Stochastic Satisfiability modulo Theories for Non-linear Arithmetic*

Tino Teige and Martin Fränzle

Carl von Ossietzky Universität, Oldenburg, Germany
{teige|fraenzle}@informatik.uni-oldenburg.de

Abstract. The stochastic satisfiability modulo theories (SSMT) problem is a generalization of the SMT problem on existential and randomized (aka. stochastic) quantification over discrete variables of an SMT formula. This extension permits the concise description of diverse problems combining reasoning under uncertainty with data dependencies. Solving problems with various kinds of uncertainty has been extensively studied in Artificial Intelligence. Famous examples are stochastic satisfiability and stochastic constraint programming. In this paper, we extend the algorithm for SSMT for decidable theories presented in [FHT08] to non-linear arithmetic theories over the reals and integers which are in general undecidable. Therefore, we combine approaches from Constraint Programming, namely the iSAT algorithm tackling mixed Boolean and non-linear arithmetic constraint systems, and from Artificial Intelligence handling existential and randomized quantifiers. Furthermore, we evaluate our novel algorithm and its enhancements on benchmarks from the probabilistic hybrid systems domain.

1 Introduction

Papadimitriou [Pap85] proposed the idea of uncertainty for propositional satisfiability by introducing *randomized* quantification in addition to existential quantification. This yields the stochastic propositional satisfiability (SSAT) problem where randomly quantified variables (randomized variables for short) are set to `true` with a certain probability. The solution of an SSAT problem Φ is a strategy to assign values to the existential variables that maximizes the overall satisfaction probability of Φ. Since the quantifier ordering of Φ, called prefix, allows an alternating sequence of existential and randomized quantifiers, the value of an existential variable depends on the values of the randomized variables with earlier appearance in the prefix. Consequently, in general such a solution is a tree of assignments to the existential variables depending on the values of preceding randomized variables. The SSAT framework is –at least theoretically– able to tackle many problems from Artificial Intelligence (AI) exhibiting uncertainty, e.g. stochastic planning problems. We just briefly note that there is

* This work has been partially supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center “Automatic Verification and Analysis of Complex Systems” (SFB/TR 14 AVACS, `www.avacs.org`).

a lot of work done on efficiently transforming AI problems into SSAT formulae, e.g. cf. [LMP01,ML98,ML03]. Littman [Lit99][1] proposed an algorithm for SSAT which extends the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [DP60,DLL62] (DPLL is the basic algorithm of most modern propositional satisfiability solver) with acceleration techniques like *unit resolution*, *purification*, and *thresholding*. For a very comprehensive survey about stochastic satisfiability confer [LMP01]. More recently, Majercik further improved the DPLL-style SSAT algorithm by introducing *non-chronological backtracking* [Maj04].

There are several attempts to extend the stochastic framework beyond the purely propositional case. Doing so yields stochastic constraint programming [Wal02,TMW06,BS06,BS07] in which the domains for all variables, also non-quantified variables, are so far still finite. In [BS07] it was shown that the stochastic constraint satisfaction problem (SCSP) is PSPACE-complete also for multiple objectives by describing an algorithm for SCSPs in non-prenex form. The authors of [FHT08] introduced the stochastic satisfiability modulo theories (SSMT) problem and its application for the reachability analysis of probabilistic hybrid automata. Moreover, they described an algorithm for SSMT for decidable theories, e.g. linear arithmetic over the reals and integers. Although quantified variables in an SSMT problem still have finite domains, this restriction is relaxed for non-quantified variables or, equivalently, the innermost set of existentially quantified variables.

In this paper, we extend and benchmark the ideas from [FHT08]. First, we propose an SSMT algorithm for *non-linear* arithmetic over the reals and integers. (Note that for the non-linear case the SSMT problem becomes undecidable in general.) Second, we implement this algorithm and prove its concept by presenting empirical results. Third, in addition to the *thresholding* pruning rules we adapt the promising idea of *solution-directed backjumping* [Maj04] to our setting. The algorithm described in this paper is strongly based on the iSAT algorithm [FHT$^+$07] for solving non-linear arithmetic constraint systems (involving transcendental functions) with complex Boolean structure over real- and integer-valued variables.[2] The iSAT approach tightly integrates the DPLL algorithm with interval constraint propagation (ICP, cf. [BG06] for an extensive survey) enriched by enhancements like conflict-driven clause learning and non-chronological backtracking. For a very detailed description of the iSAT algorithm the reader is referred to the original paper, in particular to the example on pages 217–219. As the core algorithm, iSAT is implemented in the constraint solver HySAT-II[3] which has been specifically designed for bounded model checking of hybrid (discrete-continuous) systems.

---

[1] We remark that the problem in this paper, called P-Sat, additionally contains universal quantification.

[2] Note that the input formula of iSAT is rewritten into conjunctive normal form beforehand and all arithmetic constraints are decomposed into primitive constraints [FHT$^+$07, Section 2].

[3] A HySAT-II executable, the tool documentation, and benchmarks can be found on `http://hysat.informatik.uni-oldenburg.de`.

$\Phi = \exists x \in \{0,1\}\ \rotatebox[origin=c]{180}{R}_{\langle(0,0.6),(1,0.4)\rangle} y \in \{0,1\} : (x > 0 \vee 2a \cdot \sin(4b) \geq 3) \wedge (y > 0 \vee 2a \cdot \sin(4b) < 1)$

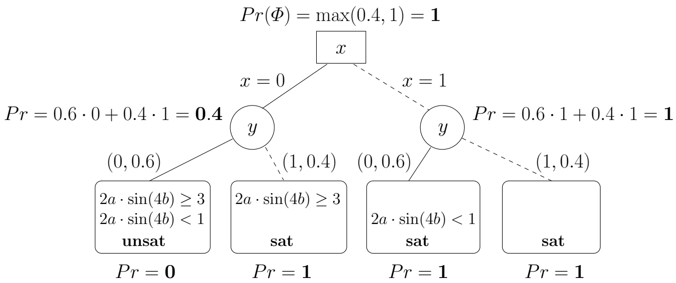

Fig. 1. Semantics of an SSMT formula depicted as a tree

Structure of the paper. In Section 2 we recall the definition of an SSMT problem while Section 3 presents an algorithm for SSMT for non-linear arithmetic theories. An experimental evaluation of that algorithm is given in Section 4. Section 5 concludes the paper and lists some directions for future work.

2 Stochastic satisfiability modulo theories

The *satisfiability modulo theories* (SMT) problem (cf., e.g., [RT06]) is a decision problem for logical formulae wrt. combinations of background theories. Thus, SMT generalizes the well-known propositional satisfiability (SAT) problem. The stochastic SMT (SSMT) problem extends SMT to support *randomized quantification* over discrete variables as known from SSAT and SCSP.

Let φ be an SMT formula in conjunctive normal form (CNF) over some quantifier-free potentially non-linear arithmetic theory T over the reals, integers, and Booleans. I.e., φ is a logical *conjunction* of clauses, and a *clause* is a logical *disjunction* of (atomic) arithmetic predicates from T, as in $\varphi = (x > 0 \vee 2a \cdot \sin(4b) \geq 3) \wedge (y > 0 \vee 2a \cdot \sin(4b) < 1)$. An SSMT problem

$$\Phi = Q_1 x_1 \in \mathrm{dom}(x_1) \ldots Q_n x_n \in \mathrm{dom}(x_n) : \varphi$$

is specified by a *prefix* $Q_1 x_1 \in \mathrm{dom}(x_1) \ldots Q_n x_n \in \mathrm{dom}(x_n)$ binding the variables x_i to the quantifier Q_i,[4] and an SMT formula φ, also called the *matrix*. We require that the domains $\mathrm{dom}(x)$ of quantified variables x are finite (and thus discrete). A quantifier Q_i, associated with variable x_i, is either *existential*, denoted as $\exists$, or *randomized*, denoted as $\rotatebox[origin=c]{180}{R}_{d_i}$ where d_i is a discrete probability distribution over $dom(x_i)$. The value of a variable x_i bound by a randomized quantifier (randomized variable for short) is determined stochastically by the corresponding distribution d_i, while the value of an existentially quantified variable can be set arbitrarily. We usually denote such a probability distribution d_i by a list $\langle(v_1, p_1), \ldots, (v_m, p_m)\rangle$ of value pairs, where p_j is understood as the probability of setting variable x_i to v_j. The list satisfies $v_j \neq v_k$ for $j \neq k$, $\forall j : p_j > 0$, $\sum_{j=1}^{m} p_j = 1$, and $\mathrm{dom}(x_i) = \{v_1, \ldots, v_m\}$. For instance,

[4] not all variables occurring in the formula φ need to be bound by a quantifier

$\reflectbox{R}_{\{(0,0.2),(1,0.5),(2,0.3)\}} x \in \{0,1,2\}$ means that the variable x is assigned the value 0, 1, or 2 with probability 0.2, 0.5, and 0.3, respectively.

The semantics of an SSMT problem is defined by the *maximum probability of satisfaction*. Intuitively, for an SSMT formula $\Phi = \exists x_1 \in \mathrm{dom}(x_1)\ \reflectbox{R}_{d_2} x_2 \in \mathrm{dom}(x_2)\ \exists x_3 \in \mathrm{dom}(x_3)\ \reflectbox{R}_{d_4} x_4 \in \mathrm{dom}(x_4) : \varphi$ determine the maximum probability s.t. there is a value for x_1 s.t. for random values of x_2 there is a value for x_3 s.t. for random values of x_4 the SMT formula φ is satisfiable. (As standard, an SMT formula φ (in CNF) is *satisfiable* iff there exists a valuation σ of the variables in φ s.t. each clause is satisfied under σ, i.e., iff at least one atom in each clause is satisfied under σ. Otherwise, φ is *unsatisfiable*.) More formally, the maximum probability of satisfaction $Pr(\Phi)$ of an SSMT formula Φ is defined recursively by the following rules where φ denotes the matrix.

1. $Pr(\varphi) = 0$ if φ is unsatisfiable.
2. $Pr(\varphi) = 1$ if φ is satisfiable.
3. $Pr(\exists x_i \in \mathrm{dom}(x_i) \ldots Q_n x_n \in \mathrm{dom}(x_n) : \varphi)$
   $= \max_{v \in \mathrm{dom}(x_i)} Pr(Q_{i+1} x_{i+1} \in \mathrm{dom}(x_{i+1}) \ldots Q_n x_n \in \mathrm{dom}(x_n) : \varphi[v/x_i])$.
4. $Pr(\reflectbox{R}_{d_i} x_i \in \mathrm{dom}(x_i) \ldots Q_n x_n \in \mathrm{dom}(x_n) : \varphi)$
   $= \sum_{(v,p) \in d_i} p \cdot Pr(Q_{i+1} x_{i+1} \in \mathrm{dom}(x_{i+1}) \ldots Q_n x_n \in \mathrm{dom}(x_n) : \varphi[v/x_i])$.

For an example see Fig. 1.

3 SSMT algorithm for non-linear arithmetic

In this section we present our algorithm SiSAT for calculating the maximum probability of satisfaction of an SSMT formula. More precisely, for a given SSMT formula Φ and a lower and upper target threshold $t_l, t_u \in [0,1]$ with $t_l \leq t_u$, the algorithm returns a witness value $p \leq Pr(\Phi)$ s.t. $p > t_u$ iff $Pr(\Phi) > t_u$, a value $p < t_l$ iff $Pr(\Phi) < t_l$, or otherwise (i.e., if $t_l \leq Pr(\Phi) \leq t_u$) the value $p = Pr(\Phi)$. If we wish to compute the exact value of $Pr(\Phi)$ we may thus simply set $t_l = 0$ and $t_u = 1$. SiSAT is an extension of the iSAT algorithm with an additional tightly integrated top layer for dealing with existential and randomized quantifiers. In the iSAT context, and thus in SiSAT, variables are interpreted over *interval valuations* which are manipulated during the proof search. As the iSAT algorithm is employed as the underlying core engine, we have to decompose all arithmetic predicates into so called *primitive constraints* by introducing additional auxiliary variables. A primitive constraint consists of exactly one relational operator, at most one arithmetic operator, and at most three variables. Note that for each (arithmetic) SMT formula there is an equi-satisfiable linearly-sized SMT formula in CNF just containing primitive constraints. For the input syntax of iSAT confer [FHT$^+$07, Section 2]. As an example, the matrix of Φ from Fig. 1 can be rewritten to, e.g., $(x > 0 \lor h_1 \cdot h_2 \geq 3) \land (y > 0 \lor h_1 \cdot h_2 < 1) \land (h_1 = 2a) \land (h_2 = \sin(h_3)) \land (h_3 = 4b)$. All algorithmic enhancements of iSAT are naturally inherited, such as *conflict-driven clause learning* & *non-chronological backtracking*, the *two-watching scheme*, as well as the combined *unit* and *inter-*

Algorithm 1 SiSAT(Pre, t_l, t_u)

In: A prefix Pre, lower and upper thresholds t_l, t_u.
Out: The satisfaction probability of the SSMT formula wrt. the thresholds.

while `true` **do**
  while `true` **do**
    $result := deduce()$. {Deducing.}
    if $result =$ `CONFLICT` **then**
      $resolved := analyze\_conflict()$. {Learning & Backjumping.}
      if not $resolved$ **then**
        return 0. {No solution for subproblem.}
      end if
    else if $result =$ `SOLUTION` **then**
      return 1. {Solution found.}
    else
      break. {Leave loop for branching.}
    end if
  end while
  {Existential quantifier.}
  if $head(Pre) = \exists x \in \mathrm{dom}(x)$ **then**
    $v \in \mathrm{dom}(x), set(x = v), \mathrm{dom}(x) := \mathrm{dom}(x) - \{v\}$.
    $p_0 = \mathrm{SiSAT}(tail(Pre), t_l, t_u)$.
    if $p_0 > t_u$ or $p_0 = 1$ or $\mathrm{dom}(x) = \emptyset$ **then**
      return p_0. {Upper threshold exceeded or maximum possible probability reached or all branches investigated.}
    end if
    $p_1 = \mathrm{SiSAT}(Pre, \max(p_0, t_l), t_u)$. {Neglect probabilities less than p_0.}
    return $\max(p_0, p_1)$. {Return maximum probability.}
  end if
  {Randomized quantifier.}
  if $head(Pre) = \text{Ɐ}_d x \in \mathrm{dom}(x)$ **then**
    $v \in \mathrm{dom}(x), (v, p_v) \in d, set(x = v), \mathrm{dom}(x) := \mathrm{dom}(x) - \{v\}$.
    $p_{remain} = \sum_{v' \in \mathrm{dom}(x), (v', p') \in d} p'$.
    $p_0 = \mathrm{SiSAT}(tail(Pre), (t_l - p_{remain})/p_v, t_u/p_v)$.
    if $(p_v \cdot p_0) > t_u$ or $(p_v \cdot p_0) = 1$ or $\mathrm{dom}(x) = \emptyset$ **then**
      return $p_v \cdot p_0$. {Upper threshold exceeded or maximum possible probability reached or all branches investigated.}
    end if
    if $p_{remain} < (t_l - p_v \cdot p_0)$ **then**
      return $p_v \cdot p_0$. {Lower threshold cannot be reached by remaining branches.}
    end if
    $p_1 = \mathrm{SiSAT}(Pre, t_l - p_v \cdot p_0, t_u - p_v \cdot p_0)$. {Update thresholds.}
    return $p_v \cdot p_0 + p_1$. {Return weighted sum.}
  end if
  {No quantifier left. Start iSAT branching.}
  if not $decide\_next\_branch()$ **then**
    return 1. {Approximative solution found.}
  end if
end while

val constraint propagation. For more details about iSAT the reader is referred to [FHT+07,THF+07].

Although we implemented SiSAT in an iterative manner, we present the basic ideas in a more intuitive recursive fashion (cf. Algorithm 1). Let $\Phi = Pre : \varphi$ be the SSMT formula to be solved and t_l, t_u be the lower and upper target thresholds, respectively. For the initial call SiSAT(Pre, t_l, t_u) the matrix φ, i.e. the clauses, of the SSMT formula Φ will be stored in a global database. New learned conflict clauses will be added to this database and, thus, will be public for all subproblems to be solved. The main loop of the SiSAT algorithm consists of the *deduction phase*, *conflict resolution*, and *branching*. Within the deduction phase the algorithm tries to conclude tighter intervals for the variables by chopping off non-solutions, starting from the domains of the variables as initial intervals. This is done by *unit propagation* and *interval constraint propagation*. Whenever a conflict occurs during search, i.e. if all constraints in a clause of the matrix are inconsistent with the current interval valuation, SiSAT analyzes the conflict. If the conflict can be resolved without revoking any assignment to a quantified variable then clause learning and backjumping are performed. Otherwise, i.e. if conflict resolution calls for undoing assignments to quantified variables, the function *analyze_conflict*() returns `false` indicating unsatisfiability of the current subproblem. Further backtracks concerning quantified variables are handled by the recursive nature of the algorithm. The branching step in the SiSAT framework corresponds to splitting an interval of a non-quantified variable or selecting a value for a quantified variable from its current domain. If a subproblem is decided to be satisfiable or unsatisfiable, the algorithm returns the probability 1 or 0 for that subproblem, resp., according to the semantics of Section 2. For the soundness of Algorithm 1, we require that the *deduce*() function returns `SOLUTION` *only* if the current quantifier prefix Pre is empty, i.e. branching for all quantified variables was performed beforehand.

The quantification issue is mainly treated within the branching step. In conformity with the semantical definition of the maximum probability, the branches for the quantified variables of the prefix are explored from left to right, and the resulting probabilities are combined correspondingly. The functions $head(Pre)$ and $tail(Pre)$ return the leftmost element $Q\, x \in \mathrm{dom}(x)$ of prefix Pre and the prefix originating from Pre where the leftmost element, i.e. $head(Pre)$, is eliminated, respectively. For a quantified variable x, we first select a value v from $\mathrm{dom}(x)$, assign v to x, and exclude v from $\mathrm{dom}(x)$. Then, we compute the probability for the branch $x = v$ by recursively calling the SiSAT procedure where the head element $Q\, x \in \mathrm{dom}(x)$ of the prefix is removed and the target thresholds are updated as follows: If x is existential then we simply preserve t_l, t_u. If x is randomized then we take the probability p_v for the value v and the maximum possible remaining probability $p_{remain} = \sum_{v' \in \mathrm{dom}(x), (v', p') \in d} p'$ for all remaining values $v' \neq v$ of x into account. I.e., the lower and upper target thresholds for this call are $(t_l - p_{remain})/p_v$ and t_u/p_v, resp., since if $t_l - p_{remain}$ cannot be reached by branch $x = v$ then t_l cannot be reached at all. (We remark that $t_l - p_{remain}$ can be a negative number and thus the new lower thresholds can be negative. This fact, however, does not influence the correctness since the termination criterion concerning lower thresholds applies only if the remaining probability $p_{remain} \geq 0$ is strictly less than the (updated) lower threshold.)

We exploit some pruning rules concerning the target thresholds which allow to return a result without visiting all branches. These rules are generalizations of the *thresholding* rules for the propositional case from [LMP01]. Let p_0 be the result of the SiSAT call. Whenever the computed probability for the branch $x = v$, i.e. either p_0 or $p_v \cdot p_0$, exceeds the upper threshold t_u, we can skip investigation of all other branches and return the (positive) result. Note that the same holds if the domain $\mathrm{dom}(x)$ becomes empty or the maximum possible probability 1 is computed. For the randomized case, it could also happen that the maximum possible probability of all remaining branches p_{remain} cannot reach the new lower target threshold $t_l - p_v \cdot p_0$. Then we are also allowed to immediately return the (negative) result without further exploration of the remaining subtree. For the remaining subtree, i.e. $\forall v' \neq v : x = v'$, we set the target thresholds as follows: If x is existential then the new lower and upper thresholds are $\max(p_0, t_l)$ and t_u, resp., since we can neglect probabilities of the remaining subtree less than the already computed value p_0. If x is randomized then both new thresholds decrease by the computed probability $p_v \cdot p_0$. Let p_1 be the result of the second SiSAT call, then we combine the computed probabilities in accordance with the SSMT semantics, namely $\max(p_0, p_1)$ for the existential and $p_v \cdot p_0 + p_1$ for the randomized case, and return the result.

If all quantified variables are currently assigned to some values, i.e. the prefix Pre is empty ($Pre = head(Pre) = \emptyset$), the algorithm applies the usual iSAT branching for all non-quantified (Boolean, integer, and real-valued) variables by splitting their intervals. Note that the iSAT algorithm is in general not able to find a solution of any mixed Boolean and non-linear arithmetic constraint formula or to prove its absence, since search algorithms based on interval splitting and interval constraint propagation over the reals are incomplete deduction systems. In order to avoid a potentially infinite sequence of splitting intervals, branching stops if for each (non-quantified) variable z the width $\omega(z)$ of the current interval of z is less than a predefined value $\varepsilon > 0$, i.e. $\omega(z) < \varepsilon$. In such a case, the algorithm found a *hull consistent* interval valuation (for more details cf. [FHT+07]) which we consider as an approximative solution. Thus, we return the probability 1.

3.1 Solution-directed backjumping

For stochastic Boolean satisfiability, *solution-directed* and *conflict-directed backjumping* was introduced by Majercik [Maj04]. We note, however, that this idea was first proposed for quantified Boolean satisfiability in [GNT03]. We adapt the promising technique of solution-directed backjumping to the stochastic mixed Boolean and (non-linear) arithmetic framework. The idea of solution-directed backjumping (SDB) is to avoid exploring the remaining branches of a quantified variable x, whenever the truth value of the formula remains the same if the cur-

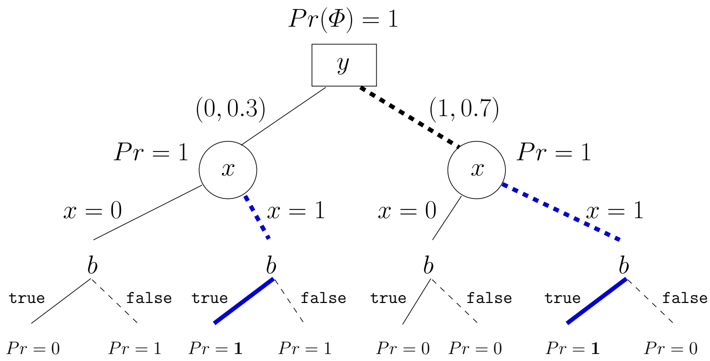

Fig. 2. Decision tree for Φ

rent value of x changed. I.e., the probability of all such subtrees are the same as for the current branch.

Motivating this heuristics we first consider an example. Given the following SSMT formula

$$\Phi = \text{Я}_{\langle(0,0.3),(1,0.7)\rangle} y \in \{0,1\}\ \exists x \in \{0,1\} : (\neg b\ \vee\ x \geq 1)\ \wedge\ (b\ \vee\ y < 1)$$

where $b \in \mathbb{B}$ is a Boolean variable[5]. The decision tree for Φ is depicted in Fig. 2. Calling the SiSAT algorithm on Φ, branching for the randomized variable y, say (1) $y = 1$, implies that $b = \texttt{true}$ (i.e. $b = 1$) by the second clause. Hence, the domain of b is narrowed to $[1, 1]$ by SiSAT's *deduce*() procedure. Then, by the first clause it follows that $x \geq 1$ has to hold, i.e. the domain of x is contracted to $\{1\}$. Thus, the only possibility for branching on the existential variable x is (2) $x = 1$. Here, *deduce*() returns `SOLUTION`. Consequently, the probability of branch (2) is 1. Since 1 is the maximum possible probability, SiSAT returns value 1 as the result for branch (1). I.e., the intermediate maximum satisfaction probability of Φ is $0.7 \cdot 1 = 0.7$. At this point, we take the idea of solution-directed backjumping into account: The assignment $y = 1$ has no impact on the satisfaction of the matrix (cf. Fig. 2). I.e., all other assignments to y also satisfy the formula and lead to the same probability. Hence, also the branch $y = 0$ results in probability 1 which means that we are able to conclude that $Pr(\Phi) = 0.7 + 0.3 \cdot 1 = 1$ without visiting the subtree for $y = 0$.

To be more formal, we first define a reason for a solution (analogously to a reason for a conflict). Given an SSMT formula $\Phi = Pre : \varphi$. Let ρ be a satisfying interval valuation of the matrix φ, i.e. $\rho(\varphi) = \texttt{true}$. If we consider hull consistency as an approximative solution then it is sufficient that ρ is hull consistent with φ, denoted as $\rho(\varphi) = \texttt{hc}$. We call a set $r \subseteq \{a : a \in c \in \varphi\}$ of atoms from φ a *reason for the satisfaction of* φ *under* ρ if the following hold:

1. $\forall c \in \varphi\ \exists a \in c : a \in r$, and

[5] The Boolean domain $\mathbb{B}$ is represented by the integer interval $[0, 1]$, where the values 0 and 1 correspond to the truth values `false` and `true`, respectively.

2. $\forall a \in r : \rho(a) = \texttt{true}$ (resp. $\rho(a) = \texttt{hc}$)

where $\rho(a)$ for an atom a gives the truth value of a under the interval valuation $\rho(x)$ of its variables x. Note that such a set r exists (while not being unique) whenever $\rho(\varphi) = \texttt{true}$ (resp. $\rho(\varphi) = \texttt{hc}$) holds. By $sat\_reasons(\varphi, \rho)$ we denote the set of all reasons r for the satisfaction of φ under ρ. In our example above, the only reason for the satisfaction is $\{x \geq 1, b\}$ where ρ is given by $\rho(y) = [1, 1], \rho(x) = [1, 1], \rho(b) = [1, 1]$.

Given a reason $r \in sat\_reasons(\varphi, \rho)$, a quantified variable x, and the current domain $\mathcal{D}_x$ of x, the predicate $no\_impact(r, \rho, x, \mathcal{D}_x)$ returns $\texttt{true}$ only if the current interval $\rho(x)$ of x has no impact on the satisfaction. More precisely,

$$no\_impact(r, \rho, x, \mathcal{D}_x) = \begin{cases} \texttt{true} & ; \forall a \in r \; \forall v_x \in \mathcal{D}_x : \\ & \quad x \notin vars(a) \vee \\ & \quad \rho[v_x/x](a) = \texttt{true} \text{ (resp. } \rho[v_x/x](a) = \texttt{hc}) \wedge \\ & \quad \forall y \in vars(a) \text{ s.t. } y \neq x : y \notin qvars(\Phi) \\ \texttt{false} & \text{otherwise} \end{cases}$$

where $vars(a)$ gives the set of all variables occurring in atom a, $qvars(\Phi)$ gives the set of all quantified variables occurring in the SSMT formula Φ, and $\rho[v_x/x]$ is the modified interval valuation ρ defined by $\rho[v_x/x](x) = [v_x, v_x]$ and $\forall y \neq x : \rho[v_x/x](y) = \rho(y)$.

If $no\_impact(r, \rho, x, \mathcal{D}_x) = \texttt{true}$ then each assignment $x = v_x$ with $v_x \in \mathcal{D}_x$ for x also satisfies each atom a from r. If x occurs in an atom $a \in r$ together with another quantified variable y, e.g. $a = (x \geq y)$, the return value is always $\texttt{false}$. This definition allows to perform solution-directed backjumping for each quantified variable locally without considering the mutual interplay with other quantified variables. For $x \geq y$, the solution $\rho(x) = [1, 1], \rho(y) = [0, 0]$, and the current domains $\mathcal{D}_x = [0, 1], \mathcal{D}_y = [0, 1]$, we could otherwise wrongly conclude that the values 1 for x and 0 for y have no impact on the satisfaction, since $\forall v_x \in \mathcal{D}_x : v_x \geq 0$ and $\forall v_y \in \mathcal{D}_y : 1 \geq v_y$. However, the assignment $x = 0, y = 1$ does not satisfy $x \geq y$. For our set of benchmarks, the SSMT formulae do not contain atoms with more than one quantified variable as we will see in Section 4. Thus, the definition of $no\_impact(r, \rho, x, \mathcal{D}_x)$ is sufficient for our application domain. However, in future work we will develop a more general and more global reasoning mechanism to tackle this issue.

The extended SiSAT algorithm supporting solution-directed backjumping is enriched by two more pruning rules which are only applied if a solution ρ with a fixed reason $r \in sat\_reasons(\varphi, \rho)$ was found. Let x be an existential variable in rule 1 and a randomized variable in rule 2, $\mathrm{dom}(x)$ be the updated domain of x, and p_0 be the currently computed probability. If x is randomized then p_v is the probability of the currently processed branch and p_{remain} the sum of the probabilities of the remaining branches (cf. Algorithm 1). The solution-directed-backjumping rules are as follows:

1. **if** $no\_impact(r, \rho, x, \mathrm{dom}(x))$ **then return** p_0.
2. **if** $no\_impact(r, \rho, x, \mathrm{dom}(x))$ **then return** $p_v \cdot p_0 + p_{remain} \cdot p_0$.

4 Evaluation of the algorithm

In this section, we evaluate our algorithm on SSMT formulae encoding discrete-time probabilistic hybrid automata. A probabilistic hybrid automaton (PHA) as described, e.g., in [FHT08] extends the notion of a hybrid automaton, where the non-deterministic selection of a transition is enriched by a probabilistic choice according to a distribution over variants of the transition. I.e., each transition carries a (discrete) probabilistic distribution. Each probabilistic choice within such a distribution leads to a potentially different successor mode while performing some discrete actions. For our case study, we are especially interested in k-bounded model checking (BMC) problems, i.e., we want to prove or disprove whether a given property P is satisfied with probability greater or equal p in a probabilistic hybrid automaton $\mathcal{H}$ along all its traces of length up to k. The automata considered for the experiments are shown in Fig. 3. These benchmarks are hand-made and serve as a first indicator for proving the concept of the approach and showing its current limits as well as the impact of the suggested algorithmic enhancements.

4.1 Description and encoding of the case studies

Let us consider the probabilistic automaton $\mathcal{H}_1$ depicted in Fig. 3. $\mathcal{H}_1$ is not hybrid since it lacks continuous state components but serves as an illustration of the idea of a probabilistic choice. The initial mode of $\mathcal{H}_1$ is s_1 (indicated by the incoming edge). The system can change its current mode by taking an outgoing transition if its transition guard evaluates to true. In our example, there is just one outgoing transition t_1 with the trivially satisfied guard `true`. After nondeterministically selecting a transition, the follower mode and action performed is given by a discrete distribution. Taking t_1 in $\mathcal{H}_1$, the probability of reaching s_1 and s_2 are 0.9 and 0.1, respectively. For a given reachability property P, say reaching mode s_2 in $\mathcal{H}_1$, the problem is to determine the maximum probability of satisfying P in k steps. I.e., the underlying problem is to find a strategy s.t. selecting a transition maximizes the probability of satisfying P. For 1 step, the probability Pr of reaching s_2 obviously is 0.1, for 2 steps $Pr = 0.1 + (0.9 \cdot 0.1) = 0.19$, and in general for $k \geq 1$ steps $Pr = \sum_{i=0}^{k-1} \left(0.1 \cdot 0.9^i\right)$. For $\mathcal{H}_1$ there are no alternative transitions over which a maximization could be achieved. However, the initial mode s_1 in $\mathcal{H}_2$ has two outgoing transitions. Assume that $k_y = 1$ and $c = 0$, then both transitions are enabled, i.e. the guards $y > c$ of t_1 and `true` of t_3 are true. Thus, we have to opt for either t_1 or t_3. For each step depth, we cannot reach the target state s_2 without taking t_1. Hence, the selection of t_3 does never yield the maximum probability of satisfying the reachability property.

We encoded the next state relation of $\mathcal{H}_1$, $\mathcal{H}_2$, and $\mathcal{H}_3$ as SSMT formulae and unwind these up to some depth k. To gain an impression of that encoding, we exemplify it for $\mathcal{H}_1$. For more details confer [FHT08]. Let k be the unwinding depth. Then, for each step $i = 1, \ldots, k$ and for the transitions t_1, t_2 we introduce existential variables $e^i_{t_1}, e^i_{t_2}$ encoding the nondeterministic choice and

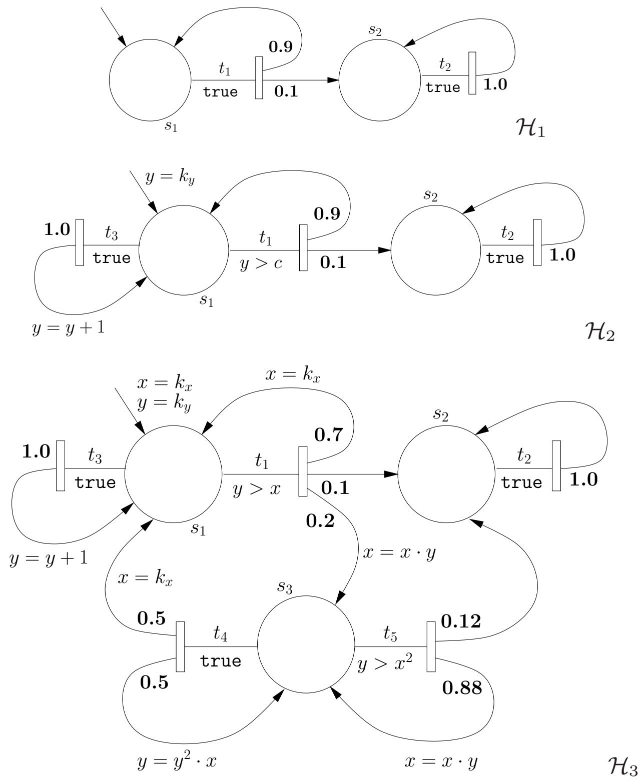

Fig. 3. Probabilistic hybrid automata $\mathcal{H}_1$ (top), $\mathcal{H}_2$ (middle), and $\mathcal{H}_3$ (bottom)

randomized variables $r^i_{t_1}, r^i_{t_2}$ encoding the probabilistic choice.[6] I.e., the prefix of the SSMT formula for step i is given by $\exists e^i_{t_1} \in \{0,1\}$ $Я_{\langle(0,0.9),(1,0.1)\rangle} r^i_{t_1} \in \{0,1\}$ $\exists e^i_{t_2} \in \{0,1\}$ $Я_{\langle(0,1.0)\rangle} r^i_{t_2} \in \{0\}$. The matrix is constructed as follows. The initial condition is $s^0_1 \wedge \neg s^0_2$, and the target property is $(s^0_2 \vee \ldots \vee s^k_2)$, where s^i_n is a Boolean variable encoding whether $\mathcal{H}_1$ is in mode s_n before step $i+1$ is executed. At each point of time, the system has to be in exactly one mode, and exactly one transition has to be taken for a mode change. I.e., $s^i_1 + s^i_2 = 1$ and $e^i_{t_1} + e^i_{t_2} = 1$. The transition relation is encoded as:

$$\begin{array}{l}
(s^{i-1}_1 \wedge (e^i_{t_1} = 1) \wedge (r^i_{t_1} = 0) \wedge s^i_1) \vee \\
(s^{i-1}_1 \wedge (e^i_{t_1} = 1) \wedge (r^i_{t_1} = 1) \wedge s^i_2) \vee \\
(s^{i-1}_2 \wedge (e^i_{t_2} = 1) \wedge (r^i_{t_2} = 0) \wedge s^i_2)
\end{array}$$

Note that an equi-satisfiable linearly-sized formula in CNF can be obtain efficiently. Moreover, we can simply arrange all sub-prefixes in front of the formula

[6] Note that [FHT08] describes an alternative approach where only one existential and one randomized variable are required per step i. For the sake of clarity, we opt for the simpler one here.

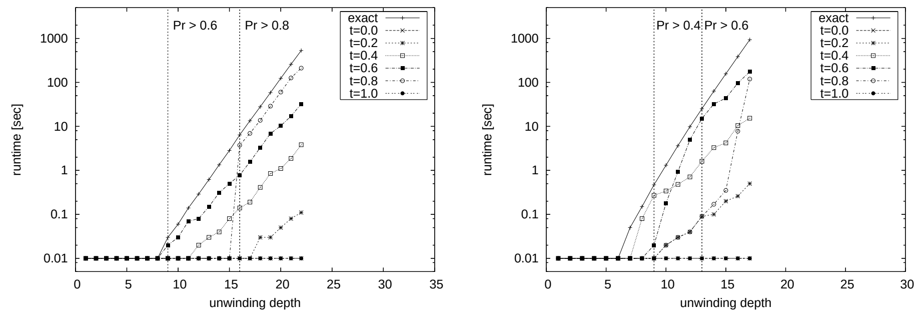

Fig. 4. Impact of thresholding for $\mathcal{H}_1$ (left) and $\mathcal{H}_2$ where $k_y = 1, c = 4$ (right)

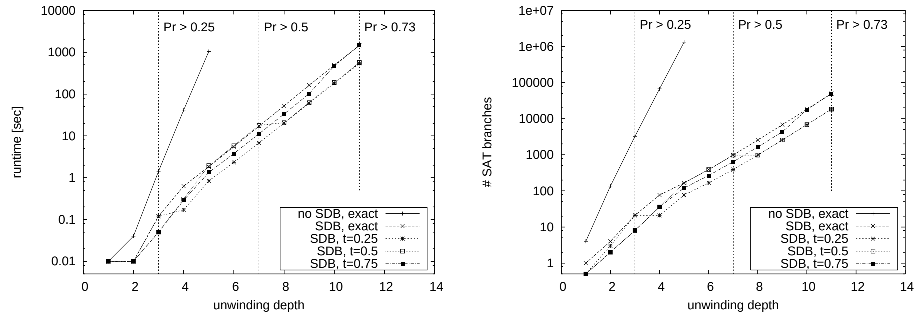

Fig. 5. Impact of solution-directed backjumping for $\mathcal{H}_3$: runtime (left) and number of found SAT branches (right)

(in ascending index-order), since all quantified variables in the transition relation for i do not occur in any other transition relation $j \neq i$. This yields an SSMT formula as required in Section 2.

For the *hybrid* case the encoding follows the same idea but we have to take account of the potentially non-linear real arithmetic *guards* of the transitions and *actions* to be performed for the probabilistic distributions. E.g., transition t_5 of $\mathcal{H}_3$ is encoded as:

$$\begin{aligned}&(s_3^{i-1} \wedge (e_{t_5}^i = 1) \wedge (y_{i-1} > (x_{i-1})^2) \wedge (r_{t_5}^i = 0) \wedge s_2^i) && \vee\\ &(s_3^{i-1} \wedge (e_{t_5}^i = 1) \wedge (y_{i-1} > (x_{i-1})^2) \wedge (r_{t_5}^i = 1) \wedge (x_i = x_{i-1} \cdot y_{i-1}) \wedge s_3^i)\end{aligned}$$

where the real-valued variables x_{i-1} and y_{i-1} represent the values of the real-valued system variables x and y, resp., before step i is executed.

4.2 Experimental results

This subsection compiles empirical results of the implemented algorithm SiSAT for the benchmarks from Subsection 4.1 encoded as SSMT formulae. The property to be checked for all automata is whether the mode s_2 can be reached.

	exact	$t = 0.0$	$t = 0.2$	$t = 0.4$	$t = 0.6$	$t = 0.8$	$t = 1.0$
unwinding depth **1**: 38 vars + 10 quantified vars, 111 clauses							
witness value	0.1	0.006	0.0	0.0	0.0	0.0	0.0
#SATs	4	1	0	0	0	0	0
#conflicts	0	0	0	0	0	0	0
runtime (sec)	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
unwinding depth **2**: 69 vars + 20 quantified vars, 212 clauses							
witness value	0.194	0.000252	0.092944	0.0392	0.0392	0.0042	0.0
#SATs	136	1	66	24	24	8	0
#conflicts	0	0	0	0	0	0	0
runtime (sec)	0.04	< 0.01	0.02	< 0.01	< 0.01	< 0.01	< 0.01
unwinding depth **3**: 100 vars + 30 quantified vars, 313 clauses							
witness value	0.2809	1.058e-05	0.201	0.1492	0.07734	0.02022	0.0
#SATs	3,248	1	2,743	1,390	832	320	0
#conflicts	6	0	6	2	0	0	0
runtime (sec)	1.43	< 0.01	1.22	0.63	0.37	0.15	< 0.01
unwinding depth **4**: 131 vars + 40 quantified vars, 414 clauses							
witness value	0.3603	4.445e-07	> 0.2	0.242	0.1339	0.04349	0.0
#SATs	67,360	1	16,167	42,891	21,088	8,380	0
#conflicts	21	0	6	20	10	10	0
runtime (sec)	41.48	< 0.01	9.81	26.42	13.02	5.13	< 0.01
unwinding depth **5**: 162 vars + 50 quantified vars, 515 clauses							
witness value	0.4323	1.867e-08	0.2002	0.4001	0.1908	0.0844	0.0
#SATs	1,322,700	1	213,560	1,126,492	447,616	201,252	0
#conflicts	35	0	21	35	29	29	0
runtime (sec)	1,044.0	< 0.01	167.7	903.6	352.2	158.6	< 0.01

Table 1. Empirical results for $\mathcal{H}_3$ where $k_x = 0, k_y = 2$

All benchmarks were performed on an 1.83 GHz Intel Core 2 Duo machine with 1 GByte physical memory running Linux. Concerning the issue of the *approximative* nature of solutions obtained by interval constraint propagation, we remark here that due to the deterministic assignments and the use of rational functions in the considered PHAs (cf. Fig. 3), we have obtained exact solutions on all benchmark runs. Hence, the computed probabilities are exact.

Concerning the performance of the SiSAT algorithm, Fig. 4 and 5 show that the runtimes dramatically grow over the BMC unwinding depths. As one can expect, the length of the quantifier prefix determines the runtimes. One acceleration technique we considered to battle against the high complexity is thresholding. Fig. 4 and Table 1 show a comparison for different thresholding parameters where *exact* means $t_l = 0$ and $t_u = 1$, and $t = k$ means $t_l = k$ and $t_u = k$. These results empirically prove the expected fact that thresholding leads to significant performance gains if the threshold parameters are *not* close to the exact maximum probability of satisfaction. Consider, e.g., the results for unwinding depth 5 of $\mathcal{H}_3$ in Table 1. The exact satisfaction probability is 0.4323. To compute this, SiSAT needed 1044 seconds, thereby visiting more than 1.3 millon satisfying branches. Setting $t_l = t_u = t = 0.4$ yields nearly the same performance while for thresholds $t < 0.4$ and $t > 0.4$ the runtimes quickly decrease. For the extreme values $t = 0$, i.e. finding just one solution, and $t = 1$, i.e. randomized quantifiers change to universal quantifiers, SiSAT terminates within fractions of a second.

While the impact of thresholding strongly depends on the pre-defined lower and upper target thresholds, solution-directed backjumping is independent from such settings but exploits the structure of the formula. Surprisingly, solution-directed backjumping yields performance gains of multiple orders of magnitude. The results for the more complex PHA $\mathcal{H}_3$ are illustrated in Fig. 5. For unwinding depth 5, the speedup factor obtained for the exact version is 567. This shows that the idea of skipping branches for which the probability remain the same actually works for our case studies. As shown on the right in Fig. 5, an enormous number of satisfying branches to be visited could be skipped when SDB was enabled. While the exact version *without* SDB was just able to solve the first 5 BMC unwindings of $\mathcal{H}_3$ within 100 minutes, the exact version *with* SDB solved 11 instances in the same time. The SSMT formula for depth 11 contains 110 quantified variables, 348 non-quantified variables, and 1121 problem clauses. Fig. 5 also indicates that on most of the BMC instances the combination of SDB and thresholding further increases the efficiency of the solver.

5 Conclusion and future work

In this paper, we presented an algorithm for stochastic SMT problems for non-linear arithmetic over the reals and integers together with experimental results from the reachability analysis of probabilistic hybrid automata. We showed that algorithmic enhancements like thresholding and solution-directed backjumping have a significant impact on the performance of the tool.

In future work, we will explore further techniques and heuristics to accelerate the SiSAT tool: For instance, further forms of backjumping within the quantified part of the decision tree. Another important aspect to improve the performance of search algorithms is to find suitable value and variable orderings. In the context of bounded model checking PHAs, we will work on an automatic translation of PHAs into SSMT formulae and bounded-model-checking optimizations like clause reusing and shifting [FH07]. Concerning the issue of approximate solutions, we will modify SiSAT to handle confidence intervals of probabilities instead of values s.t. we are able to obtain safe lower and upper bounds on the satisfaction probability when using also transcendental functions like sin or exp. Within the AVACS project[7], we will apply the SiSAT solver on benchmarks which deal with the impact of cooperative, distributed traffic management on flow of road traffic. These benchmarks are representative for a large number of hard scheduling and allocation problems and naturally show uncertain behavior.

Acknowledgements

The authors would like to thank Christian Herde, Holger Hermanns, Ralf Wimmer, Joost-Pieter Katoen, and Stephen Majercik for valuable discussions on SMT, probabilistic systems, and stochastic SAT algorithms. Furthermore, the authors are very grateful to the anonymous reviewers for their helpful comments.

[7] `www.avacs.org`

References

[BG06] F. Benhamou and L. Granvilliers, *Continuous and interval constraints*, Handbook of Constraint Programming (F. Rossi, P. van Beek, and T. Walsh, eds.), Foundations of Artificial Intelligence, Elsevier, 2006, pp. 571–603.

[BS06] T. Balafoutis and K. Stergiou, *Algorithms for Stochastic CSPs*, CP (F. Benhamou, ed.), LNCS, vol. 4204, Springer, 2006, pp. 44–58.

[BS07] L. Bordeaux and H. Samulowitz, *On the stochastic constraint satisfaction framework*, SAC, ACM, 2007, pp. 316–320.

[DLL62] M. Davis, G. Logemann, and D. Loveland, *A Machine Program for Theorem Proving*, CACM **5** (1962), 394–397.

[DP60] M. Davis and H. Putnam, *A Computing Procedure for Quantification Theory*, Journal of the ACM **7** (1960), no. 3, 201–215.

[FH07] M. Fränzle and C. Herde, *HySAT: An Efficient Proof Engine for Bounded Model Checking of Hybrid Systems*, FMSD **30** (2007), 179–198.

[FHT+07] M. Fränzle, C. Herde, T. Teige, S. Ratschan, and T. Schubert, *Efficient Solving of Large Non-linear Arithmetic Constraint Systems with Complex Boolean Structure*, JSAT Special Issue on SAT/CP Integration **1** (2007), 209–236.

[FHT08] M. Fränzle, H. Hermanns, and T. Teige, *Stochastic Satisfiability Modulo Theory: A Novel Technique for the Analysis of Probabilistic Hybrid Systems*, Proceedings of the 11th International Conference on Hybrid Systems: Computation and Control (HSCC'08), 2008.

[GNT03] E. Giunchiglia, M. Narizzano, and A. Tacchella, *Backjumping for quantified Boolean logic satisfiability*, Artif. Intell. **145** (2003), no. 1-2, 99–120.

[Lit99] M. L. Littman, *Initial Experiments in Stochastic Satisfiability*, Proc. of the 16th National Conference on Artificial Intelligence, 1999, pp. 667–672.

[LMP01] M. L. Littman, S. M. Majercik, and T. Pitassi, *Stochastic Boolean Satisfiability*, Journal of Automated Reasoning **27** (2001), no. 3, 251–296.

[Maj04] S. M. Majercik, *Nonchronological backtracking in stochastic Boolean satisfiability*, ictai **00** (2004), 498–507.

[ML98] S. M. Majercik and M. L. Littman, *MAXPLAN: A New Approach to Probabilistic Planning*, Artificial Intelligence Planning Systems, 1998, pp. 86–93.

[ML03] ———, *Contingent Planning Under Uncertainty via Stochastic Satisfiability*, Artificial Intelligence Special Issue on Planning With Uncertainty and Incomplete Information **147** (2003), no. 1-2, 119–162.

[Pap85] C. H. Papadimitriou, *Games against nature*, J. Comput. Syst. Sci. **31** (1985), no. 2, 288–301.

[RT06] S. Ranise and C. Tinelli, *Satisfiability modulo theories*, IEEE Intelligent Systems **21** (2006), no. 6.

[THF+07] T. Teige, C. Herde, M. Fränzle, N. Kalinnik, and A. Eggers, *A Generalized Two-watched-literal Scheme in a mixed Boolean and Non-linear Arithmetic Constraint Solver*, Proc. of EPIA 2007, New Trends in Artificial Intelligence, 2007, pp. 729–741.

[TMW06] A. Tarim, S. Manandhar, and T. Walsh, *Stochastic constraint programming: A scenario-based approach*, Constraints **11** (2006), no. 1, 53–80.

[Wal02] T. Walsh, *Stochastic constraint programming*, Proc. of the 15th European Conference on Artificial Intelligence (ECAI'02), IOS Press, 2002.