CARL VON OSSIETZKY universität OLDENBURG

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

Verification of Stochastic Systems by Stochastic Satisfiability Modulo Theories with Continuous Domain (CSSMT)

Dissertation zur Erlangung des Grades eines
Doktors der Ingenieurwissenschaften

genehmigte Dissertation

von **Yang Gao, M.Sc.**
geboren am 23.03.1987 in Shannxi, China

Gutachter:
Prof. Dr. Martin Fränzle
Prof. Dr. Paolo Zuliani

Tag der Disputation: 26.04.2017

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.

Yang Gao
November 2016

Acknowledgements

I would like to express my gratitude to all those who helped me during the writing of this thesis.

My most profound gratitude goes first and foremost to Prof. Dr. Martin Fränzle, my supervisor, for his constant encouragement and guidance. He has walked me through all the stages of the writing of this thesis. Without his consistent and illuminating instruction, this thesis could not have reached its present form. Martin is always kind, patient and helpful, his scientific insight and strong knowledge have brought me deeper into this field. I enjoy these years supervised by Martin.

Second, I would like to express my heartfelt gratitude to Prof. Dr. Ernst-Rüdiger Olderog, my second supervisor, who led me into the SCARE research training group and helped me from all kinds of work no matter how significant or trivial. I am also greatly indebted to the professors in the SCARE training group: Prof. Dr. Eike Best, Prof. Dr. Annegret Habel and Prof. Dr. Oliver Theel, who have instructed and helped me a lot in the past few years.

I would also like to extend my sincere gratitude to Dr. Paolo Zuliani and Dr.-Ing. Willem Hagemann, who have spent considerable efforts to review my thesis and provided me a lot of valuable advice during my Ph.D. defense.

Last my thanks would go to my beloved family for their thoughtful considerations, support from my home country and high confidence in me all through these years. I also owe my sincere gratitude to my friends and SCARE colleagues who gave me their help and time in listening to me and helping me work out my problems during the difficult course of the thesis.

Besides, this thesis is supported by the German Research Foundation through the Research Training Group DFG-GRK 1765: "System Correctness under Adverse Conditions" (SCARE, https://www.scare.uni-oldenburg.de).

Abstract

Stochastic Satisfiability Modulo Theories (SSMT) is a quantitative extension of Satisfiability Modulo Theories (SMT) inspired by stochastic logics. It extends SMT by randomized quantifiers, facilitating capture of stochastic game properties in the logic, like reachability analysis of hybrid-state Markov decision processes. Solving SSMT formulae with quantification over finite and thus discrete domain has been addressed by Tino Teige et al. A major limitation of the SSMT solving approach is that all quantifiers (except for implicit innermost existential quantification of all otherwise unbound variables) are confined to range over finite domains. As this implies that the support of probability distributions have to be finite, a large number of phenomena cannot be expressed within the SSMT framework, such as measurement error in hybrid systems. To overcome this limitation, this thesis relaxes the constraints on the domains of randomized variables, now also admitting dense probability distributions in SSMT solving, which yields SSMT over continuous quantifier domains (CSSMT).

In this thesis, we firstly extend the semantics of SSMT and introduce a rule-based solving procedure, which is an integration of SMT reasoning, constraint solving and probability analysis. The possibilities of algorithmic enhancements are then developed to improve the basic solving procedure for CSSMT. As applications, the corresponding prototype solver CSiSAT is introduced and case studies from different fields are performed to demonstrate the feasibility of the approach.

Zusammenfassung

Inspiriert durch stochastische Logiken wurde kürzlich eine quantitative Erweiterung der existierenden *Satisfiability Modulo Theory* (SMT) entwickelt. Diese, sogenannte *Stochastic Satisfiability Modulo Theory*, erweitert die SMT, indem neben den All- und Existenz-Quantoren weitere, randomisierte Quantoren eingeführt wurden. Hierdurch wird insbesondere das Erfassen von Eigenschaften von stochastischen Spielen, wie beispielsweise die probabilistische Erreichbarkeit in Markov Entscheidungsprozessen mit hybriden, diskret-kontinuiertlichen Zustandsräumen, erleichtert. Das Lösen der entsprechenden SSMT Formeln unter der Einschränkung von diskreten Quantoren wurde bereits durch Tino Teige et al. adressiert. Einer der größten limitierenden Faktoren solcher Ansätze ist, dass die Quantoren (bis auf einen impliziten Existenzquantor über alle freien Variablen der Formel) auf beschränkte Bereiche über endlichen Grundmengen limitiert sind. Somit können insbesondere nur Wahrscheinlichkeitsverteilungen über endlichen Träger betrachtet werden. Eine Vielzahl von anwendungsrelevanten Phenomenen wie Messfehler sind jedoch besser durch kontinuierliche Verteilungen beschrieben. Um solche Verteilungen in existierende SSMT Methoden zu integrieren, soll in dieser Arbeit die Erweiterung *SSMT over continuous quantifier domains* (CSSMT) entwickelt werden, welche die erwähnten Einschränkungen relaxiert, sodass nun auch SSMT Ausdrücke behandelt werden können welche Verteilungen mit kontinuierlichem Träger beinhalten.

In der vorliegenden Arbeit wird zunächst die unterliegende Semantik der SSMT Formeln erweitert sowie eine regel-basierte Vorgehen zum Lösen entwickelt, welches SMT Schlussfolgern, *constraint sovling* und Methoden der stochastischen Analysis kombiniert. Des Weiteren werden algorithmische Erweiterungen zur Steigerung der Effizienz der Basislösungsmethode. Die praktische Anwendbarkeit von CSSMT zur Verifikation von Systemen mit stochastischen Verhalten wird auf mehreren Fallstudien unterschiedlicher Bereiche mittels einer prototypischen Implementierung CSiSAT gezeigt.

Table of contents

List of figures

List of tables

Nomenclature

Roman Symbols

F_X	distribution function of X
$\mathbb{IR}$	set of intervals
$\mathbb{R}$	Reals
$\mathbb{Z}$	Integers

Greek Symbols

$(\Omega, \mathcal{F}, \mu)$	a sample space Ω with a σ-field $\mathcal{F}$ and measure μ
π_X	density function for X

Other Symbols

$\mathcal{E}(\lambda)$	exponential distribution with rate parameter λ
$\uplus$	operator to compute interval hull
$\mathcal{N}(\mu, \sigma^2)$	normal distribution with mean μ and standard variance σ
$\mathcal{U}(a, b)$	uniform distribution with minimum and maximum bounds a and b

Acronyms / Abbreviations

CDCL	Conflict Driven Clause Learning
CNF	Conjunction Normal Form
CSL	Continuous Stochastic Logic
CSSMT	Stochastic Satisfiability Modulo Theory with Continuous Domain

CTMC	Continuous Time Markov Chain
DPLL	Davis-Putnam-Logemann-Loveland procedure
DTMC	Discrete Time Markov Chain
HA	Hybrid Automata
ICP	Interval Constraint Propagation
IG	Implication Graph
MDP	Markov Decision Process
ODE	Ordinary Differential Equation
PCTL	Probabilistic Computation Tree Logic
PDF	Probability Density Function
PDMP	Piecewise Deterministic Markov Process
PHA	Probabilistic Hybrid Automata
QoS	Quantity of Service
SAT	Boolean Satisfiability Problem
SDE	Stochastic Differential Equation
SDP	Switched Diffusion Process
SHA	Stochastic Hybrid Automata
SHA	Stochastic Hybrid Automaton
SMC	Statistical Model Checking
SMT	Satisfiability Modulo Theory
SSMT	Stochastic Satisfiability Modulo Theory

Chapter 1

Introduction

The rapid development of information technology, software and hardware engineering makes it possible to implement physical processes and design controllers with embedded computer systems, which are shown to be the most common form of computing devices today. The interactions of the virtual with the physical world range from traditional control applications, like controlling an automotive powertrain, over computer-controlled active safety systems, like the anti-locking brake, the electronic stability program, or recently pedestrian detection integrated with emergency braking capabilities, to the vision of cyber-physical networks bringing even remote physical processes into our sphere of control. Such widespread applications of embedded computer systems result in functionality critical in many aspects which are required in both design phase and implement phase: 1) critical to function – systems should behave correctly in most adverse conditions such as disturbance and uncertainty introduced by the environment they are interacting with or even by themselves; 2) critical to safety – correction capabilities should be considered when systems reveal failure or crash; 3) critical to performance – systems may be influenced by power consumption and quantity of service QoS; etc. Such critical requirements have attracted a large number of researchers and engineers who devote themselves to different aspects: from system behavior formalization, over modeling and simulation, to correctness and safety verification.

The complication of modern controllers makes verification and analysis very challenging tasks, which is mainly shown in the following aspects: 1) hybrid – the modern controller is a compound system with continuous controller and discrete supervisor which interacts with the physical plant by actuator and A/D (D/A) converter. The mixture of continuity and discreteness makes it impossible to use a simple mathematical model to analyse the whole behavior of a system, more precise models which consider the interaction of continuous parts and discrete parts should be emphasized; 2) nondeterminism – the system has chance to choose its next-step behavior from feasible actions, which are called nondeterministic choices. Nondeterminism is not captured by most of the classical mathematical models. However, nondeterminism is one of the critical aspects which influence system correctness, such as a car, which is an open system as we don't know what the driver will eventually command from it. In this sense nondeterminism should be employed to assure well-behavior of the system despite lacking knowledge of actual user interaction; 3) randomness – modern controllers are not isolated systems and are generally influenced by the environment, these interactions introduce noise and disturbance to systems, which cannot be ignored especially when systems are safety critical. So mathematical models which can capture such characteristic should seriously be considered.

Regarding to such complicated systems, one critical question arises, i.e., how correctly the system can behave. Due to the interaction of continuous and discrete behavior, the influence of deterministic and random behavior, the correctness of such systems should be carefully analyzed. Consider an open system which contains all the aspects we mentioned above, it is required to behave correctly under both open decision and random disturbance, however, the system is still suffering from some malfunctions, i.e., traps into bad status or even worse – crashes. The malfunctions are inevitable for a real system. However, we can require that the probability of malfunctions is low enough, under a given tolerance. That is exactly what we are going to deal with, i.e., we want to verify the correctness of a system, i.e., the probability that the system goes to bad states under any adverse conditions is under an acceptable quantity. To achieve this goal, a lot of formal models are investigated, so that the analysis can be performed on mathematical models, which are abstractions of real-world systems concentrating on the parts of interest.

Hybrid Automata (HA, [ACHH93]) is one of the well-known facilities which aims at modeling and analyzing systems with discrete and continuous behavior. Such models support qualitative behavioral verification in the sense of showing that a system behaving according to its dynamics would never be able to engage in an undesired behavior. Hybrid automata simulate the behavior of a system, the dynamics of the system is generally represented by ordinary differential equation (ODE) and the renewal of continuous variables, meanwhile, the discrete control actions are modeled by jumps between modes. Hybrid automata mainly focus on the systems without disturbance and noise, although such uncertainties can be encoded by intervals, in many applications such as wireless sensing, more concise quantitative information about the involved uncertainties is available in terms of probabilities. To incorporate this kind of information, both the underlying models and the corresponding analysis techniques have to be adapted. Verifying reachability and safety properties within this extended setting then corresponds to obtaining statements about the probability of these properties to be satisfied. For these purposes some variants of hybrid-system models have been suggested, like Probabilistic Hybrid Automata (PHA, [Spr00]) or Stochastic Hybrid Automata (SHA, [Jul06]). In such extensions, either discrete action can feature probabilistic branching or continuous evolution can evolve stochastically along, e.g., stochastic differential equations, or both.

The resulting models are very complicated to analyze due to the introduction of different undecidable sources, and the stochastic setting makes it even more complicated to handle. The exact probability of reaching the desired targets cannot be attained in this sense. However, approximation techniques can work on it efficiently. Instead of obtaining the exact probability, an admissible bound is given by such techniques to approximate the real solution, which is able to fit the bill in a lot of practical scenarios.

In the remainder of this thesis, we will introduce a constraint-based technique to handle the verification problems with regard to the hybrid systems with stochastic behavior.

1.1 Related Work

In this section, we will recap some state-of-art techniques related to stochastic system analysis, where the main ideas and corresponding tools are introduced, as well the relations with our work.

1.1.1 Probabilistic Model Checking

Probabilistic Model Checking is a formal verification technique for modeling and analyzing systems that exhibit probabilistic behavior [HKNP06], which is based on some probabilistic models, i.e., Discrete Time Markov Chain (DTMC), Continuous Time Markov Chains (CTMS) etc. Probabilistic model checking refers to a range of techniques for calculating the probability of occurrence of certain events during the execution of the system, and can be useful to establish properties such as “shutdown occurs with probability at most 0.01” and “the video frame will be delivered within 5 ms with probability at least 0.97” [KNP02]. Applications range from areas such as randomized distributed algorithms to planning and AI, security [NS03], and even biological process modelling [LP07], wireless sensor network [AGFT14] or quantum computing [FYY13]. The properties are formulated by some specific logic, like Probabilistic Computation Tree Logic (PCTL, [HJ94]), Continuous Stochastic Logic (CSL, [ASSB00]) and so on, which are probabilistic extensions of the classical temporal logics with considering the distributions, time, rewards and some other probability-related quantities.

PRISM [KNP11] is a well-known probabilistic model checker for formal modeling and analysis of systems that exhibit random or probabilistic behavior. It has been used to analyze systems in many different application domains, including communication and multimedia protocols, randomised distributed algorithms, security protocols, biological systems and many others. PRISM can build and analyze several types of probabilistic models such as CTMC, DTMC, Markov Decision Process (MDP) and so on. The property specification language incorporates the temporal logics PCTL, CSL, etc., as well as extensions for quantitative specifications and costs/rewards.

PRISM has been successfully applied in many applications due to its capability to integrate different probabilistic models, user-friendly modeling language and interactive graphical interface. However, PRISM does not support continuous variables and distributions in the following sense.

- PRISM is confined to finite-state systems, which makes continuous variables difficult to tackle. In PRISM, continuous states are not supported, so modeling continuous quantities should be done by using some other tricks, i.e., model the continuous variables as rewards or abstract the domain of continuous variables and discretize them, but these methods are not easy to be generalized.
- The definitions of DTMC/CTMC provide that the discrete transitions are issued with discrete distributions, which makes continuous uncertainties unable to be captured. PRISM supports different types of probabilistic models. However, all of them have the mentioned characteristics.

1.1.2 Stochastic Hybrid Automata

Stochastic hybrid automata are proposed to integrate probabilities into hybrid system, which lead to a number of different notions, each from a distinct perspective [AG97, Spr00, Buj04, BLB05, APLS08]. An important problem in hybrid systems is reachability analysis, which is to verify that whether a given system can reach a given target. Here the target can be either desired states or bad states we want to avoid. With introducing the probability, the previous problem should be slightly modified, like certifying that the probability of reaching some specified states remains above (or below) a given quantitative safety target.

Stochastic hybrid systems have found various applications in diverse areas and tool support for several subclasses, a framework featuring both probabilistic behavior and continuous-time dynamics as well as nondeterminism is developed by Hahn et al. [ZSR$^+$10], which relies on state-space discretization by safe abstraction. The idea is first to consider the non-probabilistic hybrid automaton obtained by replacing probabilistic branching with nondeterministic choices. Provided that there is a finite abstraction for this classical hybrid automaton, the method then decorates this abstraction with probabilities to obtain a probabilistic abstraction, namely a finite probabilistic automaton, which allows to verify probabilistic safety properties on the abstraction: if such a property holds in the abstraction, it also holds in the concrete system.

Otherwise, refinement of the abstraction is required to obtain a more precise result. Based on this framework, a prototype verifier ProHVer [HHHK13] is implemented, which is capable of computing the unbounded reachability probability for probabilistic hybrid automata. ProHVer maintains an over-approximation of the original probabilistic hybrid automata, which can then compute a probability that is an upper bound for the maximal reachability probability for the property given in the probabilistic hybrid automaton. However, exploiting ProHVer for computing the abstractions is limited to linear dynamics and can handle even that only via an over-approximation by piecewise constant differential inclusions. Moreover, due to the manual selection of abstraction, the user must possess in-depth knowledge on such topics, for some problems, numbers of iterations should be performed to get acceptable results, which makes the tool not very easy to use. Our method to verify probabilistic systems is based on constraints solving. The behavior of interest can be encoded to constraints, which can be even nonlinear constraints with random variables. Then the verification procedure is performed on the constraints. Such framework is relatively intuitive and can be easily generalized, moreover, it doesn't require deep knowledge of users.

Another variant of stochastic hybrid system is discussed by Fedor Shmarov, Paolo Zuliani et al. [SZ15] where the hybrid systems equipped with random initial parameters are considered. The framework is mainly based on safe gridding and δ-complete decision procedure: firstly, the random initial parameters are split according to a given numerical error bound δ, i.e., the partition can be guaranteed that the resulted probability estimation is acceptable regarding the given precision. Then the bounded reachability in hybrid systems is encoded as a first-order logic formula ϕ, the satisfiability of the resulted formula should be decided, if $\phi(x)$ is $true$, the given initial value x will lead to the set B we are interested, yet $false$ means x will lead to the region out of B. Instead of determining the original formula $\phi(x)$, in their work, a relaxed formula $\phi([x])$ is verified by using the notion of δ-complete decision procedure [GKC13], which evaluates a weaker formula, i.e., on the safe partition. For an interval $[x]$ obtained by the partition, the δ-complete decision procedure concludes: if $\phi([x])$ is $unsat$, all points of $[x]$ will not lead to B for sure, such intervals $[x]$ can be used for calculating the probability over-approximation, if its complement complementary formula $\phi^C([x])$ is $unsat$, the entire $[x]$ lies in set B for sure, which can be used to calculating the probability under-approximation, for the other two cases – $\phi([x])$ or $\phi^C([x])$ is sat, $[x]$ should be partitioned and verified again. Finally, the lower and upper approximation for the reachability probability can be obtained. All the ideas have been implemented in ProbReach [SZ15]. The drawbacks of this framework are obvious: the δ-complete decision procedure should be applied on each partitioned interval, it is quite a number of computation. For high dimension problems, even an exponential explosion is suffered; another issue is that such method only deals with the hybrid systems with random initial parameters, however, the stochastic dynamics is not discussed. Thus we introduce another mechanism for checking satisfiability of formulas with random variables, which is based on branching and heuristic pruning technique so that we don't need to traverse all the domain. Moreover, the stochastic dynamics can also be encoded and solved by our method.

1.1.3 Satisfiability Modulo Theories

Another pertinent technique to analyze stochastic hybrid system is based on constraint solving for stochastic logic involving arithmetic, which encodes the behavior and properties to some constraints and solves the resulted combination of constraints by corresponding decision procedure. Such framework works on the extension of classical logics like Boolean Satisfiability Problem (SAT) and Satisfiability Modulo Theory (SMT) [BHvM09] with stochastic consideration. Our work resides in this branch.

The first idea of modeling uncertainty using randomized quantification was proposed within the framework of SAT by Papadimitriou, yielding Stochastic SAT (SSAT) featuring both classical quantifiers and randomized quantifiers [Pap85]. This work has been lifted to SMT by Fränzle, Teige et al. [FHT08, TF08] in order to symbolically reason about reachability problems of probabilistic hybrid automata (PHA). Instead of reporting true or false, an SSAT/SSMT formula Φ has a probability as semantics, which denotes the probability of satisfaction of Φ under an optimal resolution of the non-random quantifiers. SSAT and SSMT permit concise description of diverse problems combining reasoning under uncertainty with data dependencies. Applications range from AI planning [ML98, LMP01, ML99] to analysis of PHA [FHT08].

A serious limitation of the SSMT-solving approach pioneered by Teige [Tei12] is that all quantifiers (except for implicit innermost existential quantification of all otherwise unbound variables) are confined to range over finite domains. As this implies that the carriers of probability distributions have to be finite, a large number of phenomena cannot be expressed within the current SSMT framework, such as continuous noise or measurement error in hybrid systems. To overcome this limitation, we relax the constraints on the domains of randomized variables, now also admitting continuous probability distributions in SSMT solving [GF15].

Another work extending SSMT to continuous domains is by Ellen et al. [EGF14], proposing a statistical solving technique adopted from statistical AI planning algorithms and thus only being able to offer stochastic guarantees.

1.2 Contributions

SSMT and its original solving procedure were proposed by Fränzle, Hermanns, and Teige [FHT08] based on SSAT [Pap85], SMT solving by DPLL($\mathcal{T}$) [NOT06], and the iSAT algorithm [FHT$^+$07] for solving non-linear arithmetic constraint systems by using interval constraint propagation. The original formulation of SSMT is confined to finite quantifier domains. This Ph.D. work extends the semantics of SSMT so that continuous probability distributions are supported. Due to the undecidability of satisfiability with real arithmetics, we are focusing on the solving procedure with safe bounds. The contributions contain:

- We extend the semantics of SSMT with supporting continuous distributions (Continuous SSMT, CSSMT) and propose corresponding solving procedure, which is based on a combination of the CDCL($\mathcal{T}$) (Constraint Driven Clause Learning, [NOT06]) and ICP (Interval Constraint Propagation, [RVBW06, VHMK97]) algorithms, as first implemented in the iSAT solver for rich arithmetic SMT problems over $\mathbb{R}^n$ [FHT$^+$07], and on branch-and-prune rules for the quantifiers generalizing those suggested in [FHT$^+$07, Tei12]. We extend these methods so that they can deal with the SSMT formula with continuous quantifier domains. Our solving procedure therefore is divided into three layers: an SMT layer manipulating the Boolean structure of the "matrix"[1] of the formula, an interval constraint solving layer reasoning over the conjunctive constraint systems in the theory part of the formula, and a stochastic SMT layer reasoning about the quantifier prefix. Each layer is defined by a set of rules to generate, split, and combine so-called *computation cells*, where a computation cell is a box-shaped part of the $\mathbb{R}^n$, i.e., the problem domain of the constraints. The solver thereby approximates the exact satisfaction probability of the formula under investigation and terminates with a conclusive result whenever the approximation gets tight enough to conclusively answer the question whether the satisfaction probability is above or below a certain specified target.

- We implement a prototype solver CSiSAT in C++, which is a satisfiability solver for non-linear constraints with random variables, i.e., CSSMT formulae. CSiSAT takes CSSMT formulae as input and returns safe bounds to estimate probability of satisfaction, which can be applied to verify reachability properties in stochastic hybrid systems.

- We apply the theory of CSSMT and the solver CSiSAT to some application scenarios in different aspects, i.e., controller design, tasks scheduling, path planning and so on. All

[1]In SSAT parlance, this is the body of the formula after rewriting it to prenex form and stripping all the quantifiers.

the applications consider the uncertainties in the systems and analyze the properties of interest benefit from stochastic constraints solving.

1.3 Structure of the thesis

This thesis is logically divided into three parts:

- Chapter 3-5 introduce the semantics of continuous SSMT, we start with an introduction for classical satisfiability theory (SMT) and its stochastic extension – stochastic SMT, which are recapped in Chapter 3. Due to the limitation that stochastic SMT only supports discrete distributions we further extend its semantics to continuous domain (CSSMT), which is formalized in Chapter 4. The potential application of CSSMT is stressed in Chapter 5 by introducing bounded reachability problem and its relation to CSSMT.

- Chapter 6 and Chapter 7 focus on the solving procedure for CSSMT, where the rule-based solving procedure is discussed in Chapter 6, and the possibilities of algorithmic enhancements are further developed in Chapter 7.

- Chapter 8 and Chapter 9 can be categorized into the application domain. The prototype tool CSiSAT is introduced in Chapter 8 and how to use CSiSAT to solve practical problems are discussed in Chapter 9.

Apart from this, foundations and notations are briefly reviewed in Chapter 2 and finally in Chapter 10 we summarize the thesis and discuss promising points for future work.

Chapter 2

Foundations and Notations

In this chapter, we will briefly recap foundations and notations used in the following parts. Probability theory is our main basis and interval arithmetic contributes to the solving procedure; we will also review some basic conceptions of satisfiability modulo theory (SMT), which is regarded as our start point.

2.1 Probability Basis

Let Ω be a *sample set*, the set of all possible outcomes of an experiment. A pair $(\Omega, \mathcal{F})$ is said to be a *sample space* if $\mathcal{F}$ is a σ-field of subsets of Ω. A triple $(\Omega, \mathcal{F}, \mu)$ is a *probability space* if μ is a probability measure over $\mathcal{F}$: 1) $0 \leq \mu(A) \leq 1$ for all $A \in \mathcal{F}$; 2) $\mu(\emptyset) = 0$ and $\mu(\Omega) = 1$; 3) $\mu(\bigcup_{k=1}^{\infty} A_k) = \Sigma_{k=1}^{\infty} \mu(A_k)$ for disjoint $A_k \in \mathcal{F}$.

Let $(\Omega, \mathcal{F}, \mu)$ be a probability space. A function $X : \Omega \to \mathbb{R}$ is said to be a random variable iff $X^{-1}(B) \in \mathcal{F}$ for all $B \in \mathcal{B}$, where $\mathcal{B}$ is the Borel σ-algebra generated by all the open sets in $\mathbb{R}$. The *distribution function* of X is the function $F_X : \mathbb{R} \to [0,1]$ defined by $F_X(x) := \mu(X \leq x)$ for all $x \in \mathbb{R}$. If there exists a nonnegative, integrable function $\pi : \mathbb{R} \to \mathbb{R}$ such that $F_X(x) := \int_{-\infty}^{x} \pi(y)dy$, then π is called the *density function* for X. It follows then that $\mu(X \in A) = \int_A \pi(x)dx$ for all $A \in \mathcal{F}$.

In order to achieve a uniform treatment of discrete and continuous random variables, we equip $\mathbb{R}$ with the Lebesgue measure and $\mathbb{Z}$ with cardinality of its subsets as a measure. Given this convention, we can uniformly write $\int_A \pi(x)dx$ for determining the probability mass assigned by density (or, in the discrete case, distribution) π to the set A, as the measure is understood. Note that in the discrete case, $\int_A \pi(x)dx = \sum_{x \in A} \pi(x)$ due to the particular choice of the measure for $\mathbb{Z}$. This permits us to uniformly treat densities over the continuum and distributions over discrete carriers as densities.

Some frequently used densities are given below, such notations will also be used in the rest parts:

- $\mathcal{N}(\mu,\sigma^2)$, normal distribution with mean μ and standard variance σ, the probability density function (PDF) is:

$$\pi(x;\mu,\sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

- $\mathcal{U}(a,b)$, uniform distribution with minimum and maximum bounds a and b, the corresponding PDF is:

$$\pi(x;a,b) = \begin{cases} \frac{1}{b-a} & \text{for } a \le x \le b \\ 0 & \text{otherwise} \end{cases}$$

- $\mathcal{E}(\lambda)$, exponential distribution with rate parameter λ, where the CDF is:

$$\pi(x;\lambda) = \begin{cases} 1 - e^{-\lambda x} & x \ge 0 \\ 0 & x < 0 \end{cases}$$

Above we list some continuous distributions, other distributions and discrete distributions will be mentioned when they are used in the rest of thesis.

2.2 Interval Arithmetic and Constraint Solving

Interval arithmetic is a mathematical technique systematized by Rosalind Cecily Young in the 1930s [You31] and developed by mathematicians since the 1950s [S+09, MM79, AH84] as an approach to putting safe bounds on errors in numerical computation. It has later been extended to handle arithmetic constraints over continuous domains [BG06], where it enables efficient search for approximate solutions to large-scale non-linear constraint systems. Our main framework is essentially based on interval arithmetic and interval constraint propagation, so we will recap the main concepts in this section.

2.2.1 Interval Arithmetic

Let $\mathbb{R}$ denote the reals. A real interval X is defined as the set of real numbers between (or including) a given upper and lower bound, i.e. $X = [\underline{X}, \overline{X}] = \{x \in \mathbb{R} \mid \underline{X} \le x \le \overline{X}\}$ (open intervals can be defined similarly by using strict less equal operator), and the set of intervals is denoted by $\mathbb{IR}$. Interval arithmetic is a lifting of real arithmetic. For an arithmetic operation $\circ$ like $+$, $-$ or $\times$, the corresponding interval operation on intervals X and Y is defined by $X \circ Y = \mathrm{hull}(\{x \circ y \mid x \in X, y \in Y\})$, where hull(X) denotes the smallest computer-representable interval covering X. Pay attention to the case that X is divided by Y containing zero, models should be modified by extending reals $\mathbb{R}^\infty = \mathbb{R} \cup \{-\infty, +\infty\}$, in this case, divided by Y yields two intervals both containing infinity. Elementary functions are defined in the same way.

Sometimes it is useful to represent the variables in a vector manner, i.e., the variables $x_1, \ldots, x_n$ are denoted by the vector $\mathbf{x} = (x_1, \ldots, x_n)$. The domain of the variables are then denoted by the n-dimension box $\mathbf{X} = (X_1, \ldots, X_n)$, which is to say that the domain of x_i is X_i.

Interval arithmetic is often used to compute the range of a real function over a domain, a real function $f : \mathbb{R}^n \to \mathbb{R}$ can be lifted to an interval function $F : \mathbb{IR}^n \to \mathbb{IR}$ if for every $I \in X^n$ the inclusion $\{f(x) : x \in I\} \subseteq F(X)$ holds.

2.2.2 Constraint Solving

Interval arithmetic has been developed into interval constraint propagation (ICP, [RVBW06, VHMK97, GB06]) as a means of solving systems of real equalities and inequalities. Given a set of real constraints C and initial interval bounds $\rho : Vars \to \mathbb{IR}$ on their variables, ICP successively narrows the initial intervals to small intervals still covering all real solutions to the constraint system.

Example 2.2.1. *Consider the constraint c: $x_1 + x_2 = x_3$ with initially $\rho(x_1) = (-\infty, 5]$, $\rho(x_2) = (-\infty, 4]$ and $\rho(x_3) = [6, \infty)$. ICP modifies the domain of variable by representing it with other variables and then performing the interval computation, i.e., $x_3 = x_1 + x_2$ yields $\rho(x_3) = [6, \infty) \cap ((-\infty, 5] + (-\infty, 4]) = [6, \infty) \cap (-\infty, 9] = [6, 9]$. Similarly, we can obtain smaller intervals for both x_1 and x_2, which yields $\rho(x_1) = [2, 5]$ and $\rho(x_2) = [1, 4]$ respectively.*

During the narrowing process, ICP will frequently reach a fix-point given by some consistency conditions, e.g., hull consistency (see Def. 2.2.1), which is a state where ICP cannot narrow any domain of variables or make negligible progress. In such a case, interval splitting will be performed to pursue further narrowing and recursively contract the sub-intervals. This framework is called *branch-and-prune* [VHMK97] and forms a core mechanism for ICP based constraint solving. A toy example is given below.

Definition 2.2.1. *A simple constraint $c \equiv (p = q \circ r)$ is* hull consistent *w.r.t. $\rho : Vars \to \mathbb{IR}$ iff $\rho(p) = \rho(q) \circ \rho(r)$. Likewise, a simple bound $c \equiv (p \leq c)$, where c is a constant, is said to be hull consistent w.r.t. $\rho : Vars \to \mathbb{IR}$ iff $\rho(p) \subseteq (-\infty, \tilde{c}]$, where $\tilde{c}$ is the smallest computer-representable number $\geq c$.*

A set C of constraints is hull consistent w.r.t. ρ iff all its constraints are. The notation $\rho \models_{hc} C$ is used.

Example 2.2.2. *Consider the constraints $c_1 : p = q^2$ and $c_2 : p - q = 0$, and let the initial configuration be $\rho(q) = [-2,2]$ and $\rho(p) = [-2,2]$. We know that this problem has two isolated solutions $(q = 0, p = 0)$ and $(q = 1, p = 1)$. Starting by considering constraint c_1, the ICP process will give the following sequence:*

$$\begin{array}{rcl} (c_1, \rho(p) = [0,2]) & , & (c_1, \rho(q) = [0, 1.414\ldots]), \\ (c_2, \rho(p) = [0, 1.414\ldots]) & , & (c_1, \rho(q) = [0, 1.189\ldots]), \\ & \ldots & \end{array}$$

Iterative narrowing by ICP stops with a hull consistency set $\rho(q) = [0, 1.00\ldots]$ and $\rho(p) = [0, 1.00\ldots]$ (Converge to 1 but will not reach 1). In order to get close to the true results, we then choose a variable and split its domain into two parts, e.g., we could choose q and split into $\rho(q_1) = [0, 0.5]$ and $\rho(q_2) = (0.5, 1.00\ldots]$. Then ICP will be used on each sub-interval, rapidly narrowing them to small boxes enclosing the actual solutions.

The procure mentioned above forms the basic framework of constraints solving by using interval constraint propagation, which is shown in Algorithm 1.

Algorithm 1 Branch and Prune

Input: C: constraints model, ρ: interval box.
Output: L: a set of boxes which approximate the solutions of C regarding to ρ.

```
1: $L := \{\rho\}$                                  ▷ set of interval boxes
2: while True do
3:     $I := Choose(L)$;                             ▷ choice of a current box
4:     $I' := Prune(C, I)$;                          ▷ pruning by ICP
5:     if $I' \models_{hc} C$ then
6:         $L := Branch(L, I')$;                     ▷ branching if hull consistent
7:     if L is terminal then return L;               ▷ covering of the solutions of $C$
8: end while
```

In the above algorithm, the branching is applied when hull consistency is obtained, this is theoretically correct. However, in practice, the branching can be also pursued upon negligible progress for efficiency, i.e., the narrowing of intervals between two steps is small enough.

2.3 Satisfiability Modulo Theories (SMT)

Our main contribution is the extension of standard SMT, thus in this section, we will recap some basic concepts in this field. We start from the satisfiability problem of propositional formulas (SAT) to its underlying solving procedure, i.e., Davis-Putnam-Logemann-Loveland (DPLL) procedure, and then more expressive logics are discussed, i.e., Satisfiability Modulo Theories (SMT).

2.3.1 Satisfiability of Propositional Formulas (SAT)

Deciding the satisfiability of propositional formulas (SAT) is known as a NP-complete problem in complexity theory [Coo71], it is also the basis of many practical applications such as Electronic Design Automation, Verification, Artificial Intelligence and so on. The modern solving techniques make it possible to solve large-scale formulae with thousands of variables very efficiently. SAT is nowadays still a very hot topic in computer science, computer engineering and active in many aspects.

In satisfiability theory, a SAT formula is generally expressed by a standard form like $C_1 \wedge \cdots \wedge C_n$, which is called Conjunction Normal Form (CNF), as the name indicates, it is a conjunction of *clauses*. A clause C_i is a disjunction of literals with the form $l_1 \vee \cdots \vee l_n$, where a *literal* l_i is either an *atom* p or its negation $\neg p$. A unit clause only contains a single literal. We define M as an *assignment* of a CNF formula F, which assigns each literal a truth value, $M \models F$ if all its clauses are true with regard to M, in this case, M is a *model* of F. If there is no models for F, F is *unsatisfiable*.

Example 2.3.1. *Consider a four variables CNF formula:*

$$(\neg x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (x_2 \vee \neg x_3 \vee \neg x_4) \wedge (x_1 \vee x_4)$$

This CNF formula contains 5 clauses. $M(x_1) = False$*,* $M(x_2) = True$*,* $M(x_3) = False$ *and* $M(x_4) = True$ *is an assignment for the formula, and which makes all the clauses satisfied, so* M *is model the formula.*

For simplicity, a CNF formula can be represented by dropping of the $\wedge$ *connectives, i.e.,*

$$\neg x_1 \vee \neg x_2,\ x_2 \vee x_3,\ \neg x_1 \vee \neg x_3 \vee x_4,\ x_2 \vee \neg x_3 \vee \neg x_4,\ x_1 \vee x_4$$

2.3.2 DPLL Procedure

The algorithm for deciding the satisfiability of propositional formula was firstly proposed by Davis et al [DLL62], which is named as Davis-Putnam-Logemann-Loveland (DPLL) procedure.

The DPLL algorithm is the basis for a lot of modern SAT solving techniques, the basic DPLL procedure is a backtracking based technique which recursively chooses a literal and assigns a truth value to it, if the assigned literals makes the whole formula satisfied, the procedure returns the assignments as a model; otherwise if conflicts are obtained, DPLL procedure backtracks to the previous decision and redoes with the opposite assignment. If the procedure fails the backtracking, it claims that the CNF formula is unsatisfiable. The basic DPLL algorithm thus contains the following procedures:

- *Decide*, choose a literal and assign it to *True* or *False*. The way of choosing literal can be heuristically adopted according to the actual situations;
- *Unit propagate*, go through the whole formula and find unit clauses resulted by the current assignment;
- *Backtrack*, withdraw the current decision step if conflicts are met, redo the previous decision with the opposite assignments, for efficiency, this step can be improved with a smart way, i.e., Conflict-Driven-Clause-Learning (CDCL, [SS97, BJS97]). Instead of backtracking to the previous decision step, CDCL analyses the conflicts and jumps back to the decision step which leads to the conflicts;
- *Fail*, terminate the procedure if backtracking step cannot find a previous decision point to jump, which means the formula is unsatisfiable.

Example 2.3.2. *Reconsider the previous example, the following steps show how DPLL algorithm works:*

$$\begin{array}{rcl}
\emptyset || & \neg x_1 \vee \neg x_2,\ x_2 \vee x_3,\ \neg x_1 \vee \neg x_3 \vee x_4,\ x_2 \vee \neg x_3 \vee \neg x_4,\ x_1 \vee x_4 & \Rightarrow (Decide) \\
x_1^d || & False \vee \neg x_2,\ x_2 \vee x_3,\ False \vee \neg x_3 \vee x_4,\ x_2 \vee \neg x_3 \vee \neg x_4,\ True & \Rightarrow (UnitPropagate) \\
x_1^d \bar{x}_2 || & True,\ False \vee x_3,\ False \vee \neg x_3 \vee x_4,\ False \vee \neg x_3 \vee \neg x_4,\ True & \Rightarrow (UnitPropagate) \\
x_1^d \bar{x}_2 x_3 || & True,\ False \vee False \vee x_4,\ False \vee False \vee \neg x_4,\ True & \Rightarrow (UnitPropagate) \\
x_1^d \bar{x}_2 x_3 x_4 || & True,\ True,\ False,\ True & \Rightarrow (BackTrack) \\
\bar{x}_1 || & True,\ x_2 \vee x_3,\ True,\ x_2 \vee \neg x_3 \vee \neg x_4,\ False \vee x_4 & \Rightarrow (UnitPropagate) \\
\bar{x}_1 x_4 || & True,\ x_2 \vee x_3,\ True,\ x_2 \vee \neg x_3 \vee False & \Rightarrow (Decide) \\
\bar{x}_1 x_4 \bar{x}_3^d || & True,\ x_2 \vee False,\ True,\ True & \Rightarrow (UnitPropagate) \\
\bar{x}_1 x_4 \bar{x}_3^d x_2 || & True,\ True,\ True,\ True &
\end{array}$$

We find an assignment which makes all the clauses True, i.e., x_1 is assigned to False, x_2 is assigned to True, x_3 is assigned to False and x_4 is assigned to True, which is also the model of the given formula.

The modern DPLL procedure doesn't implement the classical DPLL framework, a lot of sophisticated techniques are proposed for efficiency.

2.3.3 Satisfiability Modulo Theory (SMT)

SAT formula contains only Boolean variables, which is limited in many applications. In order to support more expressive logics, some other background theories are integrated to SAT, which is known as Satisfiability Modulo Theories (SMT). SMT is a more powerful tool to formulate the constraints concerning different domains, thus arises in many industrial applications.

SMT arises to be a very interesting and challenging topic in modern constraints solving area, which combines different theories like equality [Ack54, DST80, NO80], linear arithmetic [Dan98, Pug91], bit vectors [BD02, FOR$^+$01], arrays [MP67, Rey79], pointer logic [Bur72, Lei95, Rey02] etc, the quantified logics have also been considered [Sto76, BKF95, DSW99].

Because of the success of SAT/SMT theories, a large number of tools have been implemented and widely applied to industrial applications. Successful tools varies from specific built-in theory solvers, like Boolector for arrays and bit vectors, iSAT and MiniSmt for non-linear arithmetic; over combined theory solvers, like CVC3 and Yices; to large-scale industrial strength solver, like Z3 from Microsoft Research.

The modern SMT solvers involve different kinds of sophisticated theories and mechanisms for both generality and efficiency. One of the significant frameworks is interval based solving procedure, which was first proposed by Martin Fränzle et al. [FHT$^+$07], where a tight integration of SAT solving techniques with interval based arithmetic constraint solving was investigated. Their approach is able to handle large constraint systems with complex Boolean structure involving nonlinear arithmetic. Due to the undecidability of nonlinear arithmetic [Rat06], the proposed solving procedure is not complete with sometimes *unknown* answer. However, with the help of other heuristics like *restarts* it can achieve an almost complete procedure. HySAT/iSAT [HEFT08] are interval based SMT solver which implemented the mentioned solving procedure and can be applied to either nonlinear constraints solving or bounded model checking.

Interval-based technique is also the basis of this Ph.D. thesis, our solving procedure for stochastic SMT is a combination of interval constraint propagation, classical CDCL framework and probability analysis. The rest of the thesis will focus on these aspects.

Chapter 3

Stochastic Satisfiability Modulo Theories

3.1 Motivation: From SMT to Stochastic SMT (SSMT)

As has been stated in the previous sections, SAT/SMT solvers make rapid advances and find novel uses in a wide variety of applications both in computer science and beyond, i.e., they are used for verification, proving the correctness of programs, software testing based on symbolic execution, syntheses, hardware design, etc. However, SAT/SMT solvers cannot deal with systems with uncertainties. Although such uncertainties about variables can be encoded by intervals, in many applications such as wireless sensing, more concise quantitative information about the uncertainties involved is available in terms of probabilities. To incorporate this kind of information, both the underlying models and the corresponding analysis techniques have to be adapted, i.e., extending classical SAT/SMT so that uncertainties can be captured, which yields stochastic SAT/SMT (SSAT/SSMT).

According to different focuses, the development of SSAT/SSMT can be summarized in the following:

- Stochastic Propositional Satisfiability (SSAT). This is the first stochastic extension for propositional satisfiability by Papadimitriou [Pap85], which features both classical quantifiers and randomized quantifiers. Randomized quantifiers indicate that the variables bound by them are random variables.

- Stochastic Satisfiability Modulo Theory (SSMT). SSAT has been lifted to support nonlinear arithmetic by Fränzle, Teige et al. [FHT08, TF08] in order to symbolically reason about reachability problems of probabilistic hybrid automata (PHA).

- Stochastic Satisfiability Modulo Theory with Continuous Domain (CSSMT). A major limitation of the SSMT solving approach is that all quantifiers (except for implicit innermost existential quantification of all otherwise unbound variables) are confined to range over finite domains. As this implies that the carriers of probability distributions have to be finite, a large number of phenomena cannot be expressed within the SSMT framework, such as continuous noise or measurement error in hybrid systems. To overcome this limitation, we relax the constraints on the domains of randomized variables, now also admitting continuous probability distributions in SSMT solving [GF15]. This is the main concern of this Ph.D. thesis.

All the theories mentioned above have one thing in common, i.e., instead of reporting *True* or *False*, the formula has a probability as semantics, which denotes the probability of satisfaction under optimal resolution of the non-random quantifiers. SSAT and SSMT permit concise description of diverse problems combining reasoning under uncertainty with data dependencies. Applications range from AI planning [ML98, LMP01, ML99] to analysis of probabilistic hybrid automata [FHT08].

Example 3.1.1. *Consider the following formulas:*

$$\begin{aligned}
\Phi_1 &:= \exists x \text{ꓤ}^{0.3} y : (x \vee \neg y) \wedge (\neg x \vee y) \\
\Phi_2 &:= \exists x \in \{0,1\} \text{ꓤ}_{\langle (0.5,0.6),(1,0.4)\rangle} y \in \{0.5,1\} : (x \geq 0 \vee y < 0) \wedge (x < 0.7 \vee y \geq 0) \\
\Phi_3 &:= \exists x \in [0,1] \text{ꓤ} y \in \mathcal{N}(0,1) : (x \geq 0 \vee y < 0) \wedge (x < 0.7 \vee y \geq 0)
\end{aligned}$$

Φ_1, Φ_2 and Φ_3 are SSAT, SSAMT and CSSMT respectively, all the formulas have the form $\Phi := \mathcal{Q} : \varphi$, where $\mathcal{Q}$ is a sequence of quantifiers, different with classical satisfiability formulas, a new quantifier ꓤ is introduced, which is named as randomized quantifier and indicates that the bound variables are probabilistic distributed.

Compared the three formulas, SSMT formulas assign each variable with only True or False, i.e., $\text{ꓤ}^{0.3} y$ means y is True with probability 0.3*. SSAT is lifted to SSMT by introducing arithmetics into φ, the random variables are bound with ꓤ with discrete probability distribution, i.e., in Φ_2, $\text{ꓤ}_{\langle (0.5,0.6),(1,0.4)\rangle} y$ means y has two discrete value* 0.5 *and* 1 *and their probability are* 0.6 *and* 0.4 *respectively. More general formulas can be represented by CSSMT, where continuous distributions can be bound to variables, as seen in formula Φ_3, y is normally distributed with standard Gauss distribution.*

Compared to the classical SAT/SMT, SSAT/SSMT/CSSMT provide more information for the formula:

- all of them introduce the randomized quantifier ꓤ, that is a new symbol in satisfiability modulo theory. ꓤ indicates that the variable associated is not a typical variable, and it's distribution density and domain are also specified by the randomized quantifier, which allows the modeling of random variables in an SAT/SMT formula;

- the semantics of SSAT/SSMT/CSSMT are changed, which is not *True* or *False* anymore, instead, it is a real number between 0 and 1 which shows how probable the formula can be satisfied, i.e., the maximum probability of satisfaction.

Example 3.1.2. *Now let us check the following formula* $\Phi_1 := \exists x \text{ꓤ}^{0.3} y : (x \vee \neg y) \wedge (\neg x \vee y)$ *to understand the meaning of introducing randomize quantifier* ꓤ*. We start by taking x with True, in order to make the second clause* $\neg x \vee y$*, y has to be True,* $\text{ꓤ}^{0.3} y$ *indicates that y takes True with probability* 0.3*, which yields the probability* 0.3 *of this branch; similarly, if x is False, the probability of satisfaction is* 0.7*. In all, the maximum probability of satisfaction is* 0.7*.* Φ_2 *and* Φ_3 *can be understood in the same way; the formal semantics will be introduced later.*

3.2 Decision Procedure for SMT

The solving procedures for SSMT/CSSMT formulas are mainly based on the SMT solving procedure proposed by Martin Fränzle et al. [FHT+07, FHT08], which integrates interval arithmetic with DPLL framework. The DPLL framework has been explained in the previous chapter and here we will briefly recap the basic ideas for SMT solving with interval arithmetic, which forms the basis of our work, as in Fig. 3.1.

The solving procedure takes an SMT formula Φ and its initial assignment ρ as input, it starts with establishing a list structure M to denote the list of asserted atoms. The procedure continues with searching for *unit clauses* in Φ, adds to M and repeats until all unit clauses have been processed. Interval constraint propagation is then performed to narrow the current interval valuation ρ, which yields the contractions. Contractions may generate new unit clauses; the procedure goes back the unit propagation and ICP steps, which are repeated until contraction detects some conflicts. Otherwise, if no conflicts are identified or no new clauses are generated, it applies a splitting step.

When conflicts are detected, some previous decisions have to be reverted, which is achieved by backtracking, if no previous backtracking steps are applicable, the procedure stops with *unsat*.

The solving procedure will try to find satisfying valuations if hull consistency is obtained by ICP step; if such valuations exist, it stops with *sat*.

Remark 3.2.1. *The procedure could be extremely improved by analyzing the conflicts, i.e., in case of the conflicts encountered during the search, the reason can be recorded to prevent the*

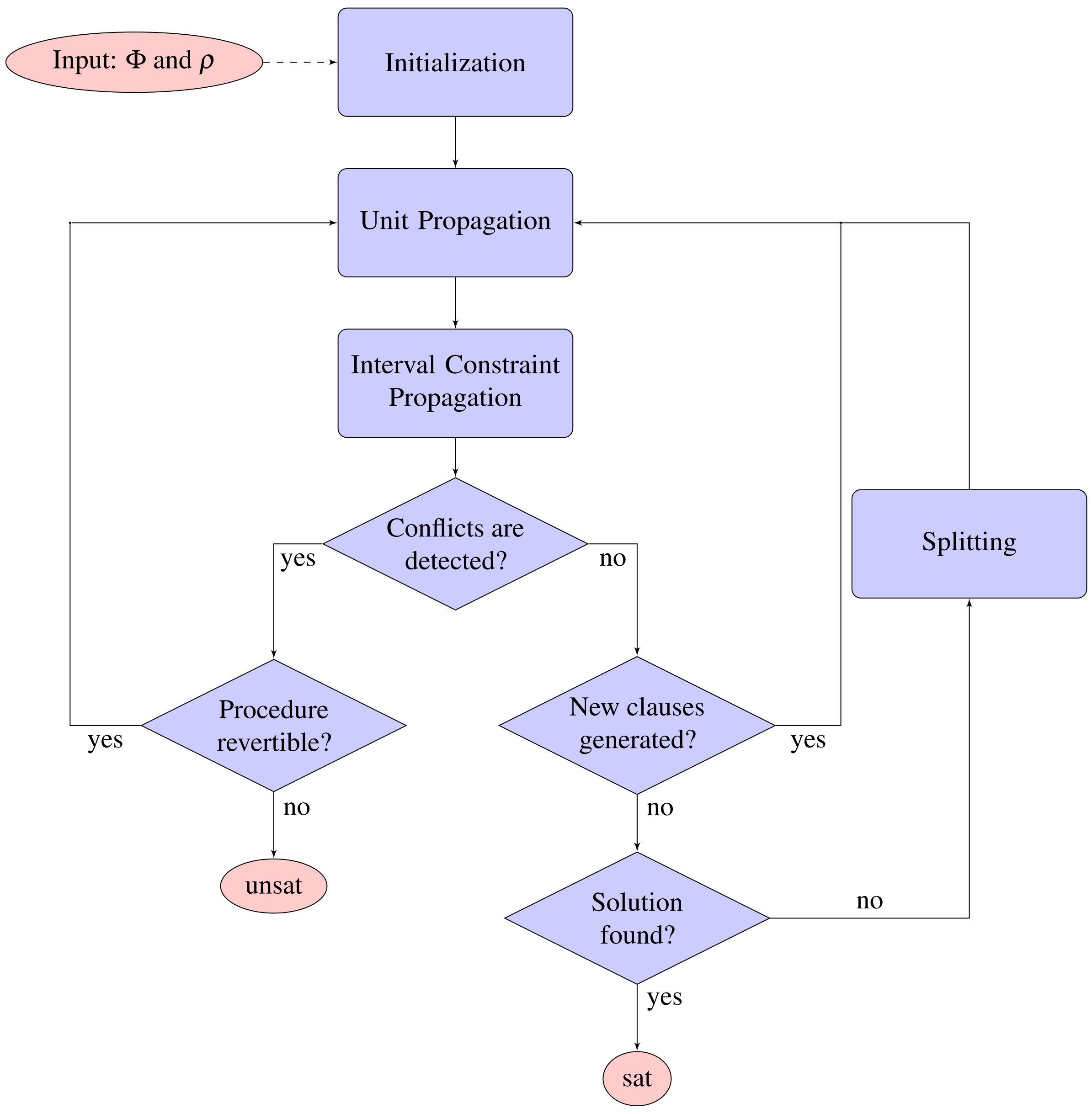


Fig. 3.1 Decision procedure for SMT integrated with interval constraint propagation

procedure from constructing other interval valuations provoking a similar conflict, which yields conflict driven learning mechanism (CDCL) and non-chronological backtracking.

Remark 3.2.2. *The base algorithm described above is not complete, due to the risk of unbounded splitting, which yields non-termination due to the density of the order on $\mathbb{R}$. In such case, the procedure fails to find a solution thus splits endlessly, thus heuristics should be selected to force the termination, i.e., set up a progress bound (precision bound), fix the splitting depth, i.e.,* *the procedure will then either lead to an 'unknown' answer equipped with candidate intervals where solutions may exist (no guarantee), or restart to do other trials.*

3.3 Algorithms for SSMT

The stochastic satisfiability modulo theories (SSMT) problem is a generalization of the SMT problem on existential and randomized quantification over discrete variables of an SMT formula, which is proposed and formolized by Tino Teige et al. [TF08, Tei12]. The formal definition of SSMT can be found as follow:

Definition 3.3.1. *An SSMT problem*

$$\Phi = Q_1 x_1 \in dom(x_1) \cdots Q_n x_n \in dom(x_n) : \varphi$$

is specified by:

- *a prefix $Q_1 x_1 \in dom(x_1) \cdots Q_n x_n \in dom(x_n)$ binding the variables x_i to the quantifiers Q_i. Q_i is either existential, denoted as $\exists$, or randomized, denoted as Я_{d_i}, where d_i is probability distribution often denoted by a list $\langle (v_1, p_1), \ldots, (v_m, p_m) \rangle$ of value pairs and p_i is understood as the probability of setting variable x_i to v_i, where $\sum_{j=1}^{m} p_j = 1$ holds;*
- *and an SMT formula φ (also called the matrix) over quantifier-free non-linear arithmetic theory over the reals, integers and Booleans. It is always represented by conjunctive normal form (CNF).*

Example 3.3.1. *According to Definition 3.3.1,*

$$\Phi = \exists x \in \{0,1\} \text{Я}_{\langle (0,0.6),(1,0.4) \rangle} y \in \{0,1\} : (x > 0 \vee 2a \cdot \sin(4b) \geq 3) \wedge (y > 0 \vee 2a \cdot \sin(4b) < 1)$$

is an SSMT formula, where x is bound by existential quantifier $\exists$, its domain is discrete with two values. y is bound by randomized quantifier Я and its discrete distribution is represented by the subscript $\langle (0,0.6),(1,0.4) \rangle$, which means y can take two values 0 *and* 1*, each has probability* 0.6 *and* 0.4 *respectively.*

The classical SMT says that a formula Φ is *satisfiable* if there exists a model (assignment) which makes Φ satisfied, Φ is *unsatisfiable* if such model cannot be found. Due to the introduction of randomized quantifier, it makes no sense to just decide the satisfiability of such formulas, instead, a quantitative semantic should be given to SSMT, that is, the *maximum probability of satisfaction*. Intuitively, for an SSMT formula $\Phi = \exists x_1 \in dom(x_1) \text{Я}_{d_2} x_2 \in$

$dom(x_2)\exists x_3 \in dom(x_3)\rotatebox[origin=c]{180}{R}_{d_4} x_2 \in dom(x_2) : \varphi$: Φ determines the maximum probability s.t. there is a value for x_1 s.t. for random variable x_2 there is value for x_3 s.t. for random values of x_4 the SMT formula ϕ is satisfiable. The formal semantics is defined as follow:

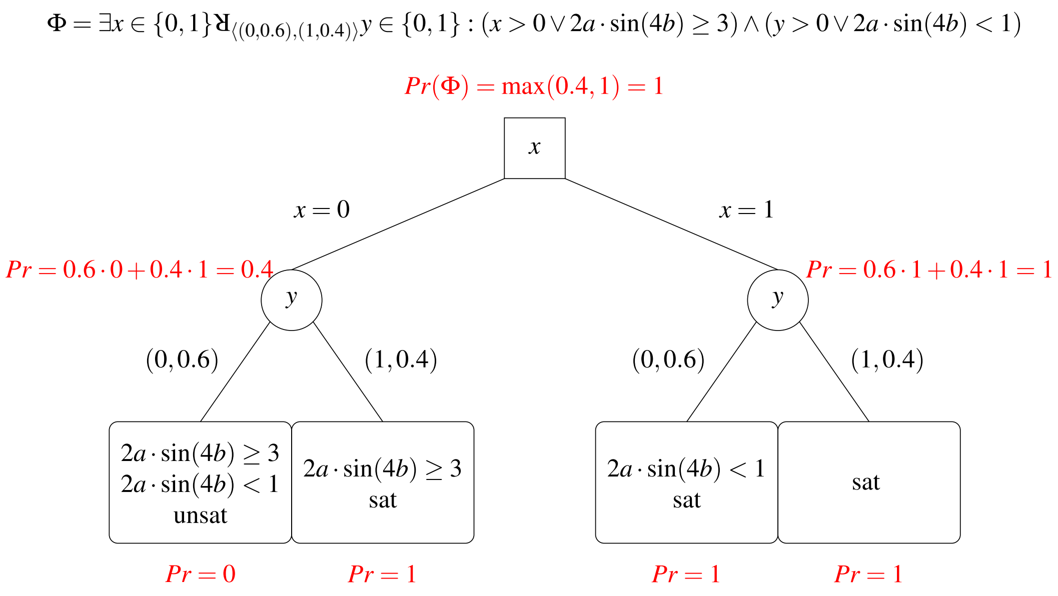
Definition 3.3.2. *The semantics [TF08] of a SSMT formula $\Phi = \mathcal{Q} : \varphi$ is defined by the maximum probability of satisfaction $Pr(\Phi)$ as follows, where ε denotes the empty quantifier prefix:*

- $Pr(\varepsilon : \varphi) = 0$ *if φ is unsatisfiable.*
- $Pr(\varepsilon : \varphi) = 1$ *if φ is satisfiable.*
- $Pr(\exists x_i \in dom(x_i) \ldots Q_n x_n \in dom(x_n) : \varphi)$
  $= \max_{v \in dom(x_i)} Pr(Q_{i+1} x_{i+1} \in dom(x_{i+1}) \ldots Q_n x_n \in dom(x_n) : \varphi[v/x_i])$.
- $Pr(\rotatebox[origin=c]{180}{R}_{\pi_i} x_i \in dom(x_i) \ldots Q_n x_n \in dom(x_n) : \varphi)$
  $= \sum_{(v,p) \in d_i} p \cdot Pr(Q_{i+1} x_{i+1} \in dom(x_{i+1}) \ldots Q_n x_n \in dom(x_n) : \varphi[v/x_i])$.

Example 3.3.2. *Back to the Example 3.3.1, the semantics can be depicted as a tree, see Figure 3.2. According to the Definition 3.3.2, the semantics of SSMT formula is interpreted by resolving the quantifiers from outermost to innermost, i.e., from top to bottom in the tree structure. x is firstly split into two branches, and for each branch, y is then split. When all the quantifiers have resolved, we obtain some SMT formulas; the satisfiability could be checked. For unsat case, we assign the probability* 0, *and* 1 *otherwise. The probabilities are then computed from bottom to top according to the types of quantifiers: 1) if the variable is bound by $\rotatebox[origin=c]{180}{R}$, the weighted sum is computed regarding the discrete distribution, which yields the expectation; 2) if the variable is bound by $\exists$, the maximum probability is chosen. For this example, we get that $x = 0$ leads to the probability of satisfaction* 0.4*, meanwhile $x = 1$ leads to the probability* 1*. We finally conclude that the maximum probability of satisfaction is* 1.

A straightforward way to solve SSMT formula can be explained as follows: since SSMT only supports discrete distributions, the domain of the corresponding random variable is discrete, i.e., finite countable. The simplest way is to substitute the matrix with discrete values, which yields SMT formulas. The satisfaction of SMT formulas can be judged by using SAT/SMT solvers, like HySAT/iSAT. Then the obtained results can be computed from bottom to top according to the type of quantifiers as described in the previous example.

The proposed solving procedure can be easily built on existing SAT/SMT solver. However, it suffers from the exponential explosion. For example, if there are k variables which are bound by $\rotatebox[origin=c]{180}{R}$, each variable contains n_k possible discrete values, then the number of resulted SMT



$\Phi = \exists x \in \{0,1\} \text{Я}_{\langle(0,0.6),(1,0.4)\rangle} y \in \{0,1\} : (x > 0 \vee 2a \cdot \sin(4b) \geq 3) \wedge (y > 0 \vee 2a \cdot \sin(4b) < 1)$
$Pr(\Phi) = \max(0.4, 1) = 1$
x
$x = 0$
$x = 1$
$Pr = 0.6 \cdot 0 + 0.4 \cdot 1 = 0.4$
$Pr = 0.6 \cdot 1 + 0.4 \cdot 1 = 1$
y
y
$(0, 0.6)$
$(1, 0.4)$
$(0, 0.6)$
$(1, 0.4)$
$2a \cdot \sin(4b) \geq 3$ $2a \cdot \sin(4b) < 1$ unsat
$2a \cdot \sin(4b) \geq 3$ sat
$2a \cdot \sin(4b) < 1$ sat
sat
$Pr = 0$
$Pr = 1$
$Pr = 1$
$Pr = 1$

Fig. 3.2 Semantics of a SSMT formula depicted as a tree [TF08].

formulas is $\prod_{i=1}^{k} n_i$. Each should be solved at least once (sometimes the solving procedure should be restarted) by SAT/SMT solver; this is very inefficient.

A better way to handle SSMT formula is to integrate pruning mechanics to the solving procedure, where parts of the searching space can be pruned through probability analysis. The semantics of SSMT formula is denoted as the probability of satisfaction, in practice, if the probability bounds of interest are given, in such cases, the solving procedure can be terminated once we get evidence that the given bounds are reached or can never be achieved. This framework has been formalized by Tino Teige [TF08], a slightly simplified version is given in Algorithm 2.

The algorithm $SSMT(\mathcal{Q}, t_l, t_u)$ takes SSMT formula $\Phi = \mathcal{Q} : \varphi$ as input, as well as the target thresholds t_l and t_u. The solving procedure will return a witness probability if the probability of satisfaction is not in this range, i.e., it exceeds the upper threshold t_u or cannot reach the lower threshold t_l, otherwise the probability of satisfaction. In this setting, if one want to compute the real probability, the target thresholds $t_l = 0$ and $t_u = 1$ can be set.

The algorithm works recursively; the base cases are treated when all the quantifiers have been resolved, the problem is reduced to quantifier-free SMT with non-linear arithmetic, this can be solved by any SMT solver which supports nonlinear arithmetic, i.e., HySAT/iSAT. This step can also be improved by tightly integrating the conflict clause learning techniques so that the reasons for conflicts can be generated and then added to the original formula. If the quantifiers are not fully resolved, the algorithm performs its recursive branch according to the type of quantifiers outermost of $\mathcal{Q}$:

Algorithm 2 $SSMT(\mathcal{Q}, t_l, t_u)$

```
Input: SSMT formula $\Phi = \mathcal{Q} : \varphi$, lower and upper thresholds $t_l$, $t_u$.
Output: The satisfaction probability of $\Phi$ w.r.t. the thresholds.
 1: while True do
    ▷ base cases
 2:     if $\varepsilon : \varphi$ is unsatisfiable then return 0;
 3:     if $\varepsilon : \varphi$ is satisfiable then return 1;
    ▷ handling $\exists$ quantifier
 4:     if $head(\mathcal{Q}) = \exists x \in dom(x)$ then
 5:         $v \in dom(x)$, $x := v$, $dom(x) = dom(x) \setminus \{v\}$;
 6:         $p_0 = SSMT(tail(\mathcal{Q}), t_l, t_u)$;
 7:         if $p_0 > t_u$ or $p_0 = 1$ or $dom(x) = \emptyset$ then return $p_0$;
 8:         $p_1 = SSMT(\mathcal{Q}, \max(p_0, t_l), t_u)$;
          return $\max(p_0, p_1)$;
    ▷ handling $\text{Я}$ quantifier
 9:     if $head(\mathcal{Q}) = \text{Я}_d x \in dom(x)$ then
10:         $v \in dom(x)$, $(v, p_v) \in d$, $x := v$, $dom(x) = dom(x) \setminus \{v\}$;
11:         $p_{remain} = \sum_{v' \in dom(x), (v', p') \in d} p'$;
12:         $p_0 = SSMT(tail(\mathcal{Q}), (t_l - p_{remain})/p_v, t_u/p_v)$;
13:         if $p_v \cdot p_0 > t_u$ or $p_v \cdot p_0 = 1$ or $dom(x) = \emptyset$ then return $p_v \cdot p_0$;
14:         if $p_{remain} < t_l - p_v \cdot p_0$ then return $p_v \cdot p_0$;
15:         $p_1 = SSMT(\mathcal{Q}, t_l - p_v \cdot p_0, t_u - p_v \cdot p_0)$;
          return $p_1 + p_v \cdot p_0$;
16: end while
```

- the outermost variable x is bound by existential quantifier $\exists$, the algorithm chooses a discrete value from its domain and substitutes the matrix with this selected value, then the probability for this branch is computed. The resulted probability (p_0) is compared to the given bound if it exceeds the upper one (t_u) or it is the last branch, i.e., no further candidates in x's domain to be chosen, p_0 is returned as witness probability, no additional computation needs to be performed; otherwise, the algorithm updates the lower bound ($\max(p_0, t_l)$) and explores other branches, which is done recursively and combined by choosing the maximum probability.

- the outermost variable x is bound by randomized quantifier $\reflectbox{R}$, a value-probability pair ((v, p_v)) is chosen from the domain of x. The remaining probability indicates that how much probability we can still obtain from the rest of x's domain. The algorithm then computes the probability regarding to the current branch, the resulted probability ($p_v \cdots p_0$) is compared with upper threshold (t_u), if it exceeds, the solving procedure terminates without further computing; another termination case is that the remaining probability cannot satisfy the requirement of the lower threshold, otherwise, the algorithm updates lower and upper thresholds and recursively explores other branches, which is combined by computing the weighted sum.

The trick played here is to introduce the upper and lower thresholds, which pursues fast termination since the algorithm doesn't need to explore all the discrete value. The remaining probability and achieved maximum probability can be evaluated during the run, according to probability analysis, once the upper threshold is exceeded, or the lower threshold cannot be reached by the remaining probability, the algorithm terminates without further explorations. This framework can speed up the solving procedure, however, in the worst case, it needs to handle all the branches. More heuristic techniques can be applied to prune the searching space, conflicts analysis and Craig Interpolation are also good candidates for algorithmic enhancement, the details can be found in Tino Teige's Ph.D. thesis [Tei12].

3.4 The Limitation of SSMT

In the previous sections, we recap the basic setting and solving procedure for SSMT, which is a significant extension w.r.t. the capability to model uncertainties. Different from the standard SAT/SMT, SSMT has the probability as semantic, which makes it possible to encode random variables in SSMT formulas and compute the maximum probability of satisfaction. However, there are some limitations which are concluded in the following:

- A serious limitation of the SSMT-solving approach is that all quantifiers are confined to range over finite domains, except for implicit innermost existential quantification of all otherwise unbound variables. From the Algorithm 2, we can see that the values in the discrete domain of variables are recursively chosen and different branch strategies are applied according to the type of quantifiers. If the domains of variables are continuous, the proposed algorithm fails due to infinite branching;
- If only the variables bound by existential quantifier are continuous, the algorithm can still work by some modifications, i.e., resolve the random variables by substituting the formula with their discrete values, which yields some substituted SMT formulas, these formulas could be handled by SAT/SMT solver. Thus the corresponding tree-like structure can be constructed to fulfill the solving procedure;
- The discreteness of random variables in SSMT implies that the carriers of probability distributions have to be finite, a large number of phenomena cannot be expressed within such frameworks, such as continuous noise or measurement error in hybrid systems.

To overcome this limitation, we relax the constraints on the domains of randomized variables, now also admitting continuous probability distributions in SSMT solving. One may think that the proposed solving procedure can be directly borrowed for the continuous case since the similar semantic and structure. However, the facts violate such intuition, which yields non-straightforward extension for SSMT and will be analyzed in the later chapters.

3.5 Related Work

Another similar work has to be stressed here, which is done by Christian Ellen, Sebastian Gerwinn and Martin Fränzle [EGF14]. They extend SSMT within their paper to also support hybrid systems with continuously-valued probabilistic and non-deterministic influences, enabling SSMT to address problems from the domain of Stochastic Hybrid Systems and stochastic optimal control. For scalability, they extend the solving procedure based on statistical model checking (SMC), which generates results which can be guaranteed with a certain level of confidence. Compared to this work, our work deals with the same problem in an alternative way, we prefer to maintain lower and upper bounds for the probability of satisfaction, although it may sacrifice the efficiency, one can see, lots of heuristic techniques can be used to make the trade-off.

3.6 Conclusion

After reviewing the related work and foundation knowledge, we will step into our main concerns on continuous SSMT (CSSMT) from next chapter. The following chapters are structured as following:

Chapter	Theme
4	Introduction to continuous SSMT and its formal semantics, the comparison with other satisfiability modulo theories.
5	Explanation of the relation between Decision Processes and formalization of reachability problem with CSSMT formulation.
6	Proposal of rule-based solving procedure for CSSMT.
7	Algorithmic enhancement for CSSMT solving, techniques to improve the efficiency.
8	Introduction to CSSMT solver, i.e., CSiSAT and its basic functions and structure.
9	Demonstration of case studies which shows the ability of CSSMT to model and analyse the systems with stochastic behavior.

Table 3.1 Thesis Structure

Chapter 4

Stochastic Satisfiability Modulo Theories with Continuous Domain

In the previous chapters, we mentioned the limitation of SSMT – a stochastic extension of SAT/SMT – is that the continuous distributions cannot be captured. In this chapter, we will relax such constraints on the domains of randomized variables, now also admitting continuous probability distributions in SSMT solving, which thus is named as continuous SSMT (CSSMT).

4.1 Motivation: Stochastic Hybrid Systems with Continuous Distribution

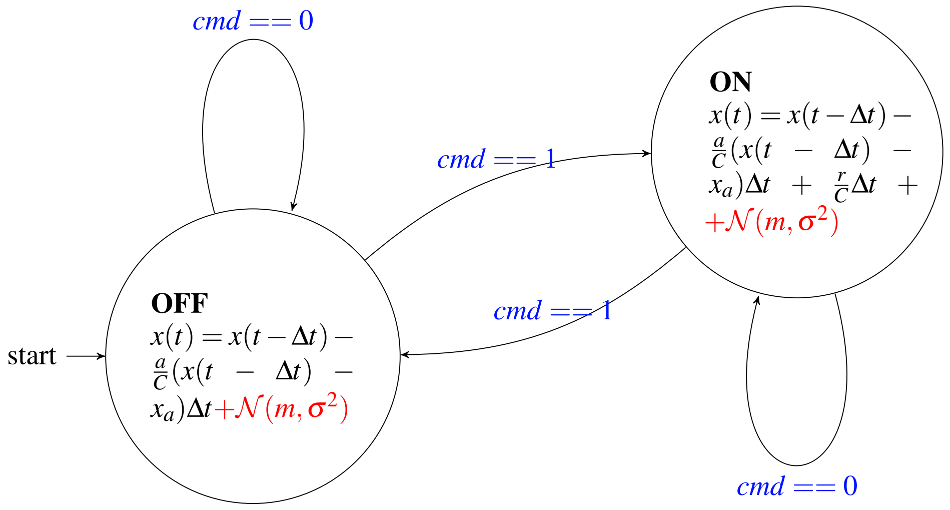

$cmd == 0$
ON
$x(t) = x(t-\Delta t) - \frac{a}{C}(x(t-\Delta t) - x_a)\Delta t + \frac{r}{C}\Delta t + \mathcal{N}(m,\sigma^2)$
$cmd == 1$
$cmd == 1$
start
OFF
$x(t) = x(t-\Delta t) - \frac{a}{C}(x(t-\Delta t) - x_a)\Delta t + \mathcal{N}(m,\sigma^2)$
$cmd == 0$

Fig. 4.1 Regulating the temperature of a room by a thermostat

Hybrid Automaton (HA) is often used to model the systems with both continuous and discrete behavior, i.e., in Figure 4.1 a simple temperature regulating system is modeled by a two-mode hybrid automaton. This model equips a thermostat with two modes $\mathcal{Q} = \{ON, OFF\}$, and the thermostat issues switching commands $cmd = \{0, 1\}$ to the heater: $cmd == 0$ means no switching command is issued and 1 oppositely, the commands can be issued at each step which decides whether to stay in the current mode or switch to the other mode, the sequence of different commands corresponds to different control strategies which yield different temperature change. Later we will see that in order to achieve the control target, i.e., to guarantee that the temperature is always in a range, the corresponding control sequence should be synthesized. The average room temperature in OFF mode changes according to the laws $x(t) = x(t-\Delta t) - \frac{a}{C}(x(t-\Delta t) - x_a)\Delta t + \mathcal{N}(m, \sigma^2)$ and in OFF mode it follows $x(t) = x(t-\Delta t) - \frac{a}{C}(x(t-\Delta t) - x_a)\Delta t + \frac{r}{C}\Delta t + \mathcal{N}(m, \sigma^2)$, where a is the average heat loss rate, C is the average thermal capacity, x_a is the ambient temperature, r is the rate of heat gain, and Δt is the discretization time interval. In order to capture the disturbance, we add a noise $\mathcal{N}(m, \sigma^2)$ term, which denote the probability measure over $(\mathbb{R}, \mathcal{B}(\mathbb{R}))$ associated with a Gaussian density function with mean m and variance σ^2.

The consideration of measurement error makes the model different from the classical hybrid model since uncertainty is introduced. For hybrid systems, we care much about the properties which specify a region and verify whether it can be reached from initial points, i.e., reachability analysis. In our framework, this setting fails due to the uncertainties of the systems, which makes the given region potentially reachable from any initial points – some with high probability but some are low. Our concerns can thus be modified as:

Given a system $\mathcal{S}$ and a region $\mathcal{C}$, probabilistic reachability analyzes how likely $\mathcal{C}$ can be reached by $\mathcal{S}$ under given initial conditions $\mathcal{I}$.

To handle probabilistic reachability problem, we should clarify three aspects for a system:

- $\mathcal{I}$, which specifies the initial conditions which should be satisfied by the system before it starts to run;
- $\mathcal{T}$, which gives the transition relation between modes and models the system behaviors;
- $\mathcal{C}$, which clarifies the region of interest that the system is expected to reach.

When all the aspects are clear, we can verify the formula $\Phi = \mathcal{I} \wedge \bigwedge_k \mathcal{T} \wedge \mathcal{C}$, which unwinds the transition relation by k steps, then yields a bounded model checking problem (BMC). If all of them are formulated by first-order logic with non-linear arithmetic, it produces a stochastic SMT formula with the continuous domain (due to the continuous random variables), i.e., a

CSSMT formula. As a conclusion, we know that CSSMT formulas can be used to model the stochastic hybrid systems and analyze their corresponding reachability properties. In the following, we will introduce the definition and semantics of CSSMT.

4.2 Definitions and Semantics for CSSMT

The syntax of stochastic SMT formulas over continuous quantifier domains agrees with the discrete version from [FHT08], except that continuous quantifier ranges are permitted.

Definition 4.2.1. *An SSMT formula with continuous domain (CSSMT) is of the form:* $\Phi = \mathcal{Q} : \varphi$*, where:*

- $\mathcal{Q} = Q_1 x_1 \in dom(x_1) \dots Q_n x_n \in dom(x_n)$ *is a sequence of quantified variables,* $dom(x_i)$ *denotes the domain of variable* x_i*, which are bounded intervals over the reals,* Q_i *is either an existential quantifier* $\exists$ *or a randomized quantifier* $\reflectbox{R}_{\pi_i}$ *with integrable probability density function over the reals* π_i *satisfying* $\int_{dom(x_i)} \pi_i(x_i)dx_i = 1$.
- φ *is an SMT formula over quantifier-free non-linear arithmetic theory* $\mathcal{T}$*. Without loss of generality, we assume that* φ *is in conjunctive normal form (CNF), i.e.,* φ *is a conjunction of clauses, and a clause is a disjuction of (atomic) arithmetic predicates.* φ *is also called the* matrix *of the formula.*

The definition is similar to its discrete version. However, the following points should be noticed:

- as in the definition of SSMT, we only allow two types quantifiers, i.e., existential quantifier $\exists$ and randomized quantifier $\reflectbox{R}$;
- all the variables can be equipped with continuous domains;
- in principle the matrix φ can be of any form, here it is limited to be CNF due to the equivalence and convenience;
- the density function π should be integrable over reals, which can be easily satisfied in most of the practical cases, i.e., uniform distribution, exponential distribution, normal distribution, etc.

Example 4.2.1. *Consider the following SSMT formula:*

$\Phi = \exists x \in [-1,1] \reflectbox{R}_{\mathcal{N}(0,1)} y \in (-\infty, +\infty) : (x^2 \leq \frac{1}{9} \vee a^3 + 2b \geq 0) \wedge (y > 0 \vee a^3 + 2b < -1)$

there are two variables x and y, x is a normal variable bound by $\exists$ *with continuous domain* $[-1,1]$*, and y is a random variable with normal distribution, i.e., mean value* 0 *and variance* 1.

The semantics of CSSMT is similar to SSMT, which denotes a real number to indicate the maximum probability of satisfaction.

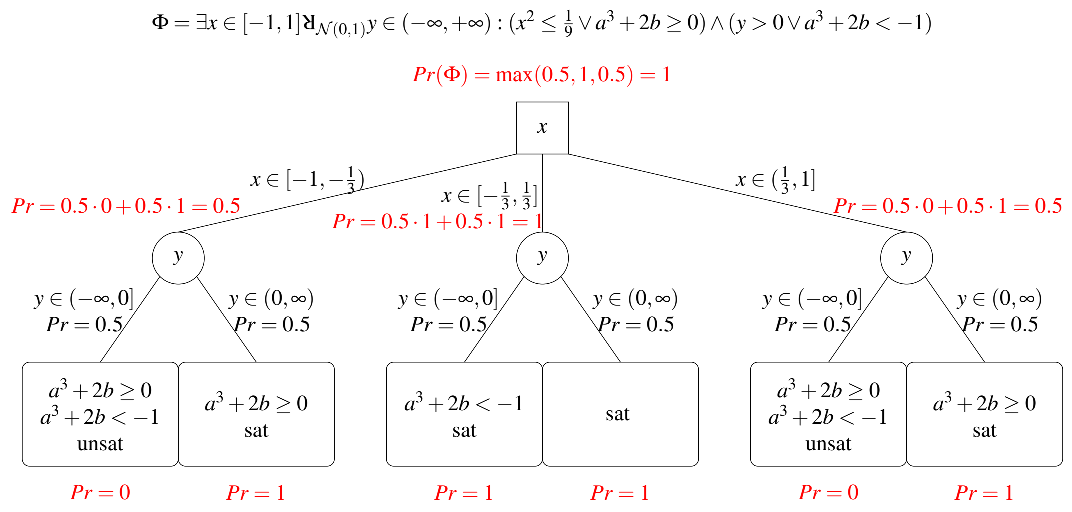
Definition 4.2.2. *The semantics of a CSSMT formula $\Phi = \mathcal{Q} : \varphi$ is defined by the maximum probability of satisfaction $Pr(\Phi)$ as follows, where ε denotes the empty quantifier prefix:*

- $Pr(\varepsilon : \varphi) = 0$ *if φ is unsatisfiable.*
- $Pr(\varepsilon : \varphi) = 1$ *if φ is satisfiable.*
- $Pr(\exists x_i \in dom(x_i) \dots Q_n x_n \in dom(x_n) : \varphi)$
  $= \sup_{v \in dom(x_i)} Pr(Q_{i+1} x_{i+1} \in dom(x_{i+1}) \dots Q_n x_n \in dom(x_n) : \varphi[v/x_i])$.
- $Pr(\text{\reflectbox{R}}_{\pi_i} x_i \in dom(x_i) \dots Q_n x_n \in dom(x_n) : \varphi)$
  $= \int_{v \in dom(x_i)} Pr(Q_{i+1} x_{i+1} \in dom(x_{i+1}) \dots Q_n x_n \in dom(x_n) : \varphi[v/x_i]) \pi_i(v) dv$.

According to Def. 4.2.2, the maximum probability of satisfaction $Pr(\Phi)$ is recursively computed by resolving the quantifiers from left to right, whereby existential quantifiers are resolved to yield highest (or actually approach the supremum) probability and randomized quantifiers yield the expectation of the remaining formulas. When all quantifiers are resolved, the satisfaction probability of the matrix φ is associated with its satisfiability, i.e., base cases for φ being either satisfiable or unsatisfiable.

Remark 4.2.1. *The semantics of CSSMT can be regarded as a direct extension of SSMT, where the summation for resolving randomized quantifiers is replaced by integration over distribution functions. Semantically, it yields a straightforward extension. However, it leads to a non-trivial extension for the solving procedure, which will be mentioned later.*

Example 4.2.2. *Fig. 4.2 constructs a tree-like structure according to the semantics of CSSMT formula which was introduced in Example 4.2.1. Semantically, Φ determines the maximum probability s.t. there are values for x which are between $[-1,1]$ s.t. for normally distributed values of y the matrix is satisfiable. As a start, we branch the domain of x into three parts, i.e., $x \in [-1, -\frac{1}{3})$, $x \in [-\frac{1}{3}, \frac{1}{3}]$ and $x \in (\frac{1}{3}, 1]$, again for each part, we branch the domain of y into two parts. Until now, all the quantified variables are resolved and we check satisfiability. Take the leftmost branch as an example, when x is in $[-1, -\frac{1}{3}]$ and y is in $(-\infty, 0]$, the matrix cannot be satisfied and we mark the probability of satisfaction as 0. When all the branches have been fully checked, we propagate the probability according to the corresponding quantifiers, for example, y is normally distributed, so the probability that y takes value from $(-\infty, 0]$ is 0.5. If we combine the probability from bottom to top and choose maximum value among the three branches for x (since x is bound by existential quantifier), then we get that the probability of* *satisfaction of* Φ *is* 1*. Notice that if we start from a finer interval partition, i.e., split x or y into more branches, we will get a different structure, but the maximum probability keeps unchanged.*



$\Phi = \exists x \in [-1,1] \text{Я}_{\mathcal{N}(0,1)} y \in (-\infty,+\infty) : (x^2 \le \frac{1}{9} \vee a^3 + 2b \ge 0) \wedge (y > 0 \vee a^3 + 2b < -1)$
$Pr(\Phi) = \max(0.5, 1, 0.5) = 1$
x
$x \in [-1, -\frac{1}{3})$
$x \in [-\frac{1}{3}, \frac{1}{3}]$
$x \in (\frac{1}{3}, 1]$
$Pr = 0.5 \cdot 0 + 0.5 \cdot 1 = 0.5$
$Pr = 0.5 \cdot 1 + 0.5 \cdot 1 = 1$
$Pr = 0.5 \cdot 0 + 0.5 \cdot 1 = 0.5$
y
y
y
$y \in (-\infty, 0]$ $Pr = 0.5$
$y \in (0, \infty)$ $Pr = 0.5$
$y \in (-\infty, 0]$ $Pr = 0.5$
$y \in (0, \infty)$ $Pr = 0.5$
$y \in (-\infty, 0]$ $Pr = 0.5$
$y \in (0, \infty)$ $Pr = 0.5$
$a^3 + 2b \ge 0$ $a^3 + 2b < -1$ unsat
$a^3 + 2b \ge 0$ sat
$a^3 + 2b < -1$ sat
sat
$a^3 + 2b \ge 0$ $a^3 + 2b < -1$ unsat
$a^3 + 2b \ge 0$ sat
$Pr = 0$
$Pr = 1$
$Pr = 1$
$Pr = 1$
$Pr = 0$
$Pr = 1$

Fig. 4.2 Semantics of a CSSMT formula depicted as a tree.

Remark 4.2.2. *Notice that for SSMT, we can construct a unique semantic tree since the domains w.r.t. both existential bound variables and randomized bound variables are discrete. For CSSMT, the case is more complicated due to the continuity of variables, the domain can be arbitrarily split, so different from the genuine semantic tree (which is an uncountably infinite branching tree directly arising from the quantifiers) the structure we used for CSSMT are based on coalescing equivalent branches in that tree into a finite number of equivalence classes, which not only facilitates graphical presentation but also forms the idea underlying the actual solving algorithm. From this point of view, the resulted tree-like structure is not unique. However, the maximum probability is always same.*

As mentioned before, the algorithm for solving CSSMT cannot be directly borrowed from SSMT solving, after introducing the semantics for CSSMT, we will explain the reasons:

- SSMT solving can collapse to SMT solving plus probability analysis, where the matrix can be substituted by discrete values and SMT solving can be performed, then the probabilities are combined reversely, i.e., from innermost quantified variable to the outermost. However, due to the dense domain for CSSMT formula, the substitution approach fails;
- The probability can be precisely computed for SSMT since for both types of quantifiers, the probabilities involved are precisely known, and the operations on them will not lose

precision, i.e., for existential quantifier, the max operator is used, and for randomized quantifier, the summation and multiplication operators are used. These cases don't work for the continuous extension;

- The SMT core can be separated from the solving procedure for SSMT, e.g., over which the probability analysis can be built, SMT core is invoked only at the bottom, i.e., when all the quantifiers have been resolved. The advantage of such separation is that one can use lots of state-of-the-art SAT/SMT solvers (certainly it should support non-linear arithmetic), since, in the end, only the satisfiability of the matrix should be given; however, for continuous case, the existing SMT solvers are not easy to be integrated into the whole framework, since for the worst case, CSSMT solving tends to find all the solutions, but most of the solvers only return a candidate model.

In the following chapters, we would like to introduce a rule-based solving procedure for CSSMT. Our approach is based on a combination of the DPLL($\mathcal{T}$) [NOT06] and ICP (Interval Constraint Propagation, [RVBW06, VHMK97]) algorithms, as first implemented in the iSAT solver for rich arithmetic SMT problems over the $\mathbb{R}^n$ [FHT$^+$07], and on branch-and-prune rules for the quantifiers generalizing those suggested in [FHT$^+$07, Tei12]. We extend these methods so that they can deal with SSMT formulas with continuous domains. Our solving procedure therefore is divided into three layers: an SMT layer manipulating the Boolean structure of the matrix of the formula, an interval constraint solving layer reasoning over the conjunctive constraint systems in the theory part of the formula, and a stochastic SMT layer reasoning about the quantifier prefix. Each layer is defined by a set of rules to generate, split, and combine so-called *computation cells*, where a computation cell is a box-shaped part of the $\mathbb{R}^n$, i.e., the problem domain of the constraints. The solver thereby approximates the exact satisfaction probability of the formula under investigation and terminates with a conclusive result whenever the approximation gets tight enough to conclusively answer the question whether the satisfaction probability is above or below a certain specified target.

Chapter 5

Stochastic Hybrid Automata and CSSMT-Based Bounded Reachability Analysis

An alternative way to deal with the interplay of randomness and continuous time is featured by Stochastic Hybrid Automata (SHA) [AKLP10, FHH$^+$11, HLS00], which is a powerful model capturing both nondeterminism and randomness, mixing both discrete jumps and continuous evolution.

SSMT – proposed by Tino Teige – has a strong connection with discrete time Markov decision process (DTMDP). Bounded reachability analysis on DTMDP can be encoded to SSMT [TF08]. The reachability problem considers a system starting from given initial states, through the behaviors modeled by DTMDP and then check whether the targets can be reached, where the initial conditions, transition relations of DTMDP and targets can be described by logical formulas, the nondeterminism of choices are linked to existential quantifiers, meanwhile the randomness of jumps is captured by randomized quantifiers, which yields an SSMT formula. The bounded reachability analysis can thus be reduced to computing the maximum probability of satisfaction of such SSMT formula.

Similarly, a CSSMT formula is able to mimic the behavior of stochastic hybrid automaton (a limited version), i.e., we can encode an SHA to a CSSMT formula and perform reachability analysis on it.

In this chapter, we will briefly introduce the definition of stochastic hybrid automaton and clarify a simplified version which can be encoded by CSSMT; then we will formalize the bounded reachability problems and show the potential applications of CSSMT. The related work is concluded at the end of this chapter.

5.1 Introduction to Stochastic Hybrid Automaton

Deterministic and non-deterministic hybrid systems have been the topic of interest in recent years, by contrast, relatively few classes of stochastic hybrid systems have been studied in detail. Even though deterministic hybrid systems can capture a wide range of behavior encountered in practice, stochastic hybrid systems play an import role in modeling. In the real world, uncertainties can appear in most of the applications.

There are lots of stochastic models proposed by researchers, i.e., from Piecewise Deterministic Markov Processes [Dav93], Switched Diffusion Processes [GAM97], over Stochastic extensions of timed automata [KNSS99], to Stochastic Hybrid Automata [HLS00]. The difference of such models is where to introduce the randomness: some models allow the randomness introduced into continuous evolution, some don't allow. In this section, we will introduce the concept of Stochastic Hybrid Automaton (SHA) where the randomness can be modeled in both continuous evolution and discrete jumps.

The stochastic hybrid automata we are going to consider consist of states and transitions, the continuous evolution is featured by relations formulated in states, and the discrete jump is named with the term "transition".

A possible execution of a stochastic hybrid automaton starts from an initial state (or specified by an initial probability measure), the variables are then updated according to the relation formulated by vector fields in given state, when some conditions are satisfied or some requirements are fulfilled, the discrete jump is triggered, the target state is specified by the transition relation or transition measure.

In the following, we will introduce the definition of stochastic hybrid automaton. In this work, we are not concentrating on the technical issues regarding the semantics stochastic hybrid automata. However, we are aiming to show that a class of stochastic hybrid automaton can be encoded to CSSMT and further handled by CSSMT solving procedure. To encode a general form of stochastic hybrid automaton is impossible since SHA can be formalized in a very general way where stochastic differential equations (SDE) are introduced to model the randomness in continuous evolution. CSSMT cannot model SDE precisely except when approximation techniques are used.

5.1.1 Definition of Stochastic Hybrid Automaton

One definition of stochastic hybrid automaton (a modified version of [HLS00]) can be found in the following:

Definition 5.1.1. *A stochastic hybrid automaton is a collection* $H = (Q, X, Dom, f, init, G, step, R)$ *where:*

- *Q is a countable set of discrete state;*
- *$X = (X_d, X_r)$, X_d is a set of deterministic variables and X_r is a set of random variables, all variables have real domain $\mathbb{R}$;*
- *$Dom : Q \to 2^X$ assigns to each state a subset of X;*
- *$f : X \times T \to 2^{X'}$ is a function mapping the current variables X to the updated variables X', which is time-dependent;*
- *$init : \mathcal{B}(Q \times X) \to [0,1]$ is an initial probability measure on $(Q \times X, \mathcal{B}(Q \times X))$ concentrated on $\bigcup_{i \in Q} \{i\} \times Dom(i)$;*
- *$G : Q \to 2^{2^X}$ assigns to each state $i \in Q$ the potential guards $G(i) \subset 2^X$ which can trigger the transitions;*
- *$step : Q \times G \to 2^{\Delta_f(Q)}$, is a measurable transition function, where $\Delta_f(Q)$ denotes the set of finite measures, we require that $step(q,g) \neq \emptyset$ for all $q \in Q$.*

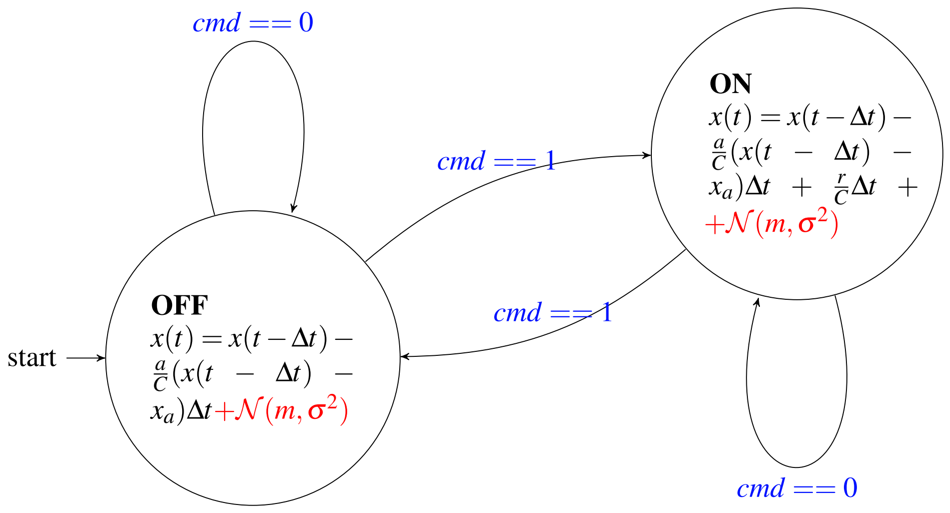

$cmd == 0$
ON
$x(t) = x(t - \Delta t) - \frac{a}{C}(x(t - \Delta t) - x_a)\Delta t + \frac{r}{C}\Delta t + +\mathcal{N}(m, \sigma^2)$
$cmd == 1$
$cmd == 1$
start
OFF
$x(t) = x(t - \Delta t) - \frac{a}{C}(x(t - \Delta t) - x_a)\Delta t + \mathcal{N}(m, \sigma^2)$
$cmd == 0$

Fig. 5.1 Regulating the temperature of a room by a thermostat

Example 5.1.1. *Let's reconsider the temperature regulating problem which we introduced in Chapter 4, the stochastic hybrid automaton has been depicted in Figure 5.1. The Definition 5.1.1 can be used to interpret the model in the following way:*

- *$Q = \{OFF, ON\}$, the thermostat contains two discrete states: ON indicates that the room temperature is increasing and OFF indicates that the room temperature is decreasing;*

- *$X = (x, \omega, cmd)$, x is a continuous variable which represents the room temperature, ω is a random variable which shows the uncertainty introduced by measurement, cmd is a discrete variable which stands for the switching commands which can be issued to the thermostat;*

- *Regarding to the domain of each variable, the physical quantity x should lie in a reasonable range, i.e., an interval $[lb, ub]$ with lower and upper bounds; the random variable ω used to model the measurement error can be any distributions depending on the problem, noticing that some continuous distribution like normal distribution have infinite domain, it may yield a negative quantity which will never happen in practice, for this we can confine such distributions in a range of interest, in most cases it is enough to consider a tailed distribution since the probability lying in such range is high enough, i.e., the range $[\mu - 3\sigma, \mu + 3\sigma]$ for a normal distribution $\mathcal{N}(\mu, \sigma^2)$ has the chance of 99.7% lying in it; at last, the discrete command cmd can be either 0 – staying in the current state, or 1 – switching to the opposite state;*

- *In the ON mode, the temperature updates according to the following formula: $x' = x(t) = x(t-\Delta t) - \frac{a}{C}(x(t-\Delta t) - x_a)\Delta t + \mathcal{N}(m, \sigma^2)$, and in the OFF mode, the corresponding formula is: $x' = x(t) = x(t-\Delta t) - \frac{a}{C}(x(t-\Delta t) - x_a)\Delta t + \frac{r}{C}\Delta t + \mathcal{N}(m, \sigma^2)$, where ω is random variables, a, C, r and x_a are some physical quantities;*

- *The initial probability density of the system can be described by a Dirac pulse if the initial temperature is given, which is:*

  $$init(q, X) = \delta_{(OFF,(x_0,0,0))}(q, X)$$

  which means that initially, the thermostat is OFF and the temperature is x_0, the measurement error and switching command are both set to 0; If the initial temperature is given by a range, we can reformulate the initial distribution by the following uniform distribution:

  $$init(q, X) = \begin{cases} \mathcal{U}(lb, ub) & \textit{for } q = OFF, \textit{ and } X = (lb \le x \le ub, 0, 0) \\ 0 & \textit{otherwise} \end{cases}$$

  which means that the initial temperature could take arbitrary value from the range bound by lb and ub in a uniform manner;

- *$G(OFF) = \{g_1, g_2\} = \{\{X = (x, \omega, cmd) \mid cmd == 0\}, \{X = (x, \omega, cmd) \mid cmd == 1\}\}$ contains two guards which indicate the commands can be issued for state OFF; the* *same definition can be applied for the ON state also with two guards, naming by g_3 and g_4 for $cmd == 0$ and $cmd == 1$ command respectively;*

- *At last, the transition functions for each mode and guard can be defined: $step(OFF, g_1) = \delta_{(OFF,(x,0,0))}(q,X)$, $step(OFF, g_2) = \delta_{(ON,(x,0,))}(q,X)$, $step(ON, g_3) = \delta_{(ON,(x,0,0))}(q,X)$, and $step(ON, g_4) = \delta_{(OFF,(x,0,0))}(q,X)$, all of the transition functions are Delta pulses since they are all deterministic transitions, the x variable is unchanged after each transition, however, ω and cmd are set to default values.*

5.1.2 Execution of Stochastic Hybrid Automaton

According to Definition 5.1.1, the execution of SHA can be easily identified, which is a run $\alpha_t = (Q(t), X(t))$.

- $\alpha_0 = (Q(0), X(0))$ is a $Q \times X$-valued random variable extracted according to the probability measure *init*;
- For $t \in [T_i, T_j)$, $Q(t) = Q(T_i)$ is a constant and X is updated according to the function f;
- T_j is the time when transition from $Q(T_i)$ is triggered given that $G(Q(T_i)) \neq \emptyset$;
- Assuming the $g \in G(Q(T_i))$ is selected, $X(T_j^-) \in g$, where $X(T_j^-)$ denotes $\lim_{t \uparrow T_j} X(t)$;
- The probability distribution of $(Q(T_j), X(T_j))$ is governed by transition measure function $step(Q(T_i), g)$;
- The process repeats the above procedure.

The semantics of the execution can be interpreted recursively on the set of runs, i.e., $\delta_t = (Q(t), X(t))$, we will clarify the probability measure on δ_t as following:

- $Pr(\delta_0) = \int_{(q,x) \in (Q(0), X(0))} init(q,x) d(q,x)$, which specifies the probability that initial state is chosen in the subset $(Q(0), X(0))$;
- $Pr(\delta_{t+\Delta t}) = Pr(\delta_t) \cdot \int_{x_r \in X_r(t+\Delta t)} \pi_{X_r}(x_r) dx_r$, if the convolution occurs in the state;
- $Pr(\delta_t) = Pr(\delta_{t^-}) \cdot \sum_{q \in Q(t), q^- \in Q(t^-), g \in G(q^-)} step(q^-, g)(q)$, if the discrete jump happens.

5.2 Translation from Stochastic Hybrid Automaton to CSSMT

In this section, we will employ a reduction procedure which reduces stochastic hybrid automata to continuous stochastic SMT, so that we can perform probabilistic bounded model checking (PBMC). Since the general SHA could have more complicated structure for continuous evolution, i.e., a function, an ordinary differential equation (ODE), a vector field, or even a stochastic differential equation (SDE), our current treatment of CSSMT doesn't contain ODE/SDE, therefore the general SHA cannot be handled by CSSMT. We would like to employ a simplified version which is still popular in system design and verification.

We consider an SHA $H = (Q, X, Dom, f, init, G, step, R)$ and simplify the function $f : X \times T \to 2^{X'}$ with discretized time steps. In reality, especially in embedded systems, time discretization is very common, since the digital components perform the computation on a discrete sequence. This can be mimicked by difference equations, discretized ODE/SDE and so on.

With this modification, we will now explain how to encode a simplified SHA $H = (Q, X, Dom, f, init, G, step, R)$ to CSSMT:

1. Encoding the initial conditions: if the initial conditions are deterministic, one can directly write $q_0 == q_{init} \wedge X_0 == X_{init}$, where Q and X are the domains of variables, q_0 and X_0 are the variables representing the initial values and should be bound by existential quantifiers, i.e., $\exists q_0 \in Q \exists X_0 \in X$; however, if the initial conditions are determined by probabilistic density, the randomized quantifier is used, i.e., $\text{Я}_{\pi_0}(q_0, X_0) \in (Q, X)$, where π_0 is initial probability distribution, here we assign to q_0 and X_0 initial values, the semantics of this formula, according to Chapter 4, is the probability that q_0 and X_0 are given to the specified values;

Example 5.2.1. *In this running example, we will show how to translate the temperature regulating problem – which is modeled by SHS as shown in Figure 5.1 – to a CSSMT formula. We will follow the procedure step by step. We firstly encode the initial conditions. Assume that the initial room temperature is uniformly distributed in a range, i.e., $[T_l^0, T_u^0]$ and the thermostat is initially switched to OFF. The condition can be formulated as follow:*

$$\underbrace{(T_l^0 \leq x_0 \leq T_u^0) \wedge (q_0 == OFF)}_{\mathcal{I}} \tag{5.1}$$

and the quantifiers are $\exists q_0 \in \{OFF, ON\} \text{Я} x_0 \in \mathcal{U}(T_l^0, T_u^0)$, where $\mathcal{U}(T_l^0, T_u^0)$ is a uniform distribution ranging over the temperature bounds.

2. Encoding the evolution in one state: if the system is in state q which contains variables $x = (xd, xr)$, the update from step $i-1$ to step i can be formulated as following: $q_i == q \wedge x_i == f(x_{i-1}, \Delta t)$, which is based on the mapping function f in state q. Notice that the variable x contains both deterministic parts and random parts, so different quantifiers are required. The existential quantifiers should be added for the state variable q_i and deterministic variable xd_i, while xr_i is bound by randomized quantifier, i.e., $\exists q_i \in Q \exists xd_i \in dom(xr) \text{ꓤ}_{\pi_{xr}} xr_i \in dom(xr)$. This formula formulates one-step update, given that the mapping function we consider is discrete;

Example 5.2.2. *In each state (ON or OFF), the room temperature is changed according to the formulas, here we use x_{i-1} and x_i to represent the previous and updated room temperature, they take value from suitable temperature range: $\exists x_{i-1} \in [T_l^{i-1}, T_u^{i-1}]$ and $\exists x_i \in [T_l^i, T_u^i]$. Note that the measurement error is modeled by a normal distributed random variable, i.e., $\text{ꓤ}\omega_i \in \mathcal{N}(m, \sigma^2)$. Then the corresponding behaviour can be formulated as a disjunction of two single state evolution:*

$$\begin{gathered}(q_i == OFF \wedge x_i == x_{i-1} - \tfrac{a}{c}(x_{i-1} - x_a)\Delta t + \omega_{i-1}) \vee \\ \underbrace{((q_i == ON \wedge x_i == x_{i-1} - \frac{a}{c}(x_{i-1} - x_a)\Delta t + \omega_{i-1} + \frac{r}{c}\Delta t)}_{\mathcal{C}_i}\end{gathered} \tag{5.2}$$

and the quantifiers are $\exists x_{i-1} \in [T_l^{i-1}, T_u^{i-1}] \exists x_i \in [T_l^i, T_u^i] \text{ꓤ}\omega_i \in \mathcal{N}(m, \sigma^2)$;

3. Encoding the transition: the transition triggered by the guards g for ith step can be formulated as following: $\bigvee_k (q_{i-1} == q \wedge g \wedge t == t_k \Rightarrow q_i = q'_k)$. When guard g triggers a transition from state q, we know that the following behaviour is dominated by the step function which contains different probability measure, we use t_k to represent all the possible measures so that it can be chosen nondeterministically, once t_k is chosen, the following jump, i.e., the target state can be decided by corresponding probability measure, we use q'_k to represent the target state, which is a random variable and its distribution function depends on which t_k is chosen. Here we only mentioned the transition triggered by g; the full representation should consider all the guard and connect them by $\vee$. From the above description, we can clarify the quantifiers for each variables involved, i.e., $\exists g \in G \exists t_k \in Trans \text{ꓤ}_{\pi_1} q'_1 \in Q \cdots \text{ꓤ}_{\pi_n} q'_n \in Q \exists q_i \in Q$. Noticing that $|Trans| = n = |step(q, g)|$, which is the number of possible measures;

Example 5.2.3. *The discrete transitions are guarded by a switching command, i.e., $u_i =$ or $u_i = 1$, we use q_{i-1} and q_i to represent the previous state and target state respectively, both* *with domain $\{OFF, ON\}$. The transitions can be formulated as following:*

$$\underbrace{(u_i == 0 \Rightarrow q_i == q_{i-1}) \vee (u_i == 1 \Rightarrow q_i == \neg q_{i-1})}_{\mathcal{T}_i} \tag{5.3}$$

and the quantifiers are $\exists q_{i-1} \in \{OFF, ON\} \exists q_i \in \{OFF, ON\} \exists u_i \in \{0,1\}$;

4. Obtaining the CSSMT formula: the corresponding CSSMT formula with k steps can be obtained by connecting the initial conditions, one state evolution, transition relations together with the targets of the problem by $\wedge$, the quantifiers identified from the previous steps are then added in front of the formula .

Example 5.2.4. *As the target, we want to regulate the room temperature so that it is neither too warm nor too cold, i.e., control the temperature in a range. For this purpose, we add a constraint for each step, i.e.,*

$$\underbrace{C_l^i \leq x_i \leq C_u^i}_{\mathcal{S}_i} \tag{5.4}$$

which indicates that the room temperature x_i is expected to be in the set $[C_l^i, C_u^i]$.

Altogether the dynamic of the system can be formalized as an CSSMT formula

$$\begin{aligned} &\text{Я} x_0 \in \mathcal{U}(T_l^0, T_u^0) \exists q_0 \in \{OFF, ON\} \exists u_i \in \{\text{“0”}, \text{“1”}\} \\ &\vdots \\ &\exists x_{i-1} \in [T_l^{i-1}, T_u^{i-1}] \exists x_i \in [T_l^i, T_u^i] \text{Я} \omega_i \in \mathcal{N}(m, \sigma^2) \cdots : \mathcal{I} \wedge \bigwedge_{i=1}^{k} (\mathcal{C}_i \wedge \mathcal{T}_i \wedge \mathcal{S}_i) \end{aligned} \tag{5.5}$$

where k is the number of computation steps analyzed.

5.3 Bounded Reachability Problems represented by CSSMT

In the previous section, we introduced how to translate the behavior of a stochastic hybrid automaton to a continuous stochastic SMT formula, which mimics the initial conditions and transition rules for each step. In computer science, an essential problem in hybrid systems theory is reachability analysis, which aims to evaluate whether a given system will reach certain unsafe states, starting from certain initial states. This problem is associated with the safety verification problem: if the system cannot reach any unsafe state, then the system is declared to be safe. In the probabilistic setting, the safety verification problem can be formulated as that of checking whether the probability that the system trajectories reach an unsafe state from its initial states can be bounded by some given probability threshold.

As we said before, the semantic of CSSMT is the maximum probability of satisfaction, i.e., how likely the formula can be satisfied, and this can be directly connected to reachability problem. For example, we have shown how to encode an SHA to CSSMT. If we formulate the target conditions as well, we take the conjunction of the encoded behavior and the target, we can get a CSSMT formula, and the probability of the CSSMT formula is exactly what we need, i.e., the maximum probability of reaching the target.

Now we can conclude the bounded reachability problem as following:

> Given a simplified stochastic hybrid system $H = (Q, X, Dom, f, init, G, step, R)$, a CSSMT formula can be obtained according to the translation procedure in the previous section. Without loss of generality, we refer the CSSMT formula as $\mathcal{Q} : \mathcal{I} \wedge \bigwedge_k \varphi_k$, where $\mathcal{Q}$ is the sequence of quantifiers, $\mathcal{I}$ formulates the initial conditions and φ_k represents the one-step transition. If we are going to name the target formula by $\mathcal{S}$, then the resulting CSSMT formula $\Phi = \mathcal{Q} : \mathcal{I} \wedge \bigwedge_k \varphi_k \wedge \mathcal{S}$ stands for the corresponding reachability problem, and $Prob(\Phi)$ is the maximum probability of reaching the target $\mathcal{S}$ in k steps starting from initial condition $\mathcal{I}$, with respect to the system H.

In the previous part, we have shown an example that shows how to encode the stochastic hybrid automata to CSSMT formulas and this problem will be solved after we introduce the detailed solving procedure for CSSMT. In the running example, since the targets are given, the semantic of the formula is the maximum probability that the temperature is controlled into the desired range with upper and lower bounds, which can be summarized in the following sense:

Proposition 5.3.1. *Let H a simplified Stochastic Hybrid Automaton and Φ its k-depth encoding under initial conditions $\mathcal{I}$ with target $\mathcal{S}$, then $Prob(\Phi) \in [lb, ub]$ iff H satisfies the k-step bounded model checking (BMC) problem w.r.t. the upper and lower probability bounds lb and ub, initial condition $\mathcal{I}$ and targets $\mathcal{S}$.*

Proof. The correctness of reduction from H to Φ has been detailed in [Tei12], the only difference is that to encode the random variables, we use continuous distributions instead. Let $I_0, I_1, \cdots, I_k$ be intervals in $\mathbb{R}^{d+r}$, the cylinder set $Cyl(Q_0, I_0, Q_1, I_1, \cdots, Q_k, I_k)$ is defined by $\{(Q(t), X(t)) \in Path \mid \forall 0 \leq i \leq k. Q(i) = Q_i \wedge \forall 0 \leq i \leq k. X(i) \subseteq I_i\}$, where $Path$ contains all the possible executions. Given the correctness of reduction, we can conclude:

$$
\begin{aligned}
& Prob(H \text{ satisfies k-step BMC }) \\
\Leftrightarrow\quad & Prob(\text{all the paths starting from } \mathcal{I} \text{ and entering } \mathcal{S} \text{ within k steps}) \\
\Leftrightarrow\quad & Prob(\{(Q(t),X(t)) \in Path \mid (Q(0),X(0)) \subseteq \mathcal{I} \wedge \exists i \leq k.((Q(i),X(i))) \subseteq \mathcal{S}\}) \\
\Leftrightarrow\quad & \sum_{i=1}^{k} Prob(Cyl(\mathcal{I} = (Q_0,I_0),Q_1,I_1,\cdots,\mathcal{S} = (Q_i,I_i))) \\
\Leftrightarrow\quad & Prob(\mathcal{Q} : \mathcal{I} \wedge \bigwedge_k \varphi_k \wedge \mathcal{S}) \\
\Leftrightarrow\quad & Prob(\Phi)
\end{aligned}
$$

□

5.4 Conclusion

In this chapter, we have connected the stochastic hybrid automaton and continuous stochastic SMT. We found that an SHA (with some limitations) can be encoded to a CSSMT formula. One may doubt that why we need such kind of translation, we can directly perform reachability analysis on SHS, this is true and has already attracted a lot of attention. However, CSSMT has strong advantages compared with other techniques in the following sense:

- Different researchers concentrate on different aspects of stochastic hybrid systems, which yields different kinds of models thus various kinds of solving techniques. To design a unique tool which covers most of such models is impossible, if we can translate the behaviour of SHS – even a simplified version – to a couple of constraints, i.e., CSSMT formulas, we can handle such models in a uniform way;

- To handle the SHS directly is a little bit tedious due to the complexity of interaction between continuous and discrete behavior, between deterministic and stochastic behavior. Thus a lot of mathematic computations need to be performed. Most of the methods for SHA verification either employ rather coarse finite-state approximations or are based on inherently approximative statistical techniques, while the CSSMT encoding is correct w.r.t. bounded properties.

Essentially, CSSMT is a satisfiability theory with the consideration of uncertainties, which is not designed only for solving reachability problems on SHS, even though it can handle a subset of those problems. In fact, CSSMT can do more than that, which is able to abstract the constraints from real applications with considering errors, uncertainties, random phenomena etc., which can be seen in the later chapters.

Chapter 6

Solving Procedure for CSSMT formula

In the previous chapters, we proposed the framework of Continuous Stochastic SMT, which is a satisfiability modulo theory with considering the uncertainties of variables. We formalized the semantics of CSSMT, which is denoted to be the maximum probability of satisfaction regarding a given CSSMT formula, which allows us to check how likely the formula can be satisfied given some of its variables are random. Additionally, CSSMT is able to be used for probabilistic model checking problem on stochastic hybrid systems, we thus provided a way to encode the behaviour of an SHA to its semantically equivalent CSSMT and clarified that the probability bounds of resulted CSSMT formula are exactly the probability bounds for bounded checking model problem on the SHA we are considering. From this chapter, we will step deep into the solving procedure and algorithmic consideration for CSSMT.

Due to the existence of non-linear arithmetic in CSSMT formula, the theory is undecidable. Generally, it is impossible to get the precise probability of CSSMT except for some extreme cases. Instead of considering the decision problem, we are trying to find proper bounds to approximate the accurate probability, i.e., the solving problem. In the first part of this chapter, we will formalize the problem we are going to consider, i.e., we want to find an upper bound and a lower bound for the probability of satisfaction under an acceptable precision.

As the solving procedure, our approach is based on a combination of the DPLL($\mathcal{T}$) [NOT06] and ICP (Interval Constraint Propagation, [RVBW06, VHMK97]) algorithms, as first implemented in the iSAT solver for rich arithmetic SMT problems over the $\mathbb{R}^n$ [FHT$^+$07], and on branch-and-prune rules for the quantifiers generalizing those suggested in [FHT$^+$07, Tei12]. We extend these methods so that they can deal with SSMT formula with continuous quantifier domains. The detailed algorithm will be explained in this chapter by a couple of rules, which define a transition system of structural operational semantics.

At the end of this chapter, we will stress the termination and soundness of the proposed solving procedure.

6.1 Problem formalization

Continuous SSMT is an extended satisfiability theory over the undecidable arithmetic domain of Boolean combinations of non-linear constraints involving transcendental functions with consideration of uncertainties. In the previous chapter, we mentioned that the semantics of CSSMT formula is the maximum probability of satisfaction, however, due to the undecidability, to precisely compute the probability is very hard to be achieved. In practice, such precise computation is not needed regarding the following perspectives:

- the numbers are representable with limited precision in computers and embedded systems, so precisely deciding the probability is unrealizable in physical world;
- in most verification problems, one is interested to know whether the probability of a given property is under a specific threshold, i.e., the probability of system error is extremely low; or the target probability can be reached, i.e., the probability that the room temperature staying in the given range should be large enough, etc;
- if the precise computation is not required, the solving procedure doesn't need to explore all the search space, instead, some heuristic pruning methods could be applied and an approximation could be given according to the problem and precision.

Undecidability. The satisfiability problem for SMT formulas 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 [AP10, AF06]. In this sense, the exact probability $Pr(Q : \phi)$ is not computable in case the matrix ϕ stems from an undecidable fragment of arithmetic, we thus formulate the goal of solving as an approximate decision problem. The problem we want our solving engine to resolve therefore is formalized as follows:

Given a CSSMT formula $\Phi = \mathcal{Q} : \phi$, a reference probability δ, and an accuracy ε, a solving procedure which upon termination returns:

- *"GE", if $Pr(\Phi)$ is greater than or equal to $\delta + \varepsilon$;*
- *"LE", if $Pr(\Phi)$ is less than or equal to $\delta - \varepsilon$;*
- *"GE" or "Inconclusive", if $Pr(\Phi) \in [\delta, \delta + \varepsilon]$;*
- *"LE" or "Inconclusive", if $Pr(\Phi) \in [\delta - \varepsilon, \delta]$.*

is called sound. It is called quasi-complete if it terminates whenever $\varepsilon > 0$ (the proof will be found later in this chapter).

According to this formalization, our goal is to decide how the probability of a CSSMT formula $\Phi = \mathcal{Q} : \phi$ can be compared with a reference probability δ under a given accuracy ε. The actual probability is comparable with the resolution $[\delta - \varepsilon, \delta + \varepsilon]$, that is to say:

- if "GE" is reported by the solving procedure, we can guarantee that the actual probability is greater equal than the reference probability, i.e., $Pr(\Phi) \geq \delta$;
- if "LE" is reported, we can guarantee that $Pr(\Phi) \leq \delta$;
- if the actual probability $Pr(\Phi)$ is in the range $[\delta - \varepsilon, \delta + \varepsilon]$, due to the resolution (precision) of the computation, $Pr(\Phi)$ is incomparable with the reference probability δ, so any answers can be reported, i.e., "GE", "LE" or "Inconclusive". However, for "GE" we know at least the $Pr(\Phi)$ is in the right half $[\delta, \delta + \varepsilon]$, for "LE" the $Pr(\Phi)$ lies in the left half $[\delta - \varepsilon, \delta]$.

In order to achieve the mentioned target, we will try to develop a solving procedure in the following sections.

6.2 Algorithm overview

In Chapter 3, we have briefly reviewed the basic algorithm for SMT and stochastic SMT, we will shortly summarize here:

- The solving procedure for SMT is based on DPLL framework and interval constraint propagation (ICP), where the boolean structure is handled by classical decision procedure for SAT, meanwhile the domains of the variables are narrowed by propagating the constraints with interval analysis;
- The solving procedure for SSMT is basically built on the procedure for SMT, the upper level is responsible for resolving the quantifiers and performing the probability analysis, and the lower level deals with the pure SMT formula and checks its satisfiability.

CSSMT relaxes the limitation of SSMT so that the random variables with continuous distributions can be formulated. Intuitively, the solving procedure for CSSMT could be similar to that for SSMT or at least slightly modify that, however, it requires major modifications. As we mentioned before, the reasons can be concluded as follows:

- Due to the discreteness of probability distributions in SSMT, the probability can be precisely manipulated. The quantifiers of SSMT are resolved by branching the discrete domain of variables, each time when the branching is performed, the probability for this branch is precisely known, the satisfiability for this branch is either “True” or “False” [1], correspondingly, the probability for this branch is either counted or ignored; However, for CSSMT it will be much more complicated due to the continuity of the variables, branching cannot be directly achieved. Instead, the interval splitting should be applied. Different with the discrete case, the probability of satisfaction for the branch is unknown, potentially it contains both solutions (which contribute to the final result) and non-solutions (which can be ignored), so more work have to be done for CSSMT;

- The SSMT solving can be reduced to SMT problem when all the quantifiers are resolved, so SSMT solver (i.e., SiSAT) can be easily built on SMT solver (i.e., iSAT/HySAT) since only satisfiability should be checked for the resulting SMT formula. Although SMT formula can also be obtained when all the quantifiers are resolved for CSSMT, checking satisfiability is not enough to compute the probability of satisfaction. Due to the continuity of the variables, we need to map all the solutions and compute the probability measure for them;

- The SSMT solving procedure can be divided into two layers, i.e., probability analysis and SMT layer, probability analysis is responsible for the branching, the decision to prune the non-related branches and synthesis the final result, meanwhile SMT layer is to check the satisfiability of pure SMT formula; CSSMT solving can be roughly divided into three layers: 1) an SMT layer manipulating the Boolean structure of the quantifier-free body of the formula, often called the “matrix”, 2) an interval constraint solving layer reasoning over the conjunctive constraint systems in the theory part of the formula, and 3) a probability analysis layer reasoning about the quantifier prefix. However, all the layers are not separated, i.e., ICP layer aims at analyzing the constraints and narrowing the domains of variables, it may also influence the probability measure for some variables if their domains are changed. Algorithm 3 shows the framework of the solving procedure and the details will be discussed later.

6.3 Rule-based Solving Procedure for CSSMT

In this section, we will present the detailed solving procedure for CSSMT which is based on a series of inference rules. Before getting into those rules, we will start with some definitions.

[1] Here we didn’t mention the “Unknown” case, which is also easy to handle when “Unknown” is obtained, the probability for the unknown branch can be considered as a range which will yield an upper and a lower bound for the final result.

Algorithm 3 Framework for CSSMT Solving

```
Input: A CSSMT formula $\Phi = \mathcal{Q} : \varphi$, a reference probability $\delta$ and precision $\varepsilon$.
Output: “GE”, “LE” or “Inconclusive” as defined before.
1: Initialization;
2: while True do
3:     SMT layer: reasoning about the Boolean structure of the matrix;
4:     CSP layer: reasoning over the constraints systems based on interval analysis;
5:     Stochastic layer: handling the quantifiers and propagating the probability;
6:     if the estimation is comparable with $\delta$ or the precision limit is reached; then
7:         Termination;
8: end while
```

6.3.1 Basic Definitions

Definition 6.3.1. *Given an CSSMT formula $\Phi = Q_1 x_1 \in dom(x_1) \ldots Q_n x_n \in dom(x_n) : \phi$, we define $x_1 \prec \cdots \prec x_n$ as the order of variables in the prefix, with x_1 being minimal.*

Definition 6.3.1 defines the order of variables, which is consistent with the order of their appearance in the quantifier prefix. The order of the prefix is necessary for computing the probability of a formula which is shown in the solving rules.

Another important concept which will frequently be mentioned later is the *computation cell*, which is a structure indicating a subset of the whole search space together with a probability estimation. All the manipulations in our algorithm are based on computation cells.

Definition 6.3.2. *A computation cell is a data structure of the form $(\Phi, \rho, C)^{(p,q)_i}$, where,*

- *Φ is the CSSMT formula we are considering;*
- *ρ: an ordered list (corresponding to the order of variables in $\mathcal{Q}$) which records the interval valuation for each variable;*
- *C is a set of constraints, which is used to record the constraints which are required to be satisfied for this cell;*
- *$(p,q)_i$ is a probability estimation for this cell, where p and q are lower and upper bounds correspondingly, and the subscript $(\cdot)_i$ means that the probability is estimated for the formula by chopping off the quantifier prefix before variable x_i.*

Essentially, the solving procedure for CSSMT splits the whole search space into subparts and works on them, and the final result is obtained by merging the boxes, as mentioned before. The solving procedure performs the reasoning on the Boolean structure of the formula, at the same time interval analysis is applied to narrow the search space, the search space is then split into subparts if no further actions can be used for the current one. Sub boxes are then merged to estimate the probability and the solving procedure terminates if the reference probability is comparable to the obtained probability. From this point of view, the computation cell is a data structure which helps to memorize some necessary information for each box, i.e., the probability bound and its searching space. Moreover, the use of computation cell makes the solving procedure more comfortable to be manipulated, which can be seen clearly in the following parts.

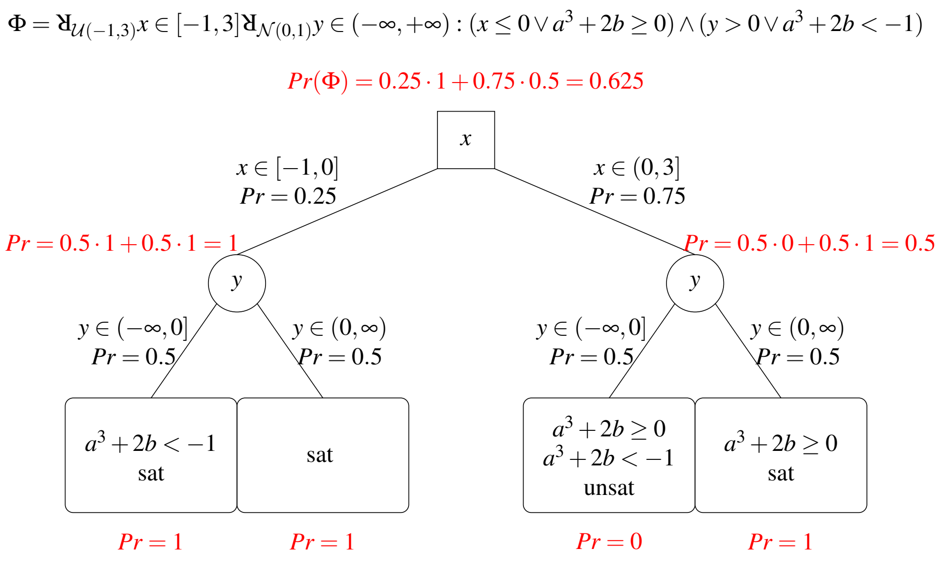

$\Phi = \text{Ⅎ}_{\mathcal{U}(-1,3)} x \in [-1,3] \text{Ⅎ}_{\mathcal{N}(0,1)} y \in (-\infty,+\infty) : (x \le 0 \vee a^3+2b \ge 0) \wedge (y > 0 \vee a^3+2b < -1)$
$Pr(\Phi) = 0.25 \cdot 1 + 0.75 \cdot 0.5 = 0.625$
x
$x \in [-1,0]$
$Pr = 0.25$
$x \in (0,3]$
$Pr = 0.75$
$Pr = 0.5 \cdot 1 + 0.5 \cdot 1 = 1$
$Pr = 0.5 \cdot 0 + 0.5 \cdot 1 = 0.5$
y
y
$y \in (-\infty,0]$
$Pr = 0.5$
$y \in (0,\infty)$
$Pr = 0.5$
$y \in (-\infty,0]$
$Pr = 0.5$
$y \in (0,\infty)$
$Pr = 0.5$
$a^3+2b < -1$
sat
sat
$a^3+2b \ge 0$
$a^3+2b < -1$
unsat
$a^3+2b \ge 0$
sat
$Pr = 1$
$Pr = 1$
$Pr = 0$
$Pr = 1$

Fig. 6.1 Semantics of a CSSMT formula depicted as a tree.

Example 6.3.1. *Consider a CSSMT formula* $\Phi = \text{Ⅎ}_{\mathcal{U}(-1,3)} x \in [-1,3] \text{Ⅎ}_{\mathcal{N}(0,1)} y \in (-\infty,+\infty) : (x \le 0 \vee a^3+2b \ge 0) \wedge (y > 0 \vee a^3+2b < -1)$ *shown in Figure 6.1, where x and y are both random variables and bound with uniform distribution and normal distribution correspondingly. The tree like structure provides a way to solve this formula. The solving procedure has shown how the domains of variables are split and how the probability is analysed. The solving procedure will be introduced later, and in this example, we will explain how to understand the computation cell.*

Consider the first time when we split x. Two branches will be obtained, if we use the notion of computation cell, we have $C_1 = (\Phi, ([-1,0], (-\infty,\infty)), C)^{(0,0.25)_1}$ *and* $C_2 = (\Phi, ((0,3], (-\infty,\infty)), C)^{(0,0.75)_1}$ *(we temporarily ignore the component C). Each cell is a part of the search space and contains the probability information, take* C_1 *ad an example, which shows that the domain of x for this cell is* $[-1,0]$*, the probability estimation is given with lower bound* 0 *and upper bound* 0.25*, this is because x is uniformly distributed within* $[-1,3]$ *and it can* *be guaranteed that the maximum probability of the branch C_1 will not exceed 0.25, however, under the current evaluation, the formula Φ is inconclusive, i.e., it contains both solution and non-solution, so we know nothing about the lower bound and 0 is given. The subscript 1 means that we are trying to resolve the quantifier for the first variable x. If we further split the variable y, we will get $C_3 = (\Phi, ([-1,0], (-\infty, 0])), C)^{(0.5,0.5)_2}$ and $C_4 = (\Phi, ([-1,0], (0, \infty)), C)^{(0.5,0.5)_2}$. Noticing that Φ is satisfiable in both cells, the probability estimation is known according to the probability distribution of y.*

The computation cells can be merged when the satisfiability for each cell is known, i.e., in our example, C_3 and C_4 can be merged, which yields $C_5 = (\Phi, ([-1,0], (-\infty, \infty)), C)^{(0.25,0.25)_1}$, this cell has the same domain but tighter probability estimation compared with C_1, so we can take C_5 as the final computation cell for the left branch.

The same can be applied to the right branch which yields another computation cell $C_6 = (\Phi, ((0,3], (-\infty, \infty)), C)^{(0.375,0.375)_1}$, C_5 and C_6 together result in the final computation cell $(\Phi, [-1,3], (-\infty, \infty)), C)^{(0.625,0.625)_1}$, which tells us the maximum probability of satisfaction of Φ is 0.625.

Remark 6.3.1. *From the example, we have seen that the computation cell is a data structure which plays an important role in recording the information of a subspace regarding the formula. The solving procedure works on the computation cells which together constitute the whole problem we are considering. Each cell is equipped with the boundaries of the domain, the probability estimation and the constraints requirement. The cells split itself when tighter probability estimation is required; it can also be merged in order to be compared with the reference probability. The role of constraints set C is not shown in above example, which is used when reasoning over the Boolean structure of the formula, which will be shown in the following.*

6.3.2 The Solving Procedure for CSSMT

In this part, we will introduce the solving procedure for a CSSMT formula. The algorithm we will present is equipped with the following notations:

- Φ: the CSSMT formula of interest with the form $Q_1 x_1 \in dom(x_1) \dots Q_n x_n \in dom(x_n) : \phi$;
- C: a set collecting the constraints which must be satisfied in the current phase;
- ρ: an ordered list (corresponding to the order of variables in $\mathcal{Q}$) which records the interval valuation for each variable;
- H: a set of computation cells.

The solving procedure is organized by a couple of rules, which are divided into four groups:

- SMT level: contains four rules, i.e., Initialization (INI), Unit Propagation (UP), Interval Constraint Propagation (ICP) and Splitting (SPL);
- Constraint solving level: contains two rules to handle the conflict (CFL) and hull-consistency (CNSIS) of constraints;
- Stochastic SMT level: contains three rules to merge the computation cells, i.e., for existential quantifier (∃-COM), for randomized quantifier (Ɐ-COM) and a rule for lifting the quantifiers (LFT);
- Termination: contains three rules to give the criteria of termination and three possible results can be given, i.e., less equal (LE), greater equal (GE) or inconclusive (INCON).

All the rules share the same structure:

$$\frac{Premises}{L \to R} \qquad \text{(Rule Name)}$$

which transform the proof state from the left of the conclusion (L) into the proof state to the right (R), while the premises (*Premises*) denote the side conditions enabling that transition.

SMT Level.

The SMT level is responsible for reasoning over the Boolean structure of the matrix ϕ. As we have shown in Chapter 2 and Chapter 3, the Boolean structure of an SMT formula is analyzed by unit propagation and interval constraint propagation, i.e., it recursively searches for unit clauses and narrows the search space. CSSMT solving shares the same idea and propagates the unit clauses as well as interval constraints in this level.

The initial configurations for the SMT level are given in the following:

Consider the CSSMT formula $\Phi = Q_1 x_1 \in dom(x_1) \dots Q_n x_n \in dom(x_n) : \phi$, we let:

$$\begin{cases} C: & = & \emptyset \\ H: & = & \emptyset \\ \rho: & = & (\rho_1, ..., \rho_n) = (dom(x_1), \dots, dom(x_n)) \end{cases}$$

i.e., there are no computation cells in the beginning, the constraints set is also empty. The evaluation is assigned with the initial domains of the variables.

From this configuration, the algorithm will start its deduction sequence, which is given by the DPLL rules at the outermost level, which in turn builds on the rules at the constraint solving and the SSMT layer. All the rules share a common structure: they manipulate a set H which contains the relevant box-shaped computation cells within the search space, which itself is a subset of the $\mathbb{R}^n$. The rules select appropriate cells by means of their premises, and they update, split, or combine them according to their conclusions.

Remark 6.3.2. *The rules presented in the following define a transition system in the tradition of structural operational semantics rather than a tree of inferences. The transitions transform the proof state from the left of the conclusion into the proof state to the right, while the premises denote the side conditions enabling that transition. A procedure thus consists of a finite sequence of state manipulating transitions, with each individual transition matching the conclusion of some rule, under the side condition that its source (and maybe some elements of the target) satisfies the premise of the transition rule.*

The first rule (INI) initializes the whole solving procedure without premises, it automatically executes when the solving procedure starts. Rule (INI) adds the first computation cell to H, which contains: 1) the formula $\mathcal{Q} : \phi$ to be decided; 2) ρ is an initial evaluation for each variable; 3) the constraints C which must be satisfied, initially an empty set; 4) a superscript $(p,q)_i = (0,1)_1$ over-approximating the satisfaction probability of the formula when chopping off the quantifier prefix before the variable x_i. Here it means that we estimate the probability for the whole formula from x_1 without chopping off any quantifiers and the lower and upper estimates are 0 and 1 respectively.

$$\frac{}{H \to H \cup \{(\mathcal{Q} : \phi, \rho, \emptyset)^{(0,1)_1}\}} \tag{INI}$$

Example 6.3.2. *We take advantage of a running example to demonstrate our algorithm. We therefore consider the CSSMT formula (as the matrix is a CNF formula, we can regard it as a set of clauses):*

$$\begin{aligned}\Phi = \ & \exists x \in [-10,10]\,\text{Я} y \in \mathcal{U}[5,25]\,\text{Я} z \in \mathcal{U}[-10,10] : \\ & x > 3 \vee y < 1, z > x^2 + 2 \vee y \leq 20, x^2 > 49 \vee y > 7x, x < 6 \vee y \geq z\end{aligned}$$

where $\mathcal{U}[a,b]$ refers to uniform distribution with range $[a,b]$. The initial configurations are $C = \emptyset$, $H = \emptyset$ and $\rho = ([-10,10],[5,25],[-10,10])$. Furthermore, we set $\delta = 0.45$ to be the reference probability.

The second rule (UP) formalizes the unit propagation mechanism in standard DPLL framework, which aims at searching the unit clauses and adds them to the constraints set. The rule reads: if all the disjuncts except one (l') in some clause cannot be satisfied w.r.t. the current evaluation ρ, then this remaining "unit" (l') must hold and is thus added to the set C. Rule (UP) traverses the whole matrix ϕ so that no unit clauses are ignored.

$$\frac{(L \vee l') \in \phi, \rho \not\models L}{H' \cup \{(Q : \phi, \rho, C)^{(p,q)_i}\} \to H' \cup \{(Q : \phi, \rho, C \cdot \langle l' \rangle)^{(p,q)_i}\}} \tag{UP}$$

Interval constraint propagation (ICP) is applied if the domain of some variables can be narrowed w.r.t. the constraints set C, where the set C records the constraints which must be satisfied in the current phase. If we perform interval analysis on such constraints, the ranges of variables may be narrowed, which yields a smaller search area. The corresponding rule reads as follows: if the range of a variable x_j can be narrowed with bound b according to the constraints C and the current evaluation ρ by means of ICP, and if ρ is not yet hull consistent w.r.t. to the new bound (represented by $\not\models_{hc}$, in this case, interval narrowing can be performed by using interval constraint propagation), we update the evaluation set and the probability estimation according to the narrowing $\rho \overset{C}{\rightsquigarrow} (x_j \sim b)$ of x_j computed by ICP:

$$\frac{\rho \overset{C}{\rightsquigarrow} (x_j \sim b), \rho \not\models_{hc} (x_j \sim b)}{H' \cup \{(Q : \Phi, \rho, C)^{(p,q)_i}\} \to H' \cup \{(Q : \Phi, update_\rho(x_j \sim b), C)^{renewal_{\rho_j}(p,q)_i}\}} \tag{ICP}$$

where

$$update_\rho(x_j \sim b)(x_i) = \begin{cases} \rho(x_j) \cap \{z | z \sim b\}, & if\ x_i = x_j \\ \rho(x_j), & otherwise \end{cases}$$

Intuitively, the *update* operator narrows the bound of variable x_j and leaves other variables unchanged. The corresponding change in the probability estimate induced by narrowing a —potentially randomized— variable x_j is reflected by

$$renewal_{\rho_j}(p,q)_i = \begin{cases} (p,q)_i, & if\ x_j \prec x_i \\ \mathbb{P}(\rho(x_i) \times \cdots \times update_\rho(x_j \sim b)(x_i) \times \cdots \rho(x_n))_i, & otherwise \end{cases}$$

where $\mathbb{P}(I_i \times \cdots \times I_n)$ is a safe, interval-arithmetic based probability estimation which returns an interval over-approximating the measure of $I_i \times \cdots \times I_n$ under the distributions attached to the quantifiers. This is generally achieved by integration over interval functions [CMR81].

Remark 6.3.3. *For the rule (ICP), the following points should be noticed:*

- $\rho \not\models_{hc} (x_j \sim b)$ *is required to trigger the rule, which says that the current evaluation* ρ *is not hull consistent regarding the new bound* $x_j \sim b$*, that is to say,* ρ *is reducible under the new bound;*

- *the domain of related variable is updated by simply using the new bound, which yields a smaller range since procedure of ICP will never lead to larger intervals;*
- *if the related variable is connected with a probability distribution, the changing of the domain will also lead to the changing of probability measure, which yields the renewal of probability estimation. Noticing that the subscript i also plays a role here, if the related variable is in front of i-th variable, the probability estimation doesn't change. The subscript i indicates that the probability is estimated for the variables after x_i.*

The rules (UP) and (ICP) are applied recursively, i.e., unit propagation tries to collect the unit clauses and adds them as constraints, interval constraint propagation narrows the domains of variables according to these constraints, which may again lead to new unit clauses.

The next rule presents the splitting of a computation cell, which happens when both rule (ICP) and rule (UP) do not yield further deductions. We say in this situation that ϕ is inconclusive on ρ. We may then perform the splitting rule (SPL) to split the current computation cell into two cells (Any splitting strategies can be applied as long as the size of some interval is reduced, in practice, bisection is applied) and update ρ as well as the probability estimation accordingly.

$$\frac{\rho_j \neq \emptyset,\ \rho_j^1 \cup \rho_j^2 = \rho_j}{\begin{array}{l} H' \cup \{(Q{:}\phi,\rho,C)^{(p,q)_i}\} \rightarrow \\ \qquad H' \cup \{(Q{:}\phi,\rho'\cdot\langle\rho_j^1\rangle\cdot\rho'',C)^{renewal_{\rho_j^1}(p,q)_j}, (Q{:}\phi,\rho'\cdot\langle\rho_j^2\rangle\cdot\rho'',C)^{renewal_{\rho_j^2}(p,q)_j}\} \end{array}} \tag{SPL}$$

The resulting cells are added to the set H for the further use. At the same time, the probability estimation for each cell should be re-evaluated since the probability measure may be changed if the split variable is bound by a randomized quantifier.

Example 6.3.3. *Let us reconsider* $\Phi = \exists x \in [-10,10]\, Ꝗ y \in \mathcal{U}[5,25]\, Ꝗ z \in \mathcal{U}[-10,10] : (x > 3 \vee y < 1) \wedge (z > x^2 + 2 \vee y \leq 20) \wedge (x^2 > 49 \vee y > 7x) \wedge (x < 6 \vee y \geq z)$ *in Example 6.3.2. According to Rule (INI), we add the first computation cell* $(\Phi, ([-10,10],[5,25],[-10,10]), \emptyset)^{(0,1)_1}$ *to the set H. Considering the clause* $x > 3 \vee y < 1$, *we observe that* $y < 1$ *violates the current evaluation, so* $x > 3$ *must be satisfied, which we add to the set C as a unit clause (UP). This yields proof state* $(\Phi, ([-10,10],[5,25],[-10,10]), \{x > 3\})^{(0,1)_1}$. *Since* $x > 3$ *must be satisfied, by interval constraint propagation we can conclude that* $\rho(x) = [-10,10] \overset{C=\{x>3\}}{\rightsquigarrow} \rho'(x) = (3,10]$. *According to the Rule (ICP), we update* ρ *and recalculate the probability interval accordingly. Since x is bound by* $\exists$, *the probability stays unchanged, yielding* $(\Phi, ((3,10],[5,25],[-10,10]), \{x > 3\})^{(0,1)_1}$. *The current evaluation makes* $z > x^2 + 1$ *unsatisfiable, so* $y \leq 20$ *will be added to C (UP). The domain of y is then narrowed to* $[5,20]$ *(ICP). Since y is bound by* Ꝗ, *we need update the probability estimation. This yields* $(\Phi, ((3,10],[5,20],[-10,10]), \{x > 3, y \leq 20\})^{(0,0.75)_1}$.

We cannot guarantee that there are solutions in $[5,20]$*, so the lower bound is* 0*, for the upper bound we can conclude that it will not exceed* 0.75 *since y is uniformly distributed and only the values in* $[5,20]$ *will be considered. We recursively apply the rule (UP) and (ICP), however, no further unit clauses can be found and no intervals can be modified. The next step is thus to apply the rule (SPL), we choose x and split its interval into two parts, then take each part and update the corresponding evaluation and upgrade the probability estimation. As x is bound by* $\exists$*, the probability interval thereby remains unchanged, giving* $H = \{(\Phi,((3,7),[5,20],[-10,10]),\{x>3,y\leq 20\})^{(0,0.75)_1},(\Phi,([7,10],[5,20],[-10,10]),\{x>3,y\leq 20\})^{(0,0.75)_1}\}$.

Constraint Solving Level.

Rules (UP), (ICP) and (SPL) can be recursively applied until two situations are met:

- the first situation is that we find the current evaluation will violate some parts of the formula, or the interval constraint propagation leads to an interval of a variable to be empty, which means that the current evaluation contains no solutions so that it can be ignored;
- the second situation leads to the opposite case, i.e., the current evaluation is hull consistent concerning the constraints set, in which the constraints are unit clauses from each clause. This means that the current cell contains solutions. In such situation, the approximation can be computed according to the requirements and precision. We can benefit from the constraint solving tools or even solve by ourselves, i.e., by a paving procedure which can generate inner approximation and outer approximation for the real solution.

The rule to handle the case of conflicts reads as following: when a conflict is obtained, i.e., if ICP under the current evaluation ρ and constraints C narrows some variables to empty sets, or if ρ violates every part in one clause, the current computation cell can be safely marked with probability 0. This is reflected by rule (CFL):

$$\frac{\rho \overset{C}{\rightsquigarrow} (x_i = \emptyset) \text{ or } L \in \phi \wedge \rho \not\models L}{H' \cup \{(Q:\phi,\rho,C)^{(p,q)_i}\} \to H' \cup \{(Q:\phi,\rho,C)^{(0,0)_1}\}} \tag{CFL}$$

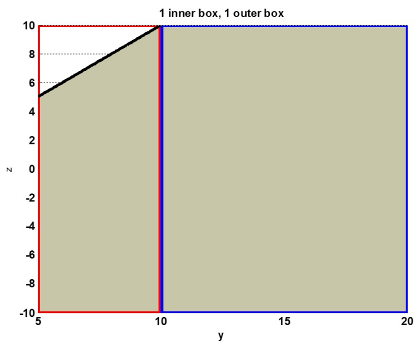
If the current evaluation ρ is hull consistent w.r.t. the actual constraint set C, a paving procedure [GB06] can be invoked to generate an inner approximation and an outer approximation of the actual solution by sets of boxes (i.e., $\{(\cdot)\}^*$ means a number of cells). By computing safe upper (resp., lower) approximations on the probability measures of the outer (resp., inner) approximations of the solution sets, we obtain a safe interval estimate on the satisfaction probability. Rule (CNSIS) assigns these.

$$\frac{\rho \models_{hc} C}{H' \cup \{(Q:\phi,\rho,C)^{(p,q)_i}\} \to H' \cup \{(Q:\phi,\rho',C)^{(p',q')_k}\}^*} \qquad \text{(CNSIS)}$$

Remark 6.3.4. *For the constraints solving layer, the following points should be noticed:*

- *The rule (CFL) marks the probability of the corresponding cell to* 0*, which means that no solution can be found in this cell, at the same time, the subscript is set to* 1*, which indicates that branch has been thoroughly explored and* 0 *is the probability estimation for the whole cell;*
- *The Rule (CNSIS) tells us that when the current evaluation ρ is hull consistent w.r.t. the constraints C, the boxes will be generated to approximate the real solution. In most of the cases, the real solution is impossible to find and the approximation is enough for most of the applications, especially we are interested in the probability of satisfaction here. To find the boxes, we can benefit from a lot of state-of-art constraints solving techniques which can obtain both inner and outer approximation, i.e., interval arithmetic based techniques [BGLC00, VSHS02], affine arithmetic based techniques [GMPK14] etc. Our work for implementation will be partially based on RealPaver [GB06], which is an interval solver using constraint satisfaction techniques. Notice that the subscript k indicates the splitting variable from which the probability is estimated. The evaluation for each subcell is correspondingly narrowed according to the constraint solving procedure, i.e., ρ' is a subset of ρ which yields approximations p' and q'.*

Example 6.3.4. *The computation cells in H are $(\Phi,((3,7),[5,20],[-10,10]),\{x>3,y\leq 20\})^{(0,0.75)_1}$ and $(\Phi,([7,10],[5,20],[-10,10]),\{x>3,y\leq 20\})^{(0,0.75)_1}$. We take the first computation cell into consideration and conclude that the evaluation violate the clause $x^2 > 49 \vee y > 7x$. According to rule (CFL) we mark this computation cell with probability* 0*. This gives $(\Phi,((3,7),[5,20],[-10,10]),\{x>3,y\leq 20\})^{(0,0)_1}$. Now we turn to consider the second computation cell. By performing rule (UP) we get $(\Phi,([7,10],[5,20],[-10,10]),\{x>3,y\leq 20,x^2>49,y\geq z\})^{(0,0.75)_1}$. We observe that the constraints in C are hull consistent w.r.t. the current evaluation ρ. In order to explain the decision procedure, here we generate one inner box and one outer box, according to Rule (CNSIS), we get two computation cells, $(\Phi,([7,10],[5,10],[-10,10]),\{x>3,y\leq 20,x^2>49,y\geq z\})^{(0,0.33*0.75)_2}$ and $(\Phi,([7,10],(10,20],[-10,10]),\{x>3,y\leq 20,x^2>49,y\geq z\})^{(0.66*0.75,0.67*0.75)_2}$, which over- and under-approximates the solutions for C w.r.t. ρ respectively. The subscript* 2 *shows the probability stems from the second variable y. As has been depicted in Fig. 6.2, a light gray area is shown, where the formula Φ is satisfiable. The red box is the corresponding outer box* *and blue is an inner. Since the variable y is bound by randomized quantifier with uniform distribution, the probability estimation should be recomputed.*



1 inner box and 1 outer box.

Fig. 6.2 Inner (blue) and outer (red) approximations for constraint solving problem: $\{x > 3, y \leq 20, x^2 > 49, y \geq z\}$ where $x \in [7,10]$, $y \in [5,20]$ and $z \in [-10,10]$.

Notice that we separate the boxes into inner boxes and outer boxes, inner box is the searching space where all the points in it are solutions, whereas an outer box contains both solutions and non-solution. For outer boxes, the lower probability is not clear since we have no idea about the solutions in them, so the lower probability is set to 0*, however, for inner boxes, both the lower probability and upper probability can be given according to the type of distributions and the precision required.*

Stochastic SMT Level.

Stochastic SMT level manipulates the computation cells and computes the probability of combined cells. It tries to find the cells being able to be merged and calculate the probability according to the quantifiers. Two computation cells are able to be merged if they have adjacent intervals for the same variable x_i. In case that x_i is bound by $\exists$, combining the two cells yields the maximum probability (Rule ($\exists$-COM)); otherwise if bound by ꓤ, the two cells can be combined by adding their probabilities (Rule ꓤ-COM).

$$\frac{\rho_i^1 \uplus \rho_i^2 \text{ is the interval hull of } \rho_i^1 \text{ and } \rho_i^2}{\begin{array}{r} H' \cup \{(Q'\exists x_i Q'' : \phi, \rho' \cdot \langle \rho_i^1 \rangle \cdot \rho'', C)^{(p_1,q_1)_i}, (Q'\exists x_i Q'' : \phi, \rho' \cdot \langle \rho_i^2 \rangle \cdot \rho'', C)^{(p_2,q_2)_i}\} \rightarrow \\ H' \cup \{(Q : \phi, \rho' \cdot \langle \rho_i^1 \uplus \rho_i^2 \rangle \cdot \rho'', C)^{max((p_1,q_1)_i,(p_2,q_2)_i)}\} \end{array}} \quad (\exists\text{-COM})$$

$$\frac{\rho_i^1 \uplus \rho_i^2 \text{ is the interval hull of } \rho_i^1 \text{ and } \rho_i^2}{\begin{array}{r} H' \cup \{(Q'\text{ꓤ} x_i Q'' : \phi, \rho' \cdot \langle \rho_i^1 \rangle \cdot \rho'', C)^{(p_1,q_1)_i}, (Q'\text{ꓤ} x_i Q'' : \phi, \rho' \cdot \langle \rho_i^2 \rangle \cdot \rho'', C)^{(p_2,q_2)_i}\} \rightarrow \\ H' \cup \{(Q : \phi, \rho' \cdot \langle \rho_i^1 \uplus \rho_i^2 \rangle \cdot \rho'', C)^{(p_1,q_1)_i+(p_2,q_2)_i}\} \end{array}} \quad (\text{ꓤ-COM})$$

where the *interval hull* of two sets I_1 and I_2 here is the smallest interval box which contains I_1 and I_2. [2] ρ_i^1 and ρ_i^2 are the intervals to be combined which are related to the i-th variable.

For the rules ($\exists$-COM) and ($\mathsf{Я}$-COM), the order of combination is irrelevant, since the cells are parts of an overall search space and probability for each cell is safely bounded. So parallel computation is possible from this point of view.

If all the computation cells w.r.t. the same variable have been tackled; the probability should be propagated to the preceding variable in the variable order. The lifting rule (LFT) checks all the computation cells in H and will propagate if all its siblings have been combined.

$$\frac{\forall (Q:\phi,\rho',C')^{(\cdot,\cdot)_j} \in H' : j < i}{H' \cup \{(Q:\phi,\rho,C)^{(p,q)_i}\} \rightarrow H' \cup \{(Q:\phi,\rho,C)^{renewal_{\rho_{i-1}}(p,q)_{i-1}}\}} \qquad \text{(LFT)}$$

The premise means that except the cell $(Q:\phi,\rho,C)^{(p,q)_i}$ there is no cell which has subscript greater than i, this means that all the cells with subscript i as well as the cells greater than i have been tackled except this one. At this moment the lifting happens. Notice that the probability estimation should also be recomputed since the estimation now is from the $(i-1)$-th variable.

Example 6.3.5. *In the running example, we have previously reached a state where there are three computation cells in H. By successively applying rules ($\mathsf{Я}$-COM), ($\exists$-COM), and (LFT), we propagate the probability, as depicted in Fig. 6.5.*

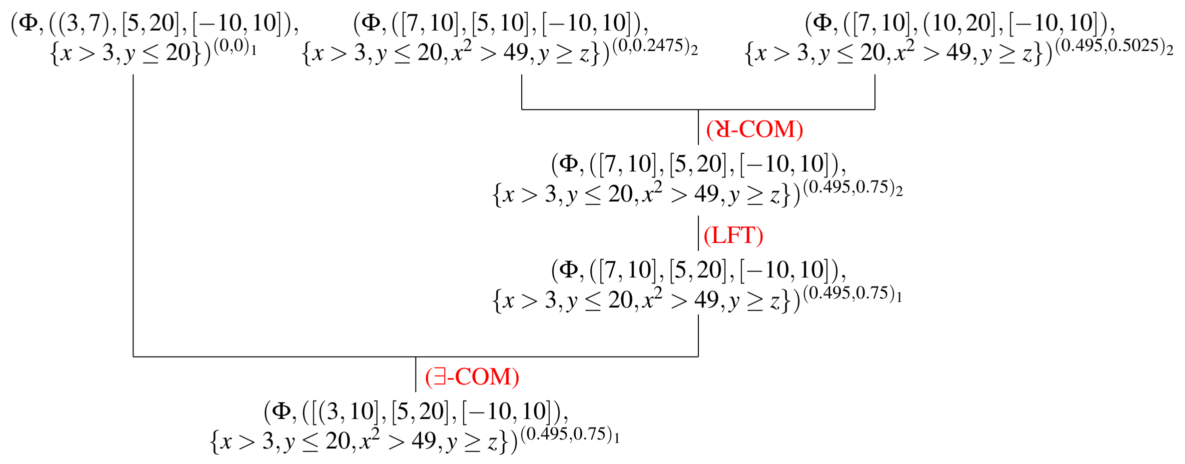

$(\Phi,((3,7),[5,20],[-10,10]),\ \{x>3,y\le 20\})^{(0,0)_1}$
$(\Phi,([7,10],[5,10],[-10,10]),\ \{x>3,y\le 20,x^2>49,y\ge z\})^{(0,0.2475)_2}$
$(\Phi,([7,10],(10,20],[-10,10]),\ \{x>3,y\le 20,x^2>49,y\ge z\})^{(0.495,0.5025)_2}$
($\mathsf{Я}$-COM)
$(\Phi,([7,10],[5,20],[-10,10]),\ \{x>3,y\le 20,x^2>49,y\ge z\})^{(0.495,0.75)_2}$
(LFT)
$(\Phi,([7,10],[5,20],[-10,10]),\ \{x>3,y\le 20,x^2>49,y\ge z\})^{(0.495,0.75)_1}$
($\exists$-COM)
$(\Phi,([(3,10],[5,20],[-10,10]),\ \{x>3,y\le 20,x^2>49,y\ge z\})^{(0.495,0.75)_1}$

Fig. 6.3 Propagate probability by using combination rules.

[2]Here we use *interval hull* instead of *union*, since we also take the rounding effect into consideration. Not every interval bound is representable in computer, thus a safe rounding is needed when we perform the union. In most of the cases, the interval hull coincides with the union.

Termination.

The combined computation cell is finally compared with the reference probability δ. Once the estimated probability for the single computation cell with subscript 1 becomes less than or equal to the reference probability δ, the original formula is concluded to satisfy $P(\Phi) \leq \delta$. Rule (LE) then reports "LE"; rule (GE) does the equivalent for the converse case.

$$\frac{q \leq \delta}{H' \cup \{(Q : \phi, \rho, C)^{(p,q)_1}\} \to \text{LE}} \tag{LE}$$

$$\frac{p \geq \delta}{H' \cup \{(Q : \phi, \rho, C)^{(p,q)_1}\} \to \text{GE}} \tag{GE}$$

If the above two cases cannot be judged under the accuracy ε, i.e., the given precision ε is reached however the probability estimation is not enough to conclude the result, the evaluation of the formula remains inconclusive w.r.t. δ :

$$\frac{q > \delta \wedge p < \delta \wedge |p-q| < \varepsilon}{H' \cup \{(Q : \phi, \rho, C)^{(p,q)_1}\} \to \text{INCON}} \tag{INCON}$$

The rule (INCON) terminates the solving procedure in the sense that no explicit result can be concluded, however, at least we know that the both the reference probability and the actual probability lie in the interval $[p,q]$ with the length no more than ε, so the error between the reference probability and the actual probability is less than ε.

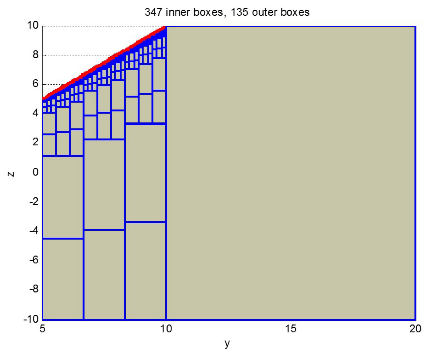
Whenever none of the above three termination rules applies, we have to restart from SMT level and generate more cells, i.e., by applying the rule (SPL) or the rule (INI) with smaller ε (the termination will be discussed later in this chapter).

Remark 6.3.5. *The whole procedure is performed on a number of computation cells maintained in a set H, which changes according to the rules. We name the configuration of set H at each decision step a* snapshot*. We can cache the snapshots in main decision points: such as when the cells are split, or when boxes are generated by other inference mechanisms. If the termination test fails, instead of restarting the whole solving procedure, the procedure can back jump to some snapshot. Backjumping to some restart points is heuristic. A straightforward, yet inefficient way is to simply backtrack to the latest snapshot. However, a more ingenious way can be performed by using Conflict Driven Clause Learning (CDCL) mechanism, allowing to backjump to the snapshot which leads to the conflict.*

Example 6.3.6. *From the previous solving steps, we got the computation cell by the rules of combining, i.e.,* $(\Phi, ([(3,10],[5,20],[-10,10]), \{x > 3, y \leq 20, x^2 > 49, y \geq z\})^{(0.495,0.75)_1}$. *The given δ for this running example is* 0.45*, according to the Rule (GE), we know that*

$Pr(\Phi) > 0.45$ since the lower bound of the estimation is 0.495. The decision procedure terminates here.

Remark 6.3.6. *In our running example, generating two boxes can reach δ. Now let us try with the new reference probability $\delta = 0.70$, a tighter approximation can be achieved by generating more boxes. As shown in Fig. 6.4, we use* RealPaver *[GB06], which is a modeling language implementing interval-based algorithms to process systems of nonlinear constraints over the real numbers, to generate the inner boxes and outer boxes so that a better result can be obtained. By doing so, we get a tighter approximation, which is $[0.7181, 0.7191]$ [3]. For this running example, the real probability can be computed easily, that is $23/32 \approx 0.71875\cdots$, which has been well approximated by the interval we have obtained.*



347 inner boxes and 135 outer boxes.

Fig. 6.4 Inner and outer approximations for constraint solving problem: $\{x > 3, y \leq 20, x^2 > 49, y \geq z\}$ where $x \in [7,10]$, $y \in [5,20]$ and $z \in [-10,10]$ by RealPaver.

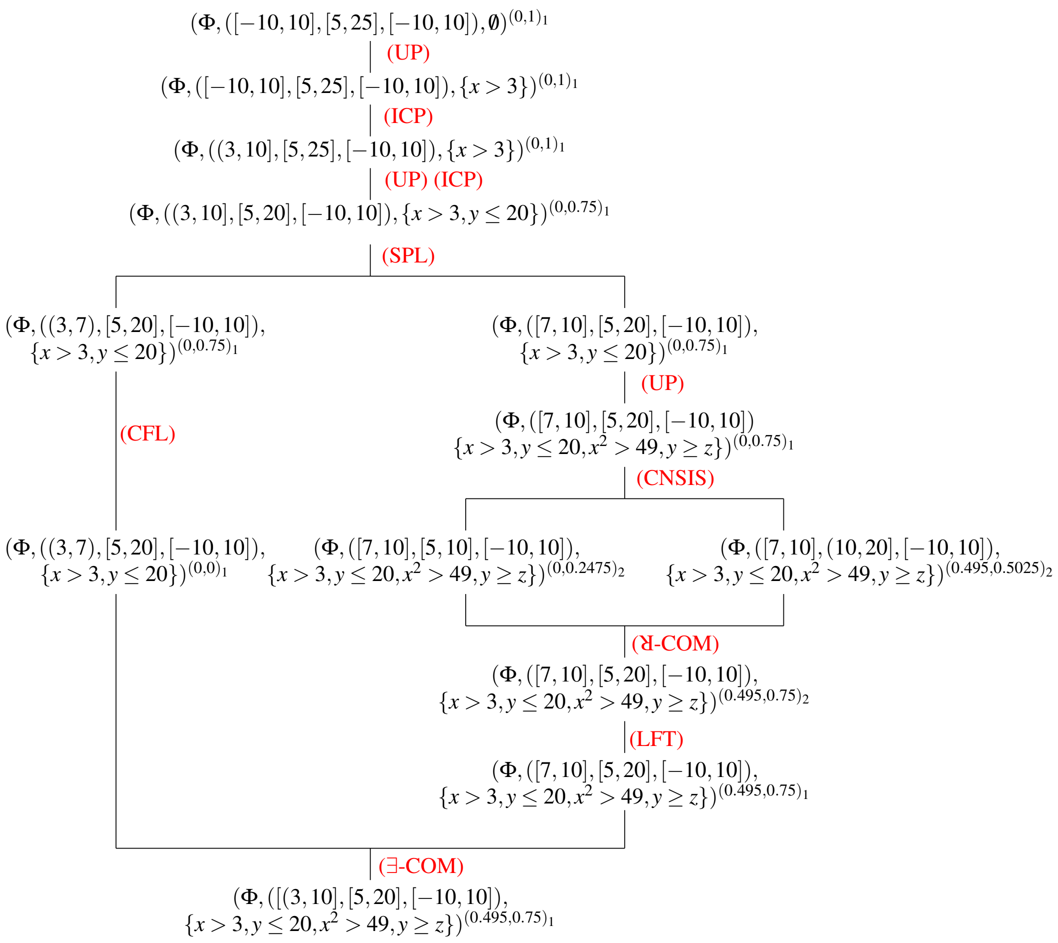
Remark 6.3.7. *The overall solving procedure starts from rule (INI), and then performs (UP), (ICP), (SPL) recurrently, thereby modifying the proof state. When a conflict is obtained or the evaluation is hull consistent regarding the constraints, (CFL) and (CNSIS) are applied to refine the probability estimation. Combining rules are then applied to build the tree-like structure. If the under-approximation is smaller than the given reference number δ or the over-approximation is greater than δ, corresponding results will be returned according to (LE) or (GE), otherwise, we should backtrack to the previous snapshot and perform this procedure again.*

[3] We define the inner box a part of search space and every point inside is a solution, while an outer box contains both solution and non-solution. In this example, 347 inner boxes give the lower probability bound, together with the 135 outer boxes they form the upper bound.

Example 6.3.7. *In Figure 6.5, we summarize the whole solving procedure for the simple CSSMT formula:*

$$\begin{aligned}\Phi = \ & \exists x \in [-10,10]\, Ⅎy \in \mathcal{U}[5,25]\, Ⅎz \in \mathcal{U}[-10,10] : \\ & x > 3 \vee y < 1, z > x^2 + 2 \vee y \leq 20, x^2 > 49 \vee y > 7x, x < 6 \vee y \geq z\end{aligned}$$

with reference probability $\delta = 0.45$. The rules used are also labelled along the arrowed lines.



$(\Phi, ([-10,10],[5,25],[-10,10]), \emptyset)^{(0,1)_1}$
(UP)
$(\Phi, ([-10,10],[5,25],[-10,10]), \{x>3\})^{(0,1)_1}$
(ICP)
$(\Phi, ((3,10],[5,25],[-10,10]), \{x>3\})^{(0,1)_1}$
(UP) (ICP)
$(\Phi, ((3,10],[5,20],[-10,10]), \{x>3, y \leq 20\})^{(0,0.75)_1}$
(SPL)
$(\Phi, ((3,7),[5,20],[-10,10]), \{x>3, y \leq 20\})^{(0,0.75)_1}$
$(\Phi, ([7,10],[5,20],[-10,10]), \{x>3, y \leq 20\})^{(0,0.75)_1}$
(UP)
(CFL)
$(\Phi, ([7,10],[5,20],[-10,10]) \{x>3, y \leq 20, x^2 > 49, y \geq z\})^{(0,0.75)_1}$
(CNSIS)
$(\Phi, ((3,7),[5,20],[-10,10]), \{x>3, y \leq 20\})^{(0,0)_1}$
$(\Phi, ([7,10],[5,10],[-10,10]), \{x>3, y \leq 20, x^2 > 49, y \geq z\})^{(0,0.2475)_2}$
$(\Phi, ([7,10],(10,20],[-10,10]), \{x>3, y \leq 20, x^2 > 49, y \geq z\})^{(0.495,0.5025)_2}$
(Ⅎ-COM)
$(\Phi, ([7,10],[5,20],[-10,10]), \{x>3, y \leq 20, x^2 > 49, y \geq z\})^{(0.495,0.75)_2}$
(LFT)
$(\Phi, ([7,10],[5,20],[-10,10]), \{x>3, y \leq 20, x^2 > 49, y \geq z\})^{(0.495,0.75)_1}$
(∃-COM)
$(\Phi, ([(3,10],[5,20],[-10,10]), \{x>3, y \leq 20, x^2 > 49, y \geq z\})^{(0.495,0.75)_1}$

Fig. 6.5 Solving procedure for Example 6.3.2

6.4 Soundness of the Solving Procedure

In this section, we would like to discuss soundness of the proposed solving procedure, i.e., we would like to show that the algorithm works correctly and terminates with the right answer.

The following lemma will lead to our conclusion:

Lemma 6.4.1. *H is a set of computation cells and $\{(Q:\phi,\rho,C)^{(p,q)_i}\}\in H$ is anyone of them, along the solving procedure the following propositions hold:*

1. *the lower estimation bound p is non-decreasing and the upper estimation bound q is non-increasing;*
2. *for all the quantified variables x_i, the domain of variables $\rho(x_i)$ is non-increasing;*

Proof. We prove Lemma 6.4.1 by checking the rules one by one, where we will explain the propositions hold after applying each rule.

The SMT layer contains four rules, i.e., (INT), (UP), (ICP), (SPL). Rule (INI) initializes the solving procedure with initial configuration, it constructs a computation cell and put it into the H, the propositions hold directly since it only generates a cell without doing anything else. The lower and upper probability are 0 and 1 respectively and ρ corresponds the the initial domain of each variables, i.e., $(Q:\phi,\rho,\emptyset)^{(0,1)_1}$.

Now let us consider a computation cell $(Q:\phi,\rho,C)^{(p,q)_i}\in H$.

Rule (UP) searches the unit clauses according to the current ρ, the unit clauses which should be satisfied are added to the constraints set C, it only changes C. However, p, q and ρ are not changed in this rule, i.e., $(Q:\phi,\rho,C')^{(p,q)_i}$, the propositions hold.

Rule (ICP) performs the interval constraint propagation according to the constraints C, which yields $(Q:\phi,\rho',C)^{(p',q')_i}$. ICP will never increase the domain of variables as it only prunes the non-solutions through the process, thus $\rho'(x_i)\subseteq\rho(x_i)$. In this sense, only the upper probability estimate q may be decreased, i.e., $q'\leq q$ and $p'=p$, which makes the propositions hold.

Rule (SPL) splits the domain of a chosen variable and generates two subcells, i.e., it changes the current configuration $H'\cup\{h=(Q:\Phi,\rho,C)^{(p,q)_i}\}$ to $H'\cup\{h_1=(Q:\Phi,\rho',C)^{(p',q')_j}$, $h_2=(Q:\Phi,\rho'',C)^{(p'',q'')_j}\}$, where h is removed and h_1, h_2 are generated. The domain of split variable is narrowed and and the probability estimates are renewed accordingly, i.e., $\rho'(x_j)\subseteq\rho(x_j)$, $\rho''(x_j)\subseteq\rho(x_j)$, $q'\leq q$, $q''\leq q$ and $p'=p$, $p''=p$. If Rule (UP) and (SPL) are recursively applied to h_1 and h_2, the propositions hold.

Constraints solving level consists of two rules (CFL) and (CNSIS). Consider the configuration $H'\cup\{h=(Q:\Phi,\rho,C)^{(p,q)_i}\}$.

Rule (CFL) says that the computation cell h contains non-solution, i.e., the conflicts are found by SMT layer which means under the current evaluation ρ, Φ is unsatisfiable, the upper bound q and lower bound p are both set to 0, yielding $H'\cup\{h=(Q:\Phi,\rho,C)^{(0,0)_i}\}$. The propositions hold.

Rule (CNSIS) handles the constraint solving problem for the cell h, it generates subcells which yields $H'\cup\bigcup_{i=1}^{k}\{h_i=(Q:\Phi,\rho_i,C)^{(p_i,q_i)_i}\}$, h disappears and will then be reconstructed by combination rules. Notice here, for each subcell h_i, if it is an inner cell, i.e., all the points in the space specified by ρ^i are solutions, the lower bound will be increased and upper bound may be decreased; if it is an outer cell which contains non-solution and solution, the upper bound may be decreased. That is to say, for each subcell h_i, $\rho_i(x_j) \subseteq \rho(x_j)$, $q_i \le q$ and $p_i \ge p$.

Stochastic SMT level contains three rules ($\exists$-COM), (Я-COM) and (LFT). Consider the configuration $H' \cup \{h = (Q : \Phi, \rho, C)^{(p,q)_i}\}$, according to the SMT level and constraint solving level, h will be split or replaced by subcells in certain steps. There are three cases which could reconstruct h:

- h is reconstructed by ($\exists$-COM) and (Я-COM) for the subcells h_i which are generated by (CNSIS), i.e., $H' \cup \bigcup_{i=1}^{k}\{h_i = (Q : \Phi, \rho_i, C)^{(p_i,q_i)_i}\}$. As we have explained, h_i is either an inner cell which increases the lower probability p_i and may decrease the upper probability q_i, or it's an outer cell which may decrease the upper probability q_i. In this sense, the accumulated lower bound can be guaranteed not less than p and the upper one is not greater than q, i.e., $\sum_{i=1}^{k} p_i \ge p$ and $\sum_{i=1}^{k} q_i \le q$. Rule (CNSIS) filters out the cells containing non-solution, thus the accumulation of ρ_i is smaller than ρ, i.e., $\bigcup_{i=1}^{k} \rho_i(x_j) \subseteq \rho(x_j)$, the propositions hold;

- h is reconstructed by ($\exists$-COM) and (Я-COM) for the split cells h_1 and h_2, i.e., $H' \cup \{h_1 = (Q : \Phi, \rho', C)^{(p',q')_j}, h_2 = (Q : \Phi, \rho'', C)^{(p'',q'')_j}\}$, h_1 and h_2 will be modified by the solving rules, i.e., they are split, checked by (CFL) and (CNSIS), generate subcells and are finally reconstruct by them, which is in the end reduced to the first case, so the propositions hold for h_1 and h_2 respectively, i.e., the evaluation are smaller than before, and the lower bound is non-decreasing and the upper bound is non-increasing. If h_1 and h_2 are combined to h, $p' + p'' \ge p$, $q' + q'' \le q$ and $(\rho' \cup \rho'') \subseteq \rho$, the propositions hold;

- h is obtained by (LFT) from cell h', i.e., $H' \cup \{h' = (Q : \Phi, \rho', C)^{(p',q')_i}\}$, where h' can be finally reduced to the first case, we conclude that the propositions hold for h', when it is lifted by (LFT), the propositions hold for h.

The rules for termination checking, i.e., (GE), (LE) and (INCON), only do the comparison work and will not influence the structure of computation cells. Thus the propositions hold directly. $\square$

From the proof of Lemma 6.4.1, we know that the satisfaction of a given computation cell h is always bounded by the probability estimation $[p,q]$. Initially, $Pr(\Phi)$ is bounded by $[0,1]$ (Rule (INI)), the upper bound is decreased if the non-solution parts can be excluded (Rule (CFL) and Rule (CNSIS)), and the lower bound is increased if the solution can be guaranteed in the considered cells, i.e., when inner cells are found (Rule (CNSIS)). The probability bound is estimated when rules (SPL) and (ICP) are applied. All the observations lead to the following lemma.

Lemma 6.4.2. *H is a set of computation cells and $\{(Q:\phi,\rho,C)^{(p,q)_i}\}\in H$ is anyone of them, along the solving procedure the probability of $Q:\phi$ under the evaluation ρ is guaranteed to be in the range $[p,q]$.*

Theorem 6.4.1. *Given a CSSMT formula $\Phi = Q:\phi$, a reference probability δ, and an accuracy ε, the solving procedure guarantees that the real probability $Pr(\Phi)$ is in a probability range $[p,q]$, where p and q are lower and upper estimates provided by the solving procedure.*

Proof. Initially $H=\{(Q:\phi,\rho,\emptyset)^{(0,1)_1}\}$, the solving procedure will yield $H'=\{(Q:\phi,\rho,C)^{(p,q)_i}\}$, according to Lemma 6.4.1, $p\geq 0$ and $q\leq 1$, and according to Lemma 6.4.2, $Pr(\Phi)$ is guaranteed in the range $[p,q]$. □

Remark 6.4.1. *In this part, we didn't talk about the termination of the solving procedure. Our algorithm is mainly based on interval analysis, so safe rounding has to be used for approximating the probability bounds, the termination property can't be ensured due to rounding eventually destroying progress. However, the algorithm will in general actually make progress until rounding hits in and will thus always terminate if the accuracy ε is set to reasonable values.*

Remark 6.4.2. *Lemma 6.4.1 and 6.4.2 hold for every cell in H, thus it also works for the initial cell which is generated by Rule (INI). The lemma guarantee that the lower probability p is non-decreasing and upper probability q is non-increasing, thus the interval $[p,q]$ gets smaller and smaller, the probability of satisfaction lies in the range $[p,q]$. The solving procedure terminates with $p\geq\delta$ or $q\leq\delta$, or the length of $[p,q]$ is smaller than ε. The results can be guaranteed with these three cases when terminates.*

6.5 Summary

In this section, we have introduced the solving procedure for CSSMT formulas, which has been divided into four groups and handled as an integration of DPLL framework, Interval constraint propagation and probability analysis. Computation cells are introduced as the main data structure to perform the solving, which are attached with probability estimations. Instead of computing the probability of satisfaction of Φ directly, we formulate the problem as a decision problem, i.e., a target is set and the solving procedure terminates once the target is reached or can never be reached. This makes it possible to benefit from heuristic pruning of the search space. Due to the undecidability, to get the precise probability is costly or even makes no sense in most of the application scenarios. To set up the target of interest is enough for most of the real applications.

A related work which has to be mentioned is done by Ellen et al. [EGF14], where a similar extension of SSMT is considered. They extended the semantics of SSMT to continuous domains in a similar manner and proposed a solving procedure based on statistical techniques adopted from statistical AI planning algorithms and thus only being able to offer stochastic guarantees. Such algorithm is faster; however, only a result with confidence interval will be provided, for the safety critical systems, it cannot one hundred percent guarantee the results. Our solving procedure, though in most cases slower than the statistical version, can provide sound results which guarantee that the real probability of satisfaction is indeed in the interval specified by the probability bounds.

The proposed solving procedure extends the standard DPLL procedure and interval analysis, however, the solving procedure is relatively straightforward and lowly efficient. In fact, by its similarity to DPLL algorithms, the base algorithm lends itself to lots of algorithmic enhancements and data structures that were introduced in SAT/SMT solver. Besides, the property of probability distribution may lead to an early termination, which saves a lot of computation cost. From this point of view, the base solving procedure can be improved in a lot of aspects, which are the topics of the next chapter.

Chapter 7

Algorithmic Enhancements

The solving procedure for CSSMT formula integrates the classical DPLL framework for SMT solving, interval constraint propagation for safe bounding and probability analysis, thus yields four groups, i.e., SMT layer, constraint solving layer, stochastic layer and a group of criteria for termination. In the previous chapter, we have formalized the solving procedure by a number of rules, which directly reflect the four groups. One should notice that all the rules are not individual and applied one by one. Instead, they cooperate with each other to find the safe bounds for the probability of satisfaction. The solving procedure works on a set of computation cells, the cells are manipulated, split or combined according to specific rules, i.e., when they satisfy the premises given by the rules.

The given solving procedure can be regarded as a basic algorithm without any optimizations. However, due to the similarity to SMT solving, a lot of algorithmic enhancements can be borrowed and customized for an efficient CSSMT solving. Moreover, some heuristic ideas to prune the search space can be explored by finding the relationship among the probability measures for computation cells, reference probability and the probability required. Thus in this chapter, we are focusing on algorithm improvement with a couple of examples for illustration. Such enhancements cannot reduce the complexity of the solving procedure due to the undecidability of CSSMT, however, for most cases, it can get the solution much faster.

In this chapter, we mainly talk about the enhancement from three aspects:

- For SMT layer, we try to improve the search algorithm for finding unit clauses, which is generally costly if we traverse every atomic formula in clauses;
- For constraint solving layer, we will try to analyze the reasons of conflicts instead of ignoring them. We will see later that this idea can help narrow the search space;
- For stochastic layer, we propose some heuristic ideas to prune the search space by analyzing the required probability.

For the further explanation, we rewrite the solving procedure in an algorithmic manner, as shown in Algorithm 4. The pseudo code gives us an intuitive understanding of how the individual rules have been organized for the whole solving procedure. We will modify the basic algorithm later after we introduce all the heuristic ideas.

Algorithm 4 Basic CSSMT Solving Procedure

```
Input: A CSSMT formula $\Phi = \mathcal{Q} : \varphi$, a reference probability $\delta$ and precision $\varepsilon$.
Output: GE: if the probability estimation can be guaranteed to be greater equal than $\delta$;
        LE: if the probability estimation can be guaranteed to be less equal than $\delta$;
        INCON: the result can not be guaranteed w.r.t. $\varepsilon$.
1: $C_0 \leftarrow (Q : \phi, \rho, C)^{(0,1)_1}$; $H \leftarrow \{C_0\}$;                    ▷ (INI)
2: while True do
3:     Take $C_i$ from $H$;
4:     $unit\_propagation(\Phi, C_i)$;                                     ▷ (UP)
5:     $interval\_constraint\_propagation(\Phi, C_i)$;                      ▷ (ICP)
6:     if Conflict then
7:         $(Q : \phi, \rho_i, C)^{(p_i,q_i)_k} \leftarrow (Q : \phi, \rho_i, C)^{(0,0)_1}$;          ▷ (CFL)
8:     else if Hull_Consistent then
9:         $H \leftarrow H \cup \{constraints\_solving(\Phi, C_i, \varepsilon)\}$;          ▷ (CNSIS)
10:    else if Inconclusive then
11:        $(C_{i1}, C_{i2}) \leftarrow split(\Phi, C_i, \varepsilon)$;                   ▷ (SPL)
12:        $H \leftarrow H \cup \{C_{i1}, C_{i2}\}$;
13:    while $|H| > 1$ do
14:        if There exist $C_1$ and $C_2$ from $H$ which can be combined regarding to $x_i$ then
15:            if $x_i$ is bound by $\exists$ then
16:                $C_{12} \leftarrow maximum\_combine(C_1, C_2)$;              ▷ ($\exists$-COM)
17:            if $x_i$ is bound by $\text{Я}$ then
18:                $C_{12} \leftarrow probabilistic\_combine(C_1, C_2)$;          ▷ ($\text{Я}$-COM)
19:            $H \leftarrow H \cup \{C_{12}\}$;
20:        else if $C$ from $H$ without combinable cells then
21:            $H \leftarrow H \cup \{lift(C)\}$;                            ▷ (LFT)
22:    end while
23:    $H = \{(Q : \phi, \rho, C)^{(pl,pu)_1}\}$;
24:    if $pu \leq \delta$ then return LE;                                ▷ (LE)
25:    else if $pl \geq \delta$ then return GE;                             ▷ (GE)
26:    else if $|pu - pl| < \varepsilon$ then return INCON;                 ▷ (INCON)
27: end while
```

7.1 Lazy Clause Evaluation

Recall that in the SMT layer of the solving procedure, unit propagation (UP) plays an important role to reason over the Boolean structure of the matrix of Φ. Unit propagation is a mechanism widely used in SAT/SMT decision procedures. It tries to check all the atomic formulas in a clause. If all the atoms except one are violated under the current evaluation ρ, this atom will be treated as a formula which must be satisfied; otherwise, the whole clause will be violated and no solution will be found. In the solving procedure for CSSMT, the rule (UP) is used to find the unit clauses, which will be propagated and added to the constraints set, all the constraints in this set have to be satisfied in the current phase. However, this step can be very inefficient and costly.

Example 7.1.1. *Let us consider the following CSSMT formula:*

$$\begin{aligned}\Phi \;=\; & \exists x \in [0,5] \reflectbox{R} y \in \mathcal{U}[8,10] \exists z \in [-10,10] : \\ & (x \geq 3 \vee y < 9 \vee z^2 > 15y \vee x > 2y+z) \wedge (y \geq \sin(z)+10 \vee x > z^2+y)\end{aligned}$$

There are two clauses in the CSSMT formula, the first clause consists of 4 atomic formulas and the second with 2 atomic formulas. The solving procedure firstly performs the unit propagation and tries to find if there are unit clauses. The basic algorithm traverses each atomic formula, we get the following result after the first round of unit propagation:

$$\begin{aligned}\Phi \;=\; & \exists x \in [0,5] \reflectbox{R} y \in \mathcal{U}[8,10] \exists z \in [-10,10] : \\ & (\underbrace{x \geq 3}_{INCON} \vee \underbrace{y < 9}_{INCON} \vee \underbrace{z^2 > 15y}_{UNSAT} \vee \underbrace{x > 2y+z}_{UNSAT}) \wedge (\underbrace{y \geq \sin(z)+10}_{INCON} \vee \underbrace{x > z^2+y}_{UNSAT})\end{aligned}$$

We find a unit clause in the second clause, which is $y \geq \sin(z)+10$ and should be added to the constraints set showing it must be satisfied under the current searching space. Interval constraint propagation is then applied w.r.t. the constraint, which says that $y \geq \sin(z)+10 = [-1,1]+10 = [9,11]$, so the domain of y can be modified, i.e., $[8,10] \cap [9,11] = [9,10]$. Since the domain of y has been changed, we should reapply unit propagation to find further unit clauses, the second round of unit propagation thus yields:

$$(\underbrace{x \geq 3}_{INCON} \vee \underbrace{y < 9}_{UNSAT} \vee \underbrace{z^2 > 15y}_{UNSAT} \vee \underbrace{x > 2y+z}_{UNSAT}) \wedge (\underbrace{y \geq \sin(z)+10}_{INCON} \vee \underbrace{x > z^2+y}_{UNSAT})$$

Another unit clause $x \geq 3$ is then found by the procedure.

Remark 7.1.1. *We may notice that the unit propagation mechanism is very costly due to the traversing of the atomic formulas in clauses, whenever the domain of variables is changed we* *need to perform the unit propagation to find potential unit clauses. In the running example, UP visited* 12 *atomic formulas in total, each round with* 6 *visits.*

Some heuristic ideas may be used to avoid such high cost, here we list some:

- *UP only visits the atom formulas in which the variable modification is involved, in our running example, only the domain of y is changed. Thus we can just visit the atomic formulas involved. By applying this, we can reduce the number of visits to* 11*;*
- *Another idea is to skip the atomic formulas which are already violated, since the ICP will only narrow the domain of variables, if there is no solution in a certain area, the narrowed area also contains no solution. By applying this strategy, the number of the visits can be further reduced to* 8.

However, a more intelligent way can be achieved if we understand the target of unit propagation better. Unit clause is a clause in which all the atom formulas are violated regarding the current evaluation except one, if we take any two atom formulas (if there are) from a unit clause, we will find the following facts hold for these two atomic formulas:

- either both of them are violated by the current evaluation;
- or only one of them is violated by the current evaluation.

We can choose two inconclusive atomic formulas as “watched atoms”, this is so-called lazy clause evaluation schema of zChaff [MMZ$^+$01]. In order to fulfill our requirement for CSSMT solving, we slightly modify the schema in the following way:

> Two atomic formulas in one clause -which are inconclusive under the current evaluation ρ- are chosen as “watched atoms”, without loss of generality, we name the two formulas l_1 and l_2. In the following procedure, we do not need to check every atom formula in the clause every time, instead, we only need to perform the checking step for these two formulas when the variables in either l_1 or l_2 are modified by interval constraint propagation, i.e., tighter bounds are obtained for some variables involved. Here we take l_1 as an example, if we get a tighter bound for some variables in l_i, which means the solving procedure narrows the search space, i.e., yields a new ρ', we then check the truth value of l_1. If l_1 gets unsatisfied under ρ', we need to choose another atom in this clause which is inconclusive w.r.t. ρ' as a watched atom. If the procedure fails to find such atom, l_2 is concluded to be a unit clause.

This schema saves a lot of work since most of the time we only need to visit at most two atomic formulas in each clause except that one of the watched atoms is violated, in average, the visiting time is quite low compared with other ideas.

Example 7.1.2. *Back to the running example, take the first clause as an example, which contains four atom formulas. We choose $x \geq 3$ and $y < 9$ as watched atoms and then start the solving procedure by UP. For the first round, UP only checks the two watched atoms and finds both of them are inconclusive; it doesn't need to check the rest formulas in this clause since for any cases this clause cannot be a unit clause. After ICP, we notice that the domain of y gets tighter and the watched atom $y < 9$ is involved. UP then checks this formula and finds it is unsatisfiable under the new evaluation, according to the lazy clause evaluation schema, we need to find another atom formula which is inconclusive; however, this cannot be achieved. We thus conclude that the other watched clause $x \geq 3$ is a unit clause.*

7.2 Conflict Driven Clause Learning

The basic algorithm doesn't handle the constraints in a special way. When conflicts are obtained, the corresponding computation cells are marked with probability zero. This works correctly but is not efficient. In fact, constraints contain a lot of information which may indicate the relationship among variables and constraints. If they can be well analyzed and used, the solving procedure can be extremely improved, which yields an extension framework for SAT/SMT solving, i.e., Conflict Driven Clause Learning (CDCL).

Example 7.2.1. *Consider the following CSSMT formula:*

$$\begin{aligned}\Phi &= \text{Я}x \in \mathcal{U}[0,10] \text{Я}y \in \mathcal{U}[-1,1] \exists z \in [-3,1] : \\ &\quad (x \geq 7 \vee y > 0.3 \vee z > x + y) \wedge (z > 0.1 \vee z \geq y^2 + 1) \wedge (z \leq 3y)\end{aligned}$$

Initially, all the atom formulas are inconclusive under the initial configuration, thus we split the domain of x into two parts from the middle point, which yields two computation cells, i.e., $H_1 = (\Phi, ([0,5],[-1,1],[-3,1],C_1)^{(p_1,q_1)_{k1}}$ and $H_2 = (\Phi, ((5,10],[-1,1],[-3,1],C_2)^{(p_2,q_2)_{k2}}$. We first take H_1 into consideration. $x \geq 7$ is violated under current evaluation, however, no further conclusion can be made in the moment. We further split the domain of y, which yields another two cells, which are $H_{11} = (\Phi, ([0,5],[-1,0],[-3,1],C_{11})^{(p_{11},q_{11})_{k11}}$ and $H_{12} = (\Phi, ([0,5],(0,1],[-3,1],C_{12})^{(p_{12},q_{12})_{k12}}$. For H_{11}, we find $y \geq 0.3$ is violated which makes the first clause unit clause, the third clause $z \leq 3y$ leads to the case that $z \leq 3y = 3 * [-1,0] = [-3,0]$, thus z must be less or equal than 0. According to the interval constraint propagation, the domain of z can be narrowed to $[-3,0]$, which makes the formula $z > 0.1$ in second clause unsatisfiable. We find another unit clause $z \geq y^2 + 1$, however, this requires that z is at least greater equal than 1. This is impossible under current evaluation, thus a conflict is obtained.

According to the rule to handle the conflict, we can conclude the cell H_{11} as a cell containing no solution, which yields $H_{11} = (\Phi, ([0,5], [-1,0], [-3,1], C_{11})^{(0,0)_1}$. Then two cells are left, i.e., H_{12} and H_2, and they will be explored by the solving procedure. However, if we step a little bit deeper into the reason for the conflict, we could easily find that $y \in [-1,0]$ is the reason which makes the atomic formulas $z \leq 3y$ and $z \geq y^2 + 1$ unsatisfiable at the same time. The search space where $y \in [-1,0]$ could thus be ignored without further exploration, i.e., for the cell H_2, the domain of y can be immediately reduced, which yields $H_2' = (\Phi, ((5,10], (0,1], [-3,1], C_2)^{(p_2,q_2)_{k2}}$, as shown in Picture 7.1.

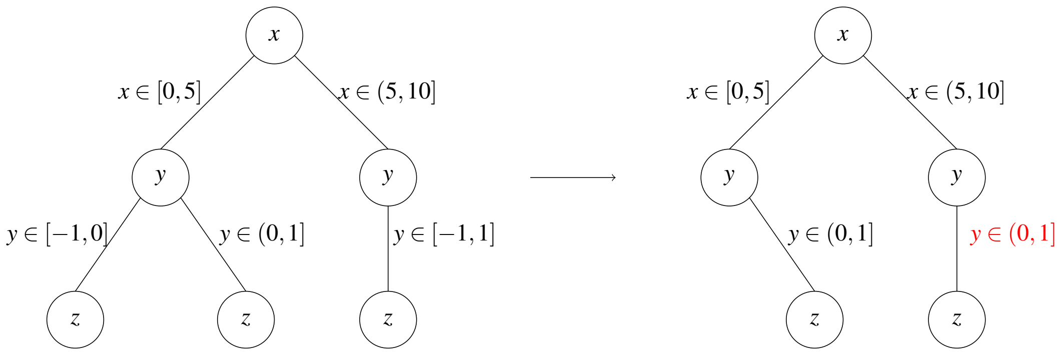


Fig. 7.1 Conflict analysis can help narrow the searching space.

From the example, we can see that the conflict analysis plays an important role in time-saving. In fact, Conflict Driven Clause Learning (CDCL) is not a new concept but is an essential mechanism in classical SAT/SMT solvers which leads to non-chronological backjumping and speeds up the decision procedure to find the solution as soon as possible ([SS97, MSS99, BJS97]).

In order to integrate the CDCL framework to our solving procedure, we will slightly modify the case when conflicts are obtained. In the basic algorithm, the probability bounds are set to 0 for cells with conflicts, which could reduce the upper bound of the probability estimation for the original formula Φ. Besides, the reasons for conflicts could be analyzed by means of an *implication graph IG* known from propositional SAT solving [ZMMM01], so that we can prune the search space accordingly. In the following, we will set up the procedure to build the implication graph, which is customized for CSSMT solving.

Formally, the implication graph is a directed acyclic graph $IG \subset A \times A$, where A contains atomic formulas and IG collects all relations from atomic formulas to atomic formula, i.e., from reasons to result. Recall the Chapter 2, where we mentioned the basic DPLL algorithm with four steps: deciding, unit propagation, backtracking and failing, which are extended by CDCL framework in the following manner:

1. *Deciding* selects a variable and assigns a truth value to it; this is same with basic DPLL framework;
2. *Unit Propagation* applies Boolean constraint propagation to find unit clauses; this is also same with DPLL;
3. *Implication Graph* builds the implication graph, which makes it different from DPLL;
4. *Backjumping*: if there is any conflict then analyze the conflict and non-chronologically backjump to the appropriate decision level. Different from backtracking in DPLL, non-chronologically back jumping doesn't require to only go back to the last decision level;
5. *SAT or UNSAT*: continue the steps until all variable values are assigned, which shows the satisfiability of the formula, or until no previous decision level can be backjumped to, which shows the unsatisfiability of the formula.

According to the mentioned procedure, the implication graph is established when:

- *deciding* step is applied, a new node is made which indicates that a variable is chosen and assigned with a truth value, at this moment, a new decision level is established;
- *unit propagation* is applied, transitions are made by the propagation, which means the new decision level may also influence the truth values of other variables.

CSSMT is different from classical SAT solving due to the continuity of the searching space. Thus the solving procedure doesn't decide truth values for variables; instead, it tries to manipulate the interval domains of variables and check the satisfiability of atomic formulas. Moreover, since CSSMT is built on non-linear arithmetic which is handled by interval constraint propagation, the Boolean structure of CSSMT formulas is not only affected by unit propagation but also by interval constraint propagation. The standard CDCL framework should then be modified:

- the decision step splits the domain of a chosen variable and explores one of its branches;
- the unit propagation applies the current evaluation to all the atom formulas to find unit clauses;
- the interval domain of variables can be also influenced by interval constraint propagation which may also be a reason of conflict.

According to all the considerations, we can now formalize the ideas to construct an implication graph during CSSMT solving. We should notice here that the construction of IG is not an individual procedure, but along the basic CSSMT algorithm.

- According to the Rule SPL (Splitting), the domain of a specific variable x_i will be split into two parts, i.e., x_{i1} and x_{i2}, one is stored to the set H, and the rest is to be explored. This is a decision step for CSSMT solving. Thus a node can be created for this IG;
- According to the Rule UP (Unit Propagation), the unit clause is chosen when all other atoms are violated under ρ, so the bounds for the variables involved in such atoms are the reasons for the unit clause. Consider a clause $l_1 \wedge l_2 \wedge \cdots l_n$: if all the l_i are violated except l_k, the bounds $x_i \sim b_i$ which makes l_i violated are the reasons for l_k, implication arcs can be established in this moment;
- According to the Rule ICP (Interval Constraint Propagation), the evaluation for some variables are narrowed by manipulating the interval analysis. The involved bounds are the reasons for the narrowing, i.e., if $(x_i \sim b_i, \cdots, x_j \sim b_j) \overset{C}{\rightsquigarrow} (x_k \sim b_k)$, $(x_i \sim b_i, \cdots, x_j \sim b_j)$ is the reason for $(x_k \sim b_k)$, implication arcs can thus be established.

After constructing the implication graph IG, the reasons which lead to conflicts can be derived by traversing the IG, where the parent nodes which result in the conflicts can be regarded as the reasons. In modern SAT/SMT solver, the negated form of the reasons can be added to the original formula so that the same conflicts can be avoided by the further procedure, which can also help us prune "bad" search space regions which contain no solution, as shown in the previous example.

Example 7.2.2. *Reconsider the Example 7.2.1 we have considered:*

$$\begin{aligned}\Phi &= \text{Я}x \in \mathcal{U}[0,10]\text{Я}y \in \mathcal{U}[-1,1]\exists z \in [-3,1]: \\ &\quad (x \geq 7 \vee y > 0.3 \vee z > x+y) \wedge (z > 0.1 \vee z \geq y^2+1) \wedge (z \leq 3y)\end{aligned}$$

We would like to show how to construct the implication graph. The solving procedure starts by splitting x into two parts. This is a decision step. Thus a node with the new domain of x can be made. By doing so, we can not find unit clauses (except $z \leq 3y$, which is naturally a unit clause since the clause contains only one atomic formula) or perform the interval constraint propagation. The further procedure chooses another variable - for example y - to split which yields another decision step. Thus a new node should be added. By performing unit propagation, $z > x+y$ is a unit clause and the implication arc can be constructed.

As next step, the interval constraint propagation is applied, and $z \leq 3y$ requires that $z \leq 0$, which makes $z > 0.1$ unsatisfied and thus unit propagation tells that $z \geq y^2 + 1$ has to be satisfied. Interval constraint propagation then concludes that z is at least 1, which results in a conflict.

The process of how the IG is constructed is depicted in the Figure 7.2, from which we can identify that decision $y \in [-1, 0]$ is the reason for conflicts.

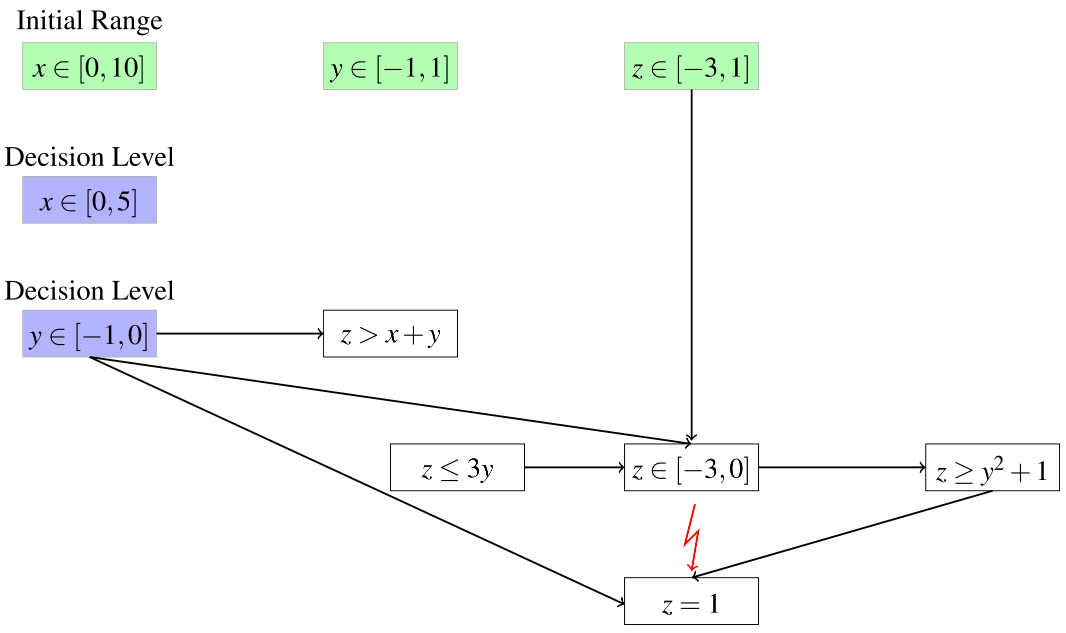


Fig. 7.2 Implication Graph.

Until now we have explored some framework from classical SAT/SMT solving which can be integrated into our basic algorithm for CSSMT solving. Lazy clause evaluation is applied in the SMT layer when we perform the unit propagation to reason about the Boolean structure of the matrix, it can extremely reduce the number of visits of the atom formulas and the unit clauses can be found faster. Lazy clause evaluation doesn't essentially improve the ideas for CSSMT solving; it is just a smart idea from which we can benefit. However, conflict clause driven learning is an inspiring mechanism which provides a methodology for SAT/SMT or even CSSMT solving. It tries to find the reasons if the conflicts are obtained during processing, the reasons will be reused in the solving procedure so that the same conflicts can be avoided and the search space can be reduced, which provides more chance to rapidly find the solution or the area which contains the solution.

Our focus is on the problem with continuously distributed variables; however, we did not look deeply into how the solving procedure can be influenced by the probability measure so far.

In fact, as handling probability is the main job of CSSMT solving, some heuristic ideas can be used to pursue early termination, which we will discuss in the next section.

7.3 Early Termination by Required (Threshold) Probability Analysis

Take a look at the basic algorithm for CSSMT solving. It is based on the structure of so-called computation cells, which is stored in a set H. If we focus on one computation cell, we will see that it can experience actions like:

- its variable domains can be modified, i.e., by performing the interval constraint propagation, the domain of variables could be narrowed, and if the variables involved are bound by the randomized quantifier, the probability estimation may be modified with a tighter version;
- the domain of a variable is split, i.e., by the splitting rule, the probability could also be recalculated depending on the type of quantifiers for the variable;
- the computation cell can be judged as “a bad cell” containing no solution, or “an inconclusive cell” where constraint solving will be applied to find solutions, in such situations, the probability estimation will be done for the subcells generated by constraint solving;
- the computation cell can be merged with another cell, in which the probability estimation is also correspondingly treated according to the type of quantifiers;
- the probability of a computation cell can be compared with a reference probability δ so that the result will be concluded.

From the above description, it is seen that the probability evaluation happens very late, i.e., after all computation cells are handled by splitting, merging, constraint solving, etc., which is not very efficient since a lot of work will be done during the solving procedure without handling the probability. In such way, the reference probability δ only plays a role at a very late stage. In fact, the probability analysis can be handled much earlier, if we have some information about the current stage, i.e., we roughly know the range of required (or threshold) probability and is clear whether it can be reached or not. This can help us judge if we need further computation for other cells.

Example 7.3.1. *Consider an intermediate step for the CSSMT solving procedure for a formula with three variables x, y and z, where x and y are uniformly distributed with range* $[-1,1]$ *and* $[0,10]$ *correspondingly, while z is a random variable with standard normal distribution. At a certain moment there are three computation cells generated, which are:* $H_1 = (\Phi, [[-1,0],[0,10],[0,0.5]], \{x \leq -0.5\})^{(0,0.1)_1}$, $H_2 = (\Phi, [(0,1],[0,3],[0,0.5]], C)^{(0,0.06)_2}$ *and* $H_3 = (\Phi, [(0,1],(3,10],[0,0.5]], C')^{(0,0.14)_2}$. *For each cell, a probability estimation is given, and the reference probability for this problem is* $\delta = 0.16$. *Let us start with the cell* H_1. *We notice that the constraint set contains one constraint* $x \leq -0.5$, *for CSSMT solving, Rule (ICP) should be applied, which yields* $H_1' = (\Phi, [[-1,-0.5],[0,10],[0,0.5]], \{x \leq -0.5\})^{(0,0.05)_1}$. *Notice here the probability estimation is also changed since the modified variable x is bound by the randomized quantifier. Without performing the next solving steps, let us first check what will happen if we merge the three computation cells, as depicted in the Figure 7.3. The result yields a maximum probability estimate* 0.15, *which can not reach the reference probability* δ *which is* 0.16, *so the solving procedure can be safely stopped at this moment with answering LE.*

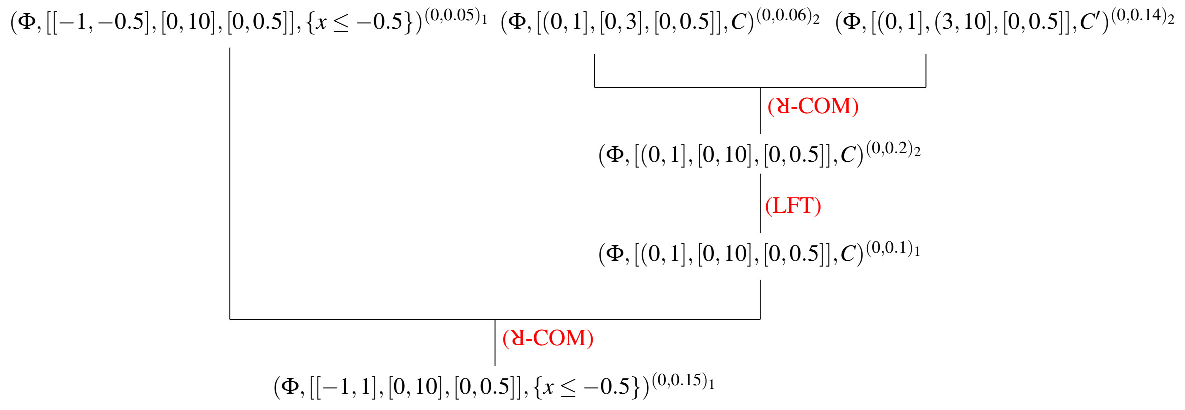

$(\Phi, [[-1,-0.5],[0,10],[0,0.5]], \{x \leq -0.5\})^{(0,0.05)_1}$
$(\Phi, [(0,1],[0,3],[0,0.5]], C)^{(0,0.06)_2}$
$(\Phi, [(0,1],(3,10],[0,0.5]], C')^{(0,0.14)_2}$
(ꓤ-COM)
$(\Phi, [(0,1],[0,10],[0,0.5]], C)^{(0,0.2)_2}$
(LFT)
$(\Phi, [(0,1],[0,10],[0,0.5]], C)^{(0,0.1)_1}$
(ꓤ-COM)
$(\Phi, [[-1,1],[0,10],[0,0.5]], \{x \leq -0.5\})^{(0,0.15)_1}$

Fig. 7.3 Propagate probability by using combination rules.

From the example, we see that the solving procedure can be terminated when we have the confidence to conclude the results without performing the further rules. This example also shows that when we handle the variables bound by randomized quantifiers, if the probability of one branch is reduced due to some reasons like ICP and so on, we can stop and check whether the accumulated probability for the rest branches is enough to still reach the reference probability. E.g., in our running example, the residual probability is maximum 0.1, together with the maximum probability 0.05 for H_1', it is not enough to reach the reference probability 0.16. The similar ideas can be used for handling the existential quantifier, or even the conflicts, which will be discussed later.

Now a question should be asked, is it necessary to apply the combination rules immediately once the probability estimation of a computation cell is changed? Since it is very costly to perform the combination rules, it is not efficient to perform probability check as what we did for Example 7.3.1. In order to avoid applying combination rules, we will introduce another concept of *required probability*. Required probability is the minimum probability needed in order to reach the reference probability, and it can be understood as an extra information for computation cells. Required probability can easily be computed and dynamically maintained; moreover, it is enough to perform the checking by finding the relationship between probability estimation and required probability, thus the tedious utilization of combination rules can be avoided.

Definition 7.3.1. *Given a CSSMT formula Φ, and reference probability δ, the required probability p_{req} for a computation cell is the minimum probability needed for reaching δ which satisfies:*

- *if the computation cell is with the initial configuration of the given CSSMT formula, i.e., generated by Rule (INI), $p_{req} = \delta$;*
- *if p_{req} is the required probability regarding a variable x_i which is bound by Ⅎ, the required probability regarding to variable x_{i+1} is $p_{req'} = p_{req}/\max(Pr(\rho(x_i)))$;*
- *if p_{req} is the required probability regarding to a variable x_i which is bound by Ⅎ, when x_i is split and thus two cells are obtained, one of the cells as probability estimation $(p,q)_i$, the required probability for the other is $p_{req2} = p_{req} - q$. This can be extended to multiple branching, i.e., $p_{req'} = p_{req} - \sum_k q_k$;*

The definition also provides us with the way to compute the required probability. It can simply traverse the tree-like structure induced by the solving procedure and the required probability for each node can be computed. There are no corresponding rules for computing required probability for the variables bound by existential quantifier; it instead is a trivial work by inheritance since $\exists$ requires the maximum probability.

Example 7.3.2. *Back to the Example 7.3.1, Figure 7.4 shows how required probability is changed after applying ICP rule. In the figure, the probability estimation is shown in red color, and the required probability for each branch and node is marked by a number with blue color.*

Before applying interval constraint propagation, the branch H_1 has a probability estimation $[0, 0.1]$. *Assume the reference probability is* 0.16, *according to the definition of required probability, we can thus compute the probability for the other branch which is* 0.06. *Then we propagate the required probability to the sub-nodes, which is shown in the left figure.*

When ICP has been performed, the probability estimation for the first branch is changed to $[0, 0.05]$, *and we modify the required probability correspondingly. We find that the required* *probability for H_3 regarding z becomes* 0.22*, which is the minimum probability which has to be reached. However, the probability estimation is* $[0,0.2]$*, and even if the maximum probability is taken, the required probability cannot be reached. Therefore, we conclude that the probability of the CSSMT formula cannot reach* 0.16*, and a LE answer can be safely reported.*

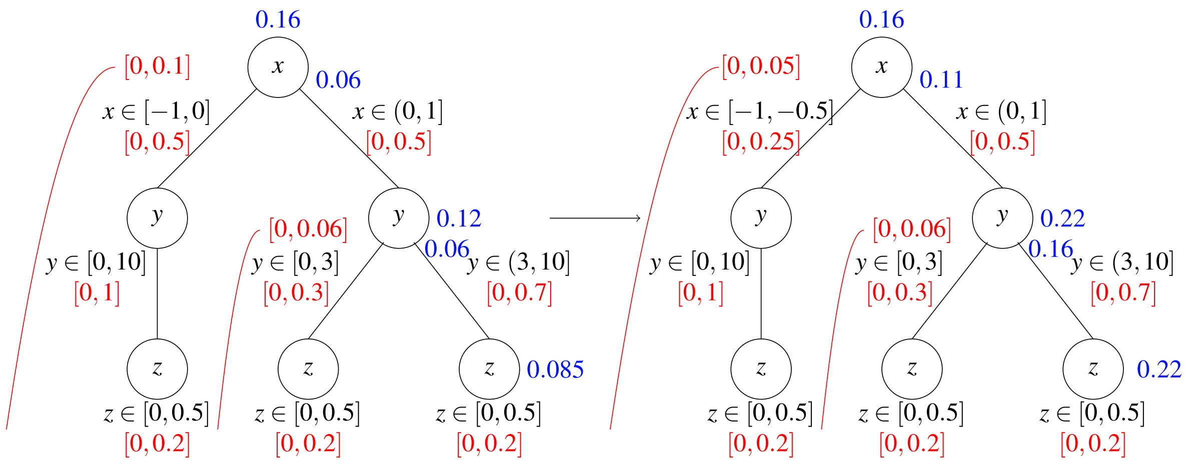


Fig. 7.4 The computation of required probability.

For the running example, we show that the analysis of required probability can yield an earlier termination of CSSMT solving procedure. A similar idea can be applied not only for the cells regarding the variables with randomized quantifiers but can also be used for the cells w.r.t. the variables with existential quantifiers, or even can work when conflicts are obtained. The intuition is as follows:

- Considering the cells which are related to the variable bound by $\exists$, when we merge the cells according to the combination rule ($\exists$-COM), it chooses the maximum probability estimation. If the probability estimation for one cell is large enough, we can ignore the rest;
- If the solving procedure tries to accumulate the probability estimation for cells regarding a random variable, the combination rule ($\reflectbox{R}$-COM) performs the addition operating. When the probability of one cell is large enough, others can be ignored.
- If conflicts are obtained, the probability estimation for the conflict cell is 0, to check the required probability for other cells may help to decide if it can still finally reach the reference probability.

Now let's formalize all the ideas. Before going to the next step, we would like to introduce another concept *threshold probability* P_{thd}, which is dual concept regarding the required probability, which defines the probability threshold for computation cells that once it is reached guarantees that the reference probability is guaranteed to be exceeded. The definition is quite similar to the one for required probability.

Definition 7.3.2. *Given a CSSMT formula Φ, and reference probability δ, the threshold probability p_{thd} for a computation cell is the probability once it is reached δ will be exceeded, which satisfies:*

- *if the computation cell is with the initial configuration of the given CSSMT formula, i.e., generated by Rule (INI), $p_{thd} = \delta$;*
- *if p_{thd} is the threshold probability regarding to a variable x_i which is bound by ᴚ, the threshold probability regarding to variable x_{i+1} is $p_{thd'} = p_{req} / \min(Pr(\rho(x_i)))$;*
- *if p_{thd} is the threshold probability regarding to a variable x_i which is bound by ᴚ, when x_i is split and thus two cells are obtained, one of the cells with probability estimation $(p, q)_i$, the threshold probability for another is $p_{thd2} = p_{thd} - p$. This can be extended to multiple branching, i.e., $p_{thd'} = p_{thd} - \sum_k p_k$;*

Notice that the definition is almost the same as the one for required probability, except that we always compute the threshold probability from the lower side of the probability estimate. Now let us move to the ideas for early termination analysis.

Randomized Quantifier

Consider a computation cell related to variable x_i which is bound by ᴚ, the probability estimation can be refined by Rule (ICP), (CONSIS) or (CONFL), the solving procedure proceeds required (threshold) probability analysis at this moment:

- if P_{req} for a cell related to x_i is greater than its upper probability bound, the procedure can report *LE* without checking the rest of cells;
- if P_{thd} for a cell related to x_i is less than its lower probability bound, the procedure can report *GE* without checking the rest of cells.

Existential Quantifier

Consider a computation cell related to variable x_i which is bound by $\exists$. The probability estimation can be refined by Rule (ICP), (CONSIS) or (CONFL), and the solving procedure proceeds exploiting required (threshold) probability analysis at this moment: if P_{thd} for a cell related to x_i is less than its lower probability bound, the procedure can report *GE* without checking the residual cells.

Conflict

Consider a computation cell related to variable x_i which is bound by ꓤ. Where a conflict is found, the solving procedure proceeds exploiting required (threshold) probability analysis at this moment. If P_{req} for a cell related to x_i is greater than its upper probability bound, the procedure can report LE without checking the residual cell.

7.4 Summary

In this chapter, we focused on the algorithmic enhancement for the CSSMT solving procedure, which is achieved by modifying the three solving layers:

- For SMT layer, the unit propagation is optimized with watched atoms so that the number of atomic formulas traversed can be reduced;
- For the constraint solving layer, the mechanism of conflict-driven clause learning is modified to fit our requirement for CSSMT solving. CDCL maintains an implication graph and try to analyze the reasons for conflicts which may reduce the search space for the original problem and thus make the algorithm more efficient;
- For the stochastic layer, we introduced the ideas of required probability and threshold probability, which indicate that in a particular moment, we can compute a threshold to decide whether the reference probability can still be reached or exceeded, which yields early termination without checking other cells and performing further rules.

The algorithm 5 is an improved version compared to the basic solving procedure proposed in the beginning of this chapter, which: 1) adds watched atoms for unit propagation; 2) establishes the implication graph during unit propagation and interval constraint propagation; 3) analyzes the reasons for conflicts when conflicts are found; 4) performs the termination analysis by computing required probability or threshold probability during the rules (ICP), (CFL) and (CNSIS).

Algorithm 5 Improved CSSMT Solving Procedure

Input: A CSSMT formula $\Phi = \mathcal{Q} : \varphi$, a reference probability δ and precision ε.
Output: GE: if the probability estimation can be guaranteed to be greater equal than δ;
LE: if the probability estimation can be guaranteed to be less equal than δ;
INCON: the result can not be guaranteed w.r.t. ε.

```
1: $C_0 \leftarrow (Q : \phi, \rho, C)^{(0,1)_1}$; $H \leftarrow \{C_0\}$;      ▷ (INI)
2: while True do
3:     Take $C_i$ from $H$;
4:     $unit\_propagation\_with\_watched\_atoms(\Phi, C_i, IG)$;      ▷ (UP)
5:     $interval\_constraint\_propagation\_with\_termination\_anlyzer(\Phi, C_i, IG)$;      ▷ (ICP)
6:     if Conflict then
7:         $(Q : \phi, \rho_i, C)^{(p_i,q_i)_k} \leftarrow (Q : \phi, \rho_i, C)^{(0,0)_1}$;      ▷ (CFL)
8:         $H \leftarrow CDCL\_with\_termination\_anlyzer(H, IG)$;
9:     else if Hull_Consistent then
10:        $H \leftarrow H \cup \{constriants\_solving\_with\_termination\_anlyzer(\Phi, C_i, \varepsilon)\}$;
11:                                                                 ▷ (CNSIS)
12:    else if Inconclusive then
13:        $(C_{i1}, C_{i2}) \leftarrow split(\Phi, C_i, \varepsilon)$;      ▷ (SPL)
14:        $H \leftarrow H \cup \{C_{i1}, C_{i2}\}$;
15:    while $|H| > 1$ do
16:        if There exist $C_1$ and $C_2$ from $H$ which can be combined regarding to $x_i$ then
17:            if $x_i$ is bound by $\exists$ then
18:                $C_{12} \leftarrow maximum\_combine(C_1, C_2)$;      ▷ ($\exists$-COM)
19:            if $x_i$ is bound by Ꭹ then
20:                $C_{12} \leftarrow probabilistic\_combine(C_1, C_2)$;      ▷ (Ꭹ-COM)
21:            $H \leftarrow H \cup \{C_{12}\}$;
22:        else if $C$ from $H$ without combinable cells then
23:            $H \leftarrow H \cup \{lift(C)\}$;      ▷ (LFT)
24:    end while
25:    $H = \{(Q : \phi, \rho, C)^{(pl,pu)_1}\}$;
26:    if $pu \leq \delta$ then return LE;      ▷ (LE)
27:    else if $pl \geq \delta$ then return GE;      ▷ (GE)
28:    else if $|pu - pl| < \varepsilon$ then return INCON;      ▷ (INCON)
29: end while
```

Chapter 8

CSiSAT: A Satisfiability Solver for CSSMT

In the previous chapters, we introduced the idea to model systems with stochastic behavior by continuous stochastic satisfiability modulo theories (CSSMT), which is an extension of standard SMT by introducing randomized quantifiers and has the probability as semantics. We further introduced a sound solving procedure to decide whether the maximum probability of satisfaction regarding a CSSMT formula $\Phi = \mathcal{Q} : \varphi$ exceeds a reference probability δ and algorithmic enhancements for the solving procedure are developed to pursue an efficient algorithm.

Based on the proposed solving procedure, we implemented a prototype solver CSiSAT, which is an SMT solver, especially for CSSMT solving. In this chapter, we will focus on the introduction of the solver. We will start by introducing the family of iSAT-based tools which are supported by the AVACS project [AVACS]. In the following, the structure of our tool CSiSAT is presented as well as the functions of CSiSAT. As a summary, we compare our solver with other similar tools which are able to handle stochastic systems. This chapter is thus regarded as an introductory material, and the demonstration examples and case studies will be supplied in the next chapter, which show the ability of CSSMT to deal with real applications.

8.1 Introduction to the Family of iSAT-based Tools

The HySAT/iSAT tool family is motivated and maintained by the AVACS project, which provided a series of constraint solvers and bounded model checkers from different aspects, as shown in Figure 8.1:

- HySAT is not only an SMT solver but also a bounded model checker [HySAT] which integrates interval constraint propagation (ICP) into conflict-driven clause learning framework (CDCL) [FHT+07] so that large Boolean structure with non-linear arithmetic can be solved efficiently. HySAT can be regarded as the origin for all other related SMT solvers in this family;

- iSAT/iSAT3 are successors to HySAT [iSAT] which optimize the main core of ICP and CDCL based SMT solving and provide a more efficient solving. Compared with HySAT, it also supports SMT solving and bounded model checking, the results are more stable and can be found faster. However, it doesn't support constraint solving with optimization, which is covered by HySAT. Based on iSAT/iSAT3, some extended tools are implemented for different targets, like iSAT-Craig which applies Craig interpolation to SMT and can prove or disprove invariant properties of transition systems, iSAT-LP which is implemented to handle the SMT especially with linear constraints, iSAT-ODE focuses on the SMT formula with ordinary differential equation (ODE) so that a larger number of dynamic systems can be captured;

- SiSAT is a stochastic SMT solver, which introduces the discrete randomized quantifiers so that it models the variables with discrete distributions [SiSAT]. However, only variables with discrete domain (except for the implicit innermost existential quantified variables) are supported by this solver. Therefore it is restricted in its ability to model the systems with continuous behavior;

- CSiSAT is a stochastic SMT solver who also supports continuously distributed variables, and it is implemented based on the algorithm mentioned in the previous chapters. The core of constraint solving for CSiSAT is partially based on the tool Realpaver [GB06], which is based on interval analysis thus safe bounds can be obtained for the results.

8.2 The Structure of CSiSAT

The prototype implementation of CSiSAT is implemented by using C++ programming language with approximately 8000 lines of source code[1]. The source code is implemented and tested on Ubuntu 15.10 with a 64-bit PC and is available at https://vhome.offis.de/~ygao/.

The overview of CSiSAT is depicted in Figure 8.2. The core algorithm is based on the previous chapter. It currently accepts a plain-text file as input which defines the CSSMT

[1] CSiSAT is still a prototype implementation, so more functions and heuristic methods are still under testing and implementation. Since it is still an ongoing project, the source code and the tool are not fully documented. If you have any problems regarding the usage, bug report and so on, drop me an email. Any comments are welcome.

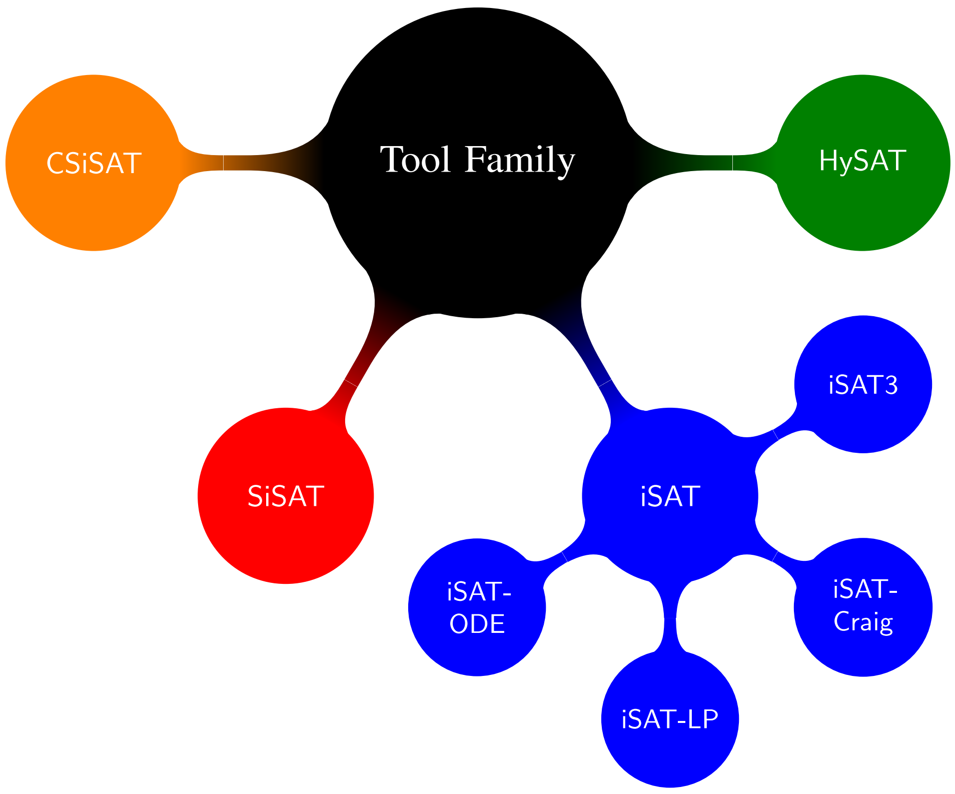


Fig. 8.1 HySAT/iSAT family.

formula by using a straightforward syntax. The input file is parsed by a parser which is done by Flex/Bison and the formula is stored by using an Abstract Syntax Graph (ASG), then the tool follows the way which we presented before: SMT layer does the reasoning on the Boolean structure, i.e., Rules (UP), (ICP) and (SPL). The constraint solving is performed in two ways, either by RealPaver [GB06] or CSiSAT itself. The output is the probability estimation for the input CSSMT formula. The current version will return the maximum and minimum probability which safely bound the probability of satisfaction regarding the input CSSMT formula.

Notice that we have two boxes in the overview structure which are drawn with the dashed line, which means these two parts are not stable for the prototype implementation. Restart procedure will try to restart the solving procedure with smaller precision if the output bounds are rough. Currently, you need reset the precision in the input file and rerun CSiSAT again. Conflict Analyzer is inspired by CDCL framework from the standard SMT community, which helps us to find the reason for the conflicts and speed up the solving procedure. In our current implementation, the cells with conflicts are marked with probability 0 and removed from the following computation, but in fact, the reasons can be analyzed and the search space can be narrowed in this sense. More details regarding optimization have been discussed in the previous chapter.

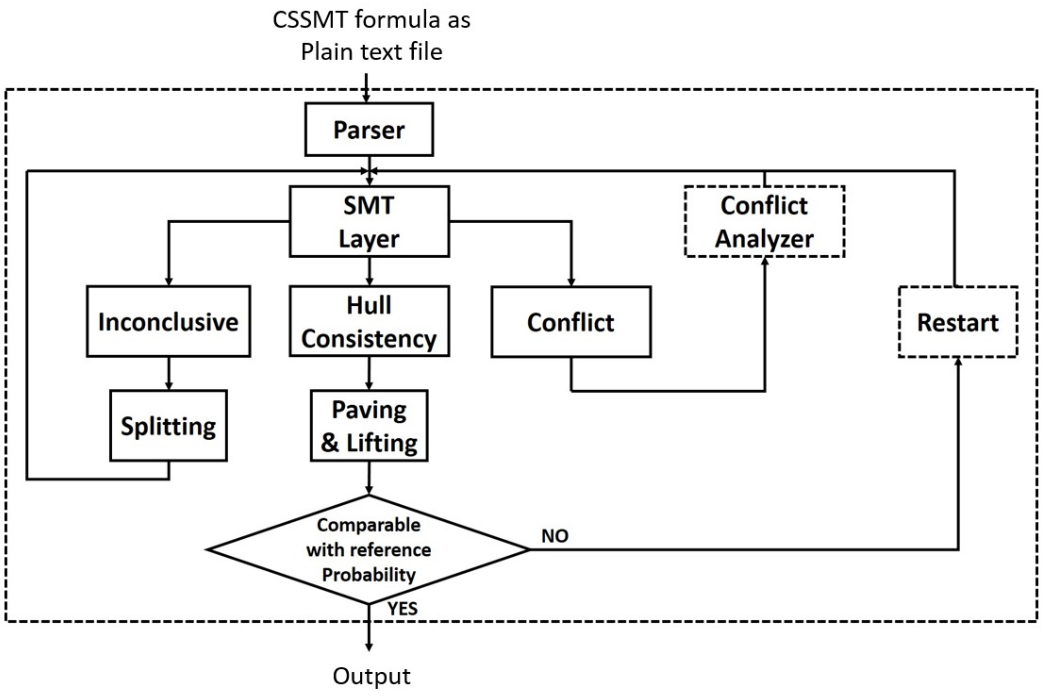


Fig. 8.2 Overview of CSiSAT

8.3 Functions of CSiSAT

CSiSAT is a pure CSSMT solver which can:

- model random variables in constraints and the distributions can be specified in the input file;
- solve the constraints in a certain way specified by users, i.e., it supports either the invoking of RealPaver or the constraint solving techniques implemented in the solver itself;
- provide the safe probability bounds of CSSMT formula under given precision, which helps decide if the target reference probability can be reached.

CSiSAT can be used to model the probabilistic constraints, analyze the bounded model checking problem, compute the reachability probability, etc, which will be seen in the next chapter.

8.4 Compared with related tools

There are a lot of tools and solvers which aim at handling systems with stochastic behaviour. In this section, we choose some similar tools which can model and analyze stochastic hybrid systems and we would like to compare our tool with them. These include SiSAT [TF08], ProHVer [HHHK13], ProbReach [SZ15] and PRISM [KNP11].

- SiSAT is a satisfiability solver which can symbolically reason about reachability problems of probabilistic hybrid automata (PHA). SiSAT denotes the probability of satisfaction of Φ under the optimal resolution of the non-random quantifiers. In this solver, all quantifiers (except for implicit innermost existential quantification of all otherwise unbound variables) are confined to range over finite domains. As this implies that the carriers of probability distributions have to be finite, a large number of phenomena cannot be expressed within the current SSMT framework, such as continuous noise or measurement error in hybrid systems;
- ProHVer is a prototype verifier for stochastic hybrid systems, which is capable of computing the unbounded reachability probability for probabilistic hybrid automata. ProHVer maintains a finite-state over-approximation of the original probabilistic hybrid automata, which can then be used to compute a probability that is an upper bound for the maximal reachability probability for the property given in the probabilistic hybrid automaton. However, exploiting ProHVer for computing the abstractions, it is limited to

linear dynamics and can handle even that only via an over-approximation by piecewise constant differential inclusions. Moreover, due to the manual selection of the abstraction, the user should at least know the related dynamics. For some problems, a number of iterations should be performed to get acceptable results, which makes the tool not very easy to use;

- ProbReach is a software for calculating bounded probabilistic reachability in hybrid systems with uncertainty in initial parameters, which is based on safe gridding and an approximation to a δ-complete decision procedure. ProbReach only deals with the hybrid systems with random initial parameters, however, stochastic dynamics is not covered.

- PRISM is a probabilistic model checker for formal modeling and analysis of systems that exhibit random or probabilistic behavior. PRISM is confined to finite-state systems like DTMC and CTMC, which makes continuous variables difficult to be tackled. In PRISM, continuous states are not supported, modeling continuous quantities should be done by using some other tricks, i.e., model the continuous variables as rewards. The definitions of DTMC/CTMC provide that the discrete transitions are issued with discrete distributions, which makes continuous uncertainties impossible to be captured unless by abstraction.

The comparison is concluded in the following table:

Name	Continuous Variables	Continuous Distributions	Probabilistic Transitions	ODE
SiSAT	$\exists$ bound variable can be continuous	×	✓	×
ProHVer	✓	✓	✓	linear dynamics
ProbReach	✓	✓	×	✓
PRISM	continuous rewards	×	✓	×
CSiSAT	✓	✓	✓	×

Table 8.1 Comparing with other tools (×: unsupported; ✓: supported.)

Chapter 9

Applications of CSiSAT

The last chapter introduced the infrastructure of the CSiSAT solver, which is a satisfiability solver for CSSMT formulae. CSiSAT takes CSSMT encodings as input and returns safe bounds for the maximum probability of satisfaction regarding the given formula, which makes it possible to model uncertainty in hybrid systems. Compared with other solvers or verifier for hybrid systems, a weakness of CSiSAT is that ordinary differential equations cannot directly be captured by CSSMT formula. But despite the lack of support for ODE, CSSMT can nevertheless be applied in many fields and real applications.

In this chapter, we are going to apply CSiSAT to model some scenarios from real applications. The case studies come from different fields and show the capabilities of CSiSAT in handling the stochastic reachability problem, stochastic constraint solving and so on. If a problem can be encoded with arithmetic constraints containing random variables, CSiSAT now is a choice.

We firstly provide an overview to get readers familiar with the input and output style of CSiSAT; a demonstration example will be given for this purpose. A couple of examples are given in the sequel, and we will provide the background of each scenario and explain how this problem can be encoded into CSSMT formulas and solved by CSiSAT.

All the experiments reported in this chapter were conducted on a 64bit-Linux (Ubuntu 16.04.1 LTS) machine with an Intel(R) Core(TM) i5-3230 processor at 2.40GHz$\times$4 equipped with 4 GB of RAM. The scripts of the models as well as the source code of CSiSAT are available at https://vhome.offis.de/~ygao/. The source code was developed under GNU GPL license and thus can be downloaded and distributed.

9.1 Demonstration Example

As the first case study, we will show a demonstration example explaining the usage of CSiSAT. As we mentioned in the previous chapter, the solver takes a CSSMT formula as input. The grammar of encoding a CSSMT formula in the input file is simple and straightforward.

Example 9.1.1. *Consider a CSSMT formula*

$$\begin{aligned}\Phi \ = \ & \exists x \in [-10,10] \,\text{Я} y \in \mathcal{U}[5,20] \,\text{Я} z \in \mathcal{U}[-10,10] : \\ & (x > 3 \vee y < 1) \wedge (z > x^2 + 2 \wedge y \leq 20) \wedge (x^2 > 49 \vee y < 0.7 * x) \wedge (x < 6 \vee y \geq z)\end{aligned}$$

where both y and z are uniformly distributed in their ranges. As input file, we can rewrite the formula as shown in Figure 9.1. The parameter precision defines the minimum length for splitting which can be modified to achieve more accurate probability estimation. Each existential quantifier is represented by "E" and a randomized quantifier is represented by "R". "U" stands for uniform distribution, other distributions can be also referred to by specific letters.

```
Branch{
precision = 0.1;
}
E x in [-10,10] R y in U[5,25] R z in U[-10,10]:
x>3 || y<1 && z>x^2+2 || y<=20 && x^2>49 || y<0.7*x && x<6 || y>=z
```

Fig. 9.1 Example input

The input file can be directly parsed into the solver and handled by the solving procedure. It will report the safe bounds which constitute lower and upper estimates of the probability of satisfaction regarding Φ under the given precision. If the results are not acceptable due to the precision, i.e., too rough to estimate the real probability, one can adjust the precision and rerun the solving procedure.

Example 9.1.2. *By running CSiSAT, we can get the lower and upper bound of the probability of satisfaction w.r.t. the formula Φ, as shown in Figure 9.2. The results say that the lower approximation is* 0.718014 *and the upper one is* 0.719719, *which means that the real probability is guaranteed to be in this range. In fact, it can easily be computed as with* $23/32 \approx 0.71875\cdots$. *The time consumed is reported as* 0.1132915s *in this case. Apart from this, some debug information will also be given.*

If we increase the precision to 0.01, *we will get a tighter estimated ranging from* 0.718658 *to* 0.718872. *However, this comes at the price of generating more boxes, the number of which increases from* 835 *boxes to* 8704, *and correspondingly consumes more time.*

```
$ ./CSiSAT test1.txt
..........................................
...........Debug Information.............
..........................................
0.718014, 0.719719
0.1132915 s
```

Fig. 9.2 Execution results

CSiSAT also supports another mode which uses RealPaver as the underlying constraint solver. This mode is activated by adding "-rp RealPaver_Path" as command parameters.

9.2 Path Planning

Like the first scenario, we would like to consider a stochastic verification task for path planning on a segment of a highway with three cars. The layout is depicted in Figure 9.3. On the highway, there is a low-speed car (blue car) and a high-speed car (red car). The ego car is now forced to make a choice between:

1. keeping in the lane, and
2. changing to the other lane.

Both options have an associated risk of collision. We want to use the framework of CSSMT to help decide which choice the car should take.

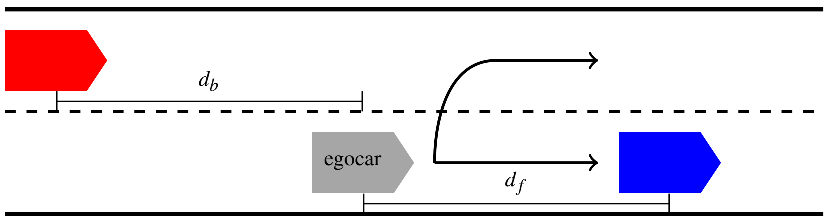


Fig. 9.3 A segment of highway with three cars.

In order to make this case study more practical, instead of fixing a specific speed for the cars, we assume that the speed of each car is influenced by the operation of the driver and the situation of the road, thus uniformly distributed in a range $[v-\delta, v+\delta]$. Similarly, the acceleration of cars is not fixed, but also obeys a uniform distribution over range $[a-\varepsilon, a+\varepsilon]$.

9.2.1 Formalization of The Problem

Determine the variables and quantifiers. According to the previous description, we will model the problem with the following variables:

- the speed of each car, i.e., v_r for the red car, v_b for the blue car and v_g for the ego car, all the variables will be bound by randomized quantifiers with uniform distribution;
- the distance between cars, i.e., d_f the distance between ego car and the car in front, d_b the distance between ego car and the car in the back. Both variables are deterministic which are given in advance, so we model the variables as single point interval;
- the acceleration, i.e., a_r for the red car, v_g for the ego car, which are both bound by randomized quantifiers with uniform distribution; we assume the car in front maintains its speed, so the acceleration for it is 0;
- the choice of maneuver, i.e., $d \in 0,1$ which is a discrete variable with two possible values, 0 means no lane change is issued, whereas 1 means the ego car will change to another road.

Determine the criteria for lane change. We will investigate the probability of collision for the actions of the ego car. The car either keeps the lane or changes to the other lane, where both actions have an associated risk of collision. Since the speed and acceleration for cars are randomly distributed, we compute the probability that no collision will occur for each case and always take action with maximum probability of avoiding a collision. The criteria for observing a collision can be formulated by a relation which involves speed, acceleration and distance, i.e., $p_1(v_b, v_g, a_g, d_f)$ for the choice of keeping the lane, and $p_2(v_r, v_g, a_b, d_b)$ for the choice of changing the lane. A straightforward criterion can be obtained by the following no-collision relation:

$$p_1 : \frac{v_g^2 - v_b^2}{2a_g} \leq d_f + v_b \cdot \frac{v_g - v_b}{a_g} \tag{9.1}$$

p_2 is formulated by using the parameters regarding the following car and ego car. Here we can add more criteria so that the decision can be precisely made, i.e., in the paper [EGF14] they also consider the relative velocity when collision occurs.

Determine the CSSMT formula. As has been stated above, the corresponding CSSMT formula can be written as follows:

$$\begin{aligned}\Phi \quad = \quad & \exists d \in \{0,1\} \text{Я} v_r \in \mathcal{U}[vr-\delta_r, vr+\delta_r] \text{Я} v_g \in \mathcal{U}[vg-\delta_g, vg+\delta_g] \\ & \text{Я} a_r \in \mathcal{U}[ar-\varepsilon_r, ar+\varepsilon_r] \text{Я} a_g \in \mathcal{U}[ag-\varepsilon_g, ag+\varepsilon_g] \exists d_f \in [df, df] \exists d_b \in [db, db] : \\ & (d \wedge p_1(v_b, v_g, a_g, d_f)) \vee (\neg d \wedge p_2(v_r, v_g, a_b, d_b)) \qquad (9.2)\end{aligned}$$

9.2.2 Experimental Results

The semantic of the Formula 9.2 gives the maximum possible probability of avoiding collision and the assignment for the variable d gives the decision which leads to this safe strategy. Figure 9.4 shows the probability of no collision regarding the speed of ego car. The error bars are obtained by solving the CSSMT formula through CSiSAT and provide safe bounds for the real probability. The results tell us that the preferred choice depends on the current speed of the ego car. If it runs relatively fast, it is better to change the lane, otherwise keeping in the current lane is safer. The crossing point of the curves gives us the speed where we should change the decision.

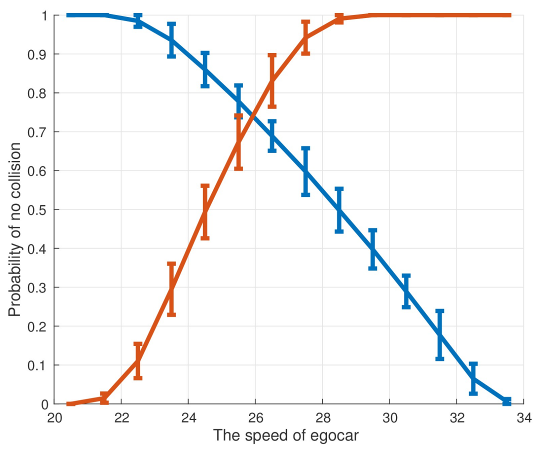


Fig. 9.4 The probability of no collision.(“red curve" for changing lane, “blue curve" for keeping in lane)

9.3 Temperature Regulation Problem

In this section, we consider the problem of regulating the temperature of a room during some time horizon $[0,N]$ by a thermostat that can switch a heater on or off. This example is a variant version taken from [APLS08], where the probabilistic reachability problem for controlled discrete time stochastic hybrid systems (DTSHS) was investigated by using dynamic programming (DP). In this work, it will be investigated by using CSSMT. This example has also been mentioned in Chapter 5, where we introduced the procedure to encode the behavior of a stochastic hybrid system into a CSSMT formula. Here we would like to review this example again and have a look at how CSSMT can help to perform the reachability analysis.

As before, we consider a room in which there is a thermostat with two modes $\mathcal{Q} = \{ON, OFF\}$, and the thermostat issues switching commands $\mathcal{U} = \{0, 1\}$ to the heater: "0" means no switching command is issued and "1" effects a change of heater state. The average room temperature changes according to the laws $m_{OFF}(x) = x - \frac{a}{C}(x - x_a)\Delta t + \mathcal{N}(m, \sigma^2)$ and $m_{ON}(x) = m_{OFF}(x) + \frac{r}{C}\Delta t$, where a is the average heat loss rate, C is the average thermal capacity, x_a is the ambient temperature, r is the rate of heat gain, and Δt is the discretization time interval. In order to capture the disturbance, we add a noise term $\mathcal{N}(m, \sigma^2)$, which denotes the probability measure over $(\mathbb{R}, \mathcal{B}(\mathbb{R}))$ associated with a Gaussian density function with mean m and variance σ^2. The formal model is depicted in Figure 9.5.

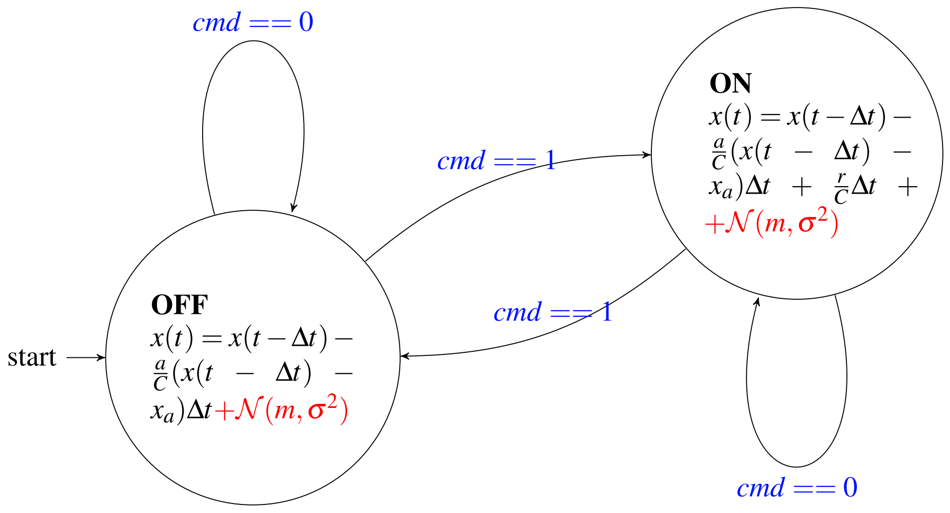

$cmd == 0$
ON
$x(t) = x(t - \Delta t) - \frac{a}{C}(x(t - \Delta t) - x_a)\Delta t + \frac{r}{C}\Delta t + +\mathcal{N}(m, \sigma^2)$
$cmd == 1$
OFF
$x(t) = x(t - \Delta t) - \frac{a}{C}(x(t - \Delta t) - x_a)\Delta t + \mathcal{N}(m, \sigma^2)$
start
$cmd == 1$
$cmd == 0$

Fig. 9.5 Regulating the temperature of a room by a thermostat

The regulation problem we consider here is the same with [APLS08]: *determine a control strategy that maximizes the probability that the average room temperature x is driven close to a given temperature with an admissible tolerance.*

9.3.1 SSMT Formalization

To formalize the problem, we translate the initial condition, the transition relation, and the desired sets into a CSSMT formula $\mathcal{Q} : \Phi$. The encoding follows the rules introduced in Chapter 5.

Initial condition.

The initial room temperature starts from any value in the set $[T_l^0, T_u^0]$ and the heater is initially switched to OFF. This condition can be formulated as follow:

$$\mathcal{I} : (T_l^0 \leq x_0 \leq T_u^0) \wedge (q_0 = OFF) \tag{9.3}$$

Since the system is considered to be well behaved for any suitable initial value, we assume that the latter is uniformly distributed in the range $[T_l^0, T_u^0]$, i.e., x_0 is bound by $\reflectbox{R} x_0 \in \mathcal{U}(T_l^0, T_u^0)$ and q_0 is a discrete variable bound by quantifier $\exists q_0 \in \{OFF, ON\}$.

Transition relation.

During each transition step the system will do two operations: 1) Choose a control action u_i by performing switching command $u_i = 0$ or $u_i = 1$. 2) Update the temperature x_i according to the thermostat state.

$$\begin{aligned} \mathcal{T}_i : \quad & ((u_i = 0 \wedge q_i = q_{i-1}) \vee (u_i = 1 \wedge \wedge q_i = \neg q_{i-1})) \\ \wedge \quad & ((q_i = ON \wedge x_i = x_{i-1} - \tfrac{a}{c}(x_{i-1} - x_a)\Delta t + \omega_{i-1} + \tfrac{r}{c}\Delta t) \vee \\ & (q_i = OFF \wedge x_i = x_{i-1} - \tfrac{a}{c}(x_{i-1} - x_a)\Delta t + \omega_{i-1}) \end{aligned} \tag{9.4}$$

u_i here is a control action which can be either 1 (switching to opposite heater state) or 0 (no switching), i.e., $\exists u_i \in \{0,1\}$. x_{i-1} and x_i are the previous and the updated room temperature. These take values from a suitable temperature range: $\exists x_{i-1} \in [T_l^{i-1}, T_u^{i-1}]$ and $\exists x_i \in [T_l^i, T_u^i]$. ω_i is a continuous random variable with Gaussian distribution: $\reflectbox{R} \omega_i \in \mathcal{N}(m, \sigma^2)$.

Target sets.

Target sets can be understood as the sets which we want to keep the room temperature inside. For each transition step, we attach a constraint to the SMT formula:

$$\mathcal{S}_i : C_l^i \leq x_i \leq C_u^i \tag{9.5}$$

which indicates that the room temperature x_i is expected to be in the set $[C_l^i, C_u^i]$.

Altogether the dynamics of the system can be formalized as an SSMT formula

$$\begin{aligned}&\text{Я}x_0 \in \mathcal{U}(T_l^0, T_u^0) \exists q_0 \in \{OFF, ON\} \exists u_i \in \{0,1\}\\ &\vdots\\ &\text{Я}\omega_i \in \mathcal{N}(m, \sigma^2) \exists x_{i-1} \in [T_l^{i-1}, T_u^{i-1}] \exists x_i \in [T_l^i, T_u^i] \cdots : \mathcal{I} \wedge \bigwedge_{i=1}^{N} (\mathcal{T}_i \wedge \mathcal{S}_i),\end{aligned} \tag{9.6}$$

where N is the number of computation steps analyzed.

9.3.2 Experimental Results

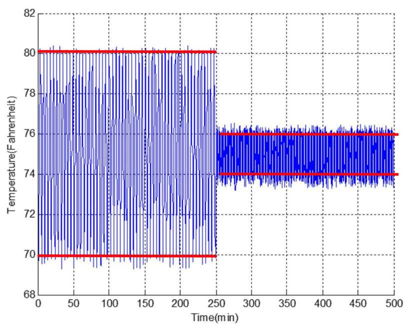
The parameters we used are the same as in [APLS08]: $x_a = 10.5°\text{F}$, $a/C = 0.1\ \text{min}^{-1}$, $r/C = 10°/\text{min}$. All the $\mathcal{N}_i$ are independent and identically distributed with mean $m = 0$ and variance $\sigma = 0.33°\text{F}$. Initially, the room temperature is in $[70, 80]°\text{F}$. In the following we consider the system dynamics during the time interval $[0, 500]$, and we specify that the room temperature should be kept within $[70, 80]°\text{F}$ when time $t \leq 250\text{min}$ and after that it is driven close to 75°F, i.e., into the range $[74, 76]°\text{F}$.

The results are depicted in Fig. 9.6, which can be understood as follows:

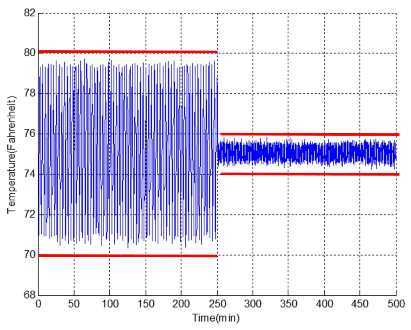
- The sub-figure (a) shows the behavior with classical threshold-driven bang-bang control, i.e., the control keeps current state unchanged until the threshold has been reached. If the measured temperature exceeds the upper bound, the controller switches to the OFF mode and vice versa. The initial temperature is uniformly chosen from the range $[70, 80]$;

- The sub-figure (b) is obtained by solving the corresponding CSSMT formula, i.e., formula 9.6. As the semantics of CSSMT formula is the maximum satisfaction of the formula, in this problem, each transition step involves optimally choosing between two possible actions, namely to stay in current mode (command "0") or to switch to the opposite mode (command "1"). By solving the CSSMT formula for each transition step, we not only obtain the maximum probability to stay in the required temperature range but can also take the action which leads to such probability as our optimal control strategy.

By inspection of the diagrams, it becomes obvious that we achieved a significantly better behavior by extracting control actions through CSSMT solving. The temperatures stay much closer to what we specified as targets, i.e., the temperature tends to stay with high probability within the given range $[70, 80]°\text{F}$ and $[74, 76]°\text{F}$, respectively.



(a) bang-bang control



(b) induced by solving CSSMT

Fig. 9.6 Experiment results.

Table 9.1 reports the results returned by CSiSAT. It shows the solving precision, time consumption, the number of boxes generated and probability approximations. Higher precision of the results is naturally achieved by investing additional solving time.

From this study, we learn that CSiSAT can be used to solve probabilistic reachability problems by encoding the behavior of a stochastic hybrid model into a CSSMT formula and then solving that by the prototype implementation. The solver provides us with both the maximum probability of reachability and the variable assignments in each step which lead to maximum probability. The latter provide controller synthesis.

Precision	Number of boxes	Time(s)	Probability bounds
10^0	182	0.01359	[0.937054,1]
10^{-1}	2288	0.0165795	[0.96041,0.975993]
10^{-2}	24564	1.42093	[0.9681,0.970115]
10^{-3}	524422	34.7482	[0.969122,0.969245]
10^{-4}	4982016	332.765	[0.969176,0.969191]

Table 9.1 Results for temperature regulation problem. (500 time steps, for each step 2 states variables, 2 control variables and 1 variable representing the temperature.)

9.4 Task Deployment and Scheduling

As a third case study, we will investigate a probabilistic scheduling problem derived from hard real-time systems. Hard real-time systems have been defined as those containing tasks that have deadlines that cannot be missed, which is guaranteed by different scheduling methods. One of them is deadline-monotonic scheduling [ABRW90], which is a static priority based algorithm for periodic processes in which the priority of each process is related to its period.

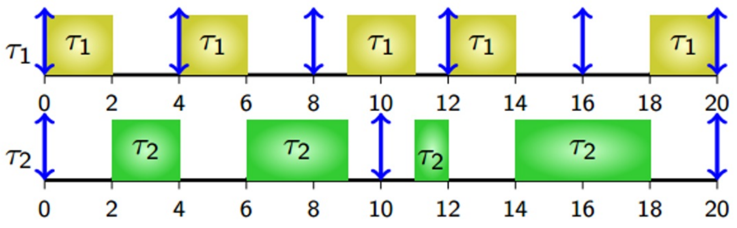


Fig. 9.7 Task scheduling.

In our scenario, we consider a system on which tasks are intended to run with two types of memory:

1. normal memory (MEM) of sufficiently large size,
2. a very small amount of scratch-pad memory (SPMEM), which has the advantage that access to it is very quick.

Tasks are called periodically (with period P_i) and do not depend on the results of each other. They have deadlines (D_i) which may not exceed their periods ($D_i \leq P_i$). The worst case execution time (C_i) of a task consists of some base time (BC_i) and the time that it needs to access its variables. These variables can be assigned to either MEM or SPMEM. Since the worst-case execution time also depends on the variable access time, it varies with the different deployment, i.e., how to distribute the variables over memory. The system is equipped with a fixed priority scheduler with preemption. In deadline-monotonic scheduling, tasks are assigned priorities according to their deadlines, the task with the shortest deadline being assigned the highest priority. Whenever a task with higher priority than the task that is just running becomes activated, this task preempts the running task, which is suspended until the processor is free again.

Let us consider a system with two tasks and 4 cells of SPMEM. The parameters are shown in Table 9.2.

We want to check whether we can find a task deployment so that no deadline is violated regarding a schedule. In standard setting, the worst-case execution time (WCET) is given as a fixed number. The schedulability test is fulfilled by examining the response time for each task, where the response time is computed by the following formula:

$$R_i = C_i + \sum_{k \in hp_i} \lceil \frac{R_i}{P_k} \rceil \cdot C_k \tag{9.7}$$

Here P_i denotes the period of task i, C_i the execution time of task i, hp_i the set of tasks with higher priority than task i; then the smallest R_i satisfying above fixed-point equation yields the

Table 9.2 System parameters

Hardware	
MEM size	4 cells
MEM access time	4 cycles
SPMEM access time	1 cycle
Task 1	
number of accesses to T1 variable 1	10
number of accesses to T1 variable 2	3
number of accesses to T1 variable 3	2
number of accesses to T1 variable 4	6
period P_1	1200
basic execution time BC_1	$\mathcal{N}(140,100)$
deadline D_1	1000
Task 2	
number of accesses to T2 variable 1	5
number of accesses to T2 variable 2	40
number of accesses to T2 variable 3	1
period P_2	200
basic execution time BC_2	$\mathcal{N}(10,10)$
deadline D_2	100

response time of task i. This recursive formula is normally solved by searching a fixed point beginning with $R_{i,0} = C_i$ as initial candidate for R_i and then inserting $R_{i,0}$ into the recursive formula as R_i. If the resulting $R_{i,1}$ equals $R_{i,0}$ the fixed point is already reached, otherwise the calculation is continued with $R_{i,1}$ as a candidate. The search can be stopped when either a fixed point is reached or the deadline exceeded. If the response time for each task is below its deadline D_i, the tasks can be scheduled without deadline violation.

However, in practice, the WCET is generally obtained by analysis and approximation, and a reliable fixed number is hard to obtain. We thus would like to consider the execution time as a random variable with a certain distribution, e.g., a clipped normal distribution, which yields the problem we want to handle:

> Given tasks with randomized basic execution time, we want to find a task deployment which can guarantee that with sufficiently high probability no deadline becomes violated.

We try to use CSiSAT to find the corresponding task deployment. Since there are only four SPMEM cells, only four of the seven tasks can be assigned to them. We can take seven variables each of them with two integers as its domain, as the first iteration step, we can start from the formula:

$$\exists x_{1\ldots4}, y_{1\ldots3} \in \{0,1\} \text{Я} BC_1 \in N[140,100] \text{Я} BC_2 \in N[10,10]:$$
$$T_1 + \lceil \frac{T_1}{P_2} \rceil \cdot T_2 \le D_1 \wedge 0 \le T_1 \le P_1 \wedge 0 \le T_2 \le D_2 \wedge \sum_{i=1}^{4} x_i + \sum_{j=1}^{3} y_j = 3 \qquad (9.8)$$

where

$$T_1 = BC_1 + (3*x_1+1)*10 + (3*x_2+1)*3 + (3*x_3+1)*2 + (3*x_4+1)*6$$
$$T_2 = BC_2 + (3*y_1+1)*5 + (3*y_2+1)*40 + (3*y_3+1)*3 \qquad (9.9)$$

which are basic execution time plus the access time for the variables. This formula can be easily encoded to CSiSAT input, and the constraint-solving mechanisms of CSiSAT will automatically find a fixed-point whenever the constraint system expresses a fixed-point equation system. In case multiple fixed-points exist, an appropriate one satisfying the remaining constraints (e.g., deadlines) will be chosen whenever such exists.

CSiSAT will give the probability regarding each possible deployment and returns the maximum one. From its results, we can learn that for some deployments, the probability is extremely low, which indicates that under such deployments the probability with deadline conflicts are very high. Such a deployment should not be used to distribute the tasks. The highest probability is found with range [0.984357, 0.985358], which is achieved by assigning the variable 3 of Task 1 and all variables of Task 2 to SPMEM.

9.5 Conclusion

In this chapter, we provided the readers with an intuition for the capabilities of CSiSAT by means of a couple of case studies, which vary from a path planning scenario over a temperature regulation problem to task scheduling. The idea underlying the solution of such problems is to encode the requirements of the problem or the behavior of systems into a corresponding CSSMT formula. That CSSMT formula is then fed to the CSiSAT solver for automatic discharge. The stochastic phenomena, measurement errors and disturbances can be modeled by random variables which are bound by randomized quantifiers, and CSiSAT will then optimally resolve existential quantifiers and return the maximum probability of satisfaction regarding to the formula of interest. It thus decides the maximum probability of satisfaction corresponding to reachability in Markov decision processes, to the optimal decision between different choices and so on.

Chapter 10

Summary and Future Work

10.1 Discussion

The verification for stochastic systems is a very tough task because of the complexity of system structure and behavior. Many modern systems integrate continuous physical components and embedded controllers, which forces us to consider both continuous and discrete behavior at the same time; nondeterminism requires a system to act correctly no matter what choices are taken for next steps; moreover, the systems are always influenced by noise in the environment and even themselves, such that uncertainties should not be ignored.

In this thesis, we proposed a framework to handle such systems with the resulting blend of nondeterministic and stochastic behavior. A new logic has been proposed in order to handle such stochastic hybrid systems in a uniform way. The individual contributions of this thesis are as follows:

- we proposed a logic named continuous stochastic satisfiability modulo theory (CSSMT), which introduces continuous randomized quantifiers so that continuous random variables can be encoded in SMT formulas directly;
- we formalized the syntax and semantics of CSSMT formulae, where the semantic of a CSSMT formula is quantitative and represents the maximum probability of satisfaction achievable under an optimal strategy for resolving existential quantifiers;
- we provided a solving procedure for CSSMT formula. In our setting, a reference probability and precision are given, and the solving procedure will try to decide the relationship between the actual satisfaction probability and the reference probability, as well as return a safe bound for the actual probability. The soundness and termination of the algorithm are also proven;

- we put forward different ideas for the algorithmic enhancements, which work in different solving layer and will lead to a practically efficient solving procedure;
- we implemented a prototype CSSMT solver, named CSiSAT, which can help solve problems from different fields.

10.2 Future Directions

The work presented in this thesis leads to some further research topics which deserve to be explored in the next research period:

Conditional CSSMT. Currently, the variables bound by randomized quantifiers are stochastically independent of each other. A possible extension adding to expressiveness thus is to introduce dependent variables, where the distribution of a random variable depends on the variables before it. It is very common that the systems contain such dependent variables, for example, the measurement error of a sensor is based on the value which has been measured. Such dependency can be formalized by joint probability distribution or conditional probability distributions.

CSSMT with ODE or SDE. As we have mentioned in previous chapters, ordinary differential equation (ODE) and stochastic differential equation (SDE) are not supported currently by CSSMT. This limits the applications of CSSMT as a modeling language since some dynamics of systems cannot be directly encoded by it. However, the extension of CSSMT to its ODE version is possible, i.e., we can compute the probability bounds by over- and under-approximating the reachable sets of ODE or SDE. A plethora of methods can be investigated here, like polytopic under-approximations [XSE16] for ODE, abstraction- or simulation-based over-approximation [ADI03, HM14] for ODE, etc. However, directly handling stochastic differential equation is a tough task, and gridding methods can be used to approximate the solution of SDE [SA11].

CSiSAT improvement. While the prototype implementation of CSiSAT can successfully discharge some non-trivial verification tasks, it can be improved in a lot of aspects: firstly, CSiSAT only supports pure CSSMT constraint formulae as input. In the HySAT/iSAT family of solvers, most of the solvers support both constraint-solving mode and bounded model checking mode (BMC). Likewise, CSiSAT could also be improved by supporting BMC mode. Secondly, CSiSAT can be made more efficient by integrating the conflict-driven clause learning (CDCL, [SS97, MSS99, BJS97]) framework and restart mechanisms, plus the heuristic enhancements

[GF16] discussed in this thesis could be fully transferred to the current solver implementation. Finally, another enhancement which makes sense is to support user-defined distributions, since CSiSAT currently only supports some widely used distributions such as the normal distribution, the uniform distribution and so on. A more flexible user interface should be provided that offers a language for defining relative distributions.

References

[ABRW90] Neil C Audsley, Alan Burns, MF Richardson, and AJ Wellings. *Deadline monotonic scheduling*. Citeseer, 1990.

[ACHH93] Rajeev Alur, Costas Courcoubetis, Thomas A Henzinger, and Pei-Hsin Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid systems*, pages 209–229. Springer, 1993.

[Ack54] Wilhelm Ackermann. Solvable cases of the decision problem. 1954.

[ADI03] Rajeev Alur, Thao Dang, and Franjo Ivančić. Progress on reachability analysis of hybrid systems using predicate abstraction. In *International Workshop on Hybrid Systems: Computation and Control*, pages 4–19. Springer, 2003.

[AF06] Jeremy Avigad and Harvey Friedman. Combining decision procedures for the reals. *arXiv preprint cs/0601134*, 2006.

[AG97] Eitan Altman and Vladimir Gaitsgory. Asymptotic optimization of a nonlinear hybrid system governed by a markov decision process. *SIAM Journal on Control and Optimization*, 35(6):2070–2085, 1997.

[AGFT14] Mohamed Abdelaal, Yang Gao, Martin Fränzle, and Oliver Theel. Eavs: Energy aware virtual sensing for wireless sensor networks. In *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), Ninth International Conference*, pages 1–6. IEEE, 2014.

[AH84] Gotz Alefeld and Jurgen Herzberger. *Introduction to interval computation*. Academic press, 1984.

[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.

[AP10] Behzad Akbarpour and Lawrence Charles Paulson. Metitarski: An automatic theorem prover for real-valued special functions. *Journal of Automated Reasoning*, 44(3):175–205, 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.

[ASSB00] Adnan Aziz, Kumud Sanwal, Vigyan Singhal, and Robert Brayton. Model-checking continuous-time markov chains. *ACM Transactions on Computational Logic (TOCL)*, 1(1):162–170, 2000.

[AVACS] AVACS. https://http://www.avacs.org/. [Online; accessed October 2016].

[BD02] Raik Brinkmann and Rolf Drechsler. RTL-datapath verification using integer linear programming. In *Proceedings of the 2002 Asia and South Pacific Design Automation Conference*, page 741. IEEE Computer Society, 2002.

[BG06] Frédéric Benhamou and Laurent Granvilliers. Continuous and interval constraints. *Handbook of constraint programming*, 2:571–603, 2006.

[BGLC00] Frédéric Benhamou, Frédéric Goualard, Éric Languénou, and Marc Christie. An algorithm to compute inner approximations of relations for interval constraints. In *Perspectives of System Informatics*, pages 416–423. Springer, 2000.

[BHvM09] Armin Biere, Marijn Heule, and Hans van Maaren. *Handbook of satisfiability*, volume 185. ios press, 2009.

[BJS97] Roberto J Bayardo Jr and Robert Schrag. Using CSP look-back techniques to solve real-world sat instances. In *AAAI/IAAI*, pages 203–208, 1997.

[BKF95] Hans Kleine Buning, Marek Karpinski, and Andreas Flogel. Resolution for quantified boolean formulas. *Information and computation*, 117(1):12–18, 1995.

[BLB05] Manuela L Bujorianu, John Lygeros, and Marius C Bujorianu. Bisimulation for general stochastic hybrid systems. In *International Workshop on Hybrid Systems: Computation and Control*, pages 198–214. Springer, 2005.

[Buj04] Manuela L Bujorianu. Extended stochastic hybrid systems and their reachability problem. In *International Workshop on Hybrid Systems: Computation and Control*, pages 234–249. Springer, 2004.

[Bur72] Rodney M Burstall. Some techniques for proving correctness of programs which alter data structures. *Machine intelligence*, 7(23-50):3, 1972.

[CMR81] Ole Caprani, Kaj Madsen, and Louis B Rall. Integration of interval functions. *SIAM Journal on Mathematical Analysis*, 12(3):321–341, 1981.

[Coo71] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

[Dan98] George Bernard Dantzig. *Linear programming and extensions*. Princeton university press, 1998.

[Dav93] Mark HA Davis. *Markov Models & Optimization*, volume 49. CRC Press, 1993.

[DLL62] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[DST80] Peter J Downey, Ravi Sethi, and Robert Endre Tarjan. Variations on the common subexpression problem. *Journal of the ACM (JACM)*, 27(4):758–771, 1980.

[DSW99] Andreas Dolzmann, Thomas Sturm, and Volker Weispfenning. Real quantifier elimination in practice. In *Algorithmic algebra and number theory*, pages 221–247. Springer, 1999.

[EGF14] Christian Ellen, Sebastian Gerwinn, and Martin Fränzle. Statistical model checking for stochastic hybrid systems involving nondeterminism over continuous domains. *International Journal on Software Tools for Technology Transfer*, pages 1–20, 2014.

[FHH$^+$11] Martin Fränzle, Ernst Moritz Hahn, Holger Hermanns, Nicolás Wolovick, and Lijun Zhang. Measurability and safety verification for stochastic hybrid systems. In *Proceedings of the 14th international conference on Hybrid systems: computation and control*, pages 43–52. ACM, 2011.

[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. *JSAT*, 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 *Hybrid Systems: Computation and Control*, pages 172–186. Springer, 2008.

[FOR$^+$01] Jean-Christophe Filliâtre, Sam Owre, Harald Rue, Natarajan Shankar, et al. Ics: Integrated canonizer and solver? In *International Conference on Computer Aided Verification*, pages 246–249. Springer, 2001.

[FYY13] Yuan Feng, Nengkun Yu, and Mingsheng Ying. Model checking quantum markov chains. *Journal of Computer and System Sciences*, 79(7):1181–1198, 2013.

[GAM97] Mrinal K Ghosh, Aristotle Arapostathis, and Steven I Marcus. Ergodic control of switching diffusions. *SIAM Journal on Control and Optimization*, 35(6):1952–1988, 1997.

[GB06] Laurent Granvilliers and Frédéric Benhamou. RealPaver: an interval solver using constraint satisfaction techniques. *ACM Transactions on Mathematical Software (TOMS)*, 32(1):138–156, 2006.

[GF15] Yang Gao and Martin Fränzle. A solving procedure for stochastic satisfiability modulo theories with continuous domain. In *Quantitative Evaluation of Systems*, volume LNCS 9259, pages 295–311. Springer, 2015.

[GF16] Y. Gao and M. Fränzle. CSiSAT: A satisfiability solver for SMT formulas with continuous probability distributions. In Erika Ábrahám and Sergiy Bogomolov, editors, *2016 International Workshop on Symbolic and Numerical Methods for Reachability Analysis (SNR)*, pages 1–6. IEEE, April 2016.

[GKC13] Sicun Gao, Soonho Kong, and Edmund M Clarke. dreal: An SMT solver for nonlinear theories over the reals. In *International Conference on Automated Deduction*, pages 208–214. Springer, 2013.

[GMPK14] Eric Goubault, Olivier Mullier, Sylvie Putot, and Michel Kieffer. Inner approximated reachability analysis. In *Proceedings of the 17th international conference on Hybrid systems: computation and control*, pages 163–172. ACM, 2014.

[HEFT08] Christian Herde, Andreas Eggers, Martin Fränzle, and Tino Teige. Analysis of hybrid systems using hysat. In *Systems, 2008. ICONS 08. Third International Conference on*, pages 196–201. IEEE, 2008.

[HHHK13] Ernst Moritz Hahn, Arnd Hartmanns, Holger Hermanns, and Joost-Pieter Katoen. A compositional modelling and analysis framework for stochastic hybrid systems. *Formal Methods in System Design*, 43(2):191–232, 2013.

[HJ94] Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal aspects of computing*, 6(5):512–535, 1994.

[HKNP06] Andrew Hinton, Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM: A tool for automatic verification of probabilistic systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 441–444. Springer, 2006.

[HLS00] Jianghai Hu, John Lygeros, and Shankar Sastry. Towards a theory of stochastic hybrid systems. In *International Workshop on Hybrid Systems: Computation and Control*, pages 160–173. Springer, 2000.

[HM14] Zhenqi Huang and Sayan Mitra. Proofs from simulations and modular annotations. In *Proceedings of the 17th international conference on Hybrid systems: computation and control*, pages 183–192. ACM, 2014.

[HySAT] HySAT. https://http://www.uni-oldenburg.de/en/hysat/. [Online; accessed October 2016].

[iSAT] iSAT. https://projects.avacs.org/projects/isat/. [Online; accessed October 2016].

[Jul06] A Agung Julius. Approximate abstraction of stochastic hybrid automata. In *International Workshop on Hybrid Systems: Computation and Control*, pages 318–332. Springer, 2006.

[KNP02] Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM: Probabilistic symbolic model checker. In *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 200–204. Springer, 2002.

[KNP11] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[KNSS99] Marta Kwiatkowska, Gethin Norman, Roberto Segala, and Jeremy Sproston. Automatic verification of real-time systems with discrete probability distributions. In *International AMAST Workshop on Aspects of Real-Time Systems and Concurrent and Distributed Software*, pages 75–95. Springer, 1999.

[Lei95] K Rustan M Leino. Towards reliable modular programs. 1995.

[LMP01] Michael L Littman, Stephen M Majercik, and Toniann Pitassi. Stochastic boolean satisfiability. *Journal of Automated Reasoning*, 27(3):251–296, 2001.

[LP07] Paola Lecca and Corrado Priami. Cell cycle control in eukaryotes: A BioSpi model. *Electronic Notes in Theoretical Computer Science*, 180(3):51–63, 2007.

[ML98] Stephen M Majercik and Michael L Littman. MAXPLAN: A new approach to probabilistic planning. In *AIPS*, volume 98, pages 86–93, 1998.

[ML99] Stephen M Majercik and Michael L Littman. Contingent planning under uncertainty via stochastic satisfiability. In *AAAI/IAAI*, pages 549–556, 1999.

[MM79] Ramon E Moore and RE Moore. *Methods and applications of interval analysis*, volume 2. SIAM, 1979.

[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 annual Design Automation Conference*, pages 530–535. ACM, 2001.

[MP67] John McCarthy and James Painter. Correctness of a compiler for arithmetic expressions. *Mathematical aspects of computer science*, 1, 1967.

[MSS99] João P Marques-Silva and Karem A Sakallah. Grasp: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.

[NO80] Greg Nelson and Derek C Oppen. Fast decision procedures based on congruence closure. *Journal of the ACM (JACM)*, 27(2):356–364, 1980.

[NOT06] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL($\mathcal{T}$). *Journal of the ACM (JACM)*, 53(6):937–977, 2006.

[NS03] Gethin Norman and Vitaly Shmatikov. Analysis of probabilistic contract signing. In *Formal Aspects of Security*, pages 81–96. Springer, 2003.

[Pap85] Christos H Papadimitriou. Games against nature. *Journal of Computer and System Sciences*, 31(2):288–301, 1985.

[Pug91] William Pugh. The omega test: a fast and practical integer programming algorithm for dependence analysis. In *Proceedings of the 1991 ACM/IEEE conference on Supercomputing*, pages 4–13. ACM, 1991.

[Rat06] Stefan Ratschan. Efficient solving of quantified inequality constraints over the real numbers. *ACM Transactions on Computational Logic (TOCL)*, 7(4):723–748, 2006.

[Rey79] John C Reynolds. Reasoning about arrays. *Communications of the ACM*, 22(5):290–299, 1979.

[Rey02] John C Reynolds. Separation logic: A logic for shared mutable data structures. In *Logic in Computer Science, 2002. Proceedings. 17th Annual IEEE Symposium on*, pages 55–74. IEEE, 2002.

[RVBW06] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of constraint programming*. Elsevier, 2006.

[S^+09] Teruo Sunaga et al. Theory of an interval algebra and its application to numerical analysis [reprint of res. assoc. appl. geom. mem. 2 (1958), 29–46]. *Japan Journal of Industrial and Applied Mathematics*, 26(2-3):125–143, 2009.

[SA11] Sadegh Esmaeil Zadeh Soudjani and Alessandro Abate. Adaptive gridding for abstraction and verification of stochastic hybrid systems. In *Quantitative Evaluation of Systems (QEST), 2011 Eighth International Conference on*, pages 59–68. IEEE, 2011.

[SiSAT] SiSAT. https://projects.avacs.org/projects/sisat/. [Online; accessed March 2015].

[Spr00] Jeremy Sproston. Decidable model checking of probabilistic hybrid automata. In *International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 31–45. Springer, 2000.

[SS97] João P Marques Silva and Karem A Sakallah. GRASP - a new search algorithm for satisfiability. In *Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design*, pages 220–227. IEEE Computer Society, 1997.

[Sto76] Larry J Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.

[SZ15] Fedor Shmarov and Paolo Zuliani. ProbReach: verified probabilistic delta-reachability for stochastic hybrid systems. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, pages 134–139. ACM, 2015.

[Tei12] Tino Teige. *Stochastic satisfiability modulo theories: a symbolic technique for the analysis of probabilistic hybrid systems*. PhD thesis, Universität Oldenburg, 2012.

[TF08] Tino Teige and Martin Fränzle. Stochastic satisfiability modulo theories for non-linear arithmetic. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 248–262. Springer, 2008.

[VHMK97] Pascal Van Hentenryck, David McAllester, and Deepak Kapur. Solving polynomial systems using a branch and prune approach. *SIAM Journal on Numerical Analysis*, 34(2):797–827, 1997.

[VSHS02] Xuan-Ha Vu, Djamila Sam-Haroud, and Marius-Calin Silaghi. Approximation techniques for non-linear problems with continuum of solutions. In *Abstraction, Reformulation, and Approximation*, pages 224–241. Springer, 2002.

[XSE16] Bai Xue, Zhikun She, and Arvind Easwaran. Under-approximating backward reachable sets by polytopes. *Computer Aided Verification*, 9779:457–476, 2016.

[You31] Rosalind Cecily Young. The algebra of many-valued quantities. *Mathematische Annalen*, 104(1):260–290, 1931.

[ZMMM01] Lintao Zhang, Conor F Madigan, Matthew H 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.

[ZSR$^+$10] Lijun Zhang, Zhikun She, Stefan Ratschan, Holger Hermanns, and Ernst Moritz Hahn. Safety verification for probabilistic hybrid systems. In *International Conference on Computer Aided Verification*, pages 196–211. Springer, 2010.