

It thus is well-motivated to enhance SSMT-based probabilistic bounded model checking of PHAs with respect to more expressive system requirements in future work. Some preliminary work on SSMT-based PBMC of PHAs for LTL specifications can be found in [Sch08]. In this approach, which is similar to the one described in [AKM11], confer Section 3.2, the LTL formula is translated into a generalized Büchi automaton which in turn can be encoded as an SSMT formula. This reduces the original problem to probabilistic bounded state reachability.

More expressive stochastic dynamics. The current model of PHAs is confined in its stochastic behavior as it only admits probabilistic events from a finite sample space within state transitions. One idea to achieve a more expressive model of randomness is to permit continuous probability distributions in discrete state changes as in [FHH$^+$11].

A slightly different but not less interesting enhancement is to support distributions that are still discrete but depend on the current discrete-continuous system state. For an example, consider Figure 10.3. If transition t is executed then the probabilities of the three probabilistic transition alternatives are not given by constant values, as in the current model, but by the values of arithmetic terms over the continuous state component x. For instance, immediately after discrete state s_1 is entered for the first time, variable x carries value 1. The probability of selecting one of the transition alternatives 1 and 2 is 0, while the probability of alternative 3 is 1. The automaton thus performs a self loop, thereby doubling the value of x to 2. That is, the probability of a second self loop decreases to $1/2 = 0.5$. Switching to discrete states s_2 and s_3 therefore is of probability

$0.8(1 - 1/2) = 0.4$ and $0.2(1 - 1/2) = 0.1$, respectively. Let us assume that the system revisits s_1 again. Then, the value of variable x was updated to 4. This implies that the probability of a third self loop in a row drops to $1/4 = 0.25$ where the probabilities of transition alternatives 1 and 2 are $0.8(1-1/4) = 0.6$ and $0.2(1-1/4) = 0.15$, respectively. It is easy to see that the probability of leaving discrete state s_1 is quickly increasing, namely according to $1 - 2^{-n}$ with n being the number of consecutive self loops. The above idea allows to describe properties like "the higher temperature the higher risk of fire" in a very precise way. In the current model, such facts can only be approximated by means of finitely many transitions and corresponding transition alternatives.

With regard to SSMT-based model checking of such enhanced PHAs, randomized quantifiers in SSMT must be extended for the purpose of supporting above state-dependent distributions, as indicated in Figure 10.3.

Generation of probabilistic counterexamples. An essential feature of model checking is the generation of counterexamples, which are system executions that violate certain system requirements. Counterexamples provide valuable information on how the system has to be redesigned with the objective of satisfying the system requirements. While a counterexample in the non-probabilistic case is simply one system run in the case of linear-time properties, a counterexample for probabilistic safety properties of the shape "the probability of reaching unsafe system states is at most θ" is rather a *tree* of runs with a probability mass exceeding the acceptable threshold value θ. More recently, several approaches to the generation of probabilistic counterexamples, in particular for finite-state Markov models were proposed in the literature [ADvR09, HKD09, AL10, ÁJW+10].

Braitling, Wimmer et al. [WBB09, BWB+11] presented SAT-based and SMT-based BMC approaches to the generation of probabilistic counterexamples for discrete-time Markov chains and Markov reward models, confer Section 3.2. Though the system models considered have finite state space and are fully probabilistic, these approaches are technically related to SSMT-based PBMC of PHAs. The technique for the generation of probabilistic counterexamples relies on collecting satisfying assignments of the corresponding BMC formulae provided by a SAT or SMT solver until their probability mass exceeds the safety threshold θ.

A similar idea can be applied to SSMT-based PBMC of PHAs. The SSMT encoding scheme ensures that satisfying assignments (of the matrix) of the PBMC formula are in one-to-one correspondence to system runs reaching the unsafe target states. Recall that the presented SSMT algorithm actually searches for such satisfying assignments during proof search which implies that the algorithm is able to generate probabilistic counterexamples. This observation was already exploited in the algorithmic enhancement of caching solutions, confer Subsection 6.5.6. That is to say, the mere construction of probabilistic counterexamples for PHAs by means of SSMT-based PBMC is a rather straightforward extension.

More challenging tasks for future work deal with the development of methods for efficiently representing and moreover compactifying probabilistic counterexamples produced by SSMT solving. From an engineering perspective, it furthermore is advantageous to devise techniques for an adequate visualization of probabilistic counterexamples within PHAs in a user-friendly way. Probabilistic counterexamples of real-world, large-scale systems can be expected to be rather large and thus not comprehensible to a human. Comprehensibility however is a crucial issue for the system engineer in order to eliminate the faulty behavior from the system under development. An important question thus is how debugging information based on the potentially large counterexample can be properly visualized within a human-readable automaton model. One idea could be a simulation-based presentation of such counterexamples. That is, the PHA is executed while non-deterministic choices are resolved by the counterexample but probabilistic choices need to be handled by the user. Another idea is to identify a rather minimal subsystem by means of removing non-deterministic and probabilistic transitions as well as locations from the original system such that this subsystem itself violates the probabilistic safety property.

Parallelization of SSMT solving algorithms. In order to improve performance of SSMT solvers in practice, another idea for future work is motivated by recent trends in hardware design towards multicore and multiprocessor systems, namely the development of parallelized SSMT algorithms. In recent work in the related area of parallel solving of quantified Boolean formulae, sometimes super-linear speed-ups were obtained due to knowledge sharing between the parallel solving processes [LMS$^+$09, LSB$^+$11]. In parallel SSMT solving, various forms of knowledge sharing based on the algorithmic enhancements from Section 6.5 are conceivable, suggesting similar performance gains as mentioned above.

Integration of statistical methods into SSMT solving. As an alternative to exhaustive probabilistic model checking, analysis approaches commonly referred to as statistical model checking (SMC) have been developed [You05a, You05b, YS06, LDB10, ZPC10, DLL$^+$11, MPL11]. SMC primarily addresses the problem of deciding whether a given probabilistic (hybrid) system meets a time-bounded property specified in a probabilistic temporal logic. SMC however is not a state-exploratory method but relies on statistical techniques like hypothesis testing. That is, a finite number of time-bounded sample runs are drawn according to the probabilistic system dynamics. For each of these runs, it can simply be checked whether or not the time-bounded property holds. It is then possible to obtain statistical estimates of the probability of satisfying the property. A simple estimate, for instance, is the empirical mean, i.e. the number of runs fulfilling the property divided by the number of all sample runs. Due to the obvious fact that the results obtained by SMC are not guaranteed to be correct, approaches were developed that allow to bound the probability of error of the statistical estimate, for instance as by Hoeffding's inequality [Hoe63]. Though SMC results are only true up to some reasonable statistical uncertainty, the main advantage of SMC is that instead of exploring the entire time-bounded system behavior, a potentially much smaller set of simulated time-bounded system runs needs to be considered. Moreover, the number of such sample runs is independent of the state space of the system and just determined by the specified bound on the probability of error.

A promising direction for future work thus is to integrate statistical methods into a systematic search-based SSMT algorithm with the objective of exploiting the benefits of both approaches. On the one hand, SSMT solving would be enhanced by utilizing statistical testing as an additional pruning mechanism potentially improving performance. On the other hand, statistical testing might benefit from SSMT solving, for instance, by reducing the number of sample runs during systematic search, namely in cases where all possible completions of a prefix of a sample run satisfy or violate the property. A particular advantage of this integration may be expected in SSMT-based PBMC. Recall that the size of PBMC formulae grows linearly for increasing step depths, while the corresponding search space, i.e. the number of variable assignments, grows exponentially. This effect clearly has a negative impact on the solving times of the SSMT algorithm. As the number of sample runs for statistical testing is completely independent of the state space, as mentioned above, the algorithmic combination of statistical testing and SSMT solving will potentially allow for PBMC problems of much larger step depths.

10.3 Closing words

The work of this thesis is chiefly motivated by industrial needs for computer-aided certification methods for safety-critical applications. Due to their rapid-growing complexity, a manual inspection of these applications would be tantamount to a Sisyphean task, causing automatic approaches to be the means of choice.

With the same certainty, the methods presented in the thesis however are far away from living up to industrial standards as, for instance, deficits in expressiveness of the supported system model and in performance for large-scale industrial applications are not concealable.

Nevertheless, we firmly believe that this thesis contributes to the state of the art of research on computer-aided formal analysis methods, namely by pioneering fully symbolic techniques for the analysis of probabilistic hybrid systems, thereby complementing existing approaches based on simulation or finite-state abstractions.

Bibliography

[AAP$^+$06a] Alessandro Abate, Saurabh Amin, Maria Prandini, John Lygeros, and Shankar Sastry. Probabilistic reachability and safe sets computation for discrete time stochastic hybrid systems. In *45th IEEE Conference on Decision and Control*, pages 258–263. IEEE, 2006.

[AAP$^+$06b] Saurabh Amin, Alessandro Abate, Maria Prandini, John Lygeros, and Shankar Sastry. Reachability analysis of controlled discrete time stochastic hybrid systems. In João P. Hespanha and Ashish Tiwari, editors, *Hybrid Systems: Computation and Control*, volume 3927 of *Lecture Notes in Computer Science*, pages 49–63. Springer, 2006.

[Aba07] Alessandro Abate. *Probabilistic Reachability for Stochastic Hybrid Systems: Theory, Computations, and Applications.* PhD thesis, EECS Department, University of California, Berkeley, 2007.

[ABCS05] Gilles Audemard, Marco Bozzano, Alessandro Cimatti, and Roberto Sebastiani. Verifying industrial hybrid systems with MathSAT. *Electr. Notes Theor. Comput. Sci.*, 119(2):17–32, 2005.

[ABDK11] Ernst Althaus, Bernd Becker, Daniel Dumitriu, and Stefan Kupferschmid. Integration of an LP solver into interval constraint propagation. In Weifan Wang, Xuding Zhu, and Ding-Zhu Du, editors, *Proceedings of the 5th International Conference on Combinatorial Optimization and Applications (COCOA 2011)*, volume 6831 of *Lecture Notes in Computer Science*, pages 343–356. Springer, 2011.

[ÁBKS05] Erika Ábrahám, Bernd Becker, Felix Klaedtke, and Martin Steffen. Optimizing bounded model checking for linear hybrid systems. In Radhia Cousot, editor, *Proceedings of the 6th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI 2005)*, volume 3385 of *Lecture Notes in Computer Science*, pages 396–412. Springer, 2005.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.

[ADI06] Rajeev Alur, Thao Dang, and Franjo Ivancic. Predicate abstraction for reachability analysis of hybrid systems. *ACM Trans. Embedded Comput. Syst.*, 5(1):152–199, 2006.

[ADvR09] Miguel E. Andrés, Pedro R. D'Argenio, and Peter van Rossum. Significant diagnostic counterexamples in probabilistic model checking. In Hana Chockler and Alan J. Hu, editors, *Proceedings of the 4th International Haifa*

Verification Conference on Hardware and Software: Verification and Testing (HVC 2008), volume 5394 of *Lecture Notes in Computer Science*, pages 129–148. Springer, 2009.

[ÁJW$^+$10] Erika Ábrahám, Nils Jansen, Ralf Wimmer, Joost-Pieter Katoen, and Bernd Becker. DTMC model checking by SCC reduction. In *Proceedings of the Seventh International Conference on the Quantitative Evaluation of Systems (QEST 2010)*, pages 37–46. IEEE Computer Society, 2010.

[AKLP10] Alessandro Abate, Joost-Pieter Katoen, John Lygeros, and Maria Prandini. Approximate model checking of stochastic hybrid systems. *European Journal of Control*, 16(6):624–641, 2010.

[AKLP11] Alessandro Abate, Joost-Pieter Katoen, John Lygeros, and Maria Prandini. A two-step scheme for approximate model checking of stochastic hybrid systems. In *Proceedings of the 18th IFAC World Congress*. IFAC, 2011.

[AKM11] Alessandro Abate, Joost-Pieter Katoen, and Alexandru Mereacre. Quantitative automata model checking of autonomous stochastic hybrid systems. In Marco Caccamo, Emilio Frazzoli, and Radu Grosu, editors, *Proceedings of the 14th ACM International Conference on Hybrid Systems: Computation and Control (HSCC 2011)*, pages 83–92. ACM, 2011.

[AL94] Martín Abadi and Leslie Lamport. An old-fashioned recipe for real time. *ACM Trans. Program. Lang. Syst.*, 16(5):1543–1571, 1994.

[AL10] Husain Aljazzar and Stefan Leue. Directed explicit state-space search in the generation of counterexamples for stochastic model checking. *IEEE Trans. Software Eng.*, 36(1):37–60, 2010.

[APLS08] Alessandro Abate, Maria Prandini, John Lygeros, and Shankar Sastry. Probabilistic reachability and safety for controlled discrete time stochastic hybrid systems. *Automatica*, 44(11):2724–2734, 2008.

[APT79] Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified Boolean formulas. *Inf. Process. Lett.*, 8(3):121–123, 1979.

[Arn74] Ludwig Arnold. *Stochastic Differential Equations: Theory and Applications.* Wiley - Interscience, 1974.

[ÁSB$^+$11] Erika Ábrahám, Tobias Schubert, Bernd Becker, Martin Fränzle, and Christian Herde. Parallel SAT solving in bounded model checking. *Journal of Logic and Computation*, 21(1):5–21, 2011.

[BB04] Henk A. P. Blom and Edwin A. Bloem. Particle filtering for stochastic hybrid systems. In *43rd IEEE Conference on Decision and Control*, volume 3, pages 3221–3226, 2004.

[BB09] Hans Kleine Büning and Uwe Bubeck. Theory of quantified Boolean formulas. In Biere et al. [BHvMW09], pages 735–760.

[BBC$^+$05] Marco Bozzano, Roberto Bruttomesso, Alessandro Cimatti, Tommi A. Junttila, Peter van Rossum, Stephan Schulz, and Roberto Sebastiani. The MathSAT 3 system. In *Conf. on Automated Deduction*, volume 3632 of *Lecture Notes in Computer Science*, pages 315–321. Springer, 2005.

[BC05] Alberto Bemporad and Stefano Di Cairano. Optimal control of discrete hybrid stochastic automata. In Manfred Morari and Lothar Thiele, editors, *Hybrid Systems: Computation and Control*, volume 3414 of *Lecture Notes in Computer Science*, pages 151–167. Springer, 2005.

[BCCZ99] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In Rance Cleaveland, editor, *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS 1999)*, volume 1579 of *Lecture Notes in Computer Science*, pages 193–207. Springer, 1999.

[BdA95] Andrea Bianco and Luca de Alfaro. Model checking of probabalistic and nondeterministic systems. In P. S. Thiagarajan, editor, *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 1026 of *Lecture Notes in Computer Science*, pages 499–513. Springer, 1995.

[Bel57] Richard Bellman. A Markovian decision process. *Journal of Mathematics and Mechanics*, 6(5):679–684, 1957.

[Ben96] Frédéric Benhamou. Heterogeneous constraint solving. In Michael Hanus and Mario Rodríguez-Artalejo, editors, *Proceedings of the 5th International Conference on Algebraic and Logic Programming (ALP 1996)*, volume 1139 of *Lecture Notes in Computer Science*, pages 62–76. Springer, 1996.

[BG06] Frédéric Benhamou and Laurent Granvilliers. Continuous and interval constraints. In Francesca Rossi, Peter van Beek, and Toby Walsh, editors, *Handbook of Constraint Programming*, Foundations of Artificial Intelligence, chapter 16, pages 571–603. Elsevier, 2006.

[BHKH05] Christel Baier, Holger Hermanns, Joost-Pieter Katoen, and Boudewijn R. Haverkort. Efficient computation of time-bounded reachability probabilities in uniform continuous-time Markov decision processes. *Theor. Comput. Sci.*, 345(1):2–26, 2005.

[BHvMW09] Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[BK98] Christel Baier and Marta Z. Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. *Distributed Computing*, 11(3):125–155, 1998.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[BKB06] Henk A. P. Blom, Jaroslav Krystul, and Bert G. J. Bakker. A particle system for safety verification of free flight in air traffic. In *Decision and Control*, pages 1574–1579. IEEE, 2006.

[BKF95] Hans Kleine Büning, Marek Karpinski, and Andreas Flögel. Resolution for quantified Boolean formulas. *Inf. Comput.*, 117(1):12–18, 1995.

[BL03] Manuela L. Bujorianu and John Lygeros. Reachability questions in piecewise deterministic Markov processes. In *Hybrid Systems: Computation and Control*, volume 2623 of *Lecture Notes in Computer Science*, pages 126–140. Springer, 2003.

[BL06] Manuela L. Bujorianu and John Lygeros. Toward a general theory of stochastic hybrid systems. In Henk A.P. Blom and John Lygeros, editors, *Stochastic Hybrid Systems: Theory and Safety Critical Applications*, volume 337 of *Lecture Notes in Control and Information Sciences*, pages 3–30. Springer, 2006.

[BMH94] Frédéric Benhamou, David A. McAllester, and Pascal Van Hentenryck. CLP(Intervals) revisited. In Maurice Bruynooghe, editor, *Proceedings of the 1994 International Symposium on Logic Programming*, pages 124–138. MIT Press, 1994.

[BPT07] Andreas Bauer, Markus Pister, and Michael Tautschnig. Tool-support for the analysis of hybrid systems and models. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE 2007)*, pages 924–929. EDA Consortium, 2007.

[Bry86] Randal E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986.

[BS06] Thanasis Balafoutis and Kostas Stergiou. Algorithms for stochastic CSPs. In Frédéric Benhamou, editor, *Proceedings of the 12th International Conference on Principles and Practice of Constraint Programming (CP 2006)*, volume 4204 of *Lecture Notes in Computer Science*, pages 44–58. Springer, 2006.

[BS07] Lucas Bordeaux and Horst Samulowitz. On the stochastic constraint satisfaction framework. In *Proceedings of the 2007 ACM Symposium on Applied Computing (SAC)*, pages 316–320. ACM, 2007.

[BSST09] Clark Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Biere et al. [BHvMW09], chapter 26, pages 825–885.

[BWB$^+$11] Bettina Braitling, Ralf Wimmer, Bernd Becker, Nils Jansen, and Erika Ábrahám. Counterexample generation for Markov chains using SMT-based bounded model checking. In Roberto Bruni and Jürgen Dingel, editors, *Proceedings of the IFIP International Conference on Formal Techniques for Distributed Systems (Joint Conference 13th FMOODS & 31st FORTE 2011)*, volume 6722 of *Lecture Notes in Computer Science*, pages 75–89. Springer, 2011.

[CL07] Christos G. Cassandras and John Lygeros, editors. *Stochastic Hybrid Systems.* CRC/Taylor & Francis, 2007.

[CM03] Christos G. Cassandras and Reetabrata Mookherjee. Receding horizon optimal control for some stochastic hybrid systems. In *Proceedings of the 42nd IEEE Conference on Decision and Control*, volume 3, pages 2162–2167, 2003.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Conference Record of Third Annual ACM Symposium on Theory of Computing*, pages 151–158. ACM, 1971.

[Cra57] William Craig. Linear reasoning. a new form of the Herbrand-Gentzen theorem. *J. Symb. Log.*, 22(3):250–268, 1957.

[Dav84] Mark H. A. Davis. Piecewise-deterministic Markov processes: A general class of non-diffusion stochastic models. *Journal of the Royal Statistical Society*, 46(3):353–384, 1984.

[Dav93] Mark H. A. Davis. *Markov Models and Optimization.* Chapman & Hall, London, 1993.

[DdM06] Bruno Dutertre and Leonardo de Moura. A fast linear-arithmetic solver for DPLL(T). In Thomas Ball and Robert B. Jones, editors, *Proceedings of the 18th Computer-Aided Verification Conference*, volume 4144 of *Lecture Notes in Computer Science*, pages 81–94. Springer, 2006.

[DJJL01] Pedro R. D'Argenio, Bertrand Jeannet, Henrik Ejersbo Jensen, and Kim Guldstrand Larsen. Reachability analysis of probabilistic systems by successive refinements. In Luca de Alfaro and Stephen Gilmore, editors, *Process Algebra and Probabilistic Methods, Performance Modeling and Verification: Joint International Workshop, PAPM-PROBMIV*, volume 2165 of *Lecture Notes in Computer Science*, pages 39–56. Springer, 2001.

[DLL62] Martin Davis, George Logemann, and Donald W. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.

[DLL$^+$11] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikucionis, and Zheng Wang. Time for statistical model checking of real-time systems. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV 2011)*, volume 6806 of *Lecture Notes in Computer Science*, pages 349–355. Springer, 2011.

[DP60] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

[EFH08] Andreas Eggers, Martin Fränzle, and Christian Herde. SAT modulo ODE: A direct SAT approach to hybrid systems. In Sung Deok Cha, Jin-Young Choi, Moonzoo Kim, Insup Lee, and Mahesh Viswanathan, editors, *Proceedings of the 6th International Symposium on Automated Technology for Verification and Analysis (ATVA 2008)*, volume 5311 of *Lecture Notes in Computer Science*, pages 171–185. Springer, 2008.

[EFH09] Andreas Eggers, Martin Fränzle, and Christian Herde. Application of constraint solving and ODE-enclosure methods to the analysis of hybrid systems. In Theodore E. Simos, George Psihoyios, and Ch. Tsitouras, editors, *NUMERICAL ANALYSIS AND APPLIED MATHEMATICS: International Conference on Numerical Analysis and Applied Mathematics 2009*, volume 1168 of *AIP Conference Proceedings*, pages 1326–1330. American Institue of Physics, 2009.

[EKK$^+$11] Andreas Eggers, Evgeny Kruglov, Stefan Kupferschmid, Karsten Scheibler, Tino Teige, and Christoph Weidenbach. Superposition modulo non-linear arithmetic. In Cesare Tinelli and Viorica Sofronie-Stokkermans, editors, *Proceedings of the 8th International Symposium on Frontiers of Combining Systems (FroCoS 2011)*, volume 6989 of *Lecture Notes in Artificial Intelligence*, pages 119–134. Springer, 2011.

[EKKT08] Andreas Eggers, Natalia Kalinnik, Stefan Kupferschmid, and Tino Teige. Challenges in constraint-based analysis of hybrid systems. In *Proceedings of the Annual ERCIM Workshop on Constraint Solving and Constraint Logic Programming (CSCLP 2008)*, 2008.

[EKKT09] Andreas Eggers, Natalia Kalinnik, Stefan Kupferschmid, and Tino Teige. Challenges in constraint-based analysis of hybrid systems. In Angelo Oddi, François Fages, and Francesca Rossi, editors, *Recent Advances in Constraints*, volume 5655 of *Lecture Notes in Artificial Intelligence*, pages 51–65. Springer, 2009.

[ERNF11] Andreas Eggers, Nacim Ramdani, Nedialko S. Nedialkov, and Martin Fränzle. Improving SAT modulo ODE for hybrid systems analysis by combining different enclosure methods. In Gilles Barthe, Alberto Pardo, and Gerardo Schneider, editors, *Proceedings of the 9th International Conference on Software Engineering and Formal Methods (SEFM 2011)*, volume 7041 of *Lecture Notes in Computer Science*, pages 172–187. Springer, 2011.

[ERNF12] Andreas Eggers, Nacim Ramdani, Nedialko S. Nedialkov, and Martin Fränzle. Set-membership estimation of hybrid systems via SAT modulo ODE. In Michel Kinnaert, editor, *Proceedings of the 16th IFAC Symposium on System Identification*, pages 440–445. IFAC, 2012.

[FH07] Martin Fränzle and Christian Herde. HySAT: An efficient proof engine for bounded model checking of hybrid systems. *Formal Methods in System Design*, 30(3):179–198, 2007.

[FHH$^+$11] Martin Fränzle, Ernst Moritz Hahn, Holger Hermanns, Nicolás Wolovick, and Lijun Zhang. Measurability and safety verification for stochastic hybrid systems. In Marco Caccamo, Emilio Frazzoli, and Radu Grosu, editors, *Proceedings of the 14th ACM International Conference on Hybrid Systems: Computation and Control (HSCC 2011)*, pages 43–52. ACM, 2011.

[FHR$^+$06] Martin Fränzle, Christian Herde, Stefan Ratschan, Tobias Schubert, and Tino Teige. Interval constraint solving using propositional SAT solving techniques. In Youssef Hamadi and Lucas Bordeaux, editors, *Proceedings of the CP 2006 First International Workshop on the Integration of SAT and CP Techniques*, pages 81–95, 2006.

[FHT$^+$07] Martin Fränzle, Christian Herde, Tino Teige, Stefan Ratschan, and Tobias Schubert. Efficient solving of large non-linear arithmetic constraint systems with complex Boolean structure. *Journal on Satisfiability, Boolean Modeling and Computation – Special Issue on SAT/CP Integration*, 1(3–4):209–236, 2007.

[FHT08] Martin Fränzle, Holger Hermanns, and Tino Teige. Stochastic satisfiability modulo theory: A novel technique for the analysis of probabilistic hybrid systems. In Magnus Egerstedt and Bud Mishra, editors, *Proceedings of the 11th International Conference on Hybrid Systems: Computation and Control (HSCC 2008)*, volume 4981 of *Lecture Notes in Computer Science*, pages 172–186. Springer, 2008.

[FHW10] Arnaud Fietzke, Holger Hermanns, and Christoph Weidenbach. Superposition-based analysis of first-order probabilistic timed automata. In Christian G. Fermüller and Andrei Voronkov, editors, *Proceedings of the 17th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-17)*, volume 6397 of *Lecture Notes in Computer Science*, pages 302–316. Springer, 2010.

[Frä99] Martin Fränzle. Analysis of hybrid systems: An ounce of realism can save an infinity of states. In Jörg Flum and Mario Rodríguez-Artalejo, editors, *Computer Science Logic (CSL 1999)*, volume 1683 of *Lecture Notes in Computer Science*, pages 126–140. Springer, 1999.

[Fre05] Goran Frehse. Phaver: Algorithmic verification of hybrid systems past hytech. In Manfred Morari and Lothar Thiele, editors, *Hybrid Systems: Computation and Control*, volume 3414 of *Lecture Notes in Computer Science*, pages 258–273. Springer, 2005.

[FTE10a] Martin Fränzle, Tino Teige, and Andreas Eggers. Engineering constraint solvers for automatic analysis of probabilistic hybrid automata. *Journal of Logic and Algebraic Programming*, 79(7):436–466, 2010.

[FTE10b] Martin Fränzle, Tino Teige, and Andreas Eggers. Satisfaction meets expectations: Computing expected values of probabilistic hybrid systems with SMT. In Dominique Méry and Stephan Merz, editors, *Proceedings of the 8th International Conference on Integrated Formal Methods (iFM 2010)*, volume 6396 of *Lecture Notes in Computer Science*, pages 168–182. Springer, 2010.

[GF06] Jürgen Greifeneder and Georg Frey. Probabilistic hybrid automata with variable step width applied to the analysis of networked automation systems. In *Proceedings of the 3rd IFAC Workshop on Discrete Event System Design (DESDes 2006)*, pages 283–288. IFAC, 2006.

[GGI$^+$10] Sicun Gao, Malay K. Ganai, Franjo Ivancic, Aarti Gupta, Sriram Sankaranarayanan, and Edmund M. Clarke. Integrating ICP and LRA solvers for deciding nonlinear real arithmetic problems. In *Proceedings of the 10th International Conference on Formal Methods in Computer-Aided Design (FMCAD 2010)*, pages 81–89. IEEE, 2010.

[GHM05] Judy Goldsmith, Matthias Hagen, and Martin Mundhenk. Complexity of DNF and isomorphism of monotone formulas. In Joanna Jedrzejowicz and Andrzej Szepietowski, editors, *Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science, MFCS 2005*, volume 3618 of *Lecture Notes in Computer Science*, pages 410–421. Springer, 2005.

[GJ90] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1990.

[GKvV95] Jan Frisco Groote, Wilco Koorn, and Sebastiaan van Vlijmen. The safety guaranteeing system at station Hoorn-Kersenboogerd. In *Conference on Computer Assurance*, pages 57–68. IEEE, 1995.

[GL04] William Glover and John Lygeros. A stochastic hybrid model for air traffic control simulation. In Rajeev Alur and George J. Pappas, editors, *Hybrid Systems: Computation and Control*, volume 2993 of *Lecture Notes in Computer Science*, pages 372–386. Springer, 2004.

[GN03] Evguenii I. Goldberg and Yakov Novikov. Verification of proofs of unsatisfiability for CNF formulas. In *Proceedings of the Design, Automation and Test in Europe Conference and Exposition (DATE 2003)*, pages 10886–10891. IEEE Computer Society, 2003.

[GNT03] Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. Backjumping for quantified Boolean logic satisfiability. *Artif. Intell.*, 145(1–2):99–120, 2003.

[HEFT08] Christian Herde, Andreas Eggers, Martin Fränzle, and Tino Teige. Analysis of hybrid systems using HySAT. In *The Third International Conference on Systems (ICONS 2008)*, pages 196–201. IEEE Computer Society, 2008.

[Her10] Christian Herde. *Efficient Solving of Large Arithmetic Constraint Systems with Complex Boolean Structure: Proof Engines for the Analysis of Hybrid Discrete–Continuous Systems.* Doctoral dissertation, Carl von Ossietzky Universität Oldenburg, Germany, 2010. Published by Vieweg+Teubner Verlag, 2011.

[Hes04] João P. Hespanha. Stochastic hybrid systems: Application to communication networks. In Rajeev Alur and George J. Pappas, editors, *Hybrid Systems: Computation and Control*, volume 2993 of *Lecture Notes in Computer Science*, pages 387–401. Springer, 2004.

[HH09] Arnd Hartmanns and Holger Hermanns. A Modest approach to checking probabilistic timed automata. In *Sixth International Conference on the Quantitative Evaluation of Systems (QEST 2009)*, pages 187–196. IEEE Computer Society, 2009.

[HHWZ10] Ernst Moritz Hahn, Holger Hermanns, Björn Wachter, and Lijun Zhang. PASS: Abstraction refinement for infinite probabilistic models. In Javier Esparza and Rupak Majumdar, editors, *16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2010)*, volume 6015 of *Lecture Notes in Computer Science*, pages 353–357. Springer, 2010.

[HJ94] Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal Asp. Comput.*, 6(5):512–535, 1994.

[HJMM04] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Kenneth L. McMillan. Abstractions from proofs. In Neil D. Jones and Xavier Leroy, editors, *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 232–244. ACM, 2004.

[HJvE01] Timothy J. Hickey, Qun Ju, and Maarten H. van Emden. Interval arithmetic: From principles to implementation. *Journal of the ACM*, 48(5):1038–1068, 2001.

[HK07] Tingting Han and Joost-Pieter Katoen. Counterexamples in probabilistic model checking. In Orna Grumberg and Michael Huth, editors, *13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 4424 of *Lecture Notes in Computer Science*, pages 72–86. Springer, 2007.

[HKD09] Tingting Han, Joost-Pieter Katoen, and Berteun Damman. Counterexample generation in probabilistic model checking. *IEEE Trans. Software Eng.*, 35(2):241–257, 2009.

[HKPV95] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*, pages 373–382. ACM, 1995.

[HLS00] Jianghai Hu, John Lygeros, and Shankar Sastry. Towards a theory of stochastic hybrid systems. In *Hybrid Systems: Computation and Control*, volume 1790 of *Lecture Notes in Computer Science*, pages 160–173. Springer, 2000.

[HNP$^+$11] Ernst Moritz Hahn, Gethin Norman, David Parker, Björn Wachter, and Lijun Zhang. Game-based abstraction and controller synthesis for probabilistic hybrid systems. In *Proceedings of the Eighth International Conference on Quantitative Evaluation of Systems (QEST 2011)*, pages 69–78. IEEE Computer Society, 2011.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes.* Series in Computer Science. Prentice Hall, 1985.

[Hoe63] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[HWZ08] Holger Hermanns, Björn Wachter, and Lijun Zhang. Probabilistic CEGAR. In Aarti Gupta and Sharad Malik, editors, *20th International Conference on Computer Aided Verification (CAV 2008)*, volume 5123 of *Lecture Notes in Computer Science*, pages 162–175. Springer, 2008.

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[KÁS$^+$10] Natalia Kalinnik, Erika Ábrahám, Tobias Schubert, Ralf Wimmer, and Bernd Becker. Exploiting different strategies for the parallelization of an SMT solver. In Manfred Dietrich, editor, *GI/ITG/GMM Workshop "Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen"*, pages 97–106. Fraunhofer Verlag, 2010.

[KB11a] Stefan Kupferschmid and Bernd Becker. Craig interpolation in the presence of non-linear constraints. In Uli Fahrenberg and Stavros Tripakis, editors, *Proceedings of the 9th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2011)*, volume 6919 of *Lecture Notes in Computer Science*, pages 240–255. Springer, 2011.

[KB11b] Stefan Kupferschmid and Bernd Becker. Craigsche Interpolation für Boolesche Kombinationen linearer und nichtlinearer Ungleichungen. In Frank Oppenheimer, editor, *Proceedings of the 14th Workshop "Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen" (MBMV 2011)*, pages 279–288. OFFIS, 2011.

[KBTF11] Stefan Kupferschmid, Bernd Becker, Tino Teige, and Martin Fränzle. Proof certificates and non-linear arithmetic constraints. In *Proceedings of the 14th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS 2011)*, pages 429–434. IEEE, 2011.

[Ked08] Nadine Keddis. Strong satisfaction. BSc thesis, Albert-Ludwigs-Universität Freiburg, Germany, 2008.

[Kle52] Stephen Cole Kleene. *Introduction to Metamathematics.* D. Van Nostrand Co./North Holland Co., New York, 1952.

[KNP11] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Prism 4.0: Verification of probabilistic real-time systems. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV 2011)*, volume 6806 of *Lecture Notes in Computer Science*, pages 585–591. Springer, 2011.

[KR06] Xenofon D. Koutsoukos and Derek Riley. Computational methods for reachability analysis of stochastic hybrid systems. In João P. Hespanha and Ashish Tiwari, editors, *Hybrid Systems: Computation and Control*, volume 3927 of *Lecture Notes in Computer Science*, pages 377–391. Springer, 2006.

[KSÁ$^+$09] Natalia Kalinnik, Tobias Schubert, Erika Ábrahám, Ralf Wimmer, and Bernd Becker. Picoso – a parallel interval constraint solver. In Hamid R. Arabnia, editor, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 473–479. CSREA Press, 2009.

[KTBF09] Stefan Kupferschmid, Tino Teige, Bernd Becker, and Martin Fränzle. Proofs of unsatisfiability for mixed Boolean and non-linear arithmetic constraint formulae. In Carsten Gremzow and Nico Moser, editors, *Proceedings of the 12th Workshop "Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen" (MBMV 2009)*, pages 27–36. Technische Universität Berlin, 2009.

[LDB10] Axel Legay, Benoît Delahaye, and Saddek Bensalem. Statistical model checking: An overview. In Howard Barringer, Yliès Falcone, Bernd Finkbeiner, Klaus Havelund, Insup Lee, Gordon J. Pace, Grigore Rosu, Oleg Sokolsky, and Nikolai Tillmann, editors, *Proceedings of the First International Conference Runtime Verification (RV 2010)*, volume 6418 of *Lecture Notes in Computer Science*, pages 122–135. Springer, 2010.

[Lee59] C. Y. Lee. Representation of switching circuits by binary-decision programs. *Bell System Technical Journal*, 38:985–999, 1959.

[Lit99] Michael L. Littman. Initial experiments in stochastic satisfiability. In *Proceedings of the 16th National Conference on Artificial Intelligence*, pages 667–672, 1999.

[LMP01] Michael L. Littman, Stephen M. Majercik, and Toniann Pitassi. Stochastic Boolean satisfiability. *Journal of Automated Reasoning*, 27(3):251–296, 2001.

[LMS04] Inês Lynce and João P. Marques-Silva. On computing minimum unsatisfiable cores. In *Online Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT 2004)*, 2004.

[LMS$^+$09] Matthew D. T. Lewis, Paolo Marin, Tobias Schubert, Massimo Narizzano, Bernd Becker, and Enrico Giunchiglia. PaQuBE: Distributed QBF solving with advanced knowledge sharing. In Oliver Kullmann, editor, *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing (SAT 2009)*, volume 5584 of *Lecture Notes in Computer Science*, pages 509–523. Springer, 2009.

[LSB$^+$11] Matthew D. T. Lewis, Tobias Schubert, Bernd Becker, Paolo Marin, Massimo Narizzano, and Enrico Giunchiglia. Parallel QBF solving with advanced knowledge sharing. *Fundam. Inform.*, 107(2–3):139–166, 2011.

[Maj04] Stephen M. Majercik. Nonchronological backtracking in stochastic Boolean satisfiability. In *16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004)*, pages 498–507. IEEE Computer Society, 2004.

[Maj07] Stephen M. Majercik. APPSSAT: Approximate probabilistic planning using stochastic satisfiability. *Int. J. Approx. Reasoning*, 45(2):402–419, 2007.

[Maj09] Stephen M. Majercik. Stochastic Boolean satisfiability. In Biere et al. [BHvMW09], chapter 27, pages 887–925.

[Mat70] Yuri V. Matiyasevich. Enumerable sets are Diophantine. *Soviet Math. Dokl.*, 11(2):354–357, 1970.

[McM03] Kenneth L. McMillan. Interpolation and SAT-based model checking. In Warren A. Hunt Jr. and Fabio Somenzi, editors, *Proceedings of the 15th International Conference on Computer Aided Verification (CAV 2003)*, volume 2725 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2003.

[McM05a] Kenneth L. McMillan. Applications of Craig interpolants in model checking. In Nicolas Halbwachs and Lenore D. Zuck, editors, *Proceedings of the 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2005)*, volume 3440 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2005.

[McM05b] Kenneth L. McMillan. An interpolating theorem prover. *Theor. Comput. Sci.*, 345(1):101–121, 2005.

[ML98a] Stephen M. Majercik and Michael L. Littman. MAXPLAN: A new approach to probabilistic planning. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems*, pages 86–93. AAAI, 1998.

[ML98b] Stephen M. Majercik and Michael L. Littman. Using caching to solve larger probabilistic planning problems. In *Proceedings of the Fifteenth National*

Conference on Artificial Intelligence (AAAI 1998), pages 954–959. AAAI, 1998.

[ML03] Stephen M. Majercik and Michael L. Littman. Contingent planning under uncertainty via stochastic satisfiability. *Artificial Intelligence Special Issue on Planning with Uncertainty and Incomplete Information*, 147(1-2):119–162, 2003.

[MMZ$^+$01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference*, pages 530–535. ACM, 2001.

[Moo66] Ramon E. Moore. *Interval Analysis.* Prentice Hall, Englewood Cliffs, NJ, 1966.

[Moo79] Ramon E. Moore. *Methods and Applications of Interval Analysis.* SIAM, Philadelphia, 1979.

[Moo80] Ramon E. Moore. Interval methods for nonlinear systems. *Computing*, Suppl. 2:113–120, 1980.

[MPL11] João Martins, André Platzer, and João Leite. Statistical model checking for distributed probabilistic-control hybrid automata with smart grid applications. In Shengchao Qin and Zongyan Qiu, editors, *Proceedings of the 13th International Conference on Formal Engineering Methods (ICFEM 2011)*, volume 6991 of *Lecture Notes in Computer Science*, pages 131–146. Springer, 2011.

[MSLM09] João P. Marques-Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. In Biere et al. [BHvMW09], chapter 4, pages 131–153.

[Neu90] Arnold Neumaier. *Interval Methods for Systems of Equations.* Cambridge University Press, Cambridge, 1990.

[Pap85] Christos H. Papadimitriou. Games against nature. *J. Comput. Syst. Sci.*, 31(2):288–301, 1985.

[Pap94] Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[PBLB03] Giordano Pola, Manuela L. Bujorianu, John Lygeros, and Maria Di Benedetto. Stochastic hybrid models: An overview with application to air traffic management. In *Proceedings of the 1st IFAC Conference on Analysis and Design of Hybrid Systems (ADHS 2003)*, pages 45–50. IFAC, 2003.

[PHLS00] Maria Prandini, Jianghai Hu, John Lygeros, and Shankar Sastry. A probabilistic approach to aircraft conflict detection. *IEEE Transactions on Intelligent Transportation Systems*, 1(4):199–220, 2000.

[Pic09] Clifford A. Pickover. *The Math Book: From Pythagoras to the 57th Dimension, 250 Milestones in the History of Mathematics*. Sterling, 1 edition, 2009.

[Pla11] André Platzer. Stochastic differential dynamic logic for stochastic hybrid programs. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *Proceedings of the 23rd International Conference on Automated Deduction (CADE-23)*, volume 6803 of *Lecture Notes in Computer Science*, pages 446–460. Springer, 2011.

[Pud97] Pavel Pudlák. Lower bounds for resolution and cutting plane proofs and monotone computations. *Journal of Symbolic Logic*, 62(3):981–998, 1997.

[PW07a] Andreas Podelski and Silke Wagner. Region stability proofs for hybrid systems. In Jean-François Raskin and P. S. Thiagarajan, editors, *Proceedings of the 5th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2007)*, volume 4763 of *Lecture Notes in Computer Science*, pages 320–335. Springer, 2007.

[PW07b] Andreas Podelski and Silke Wagner. A sound and complete proof rule for region stability of hybrid systems. In Alberto Bemporad, Antonio Bicchi, and Giorgio C. Buttazzo, editors, *Proceedings of the 10th International Workshop on Hybrid Systems: Computation and Control (HSCC 2007)*, volume 4416 of *Lecture Notes in Computer Science*, pages 750–753. Springer, 2007.

[Rob65] John Alan Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.

[Rot96] Dan Roth. On the hardness of approximate reasoning. *Artif. Intell.*, 82(1–2):273–302, 1996.

[RS07] Stefan Ratschan and Zhikun She. Safety verification of hybrid systems by constraint propagation-based abstraction refinement. *ACM Trans. Embedded Comput. Syst.*, 6(1), 2007.

[SA11] Sadegh Esmaeil Zadeh Soudjani and Alessandro Abate. Adaptive gridding for abstraction and verification of stochastic hybrid systems. In *Proceedings of the Eighth International Conference on Quantitative Evaluation of Systems (QEST 2011)*, pages 59–68. IEEE Computer Society, 2011.

[Sch78] Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, pages 216–226. ACM, 1978.

[Sch08] Christian Schmitt. Bounded model checking of probabilistic hybrid automata. Diplomarbeit (Diploma thesis), Carl von Ossietzky Universität Oldenburg, Germany, 2008.

[Sha49] Claude E. Shannon. The synthesis of two-terminal switching circuits. *Bell System Technical Journal*, 28:59–98, 1949.

[Spr00] Jeremy Sproston. Decidable model checking of probabilistic hybrid automata. In Mathai Joseph, editor, *Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT)*, volume 1926 of *Lecture Notes in Computer Science*, pages 31–45. Springer, 2000.

[Spr01] Jeremy Sproston. *Model Checking for Probabilistic Timed and Hybrid Systems.* PhD thesis, School of Computer Science, University of Birmingham, 2001.

[TEF11] Tino Teige, Andreas Eggers, and Martin Fränzle. Constraint-based analysis of concurrent probabilistic hybrid systems: An application to networked automation systems. *Nonlinear Analysis: Hybrid Systems*, 5(2):343–366, 2011.

[TF08] Tino Teige and Martin Fränzle. Stochastic satisfiability modulo theories for non-linear arithmetic. In Laurent Perron and Michael A. Trick, editors, *Proceedings of the 5th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2008)*, volume 5015 of *Lecture Notes in Computer Science*, pages 248–262. Springer, 2008.

[TF09] Tino Teige and Martin Fränzle. Constraint-based analysis of probabilistic hybrid systems. In Alessandro Giua, Cristian Mahulea, Manuel Silva, and Janan Zaytoon, editors, *Proceedings of the 3rd IFAC Conference on Analysis and Design of Hybrid Systems*, pages 162–167. IFAC, 2009.

[TF10] Tino Teige and Martin Fränzle. Resolution for stochastic Boolean satisfiability. In Christian G. Fermüller and Andrei Voronkov, editors, *Proceedings of the 17th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-17)*, volume 6397 of *Lecture Notes in Computer Science*, pages 625–639. Springer, 2010.

[TF11] Tino Teige and Martin Fränzle. Generalized Craig interpolation for stochastic Boolean satisfiability problems. In Parosh Aziz Abdulla and K. Rustan M. Leino, editors, *Proceedings of the Seventeenth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2011)*, volume 6605 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 2011.

[TF12] Tino Teige and Martin Fränzle. Generalized Craig interpolation for stochastic Boolean satisfiability problems with applications to probabilistic state reachability and region stability. *Logical Methods in Computer Science*, 8(2:16):1–32, 2012.

[THF$^+$07] Tino Teige, Christian Herde, Martin Fränzle, Natalia Kalinnik, and Andreas Eggers. A generalized two-watched-literal scheme in a mixed Boolean and non-linear arithmetic constraint solver. In José Neves, Manuel Filipe Santos, and José Manuel Machado, editors, *Proceedings of the 13th Portuguese Conference on Artificial Intelligence (EPIA 2007)*, New Trends in Artificial Intelligence, pages 729–741. APPIA, 2007.

[THFA08] Tino Teige, Christian Herde, Martin Fränzle, and Erika Ábrahám. Conflict analysis and restarts in a mixed Boolean and non-linear arithmetic constraint solver. Reports of SFB/TR 14 AVACS 34, SFB/TR 14 AVACS, 2008. ISSN: 1860-9821, `http://www.avacs.org`.

[Tij03] Henk C. Tijms. *A First Course on Stochastic Models.* John Wiley & Sons, 2003.

[Tse68] Gregory S. Tseitin. On the complexity of derivations in the propositional calculus. *Studies in Constructive Mathematics and Mathematical Logics*, Part II:115–125, 1968.

[Tur37] Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 1937.

[Wal00] Toby Walsh. SAT v CSP. In Rina Dechter, editor, *Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming (CP 2000)*, volume 1894 of *Lecture Notes in Computer Science*, pages 441–456. Springer, 2000.

[Wal02] Toby Walsh. Stochastic constraint programming. In Frank van Harmelen, editor, *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI 2002)*, pages 111–115. IOS Press, 2002.

[WBB09] Ralf Wimmer, Bettina Braitling, and Bernd Becker. Counterexample generation for discrete-time Markov chains using bounded model checking. In Neil D. Jones and Markus Müller-Olm, editors, *10th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI 2009)*, volume 5403 of *Lecture Notes in Computer Science*, pages 366–380. Springer, 2009.

[WZH07] Björn Wachter, Lijun Zhang, and Holger Hermanns. Probabilistic model checking modulo theories. In *Fourth International Conference on the Quantitative Evaluation of Systems (QEST 2007)*, pages 129–140. IEEE Computer Society, 2007.

[You05a] Håkan Lorens Samir Younes. *Verification and planning for stochastic processes with asynchronous events.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 2005.

[You05b] Håkan Lorens Samir Younes. Ymer: A statistical model checker. In Kousha Etessami and Sriram K. Rajamani, editors, *Proceedings of the 17th International Conference on Computer Aided Verification (CAV 2005)*, volume 3576 of *Lecture Notes in Computer Science*, pages 429–433. Springer, 2005.

[YS06] Håkan Lorens Samir Younes and Reid G. Simmons. Statistical probabilistic model checking with a focus on time-bounded properties. *Inf. Comput.*, 204(9):1368–1409, 2006.

[ZM03a] Lintao Zhang and Sharad Malik. Extracting small unsatisfiable cores from unsatisfiable Boolean formulas. In *Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT 2003)*, 2003.

[ZM03b] Lintao Zhang and Sharad Malik. Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications. In *Proceedings of the Design, Automation and Test in Europe Conference and Exposition (DATE 2003)*, pages 10880–10885. IEEE Computer Society, 2003.

[ZMMM01] Lintao Zhang, Conor F. Madigan, Matthew W. Moskewicz, and Sharad Malik. Efficient conflict driven learning in a Boolean satisfiability solver. In *Proceedings of the 2001 IEEE/ACM International Conference on Computer-Aided Design*, pages 279–285. IEEE Press, 2001.

[ZPC10] Paolo Zuliani, André Platzer, and Edmund M. Clarke. Bayesian statistical model checking with application to Simulink/Stateflow verification. In Karl Henrik Johansson and Wang Yi, editors, *Hybrid Systems: Computation and Control*, pages 243–252. ACM, 2010.

[ZSR$^+$10] Lijun Zhang, Zhikun She, Stefan Ratschan, Holger Hermanns, and Ernst Moritz Hahn. Safety verification for probabilistic hybrid systems. In Tayssir Touili, Byron Cook, and Paul Jackson, editors, *Proceedings of the 22nd International Conference on Computer Aided Verification, CAV 2010*, volume 6174 of *Lecture Notes in Computer Science*, pages 196–211. Springer, 2010.

Index