Constraint-Based Analysis of Concurrent Probabilistic Hybrid Systems: An Application to Networked Automation Systems[☆]

Tino Teige[*], Andreas Eggers, Martin Fränzle

Carl von Ossietzky Universität Oldenburg, Germany
Dpt. of Computing Science, Research Group Hybrid Systems

Abstract

In previous publications, the authors have introduced the notion of stochastic satisfiability modulo theories (SSMT) and the corresponding SiSAT solving algorithm, which provide a symbolic method for the reachability analysis of probabilistic hybrid systems. SSMT extends satisfiability modulo theories (SMT) with randomized (or stochastic), existential, and universal quantification, as known from stochastic propositional satisfiability. In this paper, we extend the SSMT-based procedures to the symbolic analysis of *concurrent* probabilistic hybrid systems. After formally introducing the computational model, we provide a mechanized translation scheme to encode probabilistic bounded reachability problems of concurrent probabilistic hybrid automata as linearly sized SSMT formulae, which in turn can be solved by the SiSAT tool. We furthermore propose an algorithmic enhancement which tailors SiSAT to probabilistic bounded reachability problems by caching and reusing solutions obtained on bounded reachability problems of smaller depth. An essential part of this article is devoted to a case study from the networked automation systems domain. We explain in detail the formal model in terms of concurrent probabilistic automata, its encoding into the SiSAT modeling

[☆]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`).

[*]Corresponding author

Email addresses: `tino.teige@informatik.uni-oldenburg.de` (Tino Teige), `andreas.eggers@informatik.uni-oldenburg.de` (Andreas Eggers), `fraenzle@informatik.uni-oldenburg.de` (Martin Fränzle)

language, and finally the automated quantitative analysis.

Keywords: Concurrent probabilistic hybrid systems, probabilistic logic, constraint satisfaction problems, problem solvers, automatic verification

---

1. Introduction

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

Within this article, we present an automatic approach to the constraint-based, symbolic analysis of *concurrent probabilistic hybrid systems* exhibiting both non-deterministic and probabilistic behavior as in infinite-state Markov decision processes. Our procedure belongs to the class of depth-bounded state-space exploration methods based on satisfiability solvers, which have originally been suggested for large finite-state systems by Groote et al. in [1] and Biere et al. in [2] and have since become popular under the term *bounded model checking* (BMC), now accounting for a major fraction of the industrial applications of formal verification. The idea of BMC is to encode the next-state relation of a system as a propositional formula, to unroll this to some given finite depth k, and to augment it with a corresponding finite unrav-

elling of the tableaux of (the negation of) a temporal formula in order to obtain a propositional SAT problem which is satisfiable if and only if an error trace of length k exists. Enabled by the impressive gains in performance of propositional SAT checkers in recent years, BMC can now be applied to very large finite-state designs.

Though originally formulated for discrete transition systems, the concept of BMC also applies to hybrid discrete-continuous systems. The BMC formulae arising from such systems comprise complex Boolean combinations of arithmetic constraints over real-valued variables, thus entailing the need for *satisfiability modulo theories* (SMT) solvers over arithmetic theories to solve them. Such SMT procedures are thus currently in the focus of the SAT-solving community [e.g., 3, 4, 5, 6], as is their application to and tailoring for BMC of hybrid system [e.g., 7, 8, 9, 10, 11].

The scope of the aforementioned procedures, however, is confined to qualitative (i.e., non-probabilistic) models of hybrid behavior as well as to purely Boolean queries of the form "can the system ever exhibit an undesirable behavior?", whereas requirements for safety-critical systems frequently take the form of bounds on error probability, requiring the residual probability of engaging into undesirable behavior to be below an acceptable threshold. Automatically answering such queries requires, first, models of hybrid behavior that are able to represent probabilistic effects like component breakdown and, second, algorithms for state space traversal of such hybrid models. Addressing this problem, the authors have suggested the scheme of *stochastic satisfiability modulo theories* (SSMT) [12, 13], which advances the reasoning power of SMT-based BMC to probabilistic hybrid models. Mirroring the tight integration of propositional SAT solving and theory solvers underlying the currently most successful SMT solver designs, SSMT solving borrows from stochastic propositional satisfiability (SSAT) solving [14, 15, 16, 17] and integrates the respective search algorithms with theory solving. Thereby, quantifiers of the classical *existential* and *universal* forms as well as of *randomized* (or, equivalently, *stochastic*) type become available in SMT, facilitating a symbolic encoding of probabilistic hybrid automata (PHA). The idea of the encoding is, in a nutshell, to encode the transition effects as an SMT formula, as usual, yet add the branching structure to the encoding by means of quantification, with existential quantification reflecting nondeterministic choices and randomized quantification reflecting probabilistic events [12]. This provides a symbolic encoding of the game semantics of PHA, rendering their analysis a matter of SSMT solving with the SSMT solver SiSAT [13, 18] whenever

the analysis problem can be formulated as a finite unravelling of the transition relation and the branching structure, as in probabilistic bounded model checking (PBMC). PBMC is the probabilistic extension to BMC, where the likelihood of reaching a given state within a given number of steps under an optimal strategy resolving the nondeterminism is computed rather than just reachability within that number of steps.

Previous publications provided a proof of concept in exploring symbolic encodings of PHA [12] and SSMT solving [13] for a *single*, monolithic probabilistic hybrid automaton, yet left the particular advantages of symbolic encodings on *concurrent* systems unexplored. It is well-known that in the non-probabilistic case, symbolic encodings permit an encoding of size linear in the number of parallel components, alleviating the state explosion arising from explicit construction of the product automaton, and thus enhancing the scalability of the automated analysis procedures. As this issue has not been explored for probabilistic systems so far, we devote this article to a further exploration of SSMT-based PBMC of *systems of concurrent probabilistic hybrid automata*. It is to be noted that this extension is far from trivial, as the usual mechanisms of compositionally representing concurrency in predicative semantics (e.g., using disjunction as a model of interleaving concurrency) do not directly apply due to the presence of the additional quantifiers explicating the game structure.

With its focus on scalability in the number of concurrent components and in the size of the discrete components, we consider this line of work complementary to the body of analytic techniques developed for classes of hybrid systems with more intricate stochastic dynamics. Our approach is very confined concerning the stochastic behavior; actually, it only admits probabilistic events within transitions. In the context of hybrid systems augmented with probabilities, a wealth of other, often richer models has been suggested by various authors. Basic models are the *probabilistic hybrid automata* addressed herein, where state changes forced by continuous dynamics may involve discrete random events determining both the discrete and the continuous successor state [19, 20, 12], *piecewise deterministic Markov processes* [21], where state changes may happen spontaneously in a manner similar to continuous-time Markov processes, and *stochastic differential equations* [22], where, like in Brownian motion, the random perturbation affects the dynamics continuously. In full generality, stochastic hybrid system (SHS) models can cover all such ingredients [23, 24].

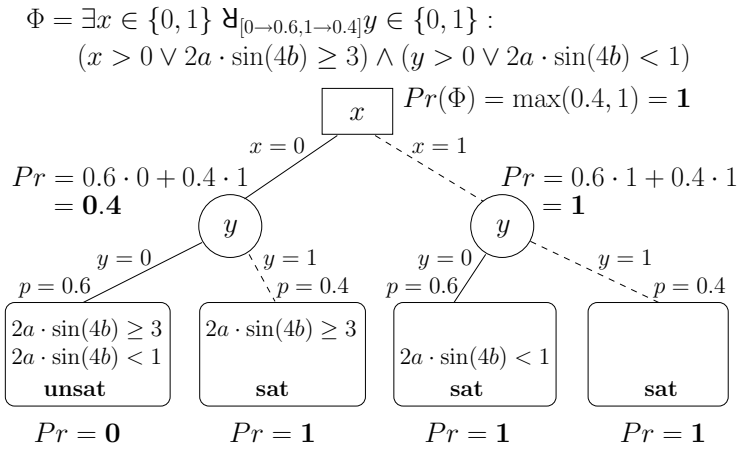


Figure 1: Semantics of an SSMT formula depicted as a tree illustrating the recursive descent through the quantifier prefix.

Structure of the paper. We first recall the notion of *stochastic satisfiability modulo theories* (SSMT) in Section 2. Thereafter, we present our computational model of a *system of concurrent discrete-time probabilistic hybrid automata* and the corresponding *probabilistic bounded reachability computation* (PBRC) and *probabilistic bounded model checking* (PBMC) problems (Section 3). To solve these PBRC and PBMC problems of concurrent probabilistic hybrid systems, we then describe the reduction from PBRC/PBMC to SSMT in Section 4 as well as the modeling language and algorithmic core of the SSMT-solver SiSAT (Section 5). In the latter section, we furthermore propose a novel PBRC/PBMC-related enhancement of the solver. To complement the presentation of the theoretical results and to show their practical feasibility, we apply the proposed approach to a realistic case study from the domain of networked automation systems (Section 6). Section 7 finally presents some conclusions.

2. Stochastic SMT

Stochastic satisfiability modulo theories (SSMT) combines the concepts of stochastic propositional satisfiability (SSAT) [14] and satisfiability modulo theories (SMT), e.g. [25]. SSMT thus enhances the reasoning power of SMT to probabilistic logics.

Let φ be a formula over some quantifier-free arithmetic theory $\mathcal{T}$ over the reals, integers, and Booleans, as in $\varphi = (x > 0 \vee 2a \cdot \sin(4b) \geq 3) \wedge (y >$

$0 \vee 2a \cdot \sin(4b) < 1$). An SSMT problem adds to the SMT problem φ a *quantifier prefix* $Q_1 x_1 \in \mathcal{D}_{x_1} \ldots Q_n x_n \in \mathcal{D}_{x_n}$ binding some variables x_i by quantifiers Q_i, yielding the SSMT formula $\Phi = Q_1 x_1 \in \mathcal{D}_{x_1} \ldots Q_n x_n \in \mathcal{D}_{x_n} : \varphi$. The quantifier-free SMT formula φ is also called the *matrix* of Φ. We demand that the individual domains $\mathcal{D}_x$ of quantified variables x are finite. A quantifier Q_i, associated with variable x_i, is either *existential*, denoted as $\exists$, *universal*, denoted as $\forall$, or *randomized*, denoted as $\bowtie_{d_i}$ where d_i is a discrete probability distribution over $\mathcal{D}_{x_i}$. The value of a variable x_i bound by a randomized quantifier (randomized variable for short) is determined stochastically according to the corresponding distribution d_i, while the value of an existentially or universally quantified variable can be set arbitrarily. We denote a probability distribution d_i by a function $[v_1 \to p_1, \ldots, v_m \to p_m]$ associating probability $p_j \geq 0$ to value v_j. The mapping $v_j \to p_j$ is understood as p_j is the probability of setting variable x_i to value v_j. The distribution satisfies $v_k \neq v_l$ for $k \neq l$, $\sum_{j=1}^{m} p_j = 1$, and $\mathcal{D}_{x_i} = \{v_1, \ldots, v_m\}$. For instance, $\bowtie_{[0 \to 0.2, 1 \to 0.5, 2 \to 0.3]} x \in \{0, 1, 2\}$ expresses that the variable x is assigned the values 0, 1, and 2 with probabilities 0.2, 0.5, and 0.3, respectively.

The semantics of an SSMT problem Φ is given by its *probability of satisfaction* $Pr(\Phi)$ defined as follows:

1. $Pr(\quad \varepsilon : \varphi \quad) = 0$ if φ is unsatisfiable.
2. $Pr(\quad \varepsilon : \varphi \quad) = 1$ if φ is satisfiable.
3. $Pr(\quad \exists x_i \in \mathcal{D}_{x_i} \cdot Q : \varphi \quad) = \max_{v \in \mathcal{D}_{x_i}} Pr(Q : \varphi[v/x_i])$.
4. $Pr(\quad \forall x_i \in \mathcal{D}_{x_i} \cdot Q : \varphi \quad) = \min_{v \in \mathcal{D}_{x_i}} Pr(Q : \varphi[v/x_i])$.
5. $Pr(\quad \bowtie_{d_i} x_i \in \mathcal{D}_{x_i} \cdot Q : \varphi \quad) = \sum_{v \in \mathrm{dom}(d_i)} d_i(v) \cdot Pr(Q : \varphi[v/x_i])$.

For an example see Figure 1.

3. Concurrent probabilistic hybrid automata

In practice, hybrid systems generally consist of multiple components evolving concurrently, both in the small, where controllers, sensor, actuators form identifiable units being coupled by one or more communication busses, or in the large, where a number of otherwise independent physical processes becomes connected via embedded control, as in a car platooning maneuver. Given the ubiquity of concurrency in such embedded control applications, it makes sense to avoid the detrimental effects of flattening concurrent systems

before verification, which have become known as state explosion in the finite-state case, and offer models directly accommodating concurrency instead.

In the sequel, we elaborate on such a model, where probabilistic hybrid automata evolve concurrently subject to a synchronous semantics involving global agreement on transitions as in CSP [26]. Within their evolution, the individual automata

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

The semantics has been defined with the goals of, firstly, permitting concise models by not imposing overly restrictive rules on use of variables and, secondly, providing separation between the possibly non-deterministic process of transition selection and the then purely probabilistic process of selection of a transition variant, as in classical, monolithic PHA [19, 20, 12]. To achieve the first, both the (then not really) local conditions for transition selection and the side effects can refer to non-local variables in both pre- and post-states, forcing parallel automata to agree on mutually consistent local transitions. The second, which is a necessary prerequisite for avoiding ill-formed probability measures due to interference between *policies* (or *schedulers, adversaries*) resolving non-determinism and the probabilistic choices, is accomplished by first committing a non-deterministic transition selection and then pursuing the probabilistic selection of a variant, yielding a deadlock if the latter experiment yields an outcome which is inconsistent to the earlier selection.

3.1. Syntax and semantics of concurrent PHA

A *system of concurrent discrete-time probabilistic hybrid automata* $\mathcal{S} = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ is given by a set of *discrete-time probabilistic hybrid automata*, where each probabilistic hybrid automaton $\mathcal{A}_i$ consists of the following.[1]

- A finite set $D_i = \{d_1^i, \ldots, d_{k_i}^i\}$ of *discrete variables* spanning the discrete state space (sometimes called the locations) of the hybrid automaton by means of the Cartesian product $\prod_{j=1}^{k_i} \mathrm{range}(d_j^i)$ of their *finite* ranges $\mathrm{range}(d_j^i)$. In order to permit non-local referencing of the state variables, we demand that $D_i \cap D_j = \emptyset$ if $i \neq j$, i.e. that the variable names used in different concurrent automata are disjoint.

- A finite vector $R_i = \{x_1^i, \ldots, x_{m_i}^i\}$ of *continuous state components* controlled by that automaton (yet visible to all others). Each continuous component x_j^i ranges over an interval $\mathrm{range}(x_j^i) = [l_{x_j^i}, u_{x_j^i}]$ within the reals $\mathbb{R}$. Again, we demand that $R_i \cap R_j = \emptyset$ if $i \neq j$. Additionally, we require discrete variable names and continuous variable names to be disjoint, i.e. $D_i \cap R_j = \emptyset$ for all i and j.

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

- A finite family $Tr_i = \{tr_1^i, \ldots, tr_{\ell_i}^i\}$ of *symbolic transitions*.

Each symbolic transition tr_j^i comprises the following.

- A *generalized transition guard* $g(tr_j^i)$ expressing the conditions on local and global variables required for establishing consensus on that transition. $g(tr_j^i)$ is an arithmetic predicate in the arithmetic theory $\mathcal{T}$ over variables in $D_1, \ldots, D_n$ and $R_1, \ldots, R_n$ as well as primed variants thereof, the latter representing the post-states. A transition guard

[1] Please note that there is an intuitive example complementing this formal description at the end of this subsection beginning on page 12.

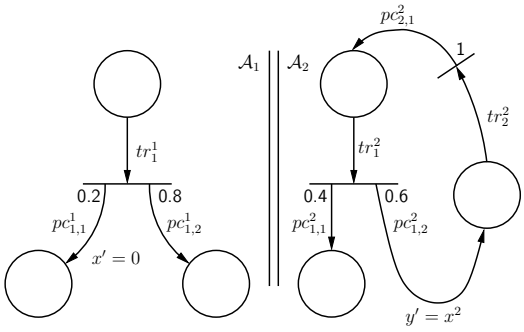
states the conditions on the discrete as well as the continuous state under which the transition may be taken. Note that the guard predicate can refer to the current states and post-states of all concurrent automata in $\mathcal{S}$. It thus provides an expressive formalism supporting synchronization through global consensus.

- A *probability distribution* $p(tr_j^i) \in P(PC_{tr_j^i})$, where $PC_{tr_j^i}$ is a finite and nonempty set of symbolic transition alternatives and $P(PC_{tr_j^i})$ denotes the set of probability distributions over $PC_{tr_j^i}$. $p(tr_j^i)$ assigns to transition tr_j^i a distribution over $|PC_{tr_j^i}|$ many transition alternatives.

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

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

  The above two requirements imply that any concurrently enabled combination of local transitions may permit at most one successor state for each possible resolution of the local probabilistic choices. This condition necessitates a global view of transitions and their related assignments, which motivates the following definitions, as illustrated in Figure 2. To obtain such a global view, let $NChoice = \prod_{i=1}^{n} Tr_i$ denote the Cartesian product of the local transition sets, thus representing the set of all potentially possible global transitions. As each local transition may have multiple probabilistic variants, the same



$$NChoice = \{(tr_1^1, tr_1^2), (tr_1^1, tr_2^2)\}$$
$$PChoice((tr_1^1, tr_1^2)) = \{(pc_{1,1}^1, pc_{1,1}^2), (pc_{1,2}^1, pc_{1,1}^2), (pc_{1,1}^1, pc_{1,2}^2), (pc_{1,2}^1, pc_{1,2}^2)\}$$
$$PChoice((tr_1^1, tr_2^2)) = \{(pc_{1,1}^1, pc_{2,1}^2), (pc_{1,2}^1, pc_{2,1}^2)\}$$
$$PChoice = PChoice((tr_1^1, tr_1^2)) \cup PChoice((tr_1^1, tr_2^2))$$
$$Assign((tr_1^1, tr_1^2), (pc_{1,1}^1, pc_{1,2}^2)) \equiv x' = 0 \wedge y' = x^2$$
$$p((tr_1^1, tr_1^2), (pc_{1,1}^1, pc_{1,2}^2)) = 0.2 \cdot 0.6 = 0.12$$

Figure 2: A parallel combination of probabilistic hybrid automata (guards omitted for the sake of clarity).

applies for global transitions. With regard to a single global transition $(tr^1, \ldots, tr^n) \in NChoice$, the set of associated probabilistic transition alternatives is $PChoice((tr^1, \ldots, tr^n)) = \prod_{i=1}^{n} PC_{tr^i}$, which is the Cartesian product of the local probabilistic transition alternatives available for the individual local transitions $tr^1, \ldots, tr^n$. Taking together all the global probabilistic alternatives of all global transitions, $PChoice := \bigcup_{nc \in NChoice} PChoice(nc)$ denotes the set of all global probabilistic choices. Given a global non-deterministic transition choice $tr = (tr^1, \ldots, tr^n) \in NChoice$ and a corresponding global probabilistic alternative choice $pc = (pc^1, \ldots, pc^n) \in PChoice(tr)$, we denote by $Assign(tr, pc) = \bigwedge_{i=1}^{n} asgn(tr^i, pc^i)$ the conjunction of the selected local assignment predicates. With these definitions, we can formalize the requirements that each concurrent execution of local transition alternatives be deterministic: We demand that for each global transition $tr \in NChoice$ and for each global probabilistic alternative $pc \in PChoice(tr)$, the associated global assignment $Assign(tr, pc)$ is deterministic or, equivalently, a partial function, i.e. it satisfies

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

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

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

Given a selection of transitions and transition alternatives, at most one post-state exists:

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

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

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

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

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

In this case, we denote by $Post(s, tr, pc)$ the unique post-state s'. Otherwise, the system deadlocks and we define $Post(s, tr, pc) = \bot$. For convenience, we define $Post(\bot, tr, pc) = \bot$ for all tr, pc.

The semantics of $\mathcal{S}$ is then defined by *runs* of $\mathcal{S}$ that are finite[2] alternating sequences of states and transitions, the latter involving both non-deterministic and probabilistic choices. A *run* $\langle s_0, (tr_1, pc_1), s_1, \ldots, (tr_k, pc_k), s_k \rangle$ *with* $s_0 \in States_{\mathcal{S}}$, $s_i \in States_{\mathcal{S}} \cup \{\bot\}$ *for* $i > 0$, *and* $(tr_i, pc_i) \in NChoice \times PChoice$ *of* $\mathcal{S}$ satisfies the following properties:

1. s_0 satisfies the initial predicate $\bigwedge_{i=1}^{n} init_i$,
2. $pc_j \in PChoice(tr_j)$ for all $1 \leq j \leq k$.
3. $s_{j+1} = Post(s_j, tr_j, pc_j)$ for all $0 \leq j \leq k-1$.

Thus, each run starts in the global initial state defined by the initial state predicates of the concurrent components. Upon each computation step, all concurrent automata first select non-deterministically among their transitions and then probabilistically under their variants. The corresponding transition step leads to a unique post-state, if existent, or to deadlock otherwise. The *probability of a transition step* from s to s' under non-deterministic choice $tr = (tr^1, \ldots, tr^n)$ and probabilistic choices $pc = (pc^1, \ldots, pc^n)$ is given by $p(tr, pc) = \prod_{i=1}^{n} p(tr^i)(pc^i)$ (cf. Figure 2). The *probability of a (finite) run* is the product of the probabilities of all transition steps of that run. The *length of a run* coincides to the number of transition steps involved. Note that the accumulated probability of all runs of a given length k under a given policy resolving non-determinism is always 1.

Example. Consider the system $\mathcal{S} = \{\mathsf{sensor}, \mathsf{controller}\}$ depicted in Figure 3. For the sake of clarity, we omitted probabilistic transition alternatives whenever just one exists, e.g. in the entire controller automaton. The idea of this very simple model is that sensor shall perform discrete state changes whenever the sine curve (evaluated over time) reaches its extremal values. That is, from sns_rise to sns_fall when hitting the maximum, and vice versa when reaching the minimum value. This switching behavior is synchronized with the controller, i.e. the controller retrieves such state changes of sensor in its guards. The controller regulates the continuous variable y that is increasing over time in discrete state ctr_rise and decreasing over time in ctr_fall. The global time of the system is modeled by variable t, and the time passage is governed by automaton sensor. A safety requirement of the system, e.g., is that the value of y may never leave the safe region $[-\pi/2, 3\pi/2]$. In case of

[2]Considering finite runs suffices for the purpose of this paper, since we investigate bounded reachability.

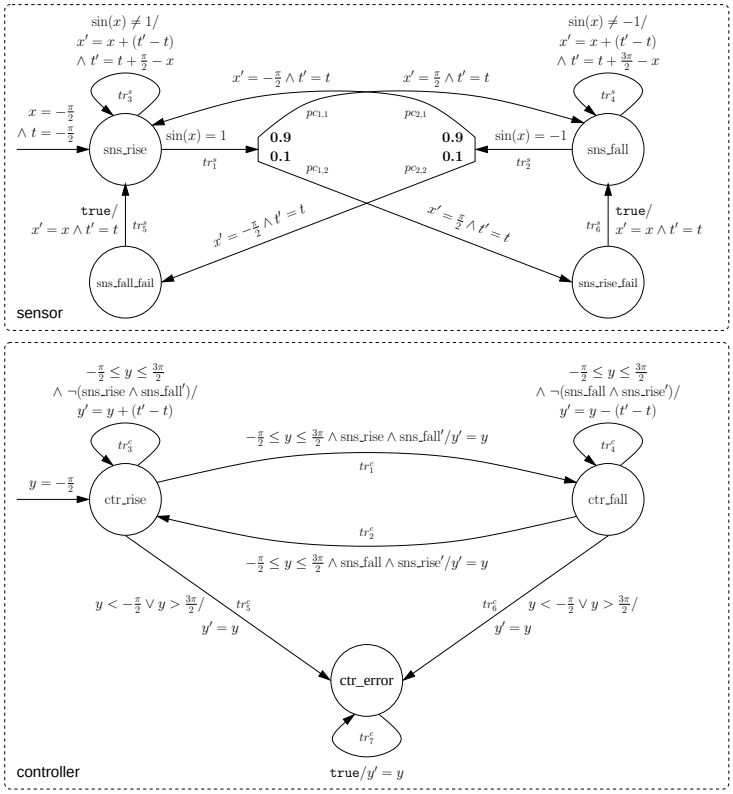


Figure 3: Two concurrent probabilistic hybrid automata sensor and controller.

violation, the controller enters the discrete state ctr_error and remains there forever. It is thus of interest whether the overall system may violate the safety property, and, if so, to quantify the system error.

The probabilistic behavior of $\mathcal{S}$ arises from the fact that the modeled sensor may overlook an optimum of the sine curve with some probability, say 0.1. In such cases, the controller may not perform a state change from ctr_rise to ctr_fall or vice versa. As depicted in Figure 3, this is modeled by the transition alternatives $pc_{1,2}$ and $pc_{2,2}$ of transitions tr_1^s and tr_2^s, respectively. Each such alternative occurs with probability 0.1 forcing the sensor to visit one of the fail states sns_rise_fail and sns_fall_fail. These fail states are then left immediately, but the effect is that the controller does not detect the discrete state change of sensor as desired.

Figure 4 depicts a possible run of $\mathcal{S}$. Initially (a) it starts in the (unique) initial state $((\mathsf{sns\_rise}, x = -\pi/2, t = -\pi/2), (\mathsf{ctr\_rise}, y = -\pi/2))$. The

	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)
sns_rise	1	1	0	0	0	1	1	0
sns_fall	0	0	0	1	1	0	0	1
sns_fall_fail	0	0	0	0	0	0	0	0
sns_rise_fail	0	0	1	0	0	0	0	0
x	$-\frac{\pi}{2}$	$\frac{\pi}{2}$	$\frac{\pi}{2}$	$\frac{\pi}{2}$	$\frac{3\pi}{2}$	$-\frac{\pi}{2}$	$\frac{\pi}{2}$	$\frac{\pi}{2}$
$(\sin(x))$	-1	1	1	1	-1	-1	1	1
t	$-\frac{\pi}{2}$	$\frac{\pi}{2}$	$\frac{\pi}{2}$	$\frac{\pi}{2}$	$\frac{3\pi}{2}$	$\frac{3\pi}{2}$	$\frac{5\pi}{2}$	$\frac{5\pi}{2}$
ctr_rise	1	1	1	1	1	1	1	0
ctr_fall	0	0	0	0	0	0	0	0
ctr_error	0	0	0	0	0	0	0	1
y	$-\frac{\pi}{2}$	$\frac{\pi}{2}$	$\frac{\pi}{2}$	$\frac{\pi}{2}$	$\frac{3\pi}{2}$	$\frac{3\pi}{2}$	$\frac{5\pi}{2}$	$\frac{5\pi}{2}$

Figure 4: Sample run of $\mathcal{S}$.

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

3.2. Probabilistic bounded reachability

In the sequel, we will be interested in the probability of reaching a set of *target* states within a given number of transition steps. In accordance with the predicative description of the system $\mathcal{S}$, we again define the target states by a predicate *Target* over variables in $D_1, \ldots, D_n$ and $R_1, \ldots, R_n$. For technical reasons, we assume the target states to be sinks of the transition relation. That is, once the system enters a target state it can remain there forever. This guarantees that each run of length k reaching the target states can be extended to a run of length $k + \ell$ also reaching the target states with the same probability.

Since the selection of transitions by the concurrent automata $\mathcal{A}_i$ involves non-determinism, a probability measure is well-defined only if considering a particular policy (scheduler, adversary) that resolves the nondeterminism. In most applications, we are interested in the *maximum probability of reaching the target states* achieved if ranging over arbitrary policies that may resolve nondeterminism using randomization, the history, etc. Such maximal probabilities are of interest in cases where the non-determinism models the choices available to a control algorithm and the target states define the desired goal, such that an optimal controller maximizes the probability of hitting the target states. Dually, maximum probabilities are also of interest when the non-determinism represents uncontrollable actions, like those of a (non-expert) human user of some plant, and the target states represent fatal system errors.

Due to considering step-bounded probabilities, we can avoid the explicit introduction of policies, and instead define the maximum probability of reaching some target states defined by a predicate *Target* within k steps directly as follows. For a proof that this coincides to the maximum probability of reaching the target within k steps, with the maximum taken over all possible schedulers, confer [18]. The proof provided there for the non-concurrent case does directly generalize to the concurrent case considered here, as the concurrent components engage into globally consistent synchronous transitions such that the overall system is semantically equivalent to the monolithic PHA spanned by the *Post* operation.

Definition 1 (Probabilistic bounded reachability). *Given a system of concurrent discrete-time probabilistic hybrid automata $\mathcal{S}$, a predicate Target over variables in $D_1, \ldots, D_n$ and $R_1, \ldots, R_n$, a depth $k \in \mathbb{N}$, and a state $z \in States_{\mathcal{S}}$, the* maximum probability of reaching states satisfying *Target*

from z in at most k steps *is denoted* $P^k_{Target}(z)$. *It is defined recursively over the depth* $k \in \mathbb{N}$ *as follows, where* $s \in States_{\mathcal{S}} \cup \{\bot\}$.

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

where $\bot$ *does not satisfy any* $\mathcal{T}$*-predicate, i.e. in particular* $\bot \not\models Target$.

Let $\mathcal{S}$ be a system of concurrent discrete-time probabilistic hybrid automata and *Target* be a predicate. Based on probabilistic bounded reachability, we denote the problem of computing the maximum probability of reaching states in $\mathcal{S}$ satisfying *Target* from the initial state of $\mathcal{S}$ within a given number of steps as the *probabilistic bounded reachability computation* (PBRC, for short) problem. The computation problem can also be stated as a decision problem, i.e. to determine whether the maximum reachability probability lies below or above a given probability threshold $\theta \in [0, 1]$. This decision problem is called *probabilistic bounded model checking* (PBMC, for short).

4. Reducing PBRC to SSMT

Probabilistic bounded reachability computation (PBRC) of a *single* probabilistic hybrid automaton can be reduced to stochastic satisfiability modulo theories (SSMT), as observed in [12], thus yielding a *symbolic* procedure for solving probabilistic bounded model checking (PBMC) problems. In this paper, we further extend this symbolic technique to systems of concurrent probabilistic hybrid automata. In contrast to explicit-state approaches (which many approaches in the realm of hybrid systems belong to with respect to the discrete state space), the predicative nature of the translation scheme to SSMT *avoids* the explicit construction of the product automaton, which grows exponentially in the number of parallel components.

As in the original paper [12], our construction proceeds in two phases: First, we generate the matrix of the SSMT formula. This matrix is an SMT formula encoding all non-deadlocked runs of system $\mathcal{S}$ of the given length $k \in \mathbb{N}$. The exclusion of deadlocked runs simplifies the matrix and is justified

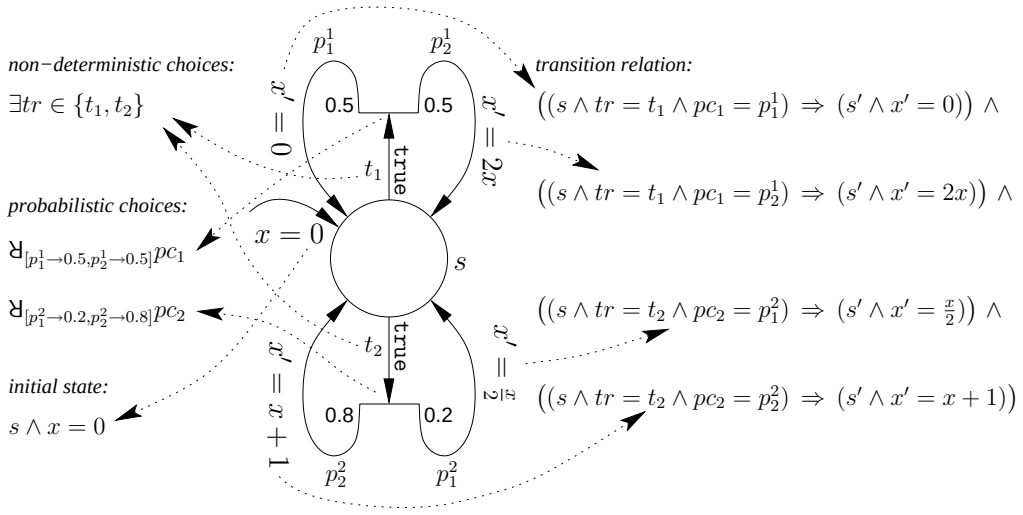


Figure 5: Example of the SSMT encoding scheme.

by the fact that such runs have no contribution to the probability of reaching the target states due to $\bot \not\models \mathit{Target}$. Thereafter, we add the quantifier prefix, which encodes the non-deterministic and the probabilistic choices of the concurrent automata, whereby non-deterministic choices yield existential quantifiers and probabilistic choices reduce to randomized quantifiers.

Before formally presenting the details of this encoding scheme in Section 4.2, we introduce the intuition by means of an example in Section 4.1.

4.1. Introductory example of the encoding

We illustrate the SSMT encoding of probabilistic hybrid automata by the simple example shown in Figure 5. For the sake of simplicity, the probabilistic automaton consists of only one location s and of one continuous variable x (initialized to zero). To perform a transition step, the automaton may non-deterministically select either transition t_1 or t_2 since both transition guards are trivially satisfied. As the definition of probabilistic bounded reachability (Definition 1) calls for *maximizing* the probability of reaching the target states, we need to select a transition for each step that maximizes the reachability probability. To do so, we encode the non-deterministic selection of transitions by existential quantification. In the example, we introduce an existentially quantified variable tr with a domain that consists of both transitions t_1 and t_2, i.e. $\exists tr \in \{t_1, t_2\}$. Transition selection is then followed by

a probabilistic choice of transition alternatives. When taking transition t_1, one of alternatives p_1^1 and p_2^1 are executed with equal probability 0.5. In case t_2 was selected, alternative p_1^2 is performed with probability 0.2 and p_2^2 with probability 0.8. This probabilistic selection of transition alternatives is mapped to randomized quantification. In the example, we introduce two randomized variables pc_1 for the probabilistic choice after transition t_1, and pc_2 for t_2, i.e. $\mathbb{R}_{[p_1^1 \to 0.5, p_2^1 \to 0.5]} pc_1$ and $\mathbb{R}_{[p_1^2 \to 0.2, p_2^2 \to 0.8]} pc_2$. By these quantified variables, we have described the non-deterministic choice of a transition and the probabilistic choice of a transition alternative for one step in the automaton.

In the following, we symbolically encode all system runs of bounded length. Each run starts in the initial system state. The initial state in the example is given by the predicate $\mathit{Init}(s, x) := s \wedge x = 0$, where s is a Boolean variable that is set to true if and only if the automaton is in location s. Thus, the satisfying valuation of $\mathit{Init}(s, x)$ represents the initial state of the automaton. In order to symbolically encode all systems runs, we have to symbolically describe all possible transition steps in the automaton, i.e. the relation between the pre- and post-state for all transitions and their transition alternatives. If the automaton is in location s, transition t_1 is selected non-deterministically, and alternative p_1^1 probabilistically, then the automaton re-enters location s and sets variable x to zero. This is described by the predicate $(s \wedge tr = t_1 \wedge pc_1 = p_1^1) \Rightarrow (s' \wedge x' = 0)$. The primed variables s' and x' represent the values of variables s and x after the transition step. The encodings for the remaining transition steps are shown in Figure 5. Conjoining the encodings for all transition steps by a logical conjunction, we obtain the transition relation predicate $\mathit{Trans}(s, x, s', x')$ that encodes all possible system steps from state (s, x) to state (s', x'). Thus, the assembled predicate $\mathit{Reach}(k) :=$

$$\mathit{Init}(s_0, x_0) \wedge \mathit{Trans}(s_0, x_0, s_1, x_1) \wedge \ldots \wedge \mathit{Trans}(s_{k-1}, x_{k-1}, s_k, x_k)$$

represents all system runs of length k. That is, each satisfying valuation of the predicate $\mathit{Reach}(k)$ encodes a run of the given probabilistic automaton. As the goal of the paper is probabilistic state reachability, we furthermore need to add a predicate that specifies the target states. Let us be interested in reaching states for which the value of variable x exceeds 100. Then, the target states predicate is given by $\mathit{Target}(s, x) := x > 100$. In reachability analysis, we are then just interested in all system runs that reach the target

states. Thus, the conjunction

$$\mathit{Reach}(k) \wedge (\mathit{Target}(s_0, x_0) \vee \ldots \vee \mathit{Target}(s_k, x_k))$$

filters out these runs.

To take into account the alternation of non-deterministic selections of transitions and probabilistic choices of transition alternatives, we need to add the quantifier prefix

$$\begin{aligned}
&\exists tr_1 \in \{t_1, t_2\}\ \text{Ƀ}_{[p_1^1 \to 0.5, p_2^1 \to 0.5]} pc_{1,1}\ \text{Ƀ}_{[p_1^2 \to 0.2, p_2^2 \to 0.8]} pc_{1,2} \\
&\ldots \\
&\exists tr_k \in \{t_1, t_2\}\ \text{Ƀ}_{[p_1^1 \to 0.5, p_2^1 \to 0.5]} pc_{k,1}\ \text{Ƀ}_{[p_1^2 \to 0.2, p_2^2 \to 0.8]} pc_{k,2}
\end{aligned}$$

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

By the construction of the overall SSMT formula $\Phi(k)$, it follows an important observation (that will be stated by Proposition 1 in the next Section 4.2): the probability of satisfaction of $\Phi(k)$, i.e. $Pr(\Phi(k))$, coincides with the maximum probability of reaching the target states within k transition steps, i.e. $Pr(\Phi(k)) = P^k_{\mathit{Target}}(\mathit{init})$ where init is the initial state of the system.

To clarify the basic idea of the symbolic SSMT encoding, we have just illustrated the translation scheme by a single probabilistic automaton. Based on this translation scheme, we can now provide a compact intuition for the SSMT encoding of *parallel* automata before presenting the formalized approach in the next subsection.

When considering a system of concurrently running probabilistic hybrid automata, we separately construct the predicates $\mathit{Reach}(k)$ for each automaton. The conjunction of these predicates then describes the runs of all single automata that are consistent among each other. Again adding k copies of the target states predicate, just system runs reaching the target states are left. For the construction of the quantifier prefix, we need to pay attention to the order of the quantified variables. Before each transition step, all automata non-deterministically select local transitions synchronously. After having established consensus on a global transition, each automaton probabilistically selects one of the available alternatives. In the quantifier prefix, for each unwinding depth, we thus first compile the existential variables of all automata

and thereafter the randomized ones representing the probabilistic alternatives. The quantifier prefix for k unwindings of the transition system is then composed by concatenating these quantifier prefixes in the same manner as was presented above for the single automaton.

4.2. Formalized encoding scheme

In the sequel, let $\mathcal{S} = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ be a system of concurrent discrete-time probabilistic hybrid automata using the definitions from Section 3.1, and *Target* be a predicate over variables in $D_1, \ldots, D_n$ and $R_1, \ldots, R_n$. Furthermore, let $k \in \mathbb{N}$ be the bound on the length of the system runs.

Phase 1: Constructing the matrix. We start by constructing the matrix $BMC_{\mathcal{S},Target}(k)$ of the resulting SSMT formula. The predicate $BMC_{\mathcal{S},Target}(k)$ encodes all runs of $\mathcal{S}$ of length k that reach some states satisfying *Target*. More formally, it will be defined by the predicates $INIT_{\mathcal{S}}(0)$, $\bigwedge_{i=1}^{k} TRANS_{\mathcal{S}}(i-1, i)$, and $\bigvee_{j=0}^{k} TARGET(j)$ representing the initial state of $\mathcal{S}$, the state changes by taking a transition from depth $i-1$ to i, and the target states at depth j, respectively.

1. For each discrete variable $d^i \in D_i$ of automaton $\mathcal{A}_i$ for $1 \leq i \leq n$, we take $k+1$ integer variables $[d^i]_j$ for $0 \leq j \leq k$, each with range range(d^i). A valuation of the variables $[d^i_1]_j, \ldots, [d^i_{k_i}]_j$ represents the discrete state of automaton $\mathcal{A}_i$ at depth j.

2. For each continuous state component $x^i \in R_i$ of $\mathcal{A}_i$ for $1 \leq i \leq n$, we take $k+1$ real-valued variables $[x^i]_j$ for $0 \leq j \leq k$, each with range range(x^i). The value of $[x^i]_j$ encodes the value of x^i at depth j.

3. For representing transitions of $\mathcal{A}_i$, for $1 \leq i \leq n$, we take k variables $[Tr^i]_j$ with domain Tr_i, for $1 \leq j \leq k$. The value of $[Tr^i]_j$ encodes the transition selection of $\mathcal{A}_i$ at step j.

4. For representing the probabilistic transition alternatives in PC_{tr^i} for each transition in $tr^i \in Tr_i$ of $\mathcal{A}_i$, for $1 \leq i \leq n$, we take k variables $[pc_{tr^i}]_j$ with domain PC_{tr^i}, for $1 \leq j \leq k$. The value of $[pc_{tr^i}]_j$ encodes the transition alternative for transition tr^i of $\mathcal{A}_i$ at step j. The value of such a variable $[pc_{tr^i}]_j$ will be irrelevant if the associated transition tr_i is not selected in step j, i.e. if $[Tr^i]_j \neq tr^i$.

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

$$INIT_{\mathcal{S}}(0) := \bigwedge_{i=0}^{n} init_i[[d^1_1]_0, \ldots, [x^n_{m_n}]_0 / d^1_1, \ldots, x^n_{m_n}]$$

where in $init_i$ each variable v is substituted by its representative $[v]_0$ at depth 0.

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

$$\bigwedge_{i=1}^{n} \bigwedge_{tr \in Tr_i} \left(\begin{array}{l} [Tr^i]_j = tr \Rightarrow \\ g(tr)[\quad [d_1^1]_{j-1}, [d_1^1]_j \ldots, [x^n_{m_n}]_{j-1}, [x^n_{m_n}]_j / \\ \qquad\quad d_1^1, (d_1^1)' \ldots, x^n_{m_n}, (x^n_{m_n})'] \end{array} \right),$$

where in $g(tr)$ each undecorated variable v is substituted by its representative $[v]_{j-1}$ at depth $j-1$, and each primed variable v' is replaced by $[v]_j$ for depth j.

7. Likewise, *assignments* at step $1 \leq j \leq k$ triggered by transition alternatives are dealt with by

$$\bigwedge_{i=1}^{n} \bigwedge_{tr \in Tr_i} \bigwedge_{pc \in PC_{tr}} \left(\begin{array}{l} [Tr^i]_j = tr \wedge [pc_{tr}]_j = pc \Rightarrow \\ asgn(tr, pc)[\quad [d_1^1]_{j-1}, [d_1^1]_j, \ldots, [x^n_{m_n}]_{j-1}, [x^n_{m_n}]_j / \\ \qquad\quad d_1^1, (d_1^1)' \ldots, x^n_{m_n}, (x^n_{m_n})'] \end{array} \right).$$

The conjunction of formulae 6 and 7 yields the predicate $TRANS_{\mathcal{S}}(j-1, j)$ encoding a system step from depth $j-1$ to j. Observe that with this predicative encoding, an infeasible choice of transitions and transition alternatives (e.g. due to inconsistent assignment predicates) that would lead to the distinguished state $\perp$ in the semantics immediately causes unsatisfiability of the formula $TRANS_{\mathcal{S}}(j-1, j)$. That is, system runs reaching the deadlock state $\perp$ do not satisfy the matrix generated by our translation scheme and are thus excluded. Considering reachability of *Target* states, this handling is correct as runs entering $\perp$ will never reach any target state and, vice versa, states reaching *Target* will never become deadlocked due to both $\perp$ and *Target* being sinks.

8. The next predicate denotes the target states for any of the steps depth $0 \leq j \leq k$.

$$TARGET(j) := Target[[d_1^1]_j, \ldots, [x^n_{m_n}]_j / d_1^1, \ldots, x^n_{m_n}]$$

Predicate $TARGET(j)$ is satisfied iff a target state is reached in step j.

9. It remains to compile the matrix $BMC_{\mathcal{S}, Target}(k)$ of the SSMT formula.

$$BMC_{\mathcal{S}, Target}(k) := INIT_{\mathcal{S}}(0) \wedge \bigwedge_{j=1}^{k} TRANS_{\mathcal{S}}(j-1, j) \wedge \left(\bigvee_{j=0}^{k} TARGET(j) \right)$$

Satisfying valuations of the quantifier-free formula $BMC_{\mathcal{S},Target}(k)$ are in one-to-one correspondence to the runs of the system $\mathcal{S}$ of length k that reach states satisfying the *Target* predicate.

Please note that we required all target states to be sinks of the transition relation in the sense that a stuttering step of probability 1 is enabled in each target state. If this is not the case then the above scheme should be replaced by

$$BMC'_{\mathcal{S},Target}(k) := INIT_{\mathcal{S}}(0) \wedge \bigwedge_{j=1}^{k} \left(\begin{array}{l} TARGET(j-1) \vee \\ TRANS_{\mathcal{S}}(j-1,j) \end{array} \right) \wedge TARGET(k)$$

in order to ensure that the matrix does not impose further constraints on the tail of a run once a target state has been reached.

Phase 2: Constructing the prefix. To construct the prefix of the SSMT formula $PBRC_{\mathcal{S},Target}(k)$, we need to encode the non-deterministic selection of transitions by all concurrent automata followed by the probabilistic choice of transition alternatives. Since we aim at maximizing the probability of reaching the target states, the non-determinism is resolved by existential quantification, while the probabilistic choices are mapped to randomized quantifiers.

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

$$NCHOICE_{\mathcal{S}}(j) := \exists [Tr^1]_j \in Tr_1 \ldots \exists [Tr^n]_j \in Tr_n$$

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

$$PCHOICE_{\mathcal{S}}(i,j) := \mathbb{R}_{pd_{tr_1^i}} [pc_{tr_1^i}]_j \in PC_{tr_1^i} \ldots \mathbb{R}_{pd_{tr_{\ell_i}^i}} [pc_{tr_{\ell_i}^i}]_j \in PC_{tr_{\ell_i}^i}$$

where the probability distributions $pd_{tr_q^i}$ are defined as $pd_{tr_q^i}(v) = p(tr_q^i)(v)$ if and only if $v \in PC_{tr_q^i}$.

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

$$CHOICE_{\mathcal{S}}(j) := NCHOICE_{\mathcal{S}}(j)\ PCHOICE_{\mathcal{S}}(1,j)\ \ldots\ PCHOICE_{\mathcal{S}}(n,j)$$

13. Finally, we construct the SSMT formula by concatenating the quantifier prefixes of the different computation steps in their natural sequence, representing the fact that the policy may draw decisions for later computation steps based on the outcomes of earlier ones, and adding the matrix representing runs of the system.

$$PBRC_{\mathcal{S},Target}(k) := CHOICE_{\mathcal{S}}(1)\ \ldots\ CHOICE_{\mathcal{S}}(k) : BMC_{\mathcal{S},Target}(k)$$

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

Proposition 1 (Correctness of reduction). *Given any system $\mathcal{S}$ and any depth $k \in \mathbb{N}$, the equality $Pr(PBRC_{\mathcal{S},Target}(k)) = P^k_{Target}(init_{\mathcal{S}})$ holds, where $init_{\mathcal{S}} \in States_{\mathcal{S}}$ is the unique initial state of $\mathcal{S}$. That is, the satisfaction probability of the symbolic encoding $PBRC_{\mathcal{S},Target}(k)$ coincides with the maximum probability of reaching a state in Target within k steps.*

5. The SSMT-based probabilistic bounded model checker SiSAT

In this section, we describe the SSMT solver SiSAT which was first published in [13]. In Section 5.1, we elaborate on the input language of the tool for modeling systems of concurrent probabilistic hybrid automata. Thereafter, we explain the algorithmic core of the tool and its optimizations in Section 5.2. Finally, we propose a novel algorithmic enhancement in the context of probabilistic bounded reachability computation, where the SiSAT tool is repeatedly called on $PBRC_{\mathcal{S},Target}(k)$ formulae with increasing depth k (Section 5.3). The main idea is to save satisfying assignments of $PBRC_{\mathcal{S},Target}(k)$ in order to reuse that information when solving $PBRC_{\mathcal{S},Target}(k')$ formulae with $k' > k$.

5.1. Modeling language

An encoding of a system of concurrent probabilistic hybrid automata – as formally introduced in Section 4 – in SiSAT requires, first, the declaration of variables to be used in the predicates[3], and the definition of the pattern of the quantifier prefix `DISTR` to encode non-deterministic and probabilistic choices

[3] An explicit declaration is necessary as SiSAT supports a many-sorted logics without imposing a naming convention on variables.

for each system step as by $CHOICE_{\mathcal{S}}(j)$. In order to describe the predicates $INIT_{\mathcal{S}}(0)$, $TRANS_{\mathcal{S}}(j-1,j)$, and $TARGET(k)$, SiSAT then provides the formula sections `INIT`, `TRANS`, and `TARGET`. Thus, a SiSAT input file consists of five sections:

1. The `DECL` section contains the declarations of all variables to be used in the individual predicates, where the types supported by SiSAT are range-bounded `float`, range-bounded `int`, and `boole`. Examples are the declaration `float [0, 1000] x` of a real-valued variable or `boole jump` of a Boolean variable. The section furthermore may contain constant declarations, like `define f = 2.0`.
2. The `INIT` section states a list of formulae comprising arbitrary Boolean combinations of arithmetic and propositional constraints. The conjunction of these formulae defines the set of possible initial states, as in `x = 0.6; !jump;`.
3. The `DISTR` section permits defining a pattern for the quantifier prefix to be unravelled during bounded model checking. This is done by stating quantifiers for unprimed variables, writing `E. x {1,2}:` for $\exists x \in \{1,2\}$, `A. y {1,2}:` for $\forall y \in \{1,2\}$, and `R. z p = [1 -> 0.8, 2 -> 0.2]:` for the randomized quantification $\text{ᴙ}_{[1\to 0.8, 2\to 0.2]} z \in \{1,2\}$.
4. In the `TRANS` section, which defines the transition relation of the system, variables may occur in both primed and unprimed form. An unprimed variable name denotes the value of the variable in the current step while a primed variable represents the value of that variable in the successor step, i.e. after the transition has taken place. For example, `jump' <-> !jump; jump -> f * x' = x; !jump -> x' = x + 2;` defines, among others, `jump` to be alternating.
5. The `TARGET` formula, finally, characterizes the set of states that shall be attempted to be reached, i.e. for which SiSAT shall check reachability, e.g. `x > 3.5;`.

Given these five parts, the SiSAT tool automatically constructs and solves the probabilistic bounded reachability formula $PBRC_{\mathcal{S},Target}(k)$ for some depth k. Moreover, the PBRC procedure works iteratively, i.e. it successively solves the unwindings of depths $s, s+1, \ldots$, where the start depth s and – if desired – the final depth can be specified by input parameters.

5.2. Algorithmic core

Algorithmically, the SiSAT tool builds on the iSAT algorithm [6] for solving the satisfiability problem of (quantifier-free) Boolean combinations of non-linear arithmetic constraints over the reals and integers. The iSAT approach tightly integrates modern *propositional satisfiability* (SAT) techniques (for an in-depth overview of SAT solving confer [27]), which are used to traverse the Boolean structure, with *interval constraint propagation* (for a comprehensive survey confer [28]) for reasoning about the non-linear arithmetics. Due to general undecidability of the problem and due to the use of interval constraint propagation, which is a highly incomplete deduction calculus, iSAT cannot decide the satisfiability of all input formulae in general. However in such undecided cases, iSAT always returns a consistent interval valuation of small volume[4] which can be considered as an *approximate* solution. The soundness properties are that unsatisfiability results are always reliable, i.e. a constraint system is unsatisfiable whenever iSAT claims it to be so, yet satisfiability results are approximate in the sense that they prove availability of a valuation which either satisfies all constraints or violates any arithmetic (in-)equality by at most a tiny, user-definable margin. In the sequel, we will call any constraint solver featuring this property an *almost-decision procedure with definable tolerance*.

Basically, SiSAT adds an additional layer for quantifier handling to iSAT which traverses the Cartesian product of the domains of the quantified variables in lexicographic order and computes the satisfaction probabilities for the individual quantifiers, as depicted in Figure 1. Upon each complete assignment to the quantified variables, iSAT is called to almost-decide the satisfiability of the corresponding quantifier-free subproblem. However, that naive approach is far from scalable as it has to solve one SMT problem per element of the Cartesian product of the quantifier ranges, which is exponential in the number of quantified variables. To overcome that problem, we have added aggressive pruning rules that save visits to major parts of the quantifier ranges based on semantic inferences (for details confer [13]). Such pruning exploits the *conflict clauses* generated by iSAT through its *conflict-driven clause learning*, as these record inconsistent value assignments which need not be probed again when traversing the set of possible assignments

[4]The maximum width of each interval occurring in that consistent interval valuation can be specified by setting an input parameter of iSAT.

to quantified variables. Dually, *solution-directed backjumping* hits when a satisfying assignment has been found: Upon satisfying assignments, SiSAT analyzes which quantified variables have *no* impact on the solution and saves probing their alternative values by directly assigning the same satisfaction probability to them. *Thresholding* exploits the fact that the PBMC problem does not ask for computing the exact satisfaction probability, but rather for deciding whether the probability of satisfaction exceeds or misses certain upper or lower target thresholds. When branching through the different value assignments permitted by a quantifier, it suffices to check whether the upper threshold is exceeded by the already processed branches or whether the lower threshold can no longer be reached by all remaining branches, abandoning the whole quantifier if one of these checks succeeds.

In the following, we explain some recent algorithmic features that are adapted from stochastic propositional satisfiability (SSAT) (an overview of SSAT algorithms and optimizations is given in [29]). *Purification* as a mechanism to prune the quantifier tree, as already considered for SSAT is adapted to the SSMT setting as follows. Assume that the remaining undecided clauses[5] of the formula are *monotonic* (or *antitonic*) in a (quantified) variable x in the sense that if $x = v$ satisfies some undecided clauses then so does $x = v'$ for each $v' > v$ (for each $v' < v$, resp.). For an existential or universal quantifier binding x, it then suffices to investigate only one of the possible values for x: for an existentially quantified x, we probe the largest possible value in case of monotonicity and the smallest in case of antitonicity, and vice versa for a universal one. Purification is in general impossible for randomized variables, since all branches give some contribution. Detecting monotonicity is hard as soon as arithmetic constraints are involved. The current version of SiSAT recognizes monotonicity for a variable x if each undecided arithmetic constraint containing x is either inside a satisfied clause or is a simple bound like $x \geq 3$ or $x \leq 6$, with all bounds outside satisfied clauses sharing the same polarity.

The efficiency of quantifier traversal heavily depends on the *value ordering decision heuristics*, i.e. the order in which possible assignments are probed, as some values trigger earlier or more aggressive pruning. SiSAT employs a dynamic decision heuristic based on *value activities*, where values

[5]The matrix of the SSMT formula is assumed here to be in *conjunctive normal form* (CNF), i.e. a conjunction of disjunctions of constraints.

with highest activities are preferred for branching. Whenever a satisfaction probability p for a branch $x = v$ is computed, the activity act_v of value v is updated as follows: If variable x is existential or randomized then act_v is incremented by p, i.e. the higher probability the higher activity. For universal variables, act_v is incremented by the complementary probability $(1 - p)$, i.e. the smaller probability the higher activity since universal quantifiers minimize probabilities.

The idea of *probability learning* (aka. *memoization* [30]) is to store probabilities for quantifier subtrees, or rather subformulae, in order to reuse that information in future search. Whenever SiSAT needs to solve the same subformula again, the corresponding stored satisfaction probability can be retrieved immediately without solving the subproblem again.

5.3. PBRC-related algorithmic tool enhancement

Let be given a system $\mathcal{S}$ of concurrent probabilistic hybrid automata and some predicate *Target* defining the system states to be reached, both encoded in the SiSAT modeling language (Section 5.1). Calling SiSAT on this input with start depth $s \geq 0$, the tool iteratively computes the satisfaction probabilities[6] of the SSMT formulae $PBRC_{\mathcal{S},Target}(s), PBRC_{\mathcal{S},Target}(s+1), \ldots$, until some specified target threshold $\theta \in [0, 1]$ is exceeded by the computed probability or the final depth is reached. In Section 3.2, we have required each *Target* state to be a sink, i.e. once the system enters a target state it remains there forever. By this assumption, we observe the following simple but important property.

Property 2. *Each system run r of length k satisfying some predicate Target can be extended to a system run $r \cdot r'$ of length $k + k'$ also satisfying Target. Moreover, the probabilities of both runs r and $r \cdot r'$ are the same.*

From Property 1 of a system $\mathcal{S}$ of concurrent probabilistic hybrid automata as introduced in Section 3, namely that the post-state of some state in $\mathcal{S}$ is

[6] In general, SiSAT cannot compute the exact probability of satisfaction due to undecidability of the arithmetic theory underlying the constraint language. As the incorporated arithmetic constraint solver iSAT is an almost-decision procedure, satisfaction probabilities can, however, at most be over-estimated, which provides safe approximations of the risk of reaching an undesirable state. Furthermore, the amount of over-estimation will generally be very small, if existent at all, due to the extremely small tolerances permitted for approximate solutions in the almost-decision procedure.

unambiguously defined by the non-deterministic transition and probabilistic transition alternative choices of the automata $\mathcal{A}_i$, it follows that each run r of $\mathcal{S}$ is unambiguously defined by the (unique) initial state and all choices in r. We denote the sequence of transitions and transition alternatives choices $((tr_1, pc_1), \ldots, (tr_k, pc_k))$ of run r by $\mathit{Choice}(r)$. The above observations facilitate the PBRC-related algorithmic tool enhancement of storing and reusing system runs satisfying *Target* for later calls of SiSAT, which we refer to as *solution caching*.

When solving an SSMT formula $\mathit{PBRC}_{\mathcal{S},\mathit{Target}}(k)$ and given that SiSAT has found a run r which satisfies *Target*, then – as mentioned above – r is unambiguously defined by all its non-deterministic and probabilistic choices. By the reduction to SSMT (cf. Section 4), these transition and transition alternative choices were mapped to variables bound by existential and randomized quantifiers, respectively, in the SSMT formula $\mathit{PBRC}_{\mathcal{S},\mathit{Target}}(k)$. Therefore, the satisfying assignment of $\mathit{PBRC}_{\mathcal{S},\mathit{Target}}(k)$ corresponding to run r just depends on the valuation v_r of the quantified variables. Moreover, by Property 2 we conclude that for each formula $\mathit{PBRC}_{\mathcal{S},\mathit{Target}}(k + k')$ with $k' \geq 0$ the valuation v_r determines a run $r \cdot r'$ that satisfies *Target*. In order to avoid recomputing the same satisfying valuations v_r in SSMT formulae of greater depths $k + k'$, SiSAT stores all such satisfying valuations v_r in a tree-like data-structure spanned by the values of the quantified variables. Whenever a stored valuation v_r is visited when solving larger SSMT formulae, SiSAT immediately deduces satisfaction probability 1 for the current quantifier subtree without exploring the remaining quantifiers in the prefix.

An optimization of solution caching that we have implemented allows to compress stored valuations v_r in size. The idea is as follows: whenever for a given run prefix r all possible extensions lead to a target state, r itself can be considered as inevitably leading to the target states as well. For runs stored in the solution cache of the form $r \cdot ((tr_1, pc_1), s_1), \ldots, r \cdot ((tr_m, pc_m), s_m)$ with states $s_1, \ldots, s_m \in \mathit{States}_{\mathcal{S}}$ satisfying *Target* and $(tr_1, pc_1), \ldots, (tr_m, pc_m)$ being all possible transition and transition alternative choices, we may thus safely conclude that already the common prefix r is a run for which only extensions exist that will finally satisfy *Target*. Under the already named assumptions that *Target* is a sink and the adversary is always trying to drive the system to *Target*, all longer runs starting with this prefix r and all valuations representing these runs, will thus satisfy *Target*. By solving the $\mathit{PBRC}_{\mathcal{S},\mathit{Target}}(k)$ formulae iteratively for increasing depth k, marking r as satisfying *Target* itself is thus a correct simplification. Let n_r be the last

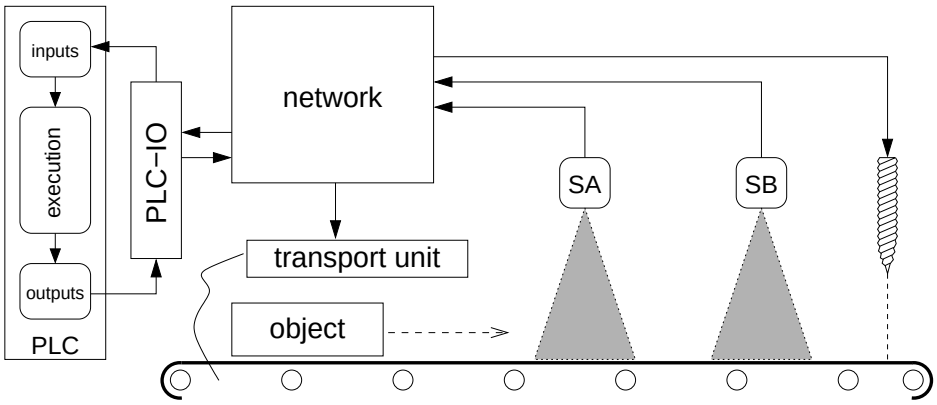


Figure 6: A networked automation system from [31].

node in the tree-like data-structure for run r (i.e. node n_r represents the last quantified variable in the valuation v_r). Then, the data-structure is updated by removing all successor nodes of node n_r, i.e. by removing the choices $(tr_1, pc_1), \ldots, (tr_m, pc_m)$. Multiple applications of this optimization lead to very compact representations of the solution set found so far, and facilitates earlier pruning of the quantifier tree due to the presence of shorter runs. In Section 6.5, we will evaluate the practical significance of solution caching on a benchmark.

6. Application to a networked automation system

In this section, we illustrate the quantitative analysis approach using the SiSAT tool. As a case study, we take the *networked automation system* (NAS) studied in [31]. After a detailed description of the NAS application in Section 6.1, we present the formal model as a system of concurrent probabilistic hybrid automata in Section 6.2, and exemplify the encoding of the system into the SSMT framework in Section 6.3. The last two subsections then deal with the automatic analysis of the model using SiSAT: in Section 6.4, we give results concerning the probabilistic behavior of the NAS model, whereas Section 6.5 evaluates the optimization of *solution caching* recently incorporated into SiSAT.

6.1. Description of the case study

A schematic overview of the *networked automation system* (NAS) studied in [31] is depicted in Figure 6. As a typical NAS, it involves networked control

by programmable logic controllers (PLCs) connected to several sensors and actuators via wired and wireless networks. Its objective is to transport a workpiece from its initial position to the drilling position by means of a transportation unit which controls the speed of the conveyor belt on which the object is transported. The PLC can set the deceleration of the belt via network messages to the transportation unit, but cannot determine the position of the object unless it hits two sensors SA and SB close to the drilling position. The sensors are connected to the IO card of the PLC over the network. When the object reaches sensor SA, the PLC reacts with sending a command to the transportation unit that forces it to decelerate to slow speed. Likewise, the transportation unit is asked to decelerate to stand-still when the PLC notices that SB has been reached. The goal is that the object halts close to the drilling position despite the uncontrollable latencies in the communication network. The parameters of the system are adopted from [31] as far as indicated. Thus, one length unit (lu) is 0.01 mm, and one time step (ts) is 1 ms. The positions of SA and SB are 699 lu and 470 lu, respectively, while the desired drilling position is at 0 lu. The initial speed of the object is 24 lu/ts and the slow speed is 4 lu/ts; the decelerations for the two types of speed changes at SA and SB are 2 and 4 $\mathrm{lu/ts^2}$, respectively. The network routing time is determined stochastically, needing 1 ts for delivery with probability 0.9 and 2 ts with probability 0.1. The cycle time of the PLC-IO card is 10 ts, and of the PLC is 7 ts. The minimum sampling interval is 1 ts. Due to the initial speed of 24 lu/ts, the initial position of the object is thus equally distributed over 24 neighboring values. In our setting, the initial position ranges between 999 and 976 lu. In this case study, no other disturbances or delays are assumed. We remark that the case study features probabilistic choices only, i.e. it lacks non-determinism.

6.2. Model of the case study

The NAS case study modeled as a system of 10 concurrent probabilistic hybrid automata is depicted in Figures 7, 8, and 9. In order to get an intuition of the rather large model and the interaction between the single automata, we intuitively explain the basic ideas of each automaton in Section 6.2.1, and exemplify the interconnections in the model by means of a sample system run in Section 6.2.2.

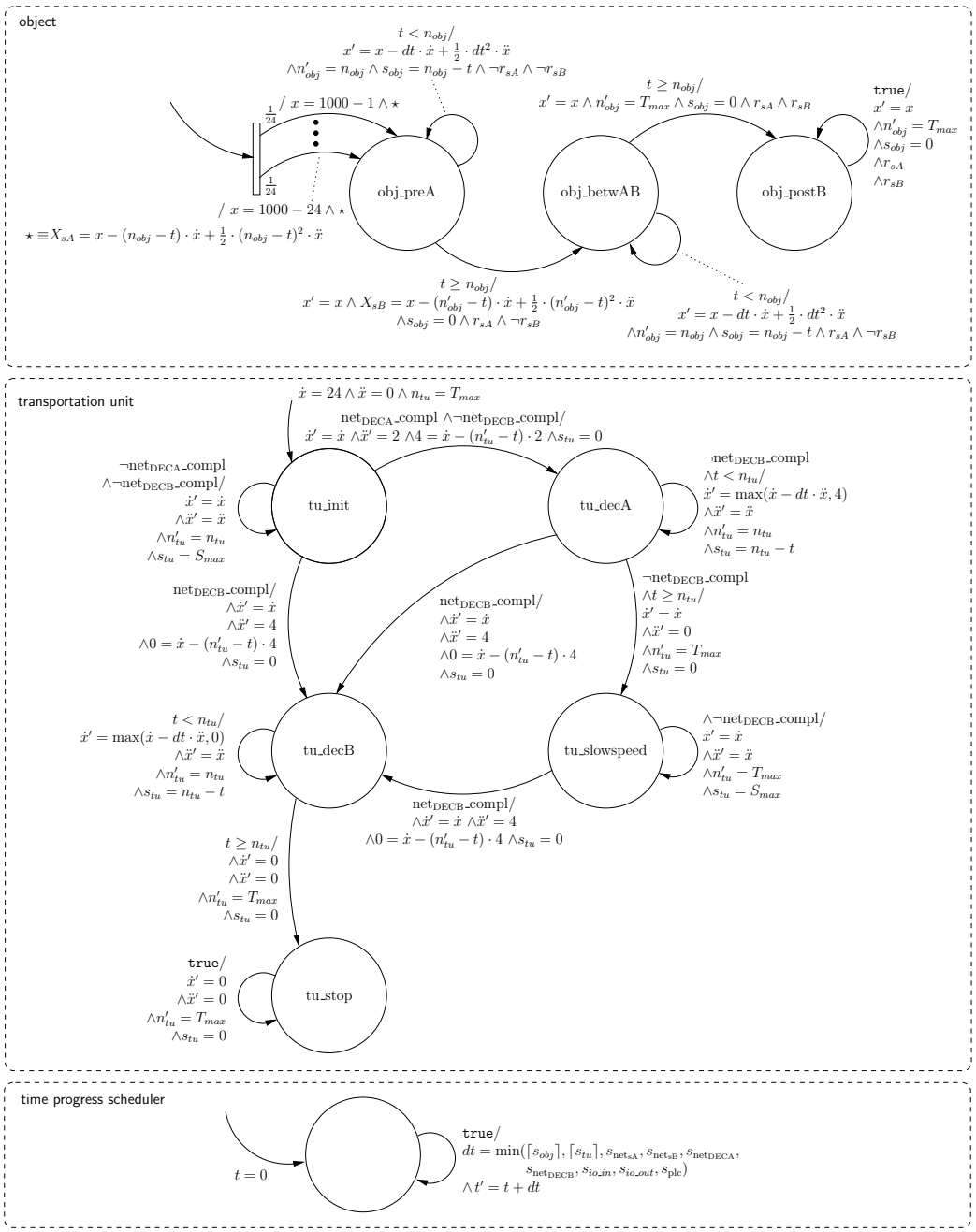


Figure 7: First part of the networked automation system. The first two automata in this figure represent the behavior of the object that is transported and the transportation unit which controls the speed of the belt. The third automaton selects the duration of the current time step dt and performs time progress.

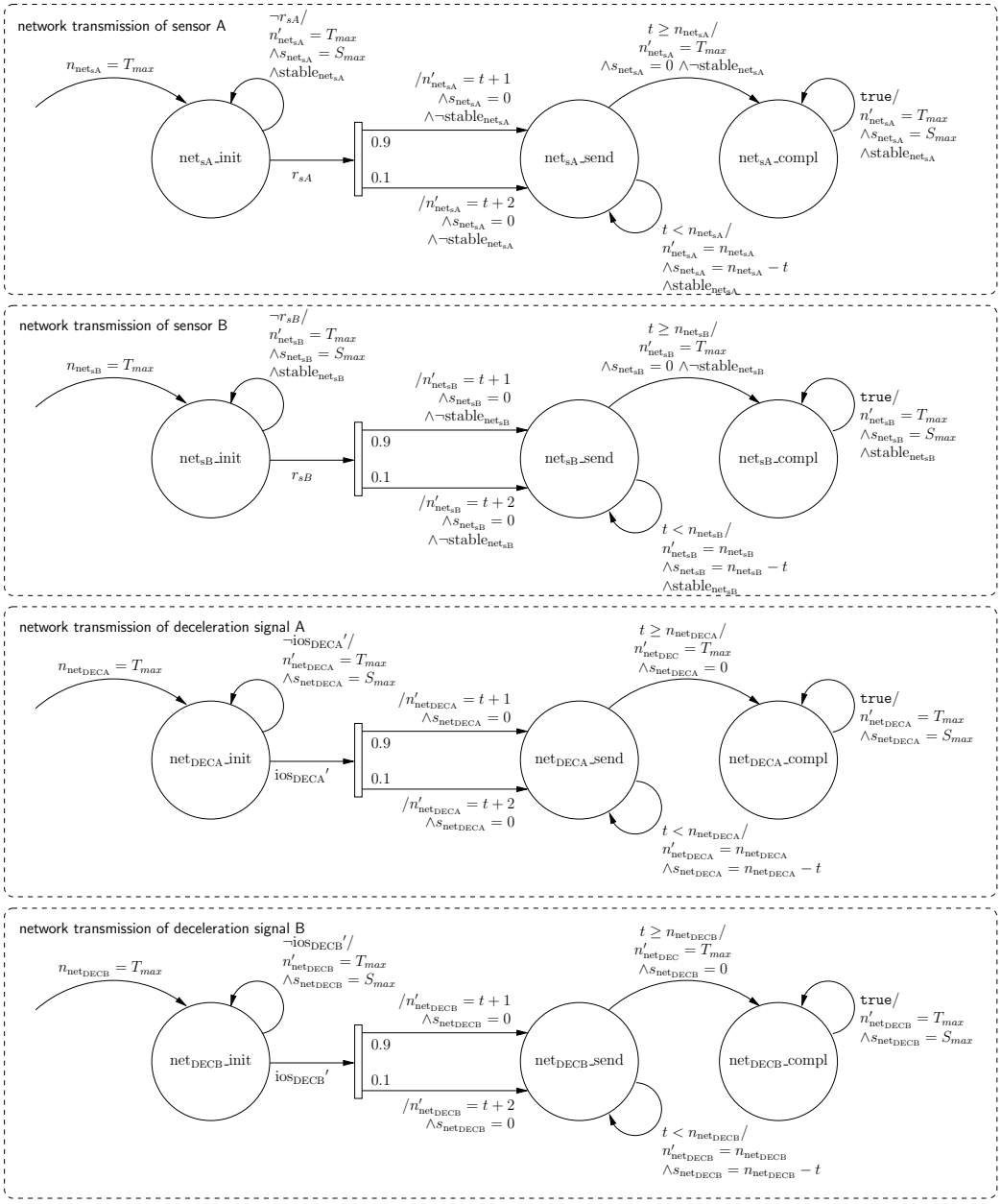


Figure 8: Second part of the NAS. The automata represent the network transmissions (first and second sensor and deceleration signals from the IO card to the transportation unit).

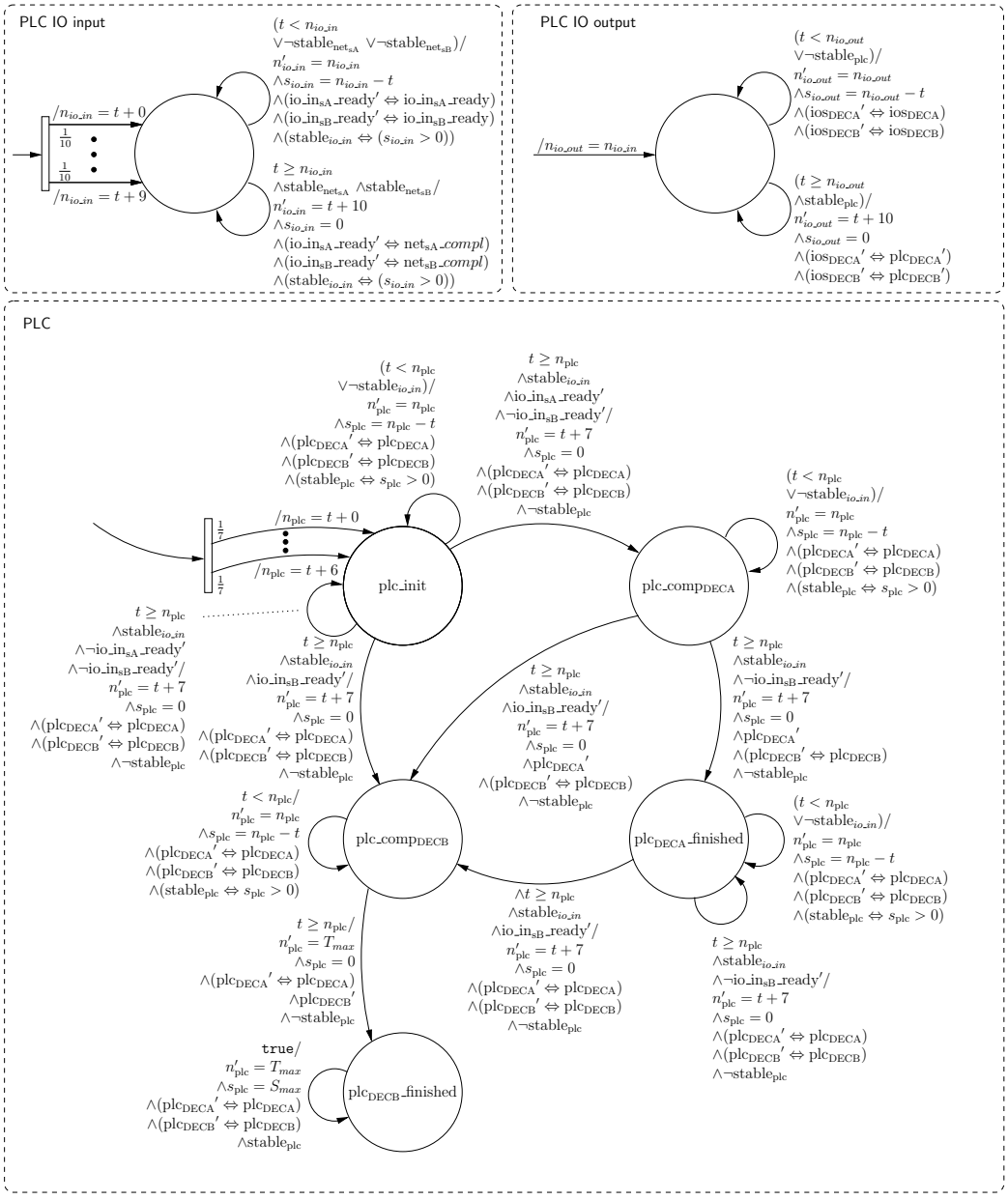


Figure 9: Third part of the NAS. The upper two automata represent the input-output parts of the PLC IO card. The lower part contains the automaton that models the behavior of the PLC which decides when to send deceleration signals depending on received messages from the sensors.

6.2.1. Explanation of the model

The global time variable t and the step duration variable dt are governed by the automaton time progress scheduler from Figure 7. The duration of a transition step of the overall system is given by the *minimum* value of the maximum step duration variables s of each automaton. The maximum step durations are local to the automata: For instance, s_{obj} is the maximum step duration variable of automaton object from Figure 7.

The automaton object models the workpiece to be transported on the conveyer belt to the drilling position at 0 lu. The automaton updates the position of the object, represented by variable x, depending on the step duration dt, the current velocity $\dot{x}$, and the current deceleration $\ddot{x}$. The velocity $\dot{x}$ and the deceleration $\ddot{x}$ are just consumed by object as they are computed by automaton transportation unit. Furthermore, object locally determines its maximum step duration s_{obj} as well as the time point n_{obj} of its next event. Initially, e.g., the next event of object is reaching sensor SA. Thus, n_{obj} is the non-negative value satisfying the equation $X_{sA} = x-(n_{obj}-t)\cdot\dot{x}+\frac{1}{2}\cdot(n_{obj}-t)^2\cdot\ddot{x}$, where X_{sA} is the fixed position of sensor SA and all other variables are assigned by their initial values. The automaton consists of the three locations obj_preA, obj_betwAB, and obj_postB meaning that the workpiece has not yet reached sensor SA, is in between SA and SB, and passed sensor SB, respectively. Reaching sensors SA and SB are indicated by setting the corresponding Boolean variables r_{sA} and r_{sB} to true. These variables are used for the communication with the automata modeling the network. The initial location is obj_preA and the initial position is uniformly distributed over the values 999 to 976, the latter fact is modeled by 24 incoming probabilistic transition alternatives. The automaton object switches to location obj_betwAB in case the global time t becomes greater than the time of the next event n_{obj}, i.e. sensor SA was reached. A similar transition to final location obj_postB is taken for passing sensor SB.

The automaton transportation unit is responsible for computing the current velocity $\dot{x}$. It furthermore sets the current deceleration $\ddot{x}$ by scanning the current locations of the automata network transmission of deceleration signal A and network transmission of deceleration signal B. Whenever, one of the latter automata has reached its location $\text{net}_{\text{DECA}}\text{\_compl}$ or $\text{net}_{\text{DECB}}\text{\_compl}$, it switches to the corresponding location tu_decA or tu_decB and setting the deceleration $\ddot{x}$ to 2 or $4\,\text{lu/ts}^2$, respectively. In case the signal to decelerate to $4\,\text{lu/ts}^2$ has not yet been transmitted by the network but the slow

speed of 4 lu/ts is reached, the automaton enters location tu_slowspeed and temporarily stops braking by setting the deceleration to zero. When switching to location tu_decB, in which braking is performed with $4\,\mathrm{lu/ts^2}$, the next event n'_{tu} is determined as the time the velocity $\dot{x}$ will become zero, i.e. the workpiece will be stopped. This can be computed by the equation $0 = \dot{x} - (n'_{tu} - t) \cdot 4$ for a fixed velocity $\dot{x}$ at time point t. Once the object comes to a standstill, detected by global time t is greater than or equal to the time of the standstill n_{tu}, transportation unit enters final location tu_stop.

The network of the system is modeled by four automata. The components network transmission of sensor A and network transmission of sensor B are responsible for the transmission of the sensor signals to the IO card of the programmable logic controller. To do so, they first sample the signals r_{sA} and r_{sB} that indicate reaching sensors SA and SB, respectively, by rising edges. If a rising edge occurs, the automata switch to their sending locations $\mathrm{net_{sA}}$_send and $\mathrm{net_{sB}}$_send. As specified in Section 6.1, the network routing time is determined stochastically, namely 1 ts with probability 0.9 and 2 ts with probability 0.1. This fact is modeled by two corresponding probabilistic transition alternatives which set the time of the next event $n'_{net_{sA}}$ (resp. $n'_{net_{sB}}$) to the current time t plus 1 or plus 2. If this next event is reached then the automata go to their final locations $\mathrm{net_{sA}}$_compl and $\mathrm{net_{sB}}$_compl. We remark here that the Boolean variables $\mathrm{stable_{net_{sA}}}$ and $\mathrm{stable_{net_{sB}}}$ indicate location switches in case they are set to false. Such location switches are of duration 0. (Observe that for a location switch the maximum step duration variable is set to 0.) Thus, several location switches may occur at the same physical time t. In order to not overlook any transmitted signal of the network, the model of the PLC IO card samples its inputs at time t not before both network automata have performed potential location switches at time t, i.e. not before both $\mathrm{stable_{net_{sA}}}$ and $\mathrm{stable_{net_{sB}}}$ are true. The automata network transmission of deceleration signal A and network transmission of deceleration signal B are responsible for forwarding the new decelerations to transportation unit, and work very similar to the previous ones. Reaching a final location immediately triggers a location switch in transportation unit as mentioned above.

The IO card of the PLC is divided into two components, where PLC IO input transmits the signals from the network to the PLC, and PLC IO output delivers the new deceleration values from the PLC to the network. Each automaton consists of only one location and two transitions, one of which is taken every 10 time steps when the corresponding signals are sampled.

Sampling points are detected by comparing the current time t with the time of the next event. The meaning of the Boolean variables $\text{stable}_{\text{net}_{\text{sA}}}$ and $\text{stable}_{\text{net}_{\text{sB}}}$ was discussed above. The same idea holds for variable $\text{stable}_{\text{plc}}$ that indicates a location switch in PLC. Two important parameters of the overall system are the initial phase shifts of the cycles of the PLC and PLC IO. A pragmatic, yet idealized, way to handle these phase shifts, e.g., is to synchronize the initial cycles of both the PLC and PLC IO. However, we will see in Section 6.4 that these phase shifts have a fundamental and non-negligible impact on the overall system behavior. To model each possible situation of the initial phase shifts, we randomly reduce the first cycle times of the PLC and PLC IO, where each random choice has equal probability. This yields seven possible initial cycle times for the PLC and ten for the PLC IO, respectively, which is modeled by initial probabilistic transition alternatives (cf. Figure 9).

Finally, the automaton PLC computes the decelerations of the object depending on the signals from sensors SA and SB. Initially, PLC resides in location plc_init and polls every 7 time steps for new inputs. A new input is detected by accessing the Boolean variables io_in_{sA}_ready' and io_in_{sB}_ready' that are set to true by the PLC IO input if signals from SA and SB were sampled, respectively. In case only io_in_{sA}_ready' is true the PLC enters location $\text{plc\_comp}_{\text{DECA}}$ for computing the deceleration for sensor SA. When the computation is finished after 7 time steps, this is indicated by leaving the location and setting the Boolean variable $\text{plc}'_{\text{DECA}}$ to true. If variable io_in_{sB}_ready' is not yet true then location plc_{DECA}_finished is visited in order to wait for this signal. If the PLC is in location plc_init, $\text{plc\_comp}_{\text{DECA}}$, or plc_{DECA}_finished and io_in_{sB}_ready' is true, the automaton goes directly to location $\text{plc\_comp}_{\text{DECB}}$ to compute the deceleration for sensor SB. That is, in case both signals from SA and SB arrive at the same time, the signal from SB is prioritized. After deceleration for SB was calculated the PLC enters its final location plc_{DECB}_finished.

6.2.2. Sample system run of the model

We exemplify the behavior of the model by means of a sample system run depicted in Figure 10. Each automaton is in its initial location, and the initial object position was probabilistically determined to be 976. After 12 ts, the object reaches sensor SA causing automaton object to enter location obj_betwAB. By this discrete state change the Boolean variable r_{sA} (meaning that SA is reached) is set to `true`, which triggers a synchronization

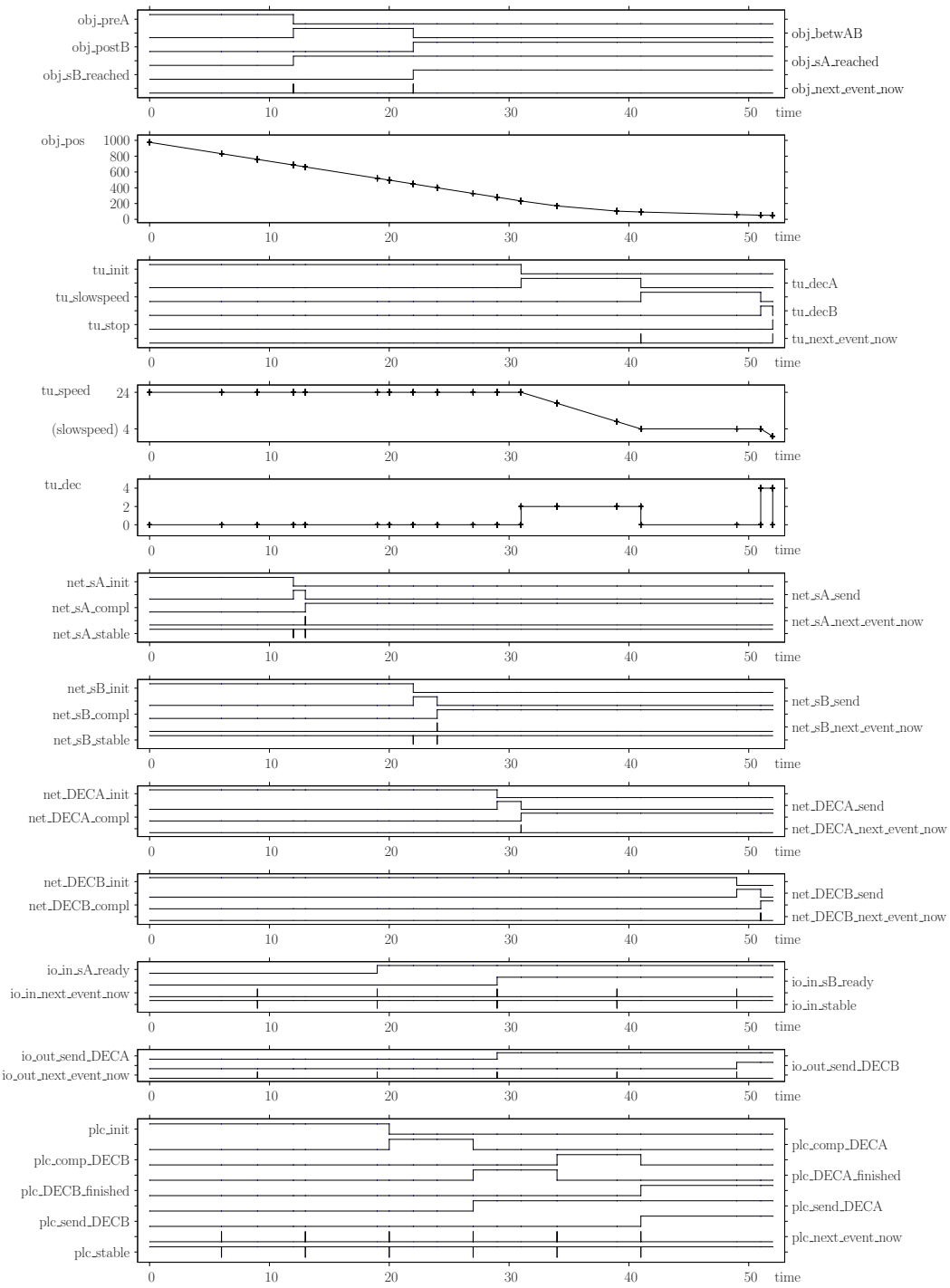


Figure 10: Sample run of the system model.

with automaton network transmission of sensor A. The latter proceeds to location net_{sA}_send, thereby probabilistically setting the network routing time to 1 ts. Thus, after 1 ts the network transmission of sensor A leaves the sending location and enters net_{sA}_compl indicating that the signal was successfully transmitted to the PLC-IO card at time point 13. The next cycle time of PLC IO input is at time point 19, i.e. at $t = 19$ the signal from sensor SA is provided for the PLC. The PLC processes this signal in its cycle from 20 to 27, and PLC IO output starts to send the new deceleration of $2\,\text{lu/ts}^2$ at time point 29 over the network. The network routing time for sending this packet, i.e. the time automaton network transmission of deceleration signal A resides in location net_{DECA}_send, is probabilistically determined to be 2 ts. Therefore, at time point 31 the transportation unit enters location tu_decA and sets the value of the deceleration to $2\,\text{lu/ts}^2$. The speed of the object is now decreasing accordingly until the slow speed of 4 lu/ts is reached at $t = 41$. The transportation unit then enters location tu_slowspeed to keep the speed constant, i.e. the deceleration is set to 0. It is kept at this value until sensor B provides a signal, resulting in setting the deceleration to the respective value of $4\,\text{lu/ts}^2$ at time point 51. Consequently, the object comes to a standstill at time point 52 with its final position 50 lu.

6.3. SSMT encoding of the case study

To enable the automatic analysis with the probabilistic bounded model checker SiSAT, we need to encode the NAS model presented in Section 6.2 into the SiSAT modeling language stated in Section 5.1. To do so, we employ the translation scheme introduced in Section 4. As an example of the result of this translation, the SSMT encoding of automaton network transmission of sensor A is depicted in Figure 11.

We would like to remark here again that this SSMT encoding facilitates a linearly sized system description. An explicit construction of the product automaton for this NAS model would result in more than 24 million discrete states, since the discrete state space is spanned by 6075 locations and 12 Boolean variables (i.e. $2^{12} = 4096$ valuations) used for synchronization.

6.4. Analysis of the case study

As mentioned in Section 6.1, the aim of the NAS application is that the workpiece stops very close to the drilling position. To quantitatively assess the analysis goal, we proceed in two phases, where all experiments were

```
DECL    -- Declaration of observable system
        -- variables.
    -- Global system time.
    float [0, MAX_TIME] time;
    ...
    -- Network sA.
    -- Locations.
    boole net_sA_init;
    boole net_sA_send;
    boole net_sA_compl;
    -- Variable to fix next event.
    float [0, MAX_TIME] net_sA_next_event;
    -- Variable to set duration of transition step.
    float [-1, MAX_TIME_STEP] net_sA_step;
    -- Indicating whether an event happens now.
    boole net_sA_next_event_now;
    -- Indicating that network is in stable state.
    boole net_sA_stable;
    ...
INIT    -- Initial condition block.
    -- Time variable initialized to 0.
    time = 0;
    ...
    -- Network sA.
    -- Initial location is net_sA_init.
     net_sA_init;
    !net_sA_send;
    !net_sA_compl;
    -- Next event is not known so far.
    net_sA_next_event = MAX_TIME;
    ...
DISTR   -- Declaration of variables bound by
        -- quantifiers.
    ...
    -- Randomized quantification of a variable,
    -- encoding network routing time.
    R. net_sA_delay p = [
                            1 -> 0.9,
                            2 -> 0.1
                        ]:
    ...
TRANS   -- Transition relation (pre-post states).
    ...
    -- Network sA.
    -- Exactly in one location after next
    -- transition.
    net_sA_init' + net_sA_send' +
                            net_sA_compl' = 1;
    -- Do we have an event now?
    net_sA_next_event_now <->
              (time >= net_sA_next_event);
    -- Network is stable if its step is
    -- not 0, i.e. no location change.
    net_sA_stable <-> (net_sA_step > 0);
    -- Remain in net_sA_init.
    (    net_sA_init
     and !obj_sA_reached)
    -> (    net_sA_init'
        and net_sA_next_event' = MAX_TIME
        and net_sA_step = MAX_TIME_STEP);

    -- Go to net_sA_send.
    (    net_sA_init
     and obj_sA_reached)
    -> (    net_sA_send'
        and net_sA_next_event' = time +
                                 net_sA_delay
        and net_sA_step = 0);

    -- Remain in net_sA_send.
    (    net_sA_send
     and !net_sA_next_event_now)
    -> (    net_sA_send'
        and net_sA_next_event' =
                        net_sA_next_event
        and net_sA_step =
                 net_sA_next_event - time);

    -- Go to net_sA_compl.
    (    net_sA_send
     and net_sA_next_event_now)
    -> (    net_sA_compl'
        and net_sA_next_event' = MAX_TIME
        and net_sA_step = 0);

    -- Remain in net_sA_compl.
    net_sA_compl
    -> (    net_sA_compl'
        and net_sA_next_event' = MAX_TIME
        and net_sA_step = MAX_TIME_STEP);
    ...
TARGET  -- Target states.
    ...
```

Figure 11: The automaton network transmission of sensor A described in the SiSAT modeling language.

performed on a 2.4 GHz AMD Opteron machine with 128 GByte physical memory running Linux.

In the *first phase*, we determine the PBRC unwinding depth k such that the workpiece has stopped in all system runs of length k. The workpiece stops in run r if and only if the run r reaches location tu_stop in the automaton transportation unit. Taking the system description in the SiSAT modeling language from Section 6.3 with `tu_stop` as the target states, we call the SiSAT tool in order to successively solve the corresponding probabilistic bounded reachability formulae $PBRC_{\mathcal{S},Target}(0), PBRC_{\mathcal{S},Target}(1), \ldots, PBRC_{\mathcal{S},Target}(k)$ until the PBMC problem $Pr(PBRC_{\mathcal{S},Target}(k)) = 1$ is decided to be true, where the latter fact means that *all* system runs (of length k) have reached location tu_stop, i.e. the object has been stopped on all paths. SiSAT found out that the desired property holds for unwinding depth $k = 44$ with a total runtime of 134 min.

Then, in the *second phase* we are able to compute the probability of stopping within some target region. This can be done by using the `Target` formula `L >= obj_pos and obj_pos >= R` with constants `L` and `R` defining the region. Assume that the target region of interest is given by `L` $= 100$ and `R` $=$ 0 (i.e. stopping with 100 lu and 0 lu). Calling SiSAT on the $PBRC_{\mathcal{S},Target}(44)$ formula with the corresponding target states, the tool computes the hit probability ≈ 0.397345 for this region within 71 min. Oftentimes, an engineer is just interested in whether the probability is below or above some given threshold θ. In such cases, the computation problem can be restated as a PBMC decision problem, and runtimes improve due to pruning by *thresholding* (cf. Section 5.2): The resulting computation times are 11 min CPU time for deciding wrt. threshold $\theta = 0.05$, 20 min for $\theta = 0.1$, 13 min for $\theta = 0.9$, and 11 min for $\theta = 0.95$.

As mentioned in Section 6.2, the initial phase shifts of the cycles of the PLC and PLC IO automata play an important role for the distribution of the final object position. Although this is not the intended use case of SiSAT, we are able to calculate the distribution of the final position by solving a set of formulae $PBRC_{\mathcal{S},Target}(44)$ with target states `L > obj_pos and obj_pos >= L-1;` for `L` $= 300, \ldots, -300$. For each such formula we get the probability of stopping within a target region of length 1, from which we can obtain the distribution. In Figure 12, four possible scenarios with fixed initial PLC and PLC IO phase shifts are depicted. From the case where the first cycles of the PLC and of the PLC IO are at time point 2, one may wrongly conclude that the model works as desired since the workpiece always stops in a very

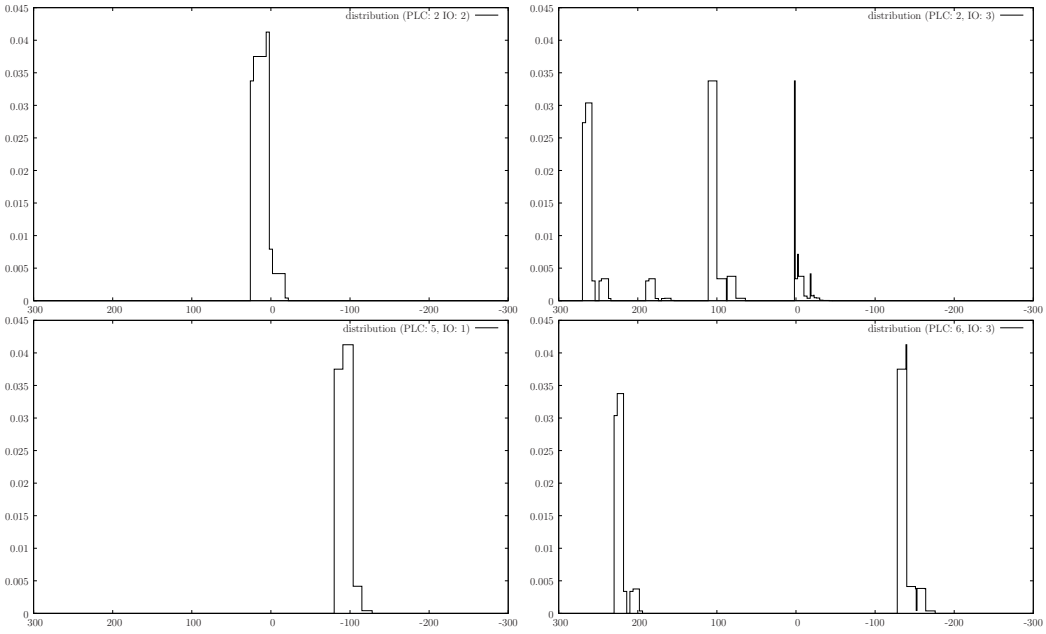


Figure 12: Different distributions of the final position for fixed initial phase shifts of the cycles of the PLC and PLC IO automata. The final object position is given on the x-axis and the probability of reaching a position within a 1 length unit strip from that position is given on the y-axis. The x-axis is reversed to reflect the movement of the object from positive to negative values of obj_pos.

small region around the drilling position. However, if the PLC and PLC IO phase shifts are 6 and 3, respectively, there is a large region around position 0 in which the object does never halt. Thus, to get an authentic picture about the distribution of the final position, one has to incorporate the initial phase shifts in a realistic way. Since there are no semantic conditions on the PLC and its IO card, like periodic resets or other synchronizations, we realistically assume that any combination of the phase shifts may arise when the object enters. In the model, the possible initial phase shifts are therefore uniformly distributed, as explained in Section 6.2. For this more authentic NAS model, we have obtained the distribution of the final object position shown in Figure 13 and roughly resembling like a normal distribution.

The CPU times for computing the distributions are 342 min for phase shifts 2 (PLC) and 2 (PLC IO), 449 min for 2 (PLC) and 3 (PLC IO), 303 min for 5 (PLC) and 1 (PLC IO), and 412 min for 6 (PLC) and 3 (PLC IO). For the distribution where the initial phase shifts are uniformly distributed,

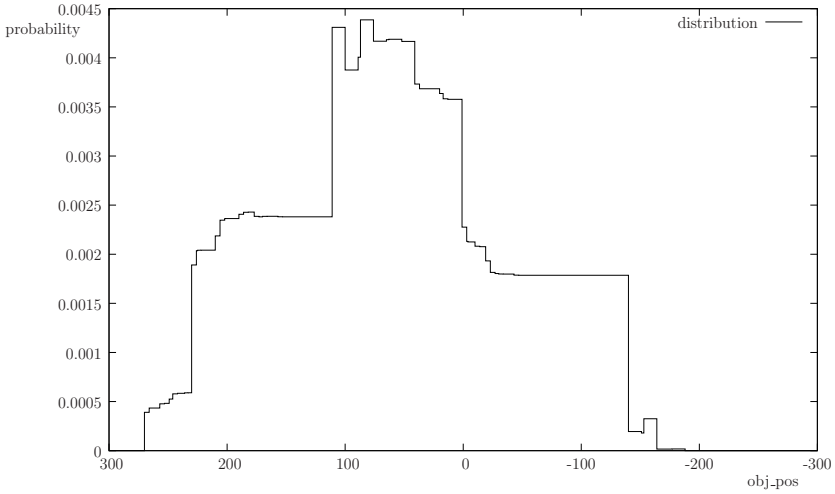


Figure 13: Distribution of the final position: probability for obj_pos to reach a position (stripes of 1 lu width were set as target and the probability of satisfaction to reach that target calculated).

the runtime is much higher due to the enormous growth of the probabilistic system behavior: the computation needs almost 32 days[7].

As mentioned above, computing such distributions is (so far) not the intended use case of SiSAT. Rather, SiSAT is designed for efficiently answering queries of the form "does the workpiece stop in a given vicinity of the target with a probability exceeding a given target value?" When deciding such decision problems, the pruning rules apply, giving tremendous speed-ups.

6.5. Empirical results of solution caching

In Section 5.3, we proposed the PBRC/PBMC-related enhancement of *solution caching*, which memorizes solutions in a tree-like data-structure when solving the formulae $PBRC_{\mathcal{S},Target}(k)$ in order to reuse these solutions when solving larger formulae $PBRC_{\mathcal{S},Target}(k+k')$. We empirically evaluate this algorithmic optimization on the NAS model from the first phase in Section 6.4, i.e. with reachability target location tu_stop, for depths 0 to 70. We investigate four settings of the solver: 1) default, 2) with solution caching, 3) with a threshold of 1, 4) with both solution caching and a threshold of 1. The

[7]Due to the possibility of running multiple SiSAT calls in parallel, the distribution was actually calculated in roughly a day on two 16 core machines.

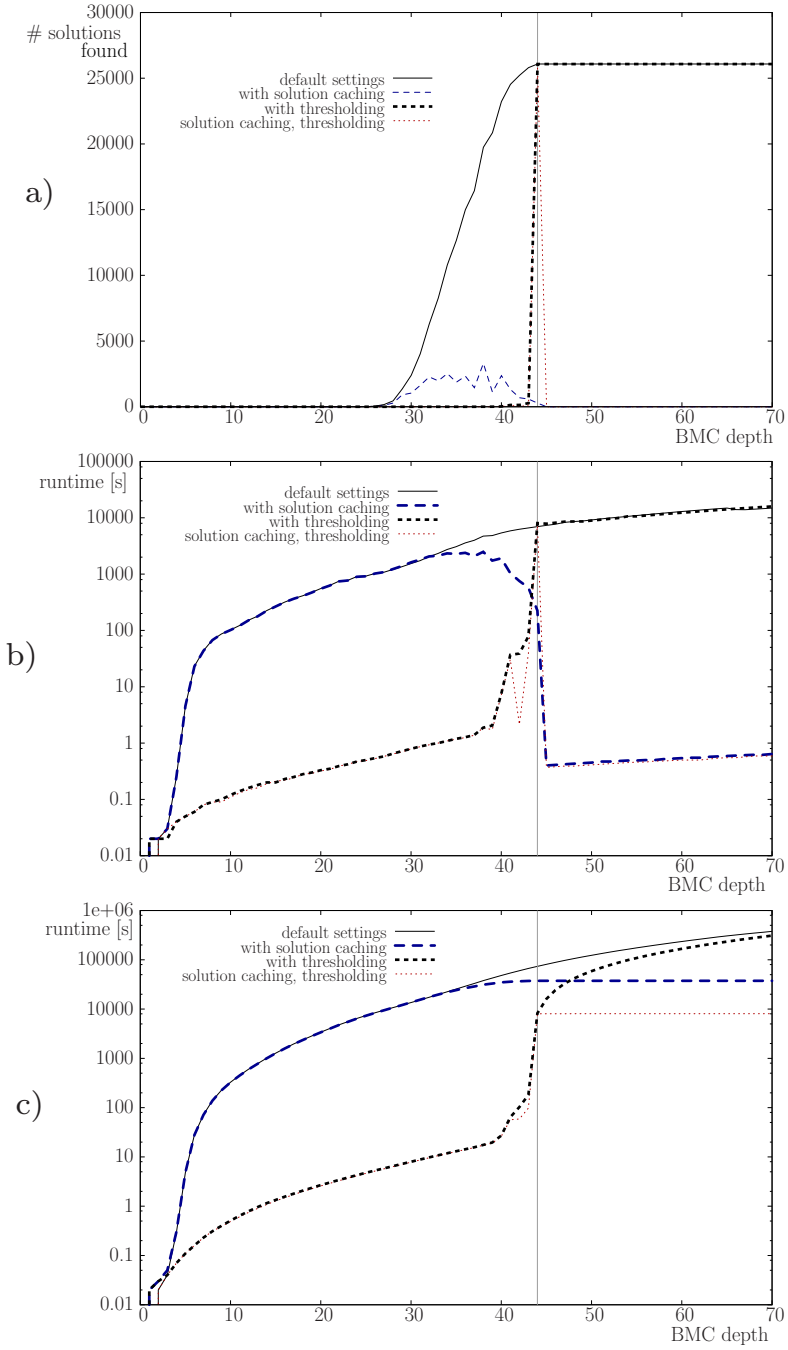


Figure 14: a) Number of solutions found in each BMC unwinding depth. b) Runtimes for solving each unwinding depth. c) Cumulative runtimes for solving the unwinding depths up to the current depth. All graphs refer to the target of reaching tu_stop which is satisfied for all system runs in BMC depth 44 (and thus also in all subsequent unwindings).

results for all these settings are shown in Figure 14. Recall that at unwinding depth 44 all system runs reach tu_stop, resulting in probability 1 for the PBRC problem of this (and larger) depth(s).

The first observation is that solving with thresholding enabled clearly outperforms its counterparts on the first 40 unwindings. This is due to the fact that upon identifying some system runs not reaching the target states, *thresholding* can start to prune quantifiers. This fact is documented by the number of solutions found in Figure 14 a). However, from unwinding depth 40 onward, the runtimes of the thresholding-enabled solvers grow heavily since more and more runs reach the target states.

The second and most important observation proves the benefit of solution caching. The additional overhead of maintaining the data-structure for caching solutions does not result in runtime penalty: both solver settings with solution caching never show a worse performance than the corresponding settings without solution caching. In particular, this also holds for the first unwindings for which no solutions were found. In total, the solution caching enabled solver was two times faster than the default setting, on unwinding depth 44, even a speed-up factor of 31 was obtained. As shown in Figure 14 b), each PBRC formula of depth greater 44 is solved by the solution caching solvers in fractions of a second. The reason is that by memorization from the run of depth 44, SiSAT is enabled to immediately deduce satisfaction probability 1 for the whole formula without probing any variable assignment (cf. Figure 14 a)).

Summarizing the findings above, we may conclude that solution caching is a powerful mechanism for accelerating the proof search of the SiSAT tool for probabilistic bounded reachability problems. An important advantage is that solution caching applies always and does not depend on user-specified parameters like thresholds.

7. Conclusion

In this paper, we proposed a symbolic technique for the bounded reachability analysis of concurrent probabilistic hybrid systems using the probabilistic bounded model checker SiSAT. We established a linearly sized predicative encoding of such concurrent probabilistic systems, alleviating the state explosion arising from explicit construction of the product automaton, and thus enhancing the scalability of the automated analysis procedure. We proved the

concept of this approach by an application to a realistic networked automation system. We furthermore introduced a novel algorithmic enhancement to SSMT solving in the context of probabilistic bounded reachability analysis, namely solution caching. The idea of this optimization is to store solutions when solving probabilistic bounded reachability formulae and then to reuse that information for formulae of greater unwinding depths. We empirically showed that solution caching can yield significant performance gains.

In future work, we will follow diverse research directions. One essential item is to further improve performance of the SiSAT tool in order to efficiently handle large-scale real-world applications. As for solution caching, a promissing issue is to exploit system-dependent properties to speed-up the proof search of the SSMT solver. Another idea is suggested by trends in hardware design towards multi-core and multiprocessor systems, namely the development of parallelized SSMT algorithms. In recent work in the related area of parallel solving of quantified Boolean formula (QBF) problems[8], sometimes super-linear speed-ups were obtained due to knowledge sharing between the parallel solving processes [32]. In parallel SSMT solving, various forms of knowledge sharing based on the algorithmic enhancements from Sections 5.2 and 5.3 are conceivable.

Within the analysis of the NAS case study we applied a naive approach to calculate distributions of the final object position. As already mentioned, this is so far not the intended use case of SiSAT. However, enhancing the SiSAT tool to more efficiently generate such distributions without solving a large set of single SSMT formulae would be a nice feature, in particular if non-determinism is present. Equally relevant for a convenient analysis process is the generation of optimal control strategies to reach the desired goal states, or dually the generation of counter-examples that lead to fatal system errors. In the probabilistic setting such strategies or counter-examples are in general not just assignments but assignment trees. Actually, SiSAT already constructs such assignment trees implicitly when traversing the quantifier tree. The main challenge here is to develop suitable data-structures that are capable of handling and manipulating such assignment trees of potentially huge size.

Another research direction goes beyond computing pure reachability probabilities. Since several industrial applications ask for more expressive quanti-

[8]QBF allows existential and universal quantification.

tative measures like expected values, we will focus our future work on various expected values, among them mean-times to failure in probabilistic hybrid systems.

Acknowledgements. We would like to express our gratitude to our colleagues in the Transregional Research Center "AVACS" (Automatic Verification and Analysis of Complex Systems) for many valuable discussions on the various topics presented herein and for jointly developing and implementing the iSAT algorithm. Additionally, we would like to thank Jürgen Greifeneder and Georg Frey for supplying detailed information on the original model of the case study and the anonymous reviewers for their advice on how to enhance readability of the article.

References

[1] J. F. Groote, J. W. C. Koorn, S. F. M. van Vlijmen, The Safety Guaranteeing System at Station Hoorn-Kersenboogerd, in: Conference on Computer Assurance, National Institute of Standards and Technology, 1995, pp. 57–68.

[2] A. Biere, A. Cimatti, E. M. Clarke, Y. Zhu, Symbolic model checking without BDDs, in: R. Cleaveland (Ed.), TACAS, Vol. 1579 of Lecture Notes in Computer Science, Springer, 1999, pp. 193–207.

[3] M. Bozzano, R. Bruttomesso, A. Cimatti, T. A. Junttila, P. van Rossum, S. Schulz, R. Sebastiani, The MathSAT 3 system, in: Conf. on Automated Deduction, Vol. 3632 of Lecture Notes in Computer Science, Springer Verlag, 2005, pp. 315–321.

[4] B. Dutertre, L. de Moura, A Fast Linear-Arithmetic Solver for DPLL(T), in: Proceedings of the 18th Computer-Aided Verification Conference, Vol. 4144 of Lecture Notes in Computer Science, Springer-Verlag, 2006, pp. 81–94.

[5] A. Bauer, M. Pister, M. Tautschnig, Tool-support for the analysis of hybrid systems and models, in: DATE '07: Proceedings of the Conference on Design, Automation and Test in Europe, EDA Consortium, San Jose, CA, USA, 2007, pp. 924–929.

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

[7] G. Audemard, M. Bozzano, A. Cimatti, R. Sebastiani, Verifying industrial hybrid systems with mathsat, Electr. Notes Theor. Comput. Sci. 119 (2) (2005) 17–32.

[8] M. Fränzle, C. Herde, HySAT: An efficient proof engine for bounded model checking of hybrid systems, Formal Methods in System Design 30 (2007) 179–198.

[9] E. Ábrahám, B. Becker, F. Klaedke, M. Steffen, Optimizing bounded model checking for linear hybrid systems, in: Proceedings of VMCAI'05 (Verification, Model Checking, and Abstraction), Vol. 3385 of Lecture Notes in Computer Science, Springer-Verlag, Paris, 2005, pp. 396–412, an extended version of this paper appeared as ATR 4.

[10] E. Ábrahám, T. Schubert, B. Becker, M. Fränzle, C. Herde, Parallel SAT solving in bounded model checking, Journal of Logic and Computation, doi:10.1093/logcom/exp002.
URL `http://dx.doi.org/10.1093/logcom/exp002`

[11] A. Eggers, M. Fränzle, C. Herde, SAT modulo ODE: A direct SAT approach to hybrid systems, in: S. S. Cha, J.-Y. Choi, M. Kim, I. Lee, M. Viswanathan (Eds.), Proceedings of the 6th International Symposium on Automated Technology for Verification and Analysis (ATVA'08), Vol. 5311 of Lecture Notes in Computer Science, Springer, 2008, pp. 171–185.

[12] M. Fränzle, H. Hermanns, T. Teige, Stochastic Satisfiability Modulo Theory: A Novel Technique for the Analysis of Probabilistic Hybrid Systems, in: M. Egerstedt, B. Mishra (Eds.), Proceedings of the 11th International Conference on Hybrid Systems: Computation and Control (HSCC'08), Vol. 4981 of LNCS, Springer, 2008, pp. 172–186.

[13] T. Teige, M. Fränzle, Stochastic Satisfiability modulo Theories for Non-linear Arithmetic, in: L. Perron, M. A. Trick (Eds.), Integration of AI and OR Techniques in Constraint Programming for Combinatorial

Optimization Problems, 5th International Conference, CPAIOR 2008, Vol. 5015 of LNCS, Springer, 2008, pp. 248–262.

[14] C. H. Papadimitriou, Games against nature, J. Comput. Syst. Sci. 31 (2) (1985) 288–301.

[15] M. L. Littman, S. M. Majercik, T. Pitassi, Stochastic Boolean Satisfiability, Journal of Automated Reasoning 27 (3) (2001) 251–296.

[16] S. Majercik, Nonchronological backtracking in stochastic Boolean satisfiability, in: Tools with Artificial Intelligence, 2004. ICTAI 2004. 16th IEEE International Conference on, IEEE Computer Society, 2004, pp. 498–507. doi:10.1109/ICTAI.2004.94.

[17] S. M. Majercik, APPSSAT: Approximate probabilistic planning using stochastic satisfiability, Int. J. Approx. Reasoning 45 (2) (2007) 402–419.

[18] M. Fränzle, T. Teige, A. Eggers, Engineering Constraint Solvers for Automatic Analysis of Probabilistic Hybrid Automata, The Journal of Logic and Algebraic Programming, to appear 2010.

[19] J. Sproston, Model checking of probabilistic timed and hybrid systems, Ph.D. thesis, University of Birmingham (2000).

[20] A. Bemporad, S. D. Cairano, Optimal control of discrete hybrid stochastic automata, in: Hybrid Systems: Computation and Control, Vol. 3414 of Lecture Notes in Computer Science, Springer, 2005, pp. 151–167.

[21] M. Davis, Markov Models and Optimization, Chapman & Hall, London, 1993.

[22] L. Arnold, Stochastic Differential Equations: Theory and Applications, Wiley - Interscience, 1974.

[23] J. Hu, J. Lygeros, S. Sastry, Towards a theory of stochastic hybrid systems, in: Hybrid Systems: Computation and Control, Vol. 1790 of Lecture Notes in Computer Science, Springer-Verlag, 2000, pp. 160–173.

[24] M. L. Bujorianu, J. Lygeros, Toward a general theory of stochastic hybrid systems, in: Stochastic Hybrid Systems: Theory and Safety Critical Applications, Vol. 337 of Lecture Notes in Control and Information Sciences, Springer-Verlag, 2006, pp. 3–30.

[25] C. Barrett, R. Sebastiani, S. A. Seshia, C. Tinelli, Satisfiability modulo theories, in: Biere et al. [27], Ch. 26, pp. 825–885.

[26] C. A. R. Hoare, Communicating Sequential Processes, Series in Computer Science, Prentice-Hall Intl., 1985.

[27] A. Biere, M. J. H. Heule, H. van Maaren, T. Walsh (Eds.), Handbook of Satisfiability, Vol. 185 of Frontiers in Artificial Intelligence and Applications, IOS Press, 2009.

[28] F. Benhamou, L. Granvilliers, Continuous and interval constraints, in: F. Rossi, P. van Beek, T. Walsh (Eds.), Handbook of Constraint Programming, Foundations of Artificial Intelligence, Elsevier, Amsterdam, 2006, Ch. 16, pp. 571–603.

[29] S. M. Majercik, Stochastic Boolean satisfiability, in: Biere et al. [27], Ch. 27, pp. 887–925.

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

[31] J. Greifeneder, G. Frey, Probabilistic hybrid automata with variable step width applied to the analysis of networked automation systems, in: Proc. 3rd IFAC Workshop on Discrete Event System Design (DESDes'06), IFAC, 2006, pp. 283–288.

[32] M. D. T. Lewis, P. Marin, T. Schubert, M. Narizzano, B. Becker, E. Giunchiglia, PaQuBE: Distributed QBF solving with advanced knowledge sharing, in: O. Kullmann (Ed.), SAT, Vol. 5584 of Lecture Notes in Computer Science, Springer, 2009, pp. 509–523.