Safety Verification for Probabilistic Hybrid Systems

Lijun Zhang
DTU Informatics, Technical University of Denmark, Denmark
zhang@imm.dtu.dk

Zhikun She
LMIB and School of Mathematics and Systems Science, Beihang University, China
zhikun.she@buaa.edu.cn

Stefan Ratschan
Institute of Computer Science, Czech Academy of Sciences, Czech Republic
stefan.ratschan@cs.cas.cz

Holger Hermanns
Department of Computer Science, Saarland University, Germany
hermanns@cs.uni-saarland.de

Ernst Moritz Hahn*
Department of Computer Science, Saarland University, Germany
emh@cs.uni-saarland.de

December 3, 2013

*Corresponding author

Abstract

The interplay of random phenomena and continuous dynamics deserves increased attention, especially in the context of wireless sensing and control applications. Safety verification for such systems thus needs to consider probabilistic variants of systems with hybrid dynamics. In safety verification of classical hybrid systems, we are interested in whether a certain set of unsafe system states can be reached from a set of initial states. In the probabilistic setting, we may ask instead whether the probability of reaching unsafe states is below some given threshold. In this paper, we consider probabilistic hybrid systems and develop a general abstraction technique for verifying probabilistic safety problems. This gives rise to the first mechanisable technique that can, in practice, formally verify safety properties of non-trivial continuous-time stochastic hybrid systems. Moreover, being based on abstractions computed by tools for the analysis of non-probabilistic hybrid systems, improvements in effectivity of such tools directly carry over to improvements in effectivity of the technique we describe. We demonstrate the applicability of our approach on a number of case studies, tackled using a prototypical implementation.

1 Introduction

Hybrid systems constitute a general and widely applicable class of dynamical systems with both discrete and continuous components. Conventional hybrid system formalisms [3, 27, 13, 30] capture many characteristics of real systems. However, in many modern application areas, also *probabilistic dynamics* occur. This is especially true for wireless sensing and control applications, where message loss probabilities and other random effects (node placement, node failure, battery drain) turn the overall control problem into a problem that can only be managed with a certain, hopefully sufficiently large, probability: Due to the influence of these random effects, the success (for example keeping the hybrid system in a safe region) can only be guaranteed with a probability less than one.

The need to integrate probabilities into hybrid systems formalisms has led to several different notions of *stochastic hybrid systems*, each from a distinct perspective [2, 32, 9, 11, 1]. The most important distinction lies in the point of attack where to introduce randomness. One option is to replace deterministic jumps by probability distributions over deterministic jumps. Another option is to generalise the differential equation components inside a mode by a stochastic differential equations component. More general models can be obtained by blending the above two choices, and by combining them with memoryless timed probabilistic jumps [8], and with nondeterminism. One prominent example of such a blend is the model of piecewise-deterministic Markov processes [15], a deterministic hybrid system model augmented with memoryless timed probabilistic jumps.

An important problem in hybrid systems theory is that of *reachability analysis*. In general terms, a reachability analysis problem consists in evaluating whether a given system may reach certain unsafe states, starting from certain initial states. This problem is associated with the safety verification problem: to prove that the system can never reach any unsafe state. In the probabilistic setting, the safety verification problem can be formulated as that of checking whether the probability that the system trajectories reach an unsafe state from an initial state can be bounded by some given probability threshold.

In this paper, we focus on the model of probabilistic hybrid automata [32], an extension of hybrid automata where jumps involve probability distributions. This adds the possibility to represent model-component failures, message losses, buffer overflows and the like. Since these phenomena are important aspects when aiming at faithful models for networked and embedded applications, the interest in this formalism is growing [16, 33]. We are striving for computational tools to support the analysis of the models of Sproston [32] and more complicated models, based on mathematically sound foundations.

Up to now, foundational results on the probabilistic reachability problem for probabilistic hybrid automata are scarce. Since they form a strict superclass of hybrid automata, this

is not surprising. Decidability results are available for probabilistic linear hybrid automata and probabilistic o-minimal hybrid automata [32].

This paper explains how we can harvest and combine recent advances in the hybrid automata and the probabilistic automata worlds, in order to treat the general case. We are doing so by computing safe overapproximations via abstractions in the continuous as well as the probabilistic domain. One of the core challenges then is how to construct a sound probabilistic abstraction over a given covering (i.e., set of abstract states, each of which is formed by a subset of the concrete state space, such that their union equals the complete concrete state space) of the state space. For this purpose, we first consider the non-probabilistic hybrid automaton obtained by replacing probabilistic branching with nondeterministic choices. Provided that we have obtained a finite abstraction for this classical hybrid automaton, we then decorate it with probabilities to obtain a probabilistic abstraction, namely a finite probabilistic automaton [31]. We show the soundness of the approach, which allows us to verify probabilistic safety properties on the abstraction: if such a property holds in the abstraction, it holds also in the concrete system. Otherwise, refinement is required to obtain a more precise result.

Our abstraction approach can be considered as an orthogonal combination of definitions of abstraction for hybrid automata [4, 30], and for Markov decision processes [14, 19]. Because of this orthogonality, abstractions of probabilistic hybrid automata can be computed via abstractions for non-probabilistic hybrid automata and Markov decision processes. To show the applicability of this combination, we implemented a prototype tool, `ProHVer`, that builds an abstraction via a combination of existing techniques for classical hybrid automata [17] as well as methods for Markov decision processes [14, 25, 19]. Subsequently, a fixed-point engine computes the reachability probabilities on the abstraction, which provides a safe upper bound on the reachability probability in the model semantics. If needed, iterative refinement of the hybrid abstraction is performed. We report several successful applications of this prototypical implementation on different case studies. To the best of our knowledge, this is the first implementation that automatically checks safety properties for probabilistic hybrid automata.

The framework considered here has the advantage of orthogonality: if a non-probabilistic abstraction is used that differs from the one we employ in this paper, then this abstraction can be extended with probabilities in a very similar fashion. Furthermore, future computational advances in hybrid automata analysis can directly be harvested for the model of probabilistic hybrid automata.

Organisation of the paper. We first discuss related work in Section 2, and then recall the definitions we use in Section 3. In Section 4, we introduce the notion of abstractions for probabilistic hybrid automata, and discuss how to compute the abstractions for safety verification problems. We illustrate our approach by applying it on a small model in Section 5. In Section 6, we describe an implementation of our algorithm and apply it on several case studies. Section 7 concludes the paper.

2 Related Work

The model considered in this paper extends (nondeterministic) hybrid automata with probabilistic discrete jumps. Davis [15] introduced piecewise-deterministic Markov processes, whose state changes are triggered spontaneously as in continuous-time Markov chains. Apart from forced and spontaneous probabilistic jumps, general stochastic hybrid systems [20, 10] comprise stochastic differential equations [5]. They can incorporate random perturbations, such as Brownian motion, into the continuous dynamics. While these models—with and without nondeterminism—enjoy a very rich variety of applications, their analysis is limited and often based on Monte-Carlo simulations [11, 22, 1, 12] of systems

without nondeterminism. Another approximate analysis technique, restricted to systems without nondeterminism, relies on the use of stochastic simulation functions [23].

From the hybrid automata perspective, the general verification problem for safety properties is known to be undecidable. Certain classes (e.g., initialised rectangular automata [18], o-minimal hybrid automata [26]) are decidable, and there are algorithms that construct finite bisimulation quotients for them. These methods have been lifted to probabilistic hybrid automata by Sproston [32], and this can be used to compute exact results, rooted in a bisimulation-based abstraction. In these special cases, we can parametrise our principal technique in such a way that it yields the very same exact results. In practice however, our method gives us the liberty to use an abstraction that overapproximates the behaviour, and is tailored to the problem at hand. The computational results will then not be exact, but overapproximating. This relies on the use of nondeterminism as a powerful abstraction mechanism, and extends to undecidable classes as well. Indeed, we treat the general case using the fact that a practical verification can—to a certain extent—circumvent the decidability barrier by a heuristic algorithm: we exploit tools which can, in practice, verify hybrid automata belonging to undecidable classes, to verify corresponding probabilistic hybrid automata (cf. also the notion of quasi-decidability [29]). While Sproston focuses on decidability results for particular classes of probabilistic hybrid models, this paper considers safety verification for models in full generality. The paper is based on an earlier conference paper [34].

Abstraction approaches have also successfully been applied to probabilistic timed automata [25, 24], a class of probabilistic hybrid automata, where only derivatives of constant value 1 occur. Their abstract analysis is based on difference-bound matrices (DBMs), and does not extend to the general setting considered here. Fränzle et al. [16, 33] use stochastic SAT to solve reachability problems on probabilistic hybrid automata. Their analysis is limited to depth-bounded reachability properties, i.e., the probability of reaching a location within at most N discrete jumps.

3 Preliminaries

In this section, we repeat the definition of conventional hybrid automata, in the style of [30], followed by the definition of probabilistic hybrid automata [32].

3.1 Hybrid Automata

3.1.1 Syntax

We fix a variable m ranging over a finite set of discrete modes $\mathtt{M} = \{m_1, \ldots, m_n\}$ and variables $x_1, \ldots, x_k$ ranging over the real numbers $\mathbb{R}$. We denote by S the resulting state space $\mathtt{M} \times \mathbb{R}^k$. For denoting the derivatives of $x_1, \ldots, x_k$ we use variables $\dot{x}_1, \ldots, \dot{x}_k$, ranging over $\mathbb{R}$ correspondingly. The primed versions $m', x'_1, \ldots, x'_k$ shall be used to describe the next state in a transition. For simplicity, we sometimes use the vector $\vec{x}$ to denote $(x_1, \ldots, x_k)$, and $(m, \vec{x})$ to denote a state. Similar notations are used for the primed and dotted versions $\vec{x}'$, $\dot{\vec{x}}$.

In order to describe hybrid automata, we use *constraints* that are arbitrary Boolean combinations of equalities and inequalities over terms. This way, constraints represent subsets of given sets. In case v is contained in the set represented by a given constraint, we say that v *satisfies* a given constraint. These constraints are used, on the one hand, to describe the possible flows and jumps and, on the other hand, to mark certain parts of the state space (e.g., the set of initial/unsafe states). A *state-space constraint* is a constraint over the variables $m, \vec{x}$, and represents a subset of $\mathtt{M} \times \mathbb{R}^k$. A *flow constraint* is a constraint over the variables m, $\vec{x}$, $\dot{\vec{x}}$. It represents a subset of $\mathtt{M} \times \mathbb{R}^k \times \mathbb{R}^k$. An example of a flow

constraint for the case $n = 2$ and $k = 1$ is:

$$\Big((m = m_1 \to \dot{x} = x) \wedge (m = m_2 \to \dot{x} = -x)\Big).$$

In general, an invariant that has to hold in a mode can be modelled by formulating a flow constraint that does not allow a continuous behaviour in certain regions. As an example, the flow constraint $(m = m_1 \to 1 \leq x \leq 5)$ expresses that in mode m_1 the invariant $1 \leq x \leq 5$ holds.

For capturing the jump behaviours, we introduce the notion of update constraints. An *update constraint* $\mathtt{u}$, also called a *guarded command*, has the form: $condition \to update$ where $condition$ is a state-space constraint over $m, \vec{x}$, and $update$ is an expression denoting a function $\mathtt{M} \times \mathbb{R}^k \to \mathtt{M} \times \mathbb{R}^k$ which is called the *reset mapping* for m and $\vec{x}$. Intuitively, assume that the state $(m, \vec{x})$ satisfies $condition$, then the mode m and variable $\vec{x}$ are updated[1] to the new state $update(m, \vec{x})$.

A *jump constraint* is a finite disjunction $\bigvee_{\mathtt{u} \in \mathtt{U}} \mathtt{u}$ where $\mathtt{U}$ is a set of guarded commands. The constraint $\bigvee_{\mathtt{u} \in \mathtt{U}} \mathtt{u}$ can be represented by the set $\mathtt{U}$ for simplicity.

Definition 3.1 *A* hybrid automaton *is a tuple $\mathcal{H} = (Flow, \mathtt{U}, Init, UnSafe)$ consisting of a flow constraint $Flow$, a finite set of update constraints $\mathtt{U}$, a state-space constraint $Init$, describing the set of initial states, and a state-space constraint $UnSafe$, describing the set of unsafe states.*

A *flow* of length l in a mode m is a function $r\colon [0, l] \mapsto \mathbb{R}^k$ such that

- if $l > 0$, then r is differentiable for all $t \in [0, l]$, and we require that $(m, r(t), \dot{r}(t))$ satisfies $Flow$, where $\dot{r}$ is the derivative of r, and
- if $l = 0$, we require that there exists a constant $c \in \mathbb{R}^k$ such that $(m, r(t), c)$ satisfies $Flow$.

Provided that the constraint of a jump is enabled, in systems modelled in a reasonable manner, it will usually be the case that the corresponding jump leads to a state in which a valid flow exists. This also restricts valid states of the hybrid automaton.

3.1.2 Transition System Semantics

The semantics of a hybrid automaton is a transition system with an uncountable set of states.

Definition 3.2 *A* transition system *is a tuple $(S, T, S_{Init}, S_{UnSafe})$ where*

- *S is the (possibly uncountable) set of states,*
- *$T \subseteq S \times S$ is the transition relation,*
- *$S_{Init} \subseteq$ is the set of initial states and*
- *$S_{UnSafe} \subseteq$ is the set of unsafe states.*

The semantics of $\mathcal{H} = (Flow, \mathtt{U}, Init, UnSafe)$ is a transition system $T_{\mathcal{H}} = (S, T, S_{Init}, S_{UnSafe})$ with state set $S = \mathtt{M} \times \mathbb{R}^k$, set of initial states $S_{Init} = \{s \in S \mid s \text{ satisfies } Init\}$, and unsafe states $S_{UnSafe} = \{s \in S \mid s \text{ satisfies } UnSafe\}$. The transition set T is defined as the union of two transition relations $T_{\mathcal{C}}, T_{\mathcal{D}} \subseteq S \times S$, where $T_{\mathcal{C}}$ corresponds to transitions due to continuous flows defined by:

[1] Our definition of jumps is deterministic, as in [4], i.e., if a jump is triggered for a state satisfying $condition$, the successor state is updated deterministically according to $update$. In [30], the jump is defined to be nondeterministic: if a state satisfies $condition$, a successor will be selected nondeterministically from a set of states. Our method can be easily extended to this. We restrict to deterministic jumps for simplicity of the presentation in this paper.

- $((m, \vec{x}), (m, \vec{x}')) \in T_{\mathcal{C}}$, if there exists a flow r of length l in mode m such that $r(0) = \vec{x}$ and $r(l) = \vec{x}'$;

and $T_{\mathcal{D}}$ corresponds to transitions due to discrete jumps. The transition due to an update constraint $\mathtt{u}\colon \mathit{condition} \to \mathit{update}$, denoted by $T_{\mathcal{D}}(\mathtt{u})$, is defined by:

- $((m, \vec{x}), (m', \vec{x}')) \in T_{\mathcal{D}}(\mathtt{u})$ if $(m, \vec{x})$ satisfies the guard $\mathit{condition}$ and it holds that $(m', \vec{x}') = \mathit{update}(m, \vec{x})$.

Then, we define $T_{\mathcal{D}} = \bigcup_{\mathtt{u} \in \mathtt{U}} T_{\mathcal{D}}(\mathtt{u})$.

In the rest of the paper, if no confusion arises, we use Init to denote both the constraint for the initial states and the set of initial states. Similarly, UnSafe is used to denote both the constraint for the unsafe states and the set of unsafe states.

3.2 Probabilistic Automata

For defining the semantics of a probabilistic hybrid automaton, we recall first the notion of a probabilistic automaton [31]. It is an extension of a transition system with probabilistic branching.

We first introduce some notation. Let S be a (possibly uncountable) set. A *distribution* over S is a function $\mu\colon S \to [0, 1]$ such that (a) the support $\mathit{Supp}(\mu) = \{s \in S \mid \mu(s) > 0\}$ is finite, and (b) it is $\sum_{s \in S} \mu(s) = 1$. Let $\mathit{Distr}(S)$ denote the set of all distributions over S. For an arbitrary but fixed state $s \in S$, a *Dirac distribution* for s, denoted by Dirac_s, is a distribution over S such that $\mathit{Dirac}_s(s) = 1$, that is, $\mathit{Supp}(\mathit{Dirac}_s) = \{s\}$. The Dirac distribution will be used to describe the continuous evolution of a probabilistic hybrid automaton.

Definition 3.3 *A* probabilistic automaton $\mathcal{M}$ *is a tuple* $(S, \mathit{Steps}, \mathit{Init}, \mathit{UnSafe})$*, where* $\mathit{Steps} \subseteq S \times \mathit{Distr}(S)$*,* $\mathit{Init} \subseteq S$*, and* $\mathit{UnSafe} \subseteq S$*. Here,*

- S *denotes the set of states,*
- Init *is the set of initial states,*
- UnSafe *the set of unsafe states, and*
- Steps *the transition relation.*

We use $s \to \mu$ as a shorthand notation for $(s, \mu) \in \mathit{Steps}$, and call μ a *successor distribution* of s. Let $\mathit{Steps}(s)$ be the set $\{\mu \mid (s, \mu) \in \mathit{Steps}\}$. We assume that $\mathit{Steps}(s) \neq \emptyset$ for all $s \in S$.

A *finite path* of $\mathcal{M}$ is finite sequence $\sigma = s_0 \mu_0 s_1 \mu_1 \dots s_n$ such that $s_i \to \mu_i$ and $\mu_i(s_{i+1}) > 0$ for all possible $0 \leq i < n$. Infinite paths are defined by taking $n = \infty$. We denote by $\mathit{first}(\sigma)$ the first state s_0 of σ, by $\sigma[i]$ the $(i+1)$-th state s_i, and, if σ is finite, by $\mathit{last}(\sigma)$ the last state of σ. Let Path be the set of all infinite paths and Path^* the set of all finite paths.

The nondeterministic choices in $\mathcal{M}$ are resolved by *adversaries*. Intuitively, an adversary of $\mathcal{M}$ is a measurable map $A\colon \mathit{Path}^* \to \mathit{Distr}(\mathit{Steps})$ such that $A(\sigma)(s, \mu) > 0$ implies that $s = \mathit{last}(\sigma)$ and $s \to \mu$. If $A(\sigma)(s, \mu) > 0$, then the successor distribution μ should be selected from state s with probability $A(\sigma)(s, \mu)$. Given an initial state s, an adversary A induces a *discrete-time Markov chain* with state space Path^* in an obvious way. We let Prob_s^A denote the corresponding unique probability measure [28] over Path.

3.3 Probabilistic Hybrid Automata

Now we recall the definition of probabilistic hybrid automata, by equipping the discrete jumps with probabilities. This is needed to model, for example, component failure or message losses.

3.3.1 Syntax

For capturing the probabilistic jump behaviours, a *probabilistic guarded command* $\mathtt{c}$ is defined to have the form

$$condition \to p_1 : update_1 + \ldots + p_{q_\mathtt{c}} : update_{q_\mathtt{c}}$$

where $q_\mathtt{c} \geq 1$ denotes the number of probabilistic branching of $\mathtt{c}$, $p_i > 0$ for $i = 1, \ldots, q_\mathtt{c}$ and $\sum_{i=1}^{q_\mathtt{c}} p_i = 1$, $condition$ is a constraint over $(m, \vec{x})$, and $update_i$ is an expression denoting a reset mapping for m and $\vec{x}$ for all $i = 1, \ldots, q_\mathtt{c}$. Intuitively, if a state $(m, \vec{x})$ satisfies the guard $condition$, a jump to states $(m_1, \vec{x}_1), \ldots, (m_{q_\mathtt{c}}, \vec{x}_{q_\mathtt{c}})$ occurs such that $(m_i, \vec{x}_i) = update_i(m, \vec{x})$ is selected with probability p_i for $i = 1, \ldots, q_\mathtt{c}$. Observe that for different $i \neq j$, it could be the case that $(m_i, \vec{x}_i) = (m_j, \vec{x}_j)$. In this paper we assume that $q_\mathtt{c}$ is finite for all $\mathtt{c}$. As for the non-probabilistic setting, we assume that conditions are enabled only if corresponding jumps lead to a state in which a flow exists.

An example for the case $n = 2$, $k = 1$ and $q_\mathtt{c} = 2$ is:

$$\Big((m = m_1 \wedge x \geq 10) \to 0.2 : (m' = m_1 \wedge x' = 0) \\ + 0.8 : (m' = m_2 \wedge x' = x + 1)\Big).$$

Definition 3.4 *A* probabilistic hybrid automaton *is a tuple* $\mathcal{H} = (Flow, \mathtt{C}, Init, UnSafe)$ *where* $Flow, Init, UnSafe$ *are the same as in the hybrid automaton, and* $\mathtt{C}$ *is a finite set of probabilistic guarded commands.*

A probabilistic hybrid automaton induces a classical hybrid automaton where probabilistic branching is replaced by nondeterministic choices. Intuitively, the semantics of the latter spans the semantics of the former.

Definition 3.5 *Let* $\mathtt{c} = (condition \to p_1 : update_1 + \ldots + p_{q_\mathtt{c}} : update_{q_\mathtt{c}})$ *be a probabilistic guarded command. It induces a set of* $q_\mathtt{c}$ *update constraints:* $ind(\mathtt{c}) = \{\mathtt{u}_1, \ldots, \mathtt{u}_{q_\mathtt{c}}\}$ *where* $\mathtt{u}_i$ *corresponds to the update constraint* $condition \to update_i$ *for* $i = 1, \ldots, q_\mathtt{c}$. *Moreover, for a set* $\mathtt{C}$ *of probabilistic guarded commands we define* $ind(\mathtt{C}) = \bigcup_{\mathtt{c} \in \mathtt{C}} ind(\mathtt{c})$.

Let $\mathcal{H} = (Flow, \mathtt{C}, Init, UnSafe)$ *be a probabilistic hybrid automaton. The* induced hybrid automaton *is a tuple* $ind(\mathcal{H}) = (Flow, ind(\mathtt{C}), Init, UnSafe)$.

3.3.2 Semantics

The semantics of a probabilistic hybrid automaton is a probabilistic automaton [32]. Let $\mathcal{H} = (Flow, \mathtt{C}, Init, UnSafe)$ be a probabilistic hybrid automaton. Let $ind(\mathcal{H})$ denote the induced hybrid automaton, and let $T_{ind(\mathcal{H})} = (S, T, Init, UnSafe)$ denote the transition system representing the semantics of $ind(\mathcal{H})$. Recall that $T = T_\mathcal{C} \cup T_\mathcal{D}$ where $T_\mathcal{C}$ corresponds to transitions due to continuous flows and $T_\mathcal{D}$ corresponds to transitions due to discrete jumps.

The semantics of $\mathcal{H}$ is the probabilistic automaton $\mathcal{M}_\mathcal{H} = (S, Steps, Init, UnSafe)$ where $S, Init, UnSafe$ are the same as in $T_{ind(\mathcal{H})}$, and $Steps$ is defined as the union of two transition relations $Steps_\mathcal{C}, Steps_\mathcal{D} \subseteq S \times Distr(S)$. Here, as in the non-probabilistic setting, $Steps_\mathcal{C}$ corresponds to transitions due to continuous flows, while $Steps_\mathcal{D}$ corresponds to transitions due to discrete jumps. Both of them are defined respectively as follows.

For each transition $((m, \vec{x}), (m, \vec{x}')) \in T_\mathcal{C}$ in $ind(\mathcal{H})$, there is a corresponding transition in $\mathcal{H}$ from $(m, \vec{x})$ to $(m, \vec{x}')$ with probability 1. So, $Steps_\mathcal{C}$ is defined by: $Steps_\mathcal{C} = \{((m, \vec{x}), Dirac_{(m,\vec{x}')}) \mid ((m, \vec{x}), (m, \vec{x}')) \in T_\mathcal{C}\}$.

Now we discuss transitions induced by discrete jumps. First, for a probabilistic guarded command $\mathtt{c}$, we define its corresponding set $Steps_\mathcal{D}(\mathtt{c})$. Let $ind(\mathtt{c}) = \{\mathtt{u}_1, \ldots, \mathtt{u}_{q_\mathtt{c}}\}$

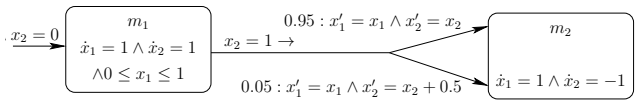

Figure 1: Illustration of Example 3.6.

be as defined in Definition 3.5. Then, for a number of $q_{\mathfrak{c}} + 1$ arbitrary states $(m, \vec{x})$, $(m_1, \vec{x}_1), \ldots, (m_{q_{\mathfrak{c}}}, \vec{x}_{q_{\mathfrak{c}}}) \in S$ satisfying the condition $((m, \vec{x}), (m_i, \vec{x}_i)) \in T_{\mathcal{D}}(\mathfrak{u}_i)$ for $i = 1, \ldots, q_{\mathfrak{c}}$, we introduce the transition $((m, \vec{x}), \mu) \in \mathit{Steps}_{\mathcal{D}}(\mathfrak{c})$ with

$$\mu(m_i, \vec{x}_i) = \sum_{j \in \{j | m_j = m_i \wedge \vec{x}_j = \vec{x}_i\}} p_j, \tag{1}$$

for $i = 1, \ldots, q_{\mathfrak{c}}$. Then, $\mathit{Steps}_{\mathcal{D}}$ is defined to be $\bigcup_{\mathfrak{c} \in \mathfrak{C}} \mathit{Steps}_{\mathcal{D}}(\mathfrak{c})$. Recall that we have assumed that $q_{\mathfrak{c}}$ is finite for all $\mathfrak{c}$. This implies $\mathit{Supp}(\mu)$ is finite for all transitions (s, μ) with $s \in S$.

3.3.3 Safety Properties

For hybrid automata, the safety property asserts that the unsafe states can never be reached. For probabilistic hybrid automata, however, the safety property expresses that the maximal probability of reaching the set UnSafe is bounded by some given threshold $\mathbf{p}$. In the following we fix a certain threshold $\mathbf{p}$.

Let $\mathit{Reach}(\mathit{UnSafe})$ denote the set of paths $\{\sigma \in \mathit{Path} \mid \exists i.\ \sigma[i] \in \mathit{UnSafe}\}$. The automaton $\mathcal{H}$ is called *safe* if for each adversary A and each initial state s of $\mathcal{M}(\mathcal{H})$, $\mathit{Prob}_s^A(\mathit{Reach}(\mathit{UnSafe})) \leq \mathbf{p}$ holds. Since safety verification in the non-probabilistic setting is an undecidable problem [18], this implies that the safety problem in the probabilistic setting is also undecidable. In the following, we develop a framework for a heuristic algorithm to deal with such a probabilistic safety verification problem for general probabilistic hybrid automata.

Example 3.6 *Consider the probabilistic hybrid automaton illustrated in Figure 1. It contains the modes m_1, m_2 and the continuous variables x_1, x_2 which both range over the interval $[0, 2]$, i.e, $S = \{m_1, m_2\} \times [0, 2] \times [0, 2]$ (note that the state space can be restricted via state-space constraints). The set of initial states is given by the constraint $\mathit{Init}(m, (x_1, x_2)) = (m = m_1 \wedge x_1 = 0 \wedge x_2 = 0)$. The constraint $\mathit{UnSafe}(m, (x_1, x_2)) = (x_1 \geq 1 \wedge x_2 \geq 1.5)$ describes the set of unsafe states. The hybrid automaton can switch modes from m_1 to m_2 with two possibilities if $x_2 = 1$, i.e.,*

$$\begin{aligned}(m = m_1 \wedge x_2 = 1) \rightarrow \\ 0.95 : (m' = m_2 \wedge x_1' = x_1 \wedge x_2' = x_2) + \\ 0.05 : (m' = m_2 \wedge x_1' = x_1 \wedge x_2' = x_2 + 0.5)\ .\end{aligned}$$

The continuous behaviour is very simple: in mode m_1, the values of the variables x_1, x_2 change with slope 1; in mode m_2, the slope of variable x_1 is 1 and variable x_2 is -1. For a flow in mode m_1, the constraint $0 \leq x_1 \leq 1$ must hold. The corresponding flow constraint is:

$$\begin{aligned}(m = m_1 \rightarrow (\dot{x}_1 = 1 \wedge \dot{x}_2 = 1 \wedge 0 \leq x_1 \leq 1)) \\ \wedge (m = m_2 \rightarrow (\dot{x}_1 = 1 \wedge \dot{x}_2 = -1))\,.\end{aligned}$$

The constraint $0 \leq x_1 \leq 1$ in Flow forces a jump from mode m_1 to m_2 if x_1 becomes 1.

The maximal reachability probability is 0.05, as can be seen as follows: From the initial state, we can only apply the probabilistic guarded command once and exactly if we

wait until $x_2 = 1$, as seen from the guard. With probability 0.95, we then move to a state with $m = m_2, x_1 = 1, x_2 = 1$. In this case, we cannot reach unsafe states, because in the following timed transitions x_2 is decreased and thus we will never have $x_2 \geq 1.5$. However, with probability 0.05 we jump to an unsafe state directly. Because of this, the probabilistic system is safe if we have a threshold of e.g. $\mathbf{p} = 0.051 > 0.05$.

3.3.4 Simulation Relations

We recall the notion of simulations between probabilistic automata. Intuitively, if $\mathcal{M}_2$ simulates $\mathcal{M}_1$, that is, $\mathcal{M}_2$ is an overapproximation of $\mathcal{M}_1$, then $\mathcal{M}_2$ can mimic all behaviours of $\mathcal{M}_1$. Thus, this allows us to verify safety properties on the abstraction $\mathcal{M}_2$ instead of $\mathcal{M}_1$. To establish the notion of simulations, we introduce first the notion of *weight functions* [21], which establish the correspondence between distributions.

Definition 3.7 *Let $\mu_1 \in Distr(S_1)$ and $\mu_2 \in Distr(S_2)$ be two distributions. For a relation $R \subseteq S_1 \times S_2$, a* weight function *for (μ_1, μ_2) with respect to R is a function $\Delta \colon S_1 \times S_2 \to [0, 1]$ such that*

1. *$\Delta(s_1, s_2) > 0$ implies $(s_1, s_2) \in R$,*
2. *$\mu_1(s_1) = \sum_{s_2 \in S_2} \Delta(s_1, s_2)$ for $s_1 \in S_1$, and*
3. *$\mu_2(s_2) = \sum_{s_1 \in S_1} \Delta(s_1, s_2)$ for $s_2 \in S_2$.*

We write $\mu_1 \sqsubseteq_R \mu_2$ if and only if there exists a weight function for (μ_1, μ_2) with respect to R.

Now, we recall the notion of simulations [31], adapted to reachability properties. The simulation requires that every successor distribution of a state of $\mathcal{M}_1$ is related to a successor distribution of its corresponding state of $\mathcal{M}_2$ via a weight function.

Definition 3.8 *Given two probabilistic automata $\mathcal{M}_1 = (S_1, Steps_1, Init_1, UnSafe_1)$ and $\mathcal{M}_2 = (S_2, Steps_2, Init_2, UnSafe_2)$, we say that $\mathcal{M}_2$* simulates *$\mathcal{M}_1$, denoted by $\mathcal{M}_1 \preceq \mathcal{M}_2$, if and only if there exists a relation $R \subseteq S_1 \times S_2$, which we will call* simulation relation *from now on, such that*

1. *for each $s_1 \in Init_1$ there exists an $s_2 \in Init_2$ with $(s_1, s_2) \in R$.*
2. *for each $s_1 \in UnSafe_1$ there exists an $s_2 \in UnSafe_2$ with $(s_1, s_2) \in R$, and there does not exist an $s' \in S_2 \setminus UnSafe_2$ such that $(s_1, s') \in R$.*
3. *for each pair $(s_1, s_2) \in R$, if there exists $(s_1, \mu_1) \in Steps_1$, then there exists a distribution $\mu_2 \in Distr(S_2)$ such that $(s_2, \mu_2) \in Steps_2$ and $\mu_1 \sqsubseteq_R \mu_2$.*

4 Abstractions for Probabilistic Hybrid Automata

Various abstraction refinement techniques have been developed for verifying safety properties of non-probabilistic hybrid automata. All of them have a common strategy: the set S is covered by a finite set of abstract states, each representing a set of concrete states. Then, an abstraction is constructed, that is an overapproximation of the original system. Afterwards, the safety property is checked on the abstraction. If no abstract unsafe state is reachable, the original system is safe since the abstraction overapproximates the original system. If not, the covering might have been chosen too coarse, and a refinement step is needed.

Following this idea, refinement techniques for abstractions based on subsuming concrete states by validity of predicates on the state variables have been used [13, 4, 30].

Let $\mathcal{H} = (Flow, \subset, Init, UnSafe)$ be a probabilistic hybrid automaton. The aim of this section is—independently of which abstraction technique is used—to develop a framework

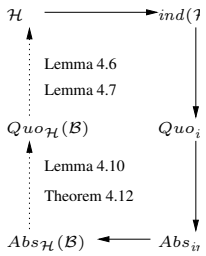

Figure 2: Computation of the abstraction.

for constructing an abstraction for $\mathcal{H}$, which is a finite probabilistic automaton. First we introduce the notion of abstract states, which form a (not necessarily disjoint) covering of the concrete state space:

Definition 4.1 *An* abstract state *is a pair (m, B) where $m \in \mathcal{M}$ and $B \subseteq \mathbb{R}^k$. A covering $\mathcal{B}$ is a finite set of abstract states such that $S = \bigcup\{(m, \vec{x}) \mid (m, B) \in \mathcal{B} \wedge \vec{x} \in B\}$.*

In the above definition, for two abstract states (m, B_1) and (m, B_2) we might have $B_1 \cap B_2 \neq \emptyset$ [2]. For instance, dependent on the abstraction technique used, B_1 and B_2 might have overlapping borders [30], or common interiors [17].

Figure 2 illustrates how this section is organised. Given a probabilistic hybrid automaton $\mathcal{H}$ and an abstract state space $\mathcal{B}$, we introduce the quotient automaton of both $ind(\mathcal{H})$ and $\mathcal{H}$ with respect to $\mathcal{B}$ in Section 4.1. In Section 4.2, we show the soundness with respect to the quotient automaton (cf. Lemma 4.6 and Lemma 4.7).

The quotient automaton is in general hard to compute. Thus, in Section 4.3 we introduce general abstractions, which overapproximate the quotient automata conservatively. In Section 4.4, we discuss how the abstraction for the given probabilistic hybrid automaton is constructed (see Figure 2): we first construct the abstraction of the induced hybrid automaton, from which the abstraction of the probabilistic setting is obtained afterwards.

4.1 Quotient Automaton for $\mathcal{H}$

We define the quotient automaton for the probabilistic hybrid automaton $\mathcal{H}$. First we define the quotient automaton for the induced hybrid automaton $ind(\mathcal{H})$. As a convention, we use $\mathcal{T}, \mathcal{I}, \mathcal{U}$ to denote the set of transitions, initial states, unsafe states in the quotient automata.

Definition 4.2 *Let $\mathcal{H} = (Flow, \mathtt{C}, Init, UnSafe)$ be a probabilistic hybrid automaton, and let $\mathcal{B}$ denote the abstract state space. Further, let $T_{ind(\mathcal{H})} = (S, T_{\mathcal{C}} \cup T_{\mathcal{D}}, Init, UnSafe)$ denote the automaton representing the semantics of $ind(\mathcal{H})$. The* quotient automaton *for $T_{ind(\mathcal{H})}$, denoted by $Quo_{ind(\mathcal{H})}(\mathcal{B})$, is a finite transition system $(\mathcal{B}, \mathcal{T}, \mathcal{I}, \mathcal{U})$ where*

- $\mathcal{I} = \{(m, B) \in \mathcal{B} \mid \exists \vec{x} \in B.\ (m, \vec{x}) \in Init\}$,
- $\mathcal{U} = \{(m, B) \in \mathcal{B} \mid \exists \vec{x} \in B.\ (m, \vec{x}) \in UnSafe\}$,
- *$\mathcal{T}_{\mathcal{C}}$ corresponds to the set of abstract transitions due to continuous flow: $\mathcal{T}_{\mathcal{C}} = \{((m, B), (m, B')) \in \mathcal{B}^2 \mid \exists \vec{x} \in B.\ \exists \vec{x}' \in B'.\ ((m, \vec{x}), (m, \vec{x}')) \in T_{\mathcal{C}}\}$,*
- *$\mathcal{T}_{\mathcal{D}}$ corresponds to the set of abstract transitions due to discrete jumps. We first define the transition induced by one fixed update $\mathtt{u} \in ind(\mathtt{C})$: $\mathcal{T}_{\mathcal{D}}(\mathtt{u}) = \{((m, B), (m', B')) \in \mathcal{B}^2 \mid \exists \vec{x} \in B.\ \exists \vec{x}' \in B'.\ ((m, \vec{x}), (m', \vec{x}')) \in T_{\mathcal{D}}(\mathtt{u})\}$. Then, let $\mathcal{T}_{\mathcal{D}} = \bigcup_{\mathtt{u} \in ind(\mathtt{C})} \mathcal{T}_{\mathcal{D}}(\mathtt{u})$.*

[2] We may also require that abstract states form a partitioning over the original state space S, with pairwise disjoint abstract states. Such abstractions are, however, harder to construct for non-trivial models.

In the following let $\mathcal{H} = (Flow, \mathtt{C}, Init, UnSafe)$ be a probabilistic hybrid automaton, let $\mathcal{M}_{\mathcal{H}} = (S, Steps_{\mathcal{C}} \cup Steps_{\mathcal{D}}, Init, UnSafe)$ denote the probabilistic automaton representing the semantics of $\mathcal{H}$, and let $\mathcal{B}$ be an abstract state space. As in the induced non-probabilistic setting, we define a quotient automaton, denoted by the probabilistic automaton $Quo_{\mathcal{H}}(\mathcal{B})$, for the abstract state space $\mathcal{B}$. For this, we first introduce the set of lifted distributions:

Definition 4.3 *Let $\mathcal{H}$, $\mathcal{M}_{\mathcal{H}}$ and $\mathcal{B}$ be as described above. Let $c \in \mathtt{C}$ and assume that $(s, \mu) \in Steps_{\mathcal{D}}(c)$ in $\mathcal{M}_{\mathcal{H}}$. By definition of $Steps_{\mathcal{D}}(c)$, there exist states $(m_1, \vec{x}_1), \ldots, (m_{q_c}, \vec{x}_{q_c}) \in S$ satisfying the condition $((m, \vec{x}), (m_i, \vec{x}_i)) \in T_{\mathcal{D}}(u_i)$ for $i = 1, \ldots, q_c$. Then, for arbitrary abstract states $(m_1, B_1), \ldots, (m_{q_c}, B_{q_c})$ with $\vec{x}_i \in B_i$ for $i = 1, \ldots, q_c$ we introduce the distribution $\mu' \in Distr(\mathcal{B})$ with:*

$$\mu'(m_i, B_i) = \sum_{\{j \mid (m_j, B_j) = (m_i, B_i)\}} \mu(m_j, \vec{x}_j).$$

The set of lifted distributions *$lift_{\mathcal{B}}(\mu)$ contains all such μ'.*

Let μ be the distribution according to a probabilistic guarded command $\mathtt{c}$. Since the covering $\mathcal{B}$ is in general not disjoint, a concrete state $(m_i, \vec{x}_i)$ might belong to more than one abstract states. In this case, μ induces more than one lifted distribution. In the above definition, this is reflected by the way of defining one specific lifted distribution μ', for which we first fix to which abstract state each concrete state $(m_i, \vec{x}_i)$ belongs. Note that if $\mathcal{B}$ is a disjoint partitioning of S, then the set $lift_{\mathcal{B}}(\mu)$ is a singleton. We now introduce the quotient automaton for the probabilistic hybrid automaton:

Definition 4.4 *Let $\mathcal{H}$, $\mathcal{M}_{\mathcal{H}}$ and $\mathcal{B}$ be as described above. The* quotient automaton *for $\mathcal{M}_{\mathcal{H}}$ with respect to $\mathcal{B}$ is defined by $Quo_{\mathcal{H}}(\mathcal{B}) = (\mathcal{B}, \mathcal{ST}, \mathcal{I}, \mathcal{U})$ where $\mathcal{I}$ and $\mathcal{U}$ are defined as for $Quo_{ind(\mathcal{H})}(\mathcal{B})$, and $\mathcal{ST} = \mathcal{ST}_{\mathcal{C}} \cup \mathcal{ST}_{\mathcal{D}}$ is the set of abstract transitions where:*

- *$\mathcal{ST}_{\mathcal{C}}$ corresponds to the set of abstract transitions due to continuous flows: $\mathcal{ST}_{\mathcal{C}} = \{((m, B), Dirac_{(m,B')}) \in \mathcal{B} \times Distr(\mathcal{B}) \mid \exists \vec{x} \in B.\ \exists \vec{x}' \in B'.\ ((m, \vec{x}), (m, \vec{x}')) \in Steps_{\mathcal{C}}\}$.*
- *$\mathcal{ST}_{\mathcal{D}}$ corresponds to the set of abstract transitions due to discrete jumps. We first define the transitions induced by one fixed probabilistic guarded command c. We let $\mathcal{ST}_D(c) = \{((m, B), \mu') \in \mathcal{B} \times Distr(\mathcal{B}) \mid \exists \vec{x} \in B.\ \exists ((m, \vec{x}), \mu) \in Steps_{\mathcal{D}}(c).\ \mu' \in lift_{\mathcal{B}}(\mu)\}$. Then, let $\mathcal{ST}_{\mathcal{D}} = \bigcup_{c \in C} \mathcal{ST}_{\mathcal{D}}(c)$.*

Example 4.5 *Consider Figure 3 and assume we have a probabilistic guarded command $c = (condition \rightarrow p_1 : up_1 + \ldots + p_4 : up_4)$. Thus $q_c = 4$. The abstract states are represented by circles, labelled with the corresponding tuple. The concrete states are represented by black points, labelled with only the evaluation of the variables (assume that all of them are different). Thus the point labelled with s_0 represents the state (m_0, s_0) and so on. Arrows are transitions in the concrete models, where the labels represent the probability p_i of the corresponding update up_i of c.*

Consider the two (there may be more) concrete transitions in $T_{ind(\mathcal{H})}$: $((m_0, s_0), (m_1, s_1))$, $((m_0, s_0), (m_1, s_2)) \in T_{\mathcal{D}}$. Both of them lead from (m_0, B_0) to the same abstract state (m_1, B_1). By Definition 4.2, we have that $((m_0, B_0), (m_i, B_i)) \in \mathcal{T}_{\mathcal{D}}$ for $i = 1, 2, 3, 4$ in $Quo_{ind(\mathcal{H})}(\mathcal{B})$.

We have a concrete transition $((m_0, s_0), \mu)$ where μ is defined by: $\mu(s_i) = p_i$ for $i = 1, 2, 3, 4$. Assume first that B_0, B_1, B_2, B_3 are disjoint. By Definition 4.3, $lift_{\mathcal{B}}(\mu) = \{\mu'\}$ where μ' is defined by: $\mu'(m_1, B_1) = p_1 + p_2$, $\mu'(m_2, B_2) = p_3$, and $\mu'(m_3, B_3) = p_4$. Then, by Definition 4.4, this induces an abstract transition $((m_0, B_0), \mu') \in \mathcal{ST}_{\mathcal{D}}$ in $Quo_{\mathcal{H}}(\mathcal{B})$.

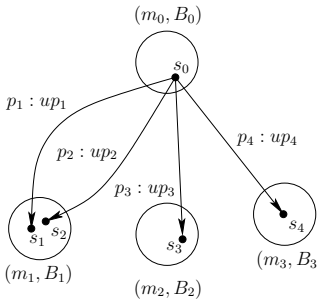

Figure 3: Illustrating the abstract discrete transitions in the quotient automaton.

Assume now that the abstract states B_1 and B_2 are not disjoint, and that s_2 is on the common border of (m_1, B_1) and (m_2, B_2) (which implies also $m_1 = m_2$). In this case, the set $lift_{\mathcal{B}}(\mu)$ contains another element μ'' which defined by: $\mu''(m_1, B_1) = p_1$, $\mu''(m_2, B_2) = p_2 + p_3$ and $\mu''(m_3, B_3) = p_4$. Again by Definition 4.4, μ'' induces another abstract transition $((m_0, B_0), \mu'')$ in $Quo_{\mathcal{H}}(\mathcal{B})$.

4.2 Soundness

Given a probabilistic hybrid automaton $\mathcal{H}$ and a set of abstract states $\mathcal{B}$, we defined a probabilistic quotient automaton $Quo_{\mathcal{H}}(\mathcal{B})$. The following lemma shows that this automaton conservatively overapproximates $\mathcal{M}_{\mathcal{H}}$.

Lemma 4.6 *$Quo_{\mathcal{H}}(\mathcal{B})$ simulates $\mathcal{M}_{\mathcal{H}}$.*

Proof: We define $R = \{((m, \vec{x}), (m', B)) \in S \times \mathcal{B} \mid m = m' \wedge \vec{x} \in B\}$. It suffices to show that R is a simulation relation. Let $((m, \vec{x}), (m, B)) \in R$ be an arbitrary pair. The first two conditions for simulation relations are trivially satisfied. It remains to show the third condition. There are two types of transitions starting from $(m, \vec{x})$ in $\mathcal{M}_{\mathcal{H}}$: the case $((m, \vec{x}), Dirac_{(m,\vec{x}')}) \in Steps_{\mathcal{C}}$ is trivial and skipped. Now consider the case $((m, \vec{x}), \mu) \in Steps_{\mathcal{D}}$: there exists then a probabilistic guarded command $\mathtt{c}$ such that $((m, \vec{x}), \mu) \in Steps_{\mathcal{D}}(\mathtt{c})$. Let $\mathtt{c}$ and $ind(\mathtt{c}) = \{\mathtt{u}_1, \ldots, \mathtt{u}_{q_{\mathtt{c}}}\}$ be as described in Definition 3.5, and let $(m_i, \vec{x}_i) = \mathtt{u}_i(m, \vec{x})$, where assume that $i = 1, \ldots, q_{\mathtt{c}}$. Note that it could be the case that, for $i \neq j$, we have $\vec{x}_i = \vec{x}_j$. Moreover, let $(m_i, B_i) \in \mathcal{B}$ denote an abstract state satisfying $\vec{x}_i \in B_i$. By construction of the relation R, we know that $((m_i, \vec{x}_i), (m_i, B_i)) \in R$. By the definition of $\mathcal{ST}$ (cf. Definition 4.4), we have that $((m, B), \mu') \in \mathcal{ST}_{\mathcal{D}}(\mathtt{c})$ where $\mu'(m_i, B_i) = \sum_{j \in \{j \mid m_j = m_i \wedge B_j = B_i\}} p_j$. Define Δ for (μ, μ') with respect to R by: $\Delta((m_i, \vec{x}_i), (m_i, B_i))$ equals $\mu(m_i, \vec{x}_i)$ for $i = 1, \ldots, q_{\mathtt{c}}$, and equals 0 otherwise. It remains to show that Δ is the proper weight function. For the first condition, assume $\Delta((m^*, \vec{x}^*), (m', B')) > 0$. By the definition of Δ, we have $m^* = m'$ and $\vec{x}^* \in B'$, implying $((m^*, \vec{x}^*), (m', B')) \in R$. Now we show the third condition (the second condition is similar). Let (m_j, B_j) be an abstract state (otherwise trivial). On the one hand, due to the definition of μ', it is $\mu'(m_j, B_j) = \sum_{i \in I} p_i$ where $I = \{i \mid m_i = m_j \wedge B_i = B_j\}$ denotes the set of all indices i such that $(m_i, B_i) = (m_j, B_j)$. On the other hand, by the definition of Δ, it holds $\sum_{i \in I} p_i = \sum_{\vec{x}_k \in B_j} \mu(m_j, \vec{x}_k) = \sum_{k \in I} \Delta((m_j, \vec{x}_k), (m_j, B_j))$ (cf. Equation (1)), which implies the third condition. ■

Since simulation on probabilistic automata preserves safety properties [31], we have the correctness of our construction:

Lemma 4.7 *The abstraction preserves the safety property: if the probability of reaching $UnSafe$ in $Quo_{\mathcal{H}}(\mathcal{B})$ is bounded by $\mathbf{p}$, this is also the case in $\mathcal{H}$.*

4.3 Abstractions for $\mathcal{H}$

Consider the probabilistic hybrid automaton $\mathcal{H}$. The computation of the exact quotient automaton $Quo_{\mathcal{H}}(\mathcal{B})$ as defined in Definition 4.4 refers to concrete states, and often is hard or even impossible. In this subsection, we introduce the notion of abstractions which over-approximate the quotient automata. As a convention, we use the primed version $\mathcal{T}', \mathcal{I}', \mathcal{U}'$ to denote the set of transitions, initial states, unsafe states in the abstraction.

Definition 4.8 *Let $\mathcal{H} = (Flow, \mathtt{C}, Init, UnSafe)$ be a probabilistic hybrid automaton, and let $\mathcal{B}$ denote the abstract state space. Then,*

- *$Abs_{ind(\mathcal{H})}(\mathcal{B}) = (\mathcal{B}, \mathcal{T}', \mathcal{I}', \mathcal{U}')$ is an* abstraction *of the quotient $Quo_{ind(\mathcal{H})}(\mathcal{B})$ iff $\mathcal{T}' = \bigcup_{\mathtt{u} \in ind(\mathtt{C})} \mathcal{T}'_{\mathcal{D}}(\mathtt{u}) \cup \mathcal{T}'_{\mathcal{C}}$ and it holds $\mathcal{T}_{\mathcal{C}} \subseteq \mathcal{T}'_{\mathcal{C}}$, $\mathcal{T}_{\mathcal{D}}(\mathtt{u}) \subseteq \mathcal{T}'_{\mathcal{D}}(\mathtt{u})$ for all $\mathtt{u} \in ind(\mathtt{C})$, and we have $\mathcal{I} \subseteq \mathcal{I}'$ and $\mathcal{U} \subseteq \mathcal{U}'$,*
- *$Abs_{\mathcal{H}}(\mathcal{B}) = (\mathcal{B}, \mathcal{ST}', \mathcal{I}', \mathcal{U}')$ is an* abstraction *of the quotient $Quo_{\mathcal{H}}(\mathcal{B})$ iff $\mathcal{ST}' = \bigcup_{\mathtt{c} \in \mathtt{C}} \mathcal{ST}'_{\mathcal{D}}(\mathtt{c}) \cup \mathcal{ST}'_{\mathcal{C}}$ and it holds $\mathcal{ST}_{\mathcal{D}}(\mathtt{c}) \subseteq \mathcal{ST}'_{\mathcal{D}}(\mathtt{c})$ for all $\mathtt{c} \in \mathtt{C}$, and it is $\mathcal{ST}_{\mathcal{C}} \subseteq \mathcal{ST}'_{\mathcal{C}}$, $\mathcal{I} \subseteq \mathcal{I}'$ and $\mathcal{U} \subseteq \mathcal{U}'$.*

In that case, we say also that $Abs_{ind(\mathcal{H})}(\mathcal{B})$ is an abstraction of the induced hybrid automaton $ind(\mathcal{H})$. Similarly, we say also that $Abs_{\mathcal{H}}(\mathcal{B})$ is an abstraction of the probabilistic hybrid automaton $\mathcal{H}$. Since the abstraction as defined may have more initial states, unsafe states and transitions than the quotient automaton, it is easy to verify that the abstraction simulates the corresponding quotient automaton. Since simulation is transitive, the abstraction also simulates the corresponding semantics automaton. Thus, the abstraction preserves also safety properties of $\mathcal{H}$.

4.4 Computing Abstractions

Let $\mathcal{H}$ be a probabilistic hybrid automaton. Existing methods [17, 4, 30] can be used to compute an abstraction $Abs_{ind(\mathcal{H})}(\mathcal{B})$ for the induced hybrid automaton $ind(\mathcal{H})$. In the following, we define an abstraction based on $Abs_{ind(\mathcal{H})}(\mathcal{B})$:

Definition 4.9 *For a probabilistic hybrid automaton $\mathcal{H}$, let $\mathcal{B}$ be the abstract state space, and $Abs_{ind(\mathcal{H})}(\mathcal{B}) = (\mathcal{B}, \mathcal{T}'_{\mathcal{D}} \cup \mathcal{T}'_{\mathcal{C}}, \mathcal{I}', \mathcal{U}')$ be an abstraction of $ind(\mathcal{H})$. We define $Abs_{\mathcal{H}}(\mathcal{B}) = (\mathcal{B}, \mathcal{ST}'_{\mathcal{C}} \cup \mathcal{ST}'_{\mathcal{D}}, \mathcal{I}', \mathcal{U}')$ for $\mathcal{H}$ as follows:*

- *$\mathcal{ST}'_{\mathcal{C}} = \{((m, B), Dirac_{(m,B')}) \in \mathcal{B} \times Distr(\mathcal{B}) \mid ((m, B), (m, B')) \in \mathcal{T}'_{\mathcal{C}}\}$,*
- *$\mathcal{ST}'_{\mathcal{D}}$ corresponds to the set of abstract transitions due to discrete jumps. We first define the transition induced by one fixed probabilistic guarded command $\mathtt{c} = (condition \rightarrow p_1 : update_1 + \ldots + p_{q_\mathtt{c}} : update_{q_\mathtt{c}})$, and $ind(\mathtt{c}) = \{\mathtt{u}_1, \ldots, \mathtt{u}_{q_\mathtt{c}}\}$ as defined in Definition 3.5. Then, for every sequence of abstract states $(m, B), (m_1, B_1), \ldots, (m_{q_\mathtt{c}}, B_{q_\mathtt{c}})$ satisfying the condition: $((m, B), (m_i, B_i)) \in \mathcal{T}'_{\mathcal{D}}(\mathtt{u}_i)$ for $i = 1, \ldots, q_\mathtt{c}$, we introduce the transition $((m, B), \mu) \in \mathcal{ST}'_{\mathcal{D}}(\mathtt{c})$ such that $\mu(m_i, B_i) = \sum_{j \in \{j \mid m_j = m_i \wedge B_j = B_i\}} p_j$ for $i = 1, \ldots, q_\mathtt{c}$. Then, $\mathcal{ST}'_{\mathcal{D}}$ is defined to be $\bigcup_{\mathtt{c} \in \mathtt{C}} \mathcal{ST}'_{\mathcal{D}}(\mathtt{c})$.*

Is $Abs_{\mathcal{H}}(\mathcal{B})$ in fact an abstraction of $\mathcal{H}$? Since $Abs_{ind(\mathcal{H})}(\mathcal{B})$ is an abstraction for $Quo_{ind(\mathcal{H})}(\mathcal{B})$, by Definition 4.8 it holds that $\mathcal{T}_{\mathcal{C}} \subseteq \mathcal{T}'_{\mathcal{C}}, \mathcal{I} \subseteq \mathcal{I}', \mathcal{U} \subseteq \mathcal{U}'$ and that $\mathcal{T}_{\mathcal{D}}(\mathtt{u}) \subseteq \mathcal{T}'_{\mathcal{D}}(\mathtt{u})$ for each $\mathtt{u} \in ind(\mathtt{C})$. Note that in general most of the inclusions above are strict [4, 30]. By the construction of $Abs_{\mathcal{H}}(\mathcal{B})$, it holds that $\mathcal{ST}_{\mathcal{C}} \subseteq \mathcal{ST}'_{\mathcal{C}}, \mathcal{I} \subseteq \mathcal{I}', \mathcal{U} \subseteq \mathcal{U}'$. The following lemma shows that it holds also $\mathcal{ST}_{\mathcal{D}} \subseteq \mathcal{ST}'_{\mathcal{D}}$:

Lemma 4.10 *Consider the abstraction $Abs_{\mathcal{H}}(\mathcal{B})$ as defined in Definition 4.9. Then, it holds that $\mathcal{ST}_{\mathcal{D}}(\mathtt{c}) \subseteq \mathcal{ST}'_{\mathcal{D}}(\mathtt{c})$, for all $\mathtt{c} \in \mathtt{C}$.*

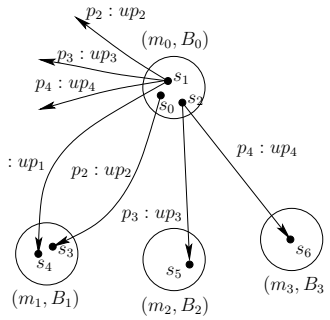

Figure 4: Abstracting abstract discrete transitions.

Proof: Fix $\mathtt{c} \in \mathbb{C}$. Assume that $((m, B), \mu') \in \mathcal{ST}_{\mathcal{D}}(\mathtt{c})$. Then, by Definition 4.4, there exists $\vec{x} \in B$ and a transition $((m, \vec{x}), \mu) \in Steps_{\mathcal{D}}(\mathtt{c})$ such that $\mu' \in lift_{\mathcal{B}}(\mu)$. For $i = 1, \ldots, q_{\mathtt{c}}$, let $(m_i, \vec{x}_i) = update_i(m, \vec{x})$, and let (m_i, B_i) be the abstract states corresponding to the distribution μ' (cf. Definition 4.3), i.e., $\mu'(m_i, B_i) = \sum_{\{j|(m_j,B_j)=(m_i,B_i)\}} \mu(m_j, \vec{x}_j)$. Obviously, $((m, \vec{x}), (m_i, \vec{x}_i)) \in T_{\mathcal{D}}(\mathtt{u}_i)$. Since $\vec{x}_i \in B_i$ it holds that $((m, B), (m_i, B_i)) \in \mathcal{T}_{\mathcal{D}}(\mathtt{u}_i) \subseteq \mathcal{T}'_{\mathcal{D}}(\mathtt{u}_i)$ for $i = 1, \ldots, q_{\mathtt{c}}$. By Definition 4.9, we have that $((m, B), \mu') \in \mathcal{ST}'_{\mathcal{D}}(\mathtt{c})$ ■

The set of transitions $\mathcal{ST}'_{\mathcal{D}}(\mathtt{c})$ is indeed an overapproximation, which is illustrated as follows.

Example 4.11 *Consider the fragment of the abstraction depicted in Figure 4 in which we assume that the transitions correspond to the probabilistic guarded command* $\mathtt{c}$ *with* $q_{\mathtt{c}} = 4$*: condition* $\rightarrow p_1 : up_1 + \ldots + p_4 : up_4$*. Assume that all of the concrete states are different and are not on borders. (Note: only parts of successor distributions are depicted, and we assume that other parts, e.g. for state* (m_0, s_1)*, lead to abstract states outside the depicted fragment.)*

Now we consider the distribution $\mu^* \in Distr(\mathcal{B})$ *which is defined as follows:* $\mu^*(m_1, B_1) = p_1 + p_2$*,* $\mu^*(m_2, B_2) = p_3$ *and* $\mu^*(m_3, B_3) = p_4$*. By the above assumption, no concrete successor distributions of* s_0, s_1 *or* s_2 *could induce* μ^* *according Definition 4.3. Thus, by Definition 4.4,* $((m_0, B_0), \mu^*) \notin \mathcal{ST}_{\mathcal{D}}(\mathtt{c})$*. On the other hand, it holds* $((m_0, B_0), (m_1, B_1)) \in \mathcal{T}'_{\mathcal{D}}(up_i)$ *for* $i = 1, 2$*,* $((m_0, B_0), (m_2, B_2)) \in \mathcal{T}'_{\mathcal{D}}(up_3)$*, and* $((m_0, B_0), (m_3, B_3)) \in \mathcal{T}'_{\mathcal{D}}(up_4)$*. Thus, by Definition 4.9 we have that* $((m_0, B_0), \mu^*) \in \mathcal{ST}'_{\mathcal{D}}(\mathtt{c})$*.*

Lemma 4.10 implies that $Abs_{\mathcal{H}}(\mathcal{B})$ is an abstraction of $Quo_{\mathcal{H}}(\mathcal{B})$. Thus:

Theorem 4.12 *For every probabilistic hybrid automaton* $\mathcal{H}$*, for every abstraction* $Abs_{ind(\mathcal{H})}(\mathcal{B})$ *of the induced hybrid automaton* $ind(\mathcal{H})$*, the safety of* $Abs_{\mathcal{H}}(\mathcal{B})$ *implies the safety of* $\mathcal{H}$*.*

4.5 Computing Reachability Probabilities

First we note that, similar as in the non-probabilistic case, only abstract states reachable from the initial abstract states are of interest. Below, we discuss briefly how to compute the maximal probability of reaching the set of abstract unsafe states in the abstraction.

Given an initial state s, the maximal probability of reaching the set of abstract unsafe states $UnSafe$ from s is denoted by

$$Prob_s^+(Reach(UnSafe)) = \sup_A Prob_s^A(Reach(UnSafe)),$$

where A ranges over all adversaries. Given the threshold $\mathbf{p}$, the safety property is satisfied if $Prob_s^+(Reach(UnSafe)) < \mathbf{p}$ for all initial states s. The probability

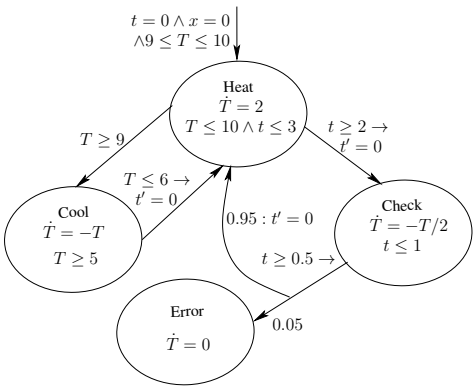

Figure 5: A probabilistic hybrid automaton for the thermostat.

$Prob_s^+(Reach(UnSafe))$ can be characterised [7] as follows: it equals 1 if $s \in UnSafe$, it is 0 if $UnSafe$ can't be reached from s at all, and otherwise it is

$$\max_{\mu \in Steps(s)} \sum_{s' \in S} \mu(s') Prob^+_{s' \in S}(Reach(UnSafe)).$$

This suggests the following well-known value iteration approach [6]. As a preprocessing, those states which cannot be reached by any initial state, or cannot reach the unsafe states can be pruned, since the probabilities for those states are 0. We iteratively compute the solution vector $\vec{v}$ during the algorithm, and let $\vec{v}^i$ denote the probability values in the i-th iteration. Initially, we let $\vec{v}^0(s) = 1$ if $s \in UnSafe$, and $\vec{v}^0(s) = 0$ otherwise. The vector $\vec{v}^{i+1}$ can then be computed recursively according to $\vec{v}^{i+1}(s) = 1$ if $s \in UnSafe$ and

$$\vec{v}^{i+1}(s) = \max_{\mu \in Steps(s)} \sum_{s' \in S} \mu(s') \vec{v}^i(s')$$

else. Hence, $\vec{v}^i(s)$ corresponds to the maximal probability of reaching $UnSafe$ within a number of i steps, and moreover, it converges to $Prob_s^+(Reach(UnSafe))$ if $i \to \infty$. In practice, a stopping criterion is needed to decide when the approximation is tight enough: in our tool (see Section 6) we choose the relative error stopping criterion, namely we stop if $\max_s \frac{||\vec{v}^{i+1}(s) - \vec{v}^i(s)||}{||\vec{v}^{i+1}(s)||} \leq \varepsilon$ with $\varepsilon = 10^{-8}$.

5 An Illustrating Example

As an illustrating example, we here consider the thermostat example depicted in Figure 5, which is extended from the one of Alur et al. [4]. There are four modes: $Cool$, $Heat$, $Check$ and $Error$. The latter mode models the occurrence of a failure, where the temperature sensor gets stuck at the temperature checked last. The set of variables is $\{t, x, T\}$, where T represents the temperature, t represents a local timer and x is used to measure the total time passed so far. Thus, in all modes it holds that $\dot{x} = 1$ and $\dot{t} = 1$. In each mode there is also an invariant constraint restricting the state space of this mode. Invariant constraints are used only for the sake of convenience and comparison with [4].

The given initial condition is $m = Heat \wedge t = 0 \wedge x = 0 \wedge 9 \leq T \leq 10$. The unsafe constraint is $m = Error \wedge x \leq 5$, which corresponds to reaching the $Error$ mode within 5 time units. Assume that the probability threshold for this risk is specified to be $\mathbf{p} = 0.2$.

First we observe that initially it is $t = 0$ and $9 \leq T \leq 10$. The system cannot stay in the mode $Heat$ for 2 time units, as this would increase the temperature by 4 units which violates the invariant $T \leq 10$ at $Heat$. This means that the system must switch to mode

Cool to decrease the temperature. It would take some amount of time (approximately 0.41) units to decrease the temperature until reaching 6 such that the system can go back to *Heat*. Note that switching back to *Heat* would reset the local timer which implies that $t = 0$ and that $x \geq 0.41$. To reach the unsafe mode *Error*, the intermediate mode *Check* must be touched. Because of the guard $t \geq 2$ between *Heat* and *Check*, once the mode *Check* is reached, it holds that $t = 0$ and $x \geq 2.41$. Then, the system waits in *Check* for at least 0.5 time units. After the probabilistic jump from *Check* is triggered, it holds that $x \geq 2.91$. Then, the unsafe state could be reached with probability 0.05. With probability 0.95 the system goes back to mode *Heat* and it holds that $t = 0$ and $x \geq 2.91$. Reiterating the above analysis, reaching *Error* from *Heat* would again take at least 2.5 time units, which implies that there is only one chance to hit the unsafe mode *Error* within 5 time units. Thus, the probability is bounded by 0.05, which implies that the safety property is indeed satisfied.

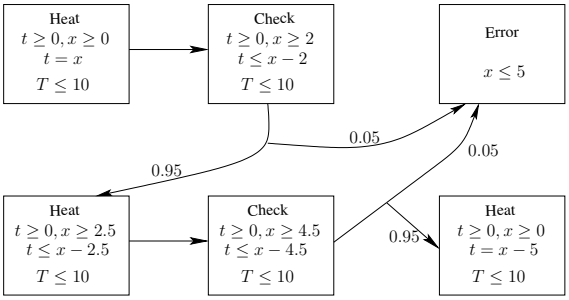

Figure 6: A fragment of the abstract states and abstract transitions of the thermostat. Each of the depicted abstract states has a self loop (which is not depicted) due to continuous flows.

Now we consider an abstraction of the thermostat example. The initial abstract state is $(Heat, B)$, where B represents concrete valuations satisfying the constraint $t \geq 0, x \geq 0$, $t = x$ and $T \leq 10$. In Figure 6, we depicted fragments of the abstract states and of those abstract transitions which lead to abstract unsafe states. Notably, in this abstraction there are two chances to touch abstract unsafe states, thus the probability amounts to $0.05 + 0.95 \cdot 0.05 = 0.0975$. The reason is that from the initial state *Heat* the abstract system does not need go back to *Cool* to let the temperature decrease, instead it can immediately switch to *Check*. This is due to the overapproximation of the abstract initial states. However, the computed probability for the threshold $\mathbf{p} = 0.2$ is still good enough to prove the safety property. If instead the threshold were set between 0.05 and 0.0975, refinement would have been needed.

6 Experiments

We implemented our method in the prototypical tool `ProHVer` (probabilistic hybrid automata verifier). It combines a modified version of `PHAVer` [17] to obtain the abstract state space with a component to compute an upper probability bound for the reachability problem using value iteration (cf. Subsection 4.5) in the induced abstract probabilistic automaton.

`PHAVer` can handle continuous dynamics with constant bounds on the derivatives exactly. In order to be able to handle affine dynamics not in this class, it uses overapproximation. Here it splits locations (introducing new discrete locations) such that the continuous state space of the resulting locations are polyhedra of a predefined maximal width. It is possible to improve the precision of the overapproximation by reducing this maximal width. However, the resulting covering and location splits can look entirely different, which is

time bound	interval length 0.15			interval length 0.15, hull			interval length 0.05, hull		
	prob.	build (s)	#states	prob.	build (s)	#states	prob.	build (s)	#states
1	0	1	38	0	1	17	0	2	56
2	0.25	3	408	0.25	2	59	0.25	9	185
3	0.5	13	3K	0.5	4	124	0.312	22	347
3.5	0.5	70	11K	1	5	145	0.5	32	425
3.6	0.5	140	14K	1	6	150	0.5	37	436
3.7	0.5	214	18K	1	6	154	-	-	-

Table 1: Bouncing ball performance figures. Where appropriate, we round probabilities to three decimal places and denote multiples of 1000 by "K".

also reflected on the probabilistic side. This phenomenon of `PHAVer` may induce situations, where that reduced width setting does not lead to tighter probability bounds. Usually, however, bounds do improve.

To show the applicability of our approach, we applied `ProHVer` to several case studies, which are small but diverse in the nature of their behaviour. In most of the examples considered, we focus on reachability probabilities with upper time bounds (obtained by using an additional clock to measure time), because these correspond to very natural verification problems for the settings considered. Notably, our method is not restricted to time-bounded reachability. Actually, time-unbounded problems are simpler (because no additional clock is needed). Experiments were run on a Pentium 4 with 2.16GHz and 2GB RAM.

6.1 Bouncing Ball

We consider a bouncing ball assembled from different materials. Assume this ball consisting of three parts: 50 per cent of its surface consists of hard material, 25 per cent consists of a material of medium hardness and the rest consists of very soft material. We drop the ball from a height $h = 2$. Further, we assume the gravity constant to be $g = 1$. If the ball hits the floor and falls on the hard side, it will jump up again with one half of the speed it had before, and if it falls down on the medium side it will jump up with a quarter of the speed. However, if it hits the ground with its soft side, it won't jump up again. We assume that the side with which the ball hits the ground is determined randomly, according to the amount of surface covered by each of the materials.

We study the probability that the ball falls on the soft side before a given time bound. Results are given in Table 1. We conducted three main analysis settings. With "interval length" we denote the splitting interval for the `PHAVer`-specific refinement technique, as described in the beginning of Section 6. In the left and medium part of the table, we used partitioning with interval length of 0.15 on the position and speed variables. For the medium part, we used the `PHAVer`-specific convex hull overapproximation [17]. For the right part of the table, we used an interval length of 0.05, and the convex hull overapproximation. Entries for which the analysis did not terminate within one hour are marked by "-".

We ascertain here that, without the convex hull overapproximation, with an interval of length 0.15, we obtain non-trivial upper bounds. However, the analysis time as well as the number of states grows very fast with increasing time bound. The reason for this to happen is, that each time the ball hits the ground, there are three possibilities with which side it will hit the ground. Thus, the number of possibilities for the amount of energy the ball still has after a number of times n the ground is hit is exponential in n. Also notice that there is some time bound up to which the ball has done an infinite number of jumps in all cases, as this case study features Zeno behaviour. This indicates that for time bounds near this value we have to use very small partitioning intervals to obtain realistic probability bounds. This

time bound	interval length 2			interval length 10		
	prob.	build (s)	#states	prob.	build (s)	#states
2	0	0	11	0	0	8
4	0.05	0	43	1	0	12
5	0.097	1	58	1	0	13
20	0.370	20	916	1	1	95
40	0.642	68	2207	0.512	30	609
80	0.884	134	4916	1	96	1717
120	0.940	159	4704	0.878	52	1502
160	0.986	322	10195	0.954	307	4260
180	0.986	398	10760	0.961	226	3768
600	1.0	1938	47609	1	1101	12617

Table 2: Thermostat performance figures

leads to an even larger time and memory consumption in the analysis.

If we use the convex hull overapproximation and an interval length of 0.15, far less resources have to be used. But we only obtain non-trivial results for time bounds up to 3. Using an interval width of 0.05, we obtain a tighter probability bound, while using still less resources than the first configuration. However, for time bound $T = 3.7$ the third configuration does not terminate within one hour.

For a time bound of 3, we can compute the probability manually: the ball can hit the ground with its soft side when it hits the floor in first place with a probability of 0.25. It hits the ground initially with its hard side with a probability of 0.5. In this case, it won't hit the ground again before $T = 3$. If it first hits the ground with its medium hard side, it will hit the ground a second time before $T = 3$. After this, there is no chance of touching the ground again before the time bound. Because of this, the overall probability of the property under consideration is $0.25 + 0.25 \cdot 0.25 = 0.3125$. This means that the result obtained by using an interval length of 0.05 and the convex hull overapproximation is exact.

6.2 Thermostat

We consider the thermostat example discussed in Section 5. For the safety property discussed there, `ProHVer` can verify this nontrivial system and property, and will answer that the system is safe, the upper bound computed is 0.097.

In Table 2, we give further probability bounds and performance statistics (time to build the abstraction—the value iteration time is negligible—and number of constructed abstract states) for different time bounds. For the left (right) part of the table, we set the interval length for the variable x to 2 (respectively 10). The time needed for the analysis as well as the number of states of the abstract transition systems grows about linearly in the time bound, though with oscillations. Comparing the left and the right side, we see that for the larger interval we need less resources, as was to be expected. Due to the way `PHAVer` splits locations along intervals, for some table entries, we see somewhat counter-intuitive behaviour. We observe that bounds do not necessarily improve with decreasing interval length. This is because `PHAVer` computes abstractions without taking into account probabilities, and hence does not guarantee abstractions with smaller intervals to be an improvement of the probability bound, though they are in most cases. Moreover, the abstractions we obtain from `PHAVer` also do not necessarily guarantee probability bounds to increase monotonically with the time bound. This is because a slightly increased time bound might induce an entirely different abstraction, leading to a tighter probability bound, and thus giving the impression of a decrease in probability, even though the actual maximal probability indeed stays the same or increases.

6.3 Water Level Control

We consider a model of a water level control system (extended from the one of Alur et al. [3]) which uses wireless sensors. Values submitted are thus subject to probabilistic delays, due to the unreliable transport medium.

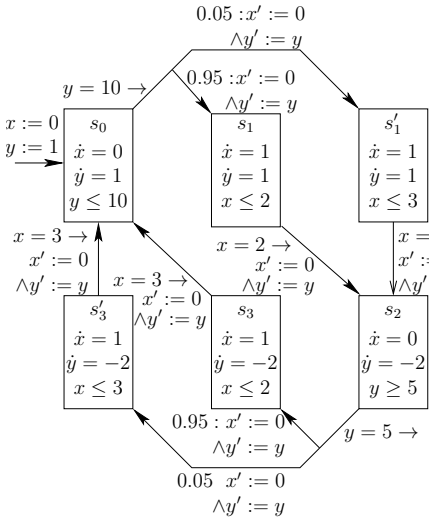

Figure 7: Water level control automaton

A sketch of the model is given in Figure 7. The water level y of a tank is controlled by a monitor. Its change is specified by a linear function. Initially, the water level is $y = 1$. When no pump is turned on (s_0), the tank is filled by a constant stream of water ($\dot{y}$). This water streams out of a larger water tank, which initially has a water level of W. We thus have $\dot{W} = -1$ in all modes of the system, and stop the system as soon as W reaches 0. For clarity, we do not explicitly included the dynamics of W in Figure 7. When a water level of $y = 10$ is seen by a sensor of the tank, the pump should be turned on. However, the pump features a certain delay, which results from submitting control data via a wireless network. With a probability of 0.95 this delay takes 2 time units (s_1), but with a probability of 0.05 it takes 3 time units (s'_1). The delay is realised by the timer x. After the delay has passed, the water is pumped out with a higher speed than it is filled into the tank ($\dot{y} = -2$ in s_2). There is another sensor to check whether the water level is below 5. If this is the case, the pump must turn off again. Again, we have a distribution over delays here (s_3 and s'_3). For the system to work correctly, the water level must stay between 1 and 12.

We are interested in the probability that the pump system violates the property given above, that is either the water level falls below 1 or grows above 12, before the larger water tank from which the water tank is filled becomes empty. We model the system in `ProHVer` and reason about this property: performance statistics are given in Table 3. Without using partitioning, we were only able to obtain exact values for W up to 82 including. Notice that we did not use the convex hull overapproximation, like in the bouncing ball case study, nor another overapproximation. For W larger than this value, we always obtained a probability limit of 1. To get tighter results, we partitioned x by an interval of length 2. For W below 83 we obtain the exact value in both table parts, whereas for 83 we obtain a useful upper bound only when using partitioning. A plot of probabilities for different W is given in Figure 8. The graph has a staircase form where wide steps alternate with narrow ones. This form results, because each time the longer time bound is randomly chosen, the tank will overflow or underflow respectively, if there is enough time left. The wide steps corresponds to the chance of overflow in the tank, the narrow ones to the chance

W	no partitioning			interval length 2		
	prob.	build (s)	#states	prob.	build (s)	#states
40	0.185	0	69	0.185	1	150
82	0.370	0	283	0.370	2	623
83	1	1	288	0.401	2	640
120	1	1	537	0.512	4	1220
500	1	38	3068	0.954	79	7158
1000	1	169	6403	0.998	365	14977

Table 3: Water level control performance figures

of underflow.

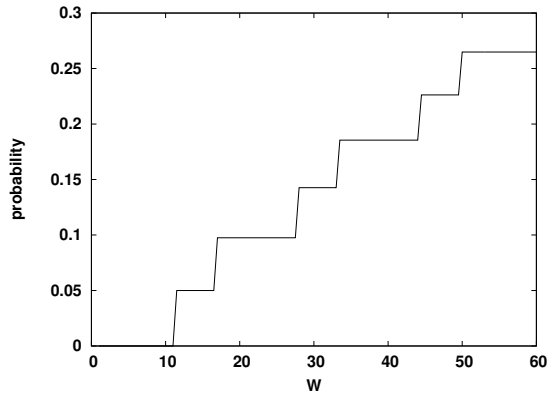

Figure 8: Graph of probabilities in water level control

6.4 Autonomous Lawn-Mower

We consider an autonomous lawn-mower that uses a probability bias to avoid patterns on lawns. This mower is started on a rectangular grassed area. When it reaches the border of the area, it changes its direction. To prevent the mower from following a simple cyclic pattern, this direction is randomly biased, such as to ensure that finally the entire area has been cut.

A sketch of the automaton is given in Figure 9. There, l is the length and h the width of the area. The position of the mower on the area is given by (x, y). With (v_x, v_y) we denote the speed in (x, y) direction, which the mower takes with a probability of 0.95 when reaching a border, whereas with (v'_x, v'_y) denotes a variation of the speed that is taken with probability 0.05. Further, (x_g, y_g) describes the mower's initial position.

At the region with $x \geq 90 \wedge x \leq 100 \wedge y \geq 170 \wedge y \leq 200$ the owner of the lawn has left a tarpaulin. We are interested in the probability that within a time bound of $t = 120$ the mower hits the tarpaulin, thereby inevitably ripping it up.

For the analysis, we set $v_x = 10$, $v_y = 10$, $v'_x = 11$, $v'_y = 9$, $l = 100$, $h = 200$, $x_g = 10$, $y_g = 20$. The creation of the labelled transition system for this automaton took 98 seconds whereas the computation time of the failure probability was negligible. The upper bound we obtained was 0.000281861. We did not use any interval specifications.

As in the other case studies, we also varied the time bound. Results are given in Table 4. For larger time bounds the analysis time as well as the number of states grows quickly. This is due to a similar effect as in the bouncing ball case study. Each time the mower reaches a

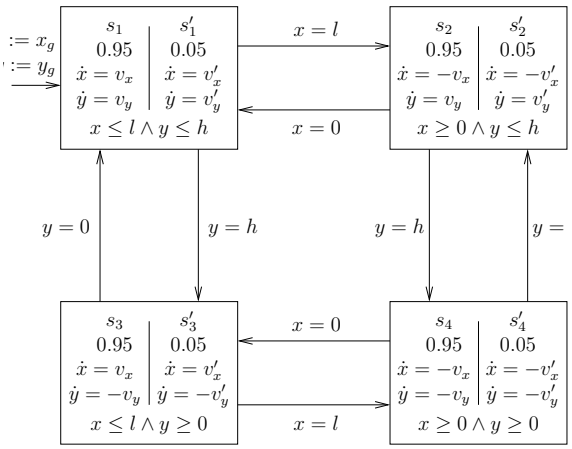

Figure 9: Probabilistic Autonomous Lawn-Mower

time bound	prob.	build (s)	#states
10	0	0.06	4
70	$1.11984 \cdot 10^{-05}$	1.22	632
100	$1.11984 \cdot 10^{-05}$	8.47	3022
110	0.000281861	65.55	9076
120	0.000281861	98.83	12660
130	0.000281861	303.40	25962
140	0.000281861	743.43	38830

Table 4: Lawn-mower performance figures

border, it may head into two different directions, which leads to a combinatorial explosion, evident by the above statistics.

We also experimented with convex hull overapproximation without interval refinement. However, the probability bounds obtained were always 1. Using interval partitioning also did not improve on this, as it only made the analyses take more time. We feel that for this case study there is not much hope of obtaining better results in resource usage. The complexity does not really live in the hybrid behaviour, but results from the excessive number of ways the mower can pass around the area. Because of this, we don't think it is possible to find an abstraction to handle this case study for larger time bounds.

6.5 Summary of Case Studies

We successfully applied our method on a number of case studies. However, it seems worthwhile to explore techniques more adapted to the generation of transition systems for probabilistic systems, especially by adjusting the splitting of states to a method better adapted to our needs. We may do so, for instance, by finding means to guarantee non-decreasing probability bounds when smaller intervals are used. Another possibility would be to do a more fine-grained splitting at places where this is useful to decrease the probability bound, while saving space by a coarser abstraction for parts of the model where this is sufficient to obtain a tight probability bound. Mostly, the upper bounds we could obtain were tight or exact (checked by manual inspection).

In Table 5, we give running times for the largest instances completed successfully: Here, ✔ means success (exact or tight upper bounds) while ✗/✔ means useful results for only some instances. Our implementation is effective in most cases considered, but there is

ball	thermostat	water	mower
214s ✔	398s ✔	365s ✔	743s ✗/✔

Table 5: Largest instances of case studies completed successfully

still room for improvement in the ProHVer implementation to handle more complex case studies. PHAVer files of the case studies can be found on the homepage of ProHVer:

http://depend.cs.uni-sb.de/tools/prohver

7 Conclusions

In this paper we have discussed how to check safety properties for probabilistic hybrid automata. These models and properties are of central importance for the design and verification of emerging wireless and embedded real-time applications. Moreover, being based on arbitrary abstractions computed by tools for the analysis of non-probabilistic hybrid automata, improvements in effectivity of such tools directly carry over to improvements in effectivity of the technique we describe. The applicability of our approach has been demonstrated on a number of case studies, tackled using a prototypical implementation.

As future work we are investigating whether our approach can be adapted to the safety verification problem for more general probabilistic hybrid systems [9, 11], that is, systems with stochastic differential equations instead of ordinary differential equations.

Acknowledgement. The work of Lijun Zhang was partially supported by MT-LAB, a VKR Centre of Excellence. Part of this work was done while Lijun Zhang was at *Saarland University* and at the *Computer Science, University of Oxford*. The work of Zhikun She was partly supported by NSFC-61003021 and Beijing Nova Program, the one of Stefan Ratschan has been supported by GAČR grant 201/08/J020, MŠMT grant OC10048, and by the institutional research plan AV0Z100300504. Holger Hermanns and Ernst Moritz Hahn were supported by the NWO-DFG bilateral project ROCKS, by the DFG as part of the Transregional Collaborative Research Centre SFB/TR 14 AVACS, and by the European Community's Seventh Framework Programme under grant agreement n^o 214755. The work of Ernst Moritz Hahn has been partially supported by the Graduiertenkolleg "Leistungsgarantien für Rechnersysteme".

References

[1] A. Abate, M. Prandini, J. Lygeros, and S. Sastry. Probabilistic reachability and safety for controlled discrete time stochastic hybrid systems. *Automatica*, 44(11):2724–2734, 2008.

[2] E. Altman and V. Gaitsgory. Asymptotic optimization of a nonlinear hybrid system governed by a Markov decision process. *SIAM Journal of Control and Optimization*, 35(6):2070–2085, 1997.

[3] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

[4] R. Alur, T. Dang, and F. Ivancic. Predicate abstraction for reachability analysis of hybrid systems. *ACM Transactions on Embedded Computing Systems*, 5:152–199, 2006.

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

[6] D. Bertsekas. *Dynamic Programming and Optimal Control*, volume 2. Athena Scientific, 1995.

[7] A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In P. S. Thiagarajan, editor, *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 1026 of *Lecture Notes in Computer Science (LNCS)*, pages 499–513. Springer Berlin / Heidelberg, 1995.

[8] H. Blom and J. Lygeros. *Stochastic Hybrid Systems: Theory and Safety Critical Applications*, volume 337 of *Lecture Notes in Control and Information Sciences (LNCIS)*. Springer Berlin / Heidelberg, 2006.

[9] M. L. Bujorianu. Extended stochastic hybrid systems and their reachability problem. In R. Alur and G. J. Pappas, editors, *Hybrid Systems: Computation and Control (HSCC)*, volume 2993 of *Lecture Notes in Computer Science (LNCS)*, pages 234–249. Springer Berlin / Heidelberg, 2004.

[10] M. L. Bujorianu and J. Lygeros. Toward a general theory of stochastic hybrid systems. In Blom and Lygeros [8], pages 3–30.

[11] M. L. Bujorianu, J. Lygeros, and M. C. Bujorianu. Bisimulation for general stochastic hybrid systems. In M. Morari and L. Thiele, editors, *Hybrid Systems: Computation and Control (HSCC)*, volume 3414 of *Lecture Notes in Computer Science (LNCS)*, pages 198–214. Springer Berlin / Heidelberg, 2005.

[12] M. L. Bujorianu, J. Lygeros, and R. Langerak. Reachability analysis of stochastic hybrid systems by optimal control. In M. Egerstedt and B. Mishra, editors, *Hybrid Systems: Computation and Control (HSCC)*, volume 4981 of *Lecture Notes in Computer Science (LNCS)*, pages 610–613. Springer Berlin / Heidelberg, 2008.

[13] E. Clarke, A. Fehnker, Z. Han, B. Krogh, O. Stursberg, and M. Theobald. Verification of hybrid systems based on counterexample-guided abstraction refinement. In H. Garavel and J. Hatcliff, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 2619 of *Lecture Notes in Computer Science (LNCS)*, pages 192–207. Springer Berlin / Heidelberg, 2003.

[14] P. R. D'Argenio, B. Jeannet, H. E. Jensen, and K. G. Larsen. Reachability analysis of probabilistic systems by successive refinements. In L. de Alfaro and S. Gilmore, editors, *Process Algebra and Probabilistic Methods, Performance Modeling and Verification: Joint International Workshop (PAPM-PROBMIV)*, volume 2165 of *Lecture Notes in Computer Science (LNCS)*, pages 39–56. Springer Berlin / Heidelberg, 2001.

[15] M. H. A. Davis. Piecewise-deterministic Markov processes: A general class of non-diffusion stochastic models. *Journal of the Royal Statistical Society*, 46(3):353–388, 1984.

[16] M. Fränzle, H. Hermanns, and T. Teige. Stochastic satisfiability modulo theory: A novel technique for the analysis of probabilistic hybrid systems. In M. Egerstedt and B. Mishra, editors, *Hybrid Systems: Computation and Control (HSCC)*, volume 4981 of *Lecture Notes in Computer Science (LNCS)*, pages 172–186. Springer Berlin / Heidelberg, 2008.

[17] G. Frehse. PHAVer: Algorithmic verification of hybrid systems past HyTech. In M. Morari and L. Thiele, editors, *Hybrid Systems: Computation and Control (HSCC)*, volume 3414 of *Lecture Notes in Computer Science (LNCS)*, pages 258–273. Springer Berlin / Heidelberg, 2005.

[18] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata. *Journal of Computer and System Sciences*, 57:94–124, 1998.

[19] H. Hermanns, B. Wachter, and L. Zhang. Probabilistic CEGAR. In A. Gupta and S. Malik, editors, *Computer-Aided Verification (CAV)*, volume 5123 of *Lecture Notes in Computer Science (LNCS)*, pages 162–175. Springer Berlin / Heidelberg, 2008.

[20] J. Hu, J. Lygeros, and S. Sastry. Towards a theory of stochastic hybrid systems. In N. A. Lynch and B. H. Krogh, editors, *Hybrid Systems: Computation and Control (HSCC)*, volume 1790 of *Lecture Notes in Computer Science (LNCS)*, pages 160–173. Springer Berlin / Heidelberg, 2000.

[21] B. Jonsson and K. G. Larsen. Specification and refinement of probabilistic processes. In *IEEE Symposium on Logic in Computer Science (LICS)*, pages 266–277. IEEE Computer Society, 1991.

[22] A. A. Julius. Approximate abstraction of stochastic hybrid automata. In J. P. Hespanha and A. Tiwari, editors, *Hybrid Systems: Computation and Control (HSCC)*, volume 3927 of *Lecture Notes in Computer Science (LNCS)*, pages 318–332. Springer Berlin / Heidelberg, 2006.

[23] A. A. Julius and G. J. Pappas. Approximations of stochastic hybrid systems. *Automatic Control, IEEE Transactions on*, 54(6):1193–1203, 2009.

[24] M. Kwiatkowska, G. Norman, and D. Parker. Stochastic games for verification of probabilistic timed automata. In J. Ouaknine and F. W. Vaandrager, editors, *Int'l Conf. on Formal Modeling and Analysis of Timed Systems (FORMATS)*, volume 5813 of *Lecture Notes in Computer Science (LNCS)*, pages 212–227. Springer Berlin / Heidelberg, 2009.

[25] M. Z. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, 282(1):101–150, 2002.

[26] G. Lafferriere, G. J. Pappas, and S. Yovine. A new class of decidable hybrid systems. In F. W. Vaandrager and J. H. van Schuppen, editors, *Hybrid Systems: Computation and Control (HSCC)*, volume 1569 of *Lecture Notes in Computer Science (LNCS)*, pages 137–151. Springer Berlin / Heidelberg, 1999.

[27] J. Preußig, S. Kowalewski, H. Wong-Toi, and T. Henzinger. An algorithm for the approximative analysis of rectangular automata. In A. P. Ravn and H. Rischel, editors, *Formal Techniques in Real-Time and Fault Tolerant System (FTRTFT)*, volume 1486 of *Lecture Notes in Computer Science (LNCS)*, pages 228–240. Springer Berlin / Heidelberg, 1998.

[28] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 1994.

[29] S. Ratschan. Safety verification of non-linear hybrid systems is quasi-semidecidable. In *TAMC 2010: 7th Annual Conference on Theory and Applications of Models of Computation*, volume 6108 of *LNCS*, pages 397–408. Springer, 2010.

[30] S. Ratschan and Z. She. Safety verification of hybrid systems by constraint propagation-based abstraction refinement. *ACM Transactions on Embedded Computing Systems*, 6(1), 2007.

[31] R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing (NJC)*, 2(2):250–273, 1995.

[32] J. Sproston. Decidable model checking of probabilistic hybrid automata. In M. Joseph, editor, *Formal Techniques in Real-Time and Fault Tolerant System (FTRTFT)*, volume 1926 of *Lecture Notes in Computer Science (LNCS)*, pages 31–45. Springer Berlin / Heidelberg, 2000.

[33] T. Teige and M. Fränzle. Constraint-based analysis of probabilistic hybrid systems. In A. Giua, C. Mahulea, M. Silva, and J. Zaytoon, editors, *Analysis and Design of Hybrid Systems (ADHS)*, pages 162–167. IFAC, 2009.

[34] L. Zhang, Z. She, S. Ratschan, H. Hermanns, and E. M. Hahn. Safety verification for probabilistic hybrid systems. In T. Touili, B. Cook, and P. Jackson, editors, *Computer-Aided Verification (CAV)*, volume 6174 of *Lecture Notes in Computer Science (LNCS)*, pages 196–211. Springer Berlin / Heidelberg, 2010.