Rewarding Probabilistic Hybrid Automata

Ernst Moritz Hahn
University of Oxford, United Kingdom*  
Holger Hermanns
Saarland University, Germany

ABSTRACT
The joint consideration of randomness and continuous time is important for the formal verification of many real systems. Considering both facets is especially important for wireless sensor networks, distributed control applications, and many other systems of growing importance. Apart from proving the quantitative safety of such systems, it is important to analyse properties related to resource consumption (energy, memory, bandwidth, etc.) and properties that lie more on the economical side (monetary gain, the expected time or cost until termination, etc.). This paper provides a framework to decide such reward properties effectively for a generic class of models which have a discrete-continuous behaviour and involve both probabilistic as well as nondeterministic decisions. Experimental evidence is provided demonstrating the applicability of our approach.


General Terms: Performance; Reliability; Verification.

Keywords: probabilistic hybrid automaton; abstraction; model checking; expected rewards; probabilistic automaton; performance evaluation; performability; continuous time; nondeterminism; simulation relation.

1. INTRODUCTION
The inclusion of stochastic phenomena in the hybrid systems framework is crucial for a spectrum of application domains, ranging from wireless communication and control to air traffic management and to electric power grid operation [13, 23]. As a consequence, many different stochastic hybrid system models have been proposed [2, 35, 11, 12, 1, 28], together with a vast body of mathematical tools and techniques.

Recently, model checkers for stochastic hybrid systems have emerged [36, 40, 17]. In this context, the model of probabilistic hybrid automata [35] is of particular interest, since it pairs expressiveness and modelling convenience in a way that model checking is indeed possible. In particular, it enables to piggyback on existing hybrid system solvers. Tak

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

HSCC'13, April 8–11, 2013, Philadelphia, Pennsylvania, USA.
Copyright 2013 ACM 978-1-4503-1567-8/13/04 ...$15.00.

on approximating or bounding reach probabilities in probabilistic hybrid automata. This is appropriate for quantifying system safety and reliability, but not for availability, survivability, throughput and resource consumption questions.

In this paper, we aim to overcome this restriction in a framework that is as general as possible, while retaining the idea of piggybacking on existing hybrid system solvers. Taking up initial ideas [22], we decorate probabilistic hybrid automata with rewards which can be considered as costs or bonuses. We discuss a method for handling properties that quantify expected rewards. The properties we consider are either the minimal or maximal expected total accumulated reward over all executions of a model or the minimal or maximal expected time-average reward over all executions of the model. We will need to postpone the precise formalisation of these notions (cf. Definition 15), until we have defined the semantics of our models. Using appropriate reward structures and properties, this approach allows us to reason about the cost accumulated until termination, the long-run cost of system operation, system availability [16] and survivability [14], time or cost until stabilisation, and many other properties of interest. Proofs backing up the results presented in this paper can be found in [21].

Related Work. Reward properties for classical (i.e. nonstochastic) timed automata have been considered by Bouyer et al. [10, 9]. Rutkowski et al. [32] considered a controller synthesis problem for average-reward properties in classical hybrid automata. Discrete-time stochastic hybrid automata have been considered for the analysis of reward properties [37, 36, 18], and have been studied with importance sampling techniques recently [41]. Methods which approximate continuous-time stochastic hybrid automata by Markov chains [29, 24] also allow for an extension to reward-based properties. To the best of our knowledge, the present paper is the first to address reward-based properties of probabilistic hybrid automata involving nondeterministic, stochastic behaviour as well as continuous time in full generality, harvesting well-understood and effective methods originally developed for the verification of classical hybrid automata.

2. PROBABILISTIC HYBRID AUTOMATA

This section introduces the notion of probabilistic hybrid automata we are going to use, and describes how rewards are integrated into the model. To get started, we first define a generic multi-dimensional post operator, which will be used to describe the continuous behaviour of our model. In this operator, we reserve the first two dimensions for the accumulation of reward, respectively the advance of time. In the context of hybrid systems [3, 4], post operators are often described by differential (in)equations. However, our notion is independent of the formalism used.

Definition 1. A k-dimensional post operator with $w \in \mathbb{N}$ and $k \geq 2$ is a function

$$
Post : \mathbb{R}^k \rightarrow 2^{\mathbb{R}^k}.
$$
Post\((r, t, v)\) will be used to describe the possible values of the continuous variables after a timed transition. This implies an update of the reward and the time dimension. If there is a constant \(c \in \mathbb{R}_{\geq 0}\) satisfying that for any \(r, t \in \mathbb{R}, v \in \mathbb{R}^{k-2}\),
\[
\text{Post}\((r, t, v)\) \subseteq \{(r + ct, t + t) \mid t \in \mathbb{R}_{\geq 0}\} \times \mathbb{R}^{k-2},
\]
we call the post operator reward affine. Models which use only reward affine post operators will turn out to allow for abstractions which are particularly precise.

**Example 1.** Consider \(\text{Post}_{\text{Check}} : \mathbb{R}^3 \to 2^\mathbb{R}^3\) with
\[
\text{Post}_{\text{Check}}(r, t, T) = \{(r + t + T \exp(-0.5t)) \mid t \in \mathbb{R}_{\geq 0} \land t + T \leq 1\}.
\]

For \((r, t, T) = (0, 0, 5)\), the behaviour is depicted in Figure 1. The graph denotes the set of points which can be reached by a timed transition. The axis labelled with \(t\) denotes both the values of the time passed as well as the continuous variable \(r\) (and here also the value of variable \(r\)). The axis \(T\) displays the third dimension. After time 0.25, it has a value of \(\approx 4.41\). Post operators will appear in the definition of the probabilistic hybrid automaton model we consider. As a preparation, we first define classical hybrid automata.

**Definition 2.** A classical hybrid automaton (HA) is a tuple
\[
\mathcal{H} = (M, k, \pi, \{\text{Post}_m\}_{m \in M}, \text{Cmds}, \text{Rew}),
\]
where
- \(M\) is a finite set of modes,
- \(k \in \mathbb{N}\) with \(k \geq 2\) is the dimension,
- \(\pi \in M\) is the initial mode,
- \(\text{Post}_m\) is a \(k\)-dimensional post operator for each \(m\),
- \(\text{Cmds}\) is a finite set of guarded commands of the form
  \[
g \to u,\ 
\]
  where
  - \(g \subseteq M \times \mathbb{R}^k\) is a guard,
  - \(u : (M \times \mathbb{R}^k) \to 2^{M \times \mathbb{R}^k}\) is an update function with
    \[
    u(s) \subseteq M \times \{(0, 0)\} \times \mathbb{R}^{k-2},
    \]
  - if \(s \in g\) then \(u(s) \neq \emptyset\),
- for each \(s = (m, v)\) with \(\text{Post}_m(v) = \emptyset\), there is a command with guard \(g\) and \(s \in g\), and
- \(\text{Rew} : ((M \times \mathbb{R}^k) \times \text{Cmds}) \to \mathbb{R}_{\geq 0}\) is a reward structure.

The continuous-time behaviour of an HA in a given mode \(m\) is determined by the corresponding post operator. Whenever the guard of a guarded command is satisfied, the command can be executed in zero time. If executed, a nondeterministic choice over successor updates and modes results. Multiple guards of commands may be satisfied at the same time, implying a nondeterministic selection over these commands.

Another obvious concept needed for the setting considered is that of a probability distribution.

**Definition 3.** A finite probability distribution over a set \(\Omega\) is a function \(\mu : \Omega \to [0, 1]\), where there are only finitely many \(a \in \Omega\) with \(\mu(a) > 0\), and it is \(\sum_{a \in \Omega} \mu(a) = 1\). In a Dirac probability distribution \(\mu\), there is only a single \(a \in \Omega\) with \(\mu(a) = 1\). With \(\text{Distr}(\Omega)\), we denote the set of all finite probability distributions over \(\Omega\). Given \(n\) pairwise different elements \(a_1, \ldots, a_n \in \Omega\) and probabilities \(p_i \geq 0\), \(1 \leq i \leq n\) with \(\sum_{i=1}^{n} p_i = 1\), we use \([a_1 \to p_1, \ldots, a_n \to p_n]\) to denote the probability distribution \(\mu \) with \(\mu(a_i) = p_i\).

With this extension, we can now specify probabilistic hybrid automata (similar to Sproston [35] Section 2).

**Definition 4.** A probabilistic hybrid automaton (PHA) is a tuple
\[
\mathcal{H} = (M, k, \pi, \{\text{Post}_m\}_{m \in M}, \text{Cmds}, \text{Rew}),
\]
where all components of this tuple are as in **Definition 2** and satisfy the same constraints, except for
- \(\text{Cmds}\), which is a finite set of probabilistic guarded commands of the form
  \[
g \to [a_1 \to p_1, \ldots, a_n \to p_n],\ 
\]
  where
  - \(g \subseteq M \times \mathbb{R}^k\) is a guard,
  - \(a_i : (M \times \mathbb{R}^k) \to 2^{M \times \mathbb{R}^k}\) is an update function with
    \[
    a_i(s) \subseteq M \times \{(0, 0)\} \times \mathbb{R}^{k-2},
    \]
  - if \(s \in g\) then \(a_i(s) \neq \emptyset\) for \(1 \leq i \leq n\).

\(\mathcal{H}\) is reward affine if all its post operators are reward affine and if \(\text{Rew}(\cdot, c)\) is constant for all \(c \in \text{Cmds}\). Under these assumptions we can consider \(\text{Rew}\) as a function
\[
\text{Rew} : \text{Cmds} \to \mathbb{R}_{\geq 0}.
\]
Probabilities are interpreted in this definition as part of the commands that update mode and variables according to the probabilities \(p_i\) associated with the \(i\)th update option \(a_i\).

A HA can now be viewed as a PHA where each guarded command has only a single update option, to be chosen by a Dirac distribution, or (a possibly uncountable) nondeterministic choice over Dirac distributions.

**Example 2.** Figure 2 depicts a PHA model of a simple unreliable thermostat, where \(r = 0\) and \(l = 1\) in each mode. The model can switch between modes Heat and Cool to adjust the temperature of its environment. At certain occasions the system may enter a Cool mode. The post operator of Check has been described in **Example 1**, the other ones are similar. On execution of the command \(c_{\text{Ch}}\), the system moves to mode Heat with probability 0.95, and to mode Error with probability 0.05. We thus have
\[
c_{\text{Ch}} = (g \to [u_{\text{ChH}} \to 0.95, u_{\text{ChE}} \to 0.05]),\ 
\]
where
- \(g = \{\text{Check}\} \times \mathbb{R} \times [0.5, \infty) \times \mathbb{R}\),
- \(u_{\text{ChH}}(m, r, t, T) = \{\{\text{Heat}, (0, 0, T)\}\},\) and
- \(u_{\text{ChE}}(m, r, t, T) = \{\{\text{Error}, (0, 0, T)\}\}.

The formalisation of the other commands is similar, but does not include nontrivial probabilities.
In the post operators, we have already integrated means to refer to reward and time. Indeed, the first two dimensions of a PHA are used to record reward accumulation, and time advances as the system lifetime time progresses. We will “collect” the reward and times whenever a command is executed, and therefore reset these dimensions whenever executing commands. The component Rew associates rewards to discrete transitions of a PHA.

As time and reward are present as explicit dimensions in our construction, guards and invariants can relate to them. This is in contrast to e.g. priced timed automata [10], where this is forbidden, so as to avoid crossing the undecidability boundary. In the setting considered here, this makes no difference because we build on machinery (for HAs) that is developed for undecidable theories, in form of heuristics.

Example 3. Consider the thermostat of Figure 2. Here it is not possible to leave the Error mode once entered. We define a reward structure Rewact with Rewact(c) ≡ 0 if c ∈ {q0, q1} and Rewact(c) ≡ 1 else. The system thus earns a reward for executing any command, except for the one of Error, and the one to initialise the system. With this reward structure, the minimal sum of reward values accumulated, expresses the minimal expected number of commands executed until an error happens.

Now, assume we have extended mode Error so that the system can recover after a certain time (e.g. by adding a reset transition back to the initial state). In such a system, it makes sense to consider the long-run behaviours. We can for instance look at a reward structure Rewact assigning constant 0 to each command. If in addition we modify the post operator in such a way that r is increased by 1 per time unit in mode Error, we can use this to reason about the percentage of time the system is not operational on the long run.

3. PHA SEMANTICS

In this section, we describe the semantics of PHAs. The semantics maps variations of infinite-state Markov decision processes [30], known as probabilistic automata [33].

Definition 5. A probabilistic automaton (PA) is a tuple M = (S, A, Act, T), where
- S is a set of states,
- A ∈ S is the initial state,
- Act is a set of actions, and the
- transition matrix T : (S × Act) → 2^{Distr(S)} assigns sets of probability distributions to state-action pairs.

For each s ∈ S, we require {a ∈ Act | T(s, a) ≠ ∅} ≠ ∅. PAs contain a (possibly uncountable) set of states, whereof one is initial. In each s ∈ S, there is a nondeterministic choice of actions a ∈ Act and distributions µ ∈ T(s, a) over successor states.

Example 4. In Figure 3 we depict a finite example PA M = (S, Act, T). Here, we have S = {s0, s1, s2}, Act = {a, b}, and

\[ T(s_0, a) \equiv \emptyset, T(s_0, b) \equiv \{(s_0 \mapsto 0.25, s_1 \mapsto 0.75), [s_1 \mapsto 1]\}, \]
\[ T(s_1, a) \equiv \{(s_1 \mapsto 1), [s_2 \mapsto 1]\}, T(s_1, b) \equiv \emptyset, \]
\[ T(s_2, a) \equiv \emptyset, T(s_2, b) \equiv \{(s_2 \mapsto 1), [s_0 \mapsto 0.25, s_1 \mapsto 0.75]\}. \]

Definition 6. A finite path of a PA M = (S, Act, T) is a tuple

\[ b_n = s_0a_0a_1 \ldots a_{n-1}a_n \in (S × Act × Distr(S))^n × S, \]
where so = s and for all i with 0 ≤ i < n it is µi ∈ T(s_i, a_i). An infinite path is a tuple

\[ b_\text{inf} = s_0a_0a_1 \ldots ∈ (S × Act × Distr(S))^\omega, \]
where so = s and µi ∈ T(s_i, a_i) holds for all i ≥ 0. By PathfinM we denote the set of all finite paths and by PathinfM we denote the set of all infinite paths of M.

We let βfin[i] ≡ βinf[i] ≡ s, denote the (i + 1)th state of a finite or infinite path (for the i-s defined). By last(βfin) ≡ sn we denote the last state of a finite path. For β, β′ ∈ PathfinM ⊔ PathinfM we write β ≤ β′ in case either β = β′ or if β is a finite prefix of β′.

By trace(βfin) = a0a1 \ldots an, we denote the trace of a finite path, and accordingly for infinite paths. The sets of all finite and infinite traces are defined as TracefinM ≡ Act* and TraceinfM ≡ Actω. Given γ = a0a1 \ldots ∈ TracefinM ⊔ TraceinfM, we define γ[i] ≡ ai, as the (i + 1)th action on the trace.

Consider a subset Actfair ⊆ Act of the actions of M. We consider a path β ∈ PathfinM as Actfair-fair if there are infinitely many i ≥ 0 with trace(β)[i] ∈ Actfin. By PathfinM Actfair we denote the set of all Actfair-fair paths of M.

Example 5. A finite path in the PA of Figure 3 is

\[ b_n ≡ 0s_0b[s_0 \mapsto 0.25, s_1 \mapsto 0.75]s_0, \]
and with Actfair = \{b\}, an Actfair-fair infinite path is

\[ b_\text{inf} ≡ 0s_0(b[s_0 \mapsto 0.25, s_1 \mapsto 0.75]s_0)^\omega. \]

We have

\[ \text{trace}(b_\text{fin}) = b, \text{trace}(b_\text{inf}) = b^\omega, \]
\[ \text{last}(b_\text{fin}) = s_0, b_\text{inf}[0] = s_0, b_\text{inf}[15] = s_0. \]

Path sets are by themselves not sufficient to describe the properties of PAs. This is because nondeterministic behaviour is intertwined with probabilistic behaviour. This asks for instances to resolve the nondeterminism. These instances, called schedulers, induce a purely probabilistic behaviour, which can be subjected to stochastic analyses.

Definition 7. A scheduler for PA M = (S, Act, T) is a function ρ : PathfinM → Distr(ACT × Distr(S)). For β ∈ PathfinM, we require ρ(β)(a, µ) > 0 implies µ ∈ T(last(β), a).

With SchedM we denote the set of schedulers of M.

A scheduler ρ is called simple if it only maps to Dirac distributions and if for all β, β′ ∈ PathfinM with last(β) = last(β′) we have ρ(β) = ρ(β′). We can interpret it as being of the form ρ : S → (ACT × Distr(S)).

In this paper, we develop results valid for general schedulers, thus not restricted to simple schedulers. But since simple schedulers are simpler to describe and understand, they appear in illustrating examples. We are now in the position to define probability measures on paths.

Definition 8. We define PrM,ρ : PathfinM → [0,1] for PA M = (S, Act, T) and scheduler ρ : PathfinM → Distr(ACT ×
We define two stochastic processes associated to a PA. Let \(\rho\) be a reward structure of \(\mathcal{M}\) and \(\mathcal{D}_{\text{num}}\) be the accumulated reward would be infinite, so that we only consider the long-run average value, which we consider as being unrealistic. Fairness can prevent this problem.

**Example 7.** Reconsider the PA of Figure 3. In this model, the accumulated reward would be infinite, so that we only consider the fractional long-run average reward. We derive for \(\text{Act}_{\text{fair}}\) that

\[
\text{val}^{+}_{\text{Act}_{\text{fair}}} = \frac{12}{5} = 2.4, \quad \text{val}^{-}_{\text{Act}_{\text{fair}}} = \frac{7}{3} \approx 2.333,
\]

which is for instance obtained by the simple schedulers \(\rho^+\) and \(\rho^-\) with

\[
\rho^+(s_0) = \{b, [s_0, 0.25, s_1, 0.75]\}, \quad \rho^+(s_1) = \{a, [s_2, 1]\},
\]

\[
\rho^-(s_2) = \{b, [s_1, 1]\}, \quad \rho^-(s_1) = \{a, [s_2, 1]\},
\]

We now define the semantics of PHAs as PAs formally.

**Definition 13.** For a PA \(\mathcal{H} = (M, k, \overline{m}, \langle \text{Post}_{\text{num}} \rangle_{m \in M}, \text{Cmds}, \text{Rew})\), the semantics is a PA

\[
[\mathcal{H}] = (S, \overline{\pi}, \text{Act}, T), \quad \text{where}
\]

- \(S = M \times \mathbb{R}_k\),
- \(\overline{\pi} = (\overline{m}, 0, \ldots, 0)\),
- \(\text{Act} = \text{Cmds} \cup \{\tau\}\),
- for \(s = (m, v) \in S\) we have that

\[
\text{Distr}(S): \text{Given } \beta = s_0 a_0 \overline{m}_0 s_1 a_1 \overline{m}_1 \ldots s_n \in \text{Path}^\text{inf}_{\mathcal{M}}, \text{ let}
\]

\[
\text{Pr}_{\mathcal{M}, \rho}(\beta) \overset{\text{def}}{=} \rho(s_0)(a_0, \overline{m}_0)(s_1)(a_1, \overline{m}_1)(s_2) \cdots \mu_{n-1}(s_n).
\]
for $c = (g \rightarrow [u_1 \mapsto p_1, \ldots, u_n \mapsto p_n]) \in \text{Cmds}$ it is $T(s, c) \equiv \emptyset$ if $s \notin g$ and else:

$$T(s, c) \equiv \{ \mu \in \text{Distr}(S) \mid \exists s' \in u_1(s), \ldots, s'_n \in u_n(s). \forall s' \in S. \mu(s') = \sum_{s'_i = s'_i} p_i \},$$

and it is $T(s, \tau) \equiv \{(m, v') \mapsto 1 \mid v' \in \text{Post}_m(v)\}$.

The semantics is similar to usual notions of HAs, which are usually given in terms of labelled transition systems [29]. The difference is in the probabilistic guarded commands, where we can have a probabilistic choice over successor states in addition to nondeterminism.

**Example 8. Consider the PHA $H$ of Figure 2.** Then $\llbracket H \rrbracket = (S, \pi, Act, T)$, with

- $S = M \times \mathbb{R}^3$,
- $\pi = (\text{Init}, 0, 0, 0)$,
- $Act = \{c_{\text{H}}, c_{\text{C}, \text{C}}, c_{\text{C}, \text{H}}, c_{\text{C}, \text{E}}, c_{\text{E}, \text{C}, \text{H}}, \tau\}$,
- $T : (S \times Act) \rightarrow \text{Distr}(S)$.

For $s = (\text{Check}, r, t, T) \in \{\text{Check}\} \times \mathbb{R}^3$, we have

$$T(s, c_{\text{H}}) = T(s, c_{\text{C}, \text{C}}) = T(s, c_{\text{C}, \text{H}}) = T(s, c_{\text{C}, \text{E}}) = T(s, c_{\text{E}, \text{C}, \text{H}}) = \emptyset,$$

it is $T(s, c_{\text{E}}) = \emptyset$ if $t < 0.5$ and we have $T(s, c_{\text{C}, \text{H}}) = \{(0.0, 0.0, T) \mapsto 0.95, (\text{Error}, 0.0, T) \mapsto 0.05\}$ else. Further,

$$T(s, \tau) = \{(\text{Check}, r, t + \tau, T \exp((-0.5t)) \mapsto 1) \mid t \in \mathbb{R}_{\geq 0} \land t + \tau \leq 1\}.$$

With these preparations, we can define the semantics of reward structures of PHAs.

**Definition 14. Given PHA $H = (M, k, \pi, (\text{Post}_m)_{m \in M}, \text{Cmds}, \text{Rew})$ with semantics $\llbracket H \rrbracket = (S, \pi, Act, T)$, the reward semantics is the reward structure $\text{rew}(H) \equiv (\text{rew}_{\text{num}}, \text{rew}_{\text{dan}})$ associated to $\llbracket H \rrbracket$. For $s = (m, r, t, v) \in S$ and $c \in \text{Cmds}$, let $\text{rew}_{\text{num}}(s, c) \equiv \text{Rew}(s, c) + r$, $\text{rew}_{\text{dan}}(s, c) \equiv t$, and $\text{rew}_{\text{num}}(s, \tau) \equiv 0$**.

In this definition, whenever a command is executed, the reward of this command is obtained. Additionally, the timed rewards and the time accumulated until the execution of this command become effective here. As in our model only paths of infinitely many commands are relevant, this is equivalent to a reward semantics in which rewards and passage of time are attached directly to timed transitions. By postponing the collection of timed rewards to the execution of subsequent commands, we will be able to simplify the computation of abstractions of PHAs.

We are now in the position to define the values of reward properties, the technical core of our approach, using the semantics of PHAs and their reward structures.

**Definition 15. Given PHA $H = (M, k, \pi, (\text{Post}_m)_{m \in M}, \text{Cmds}, \text{Rew})$, we define the maximal and minimal time-average reward as $\text{val}_{\text{M}, \text{Ira}}^{\text{H}, \text{Ira}} \equiv \text{val}_{\text{H}, \text{Ira}}^{\text{H}, \text{Rew}(H), \text{Ira}}$ and $\text{val}_{\text{M}, \text{Ira}}^{\text{H}, \text{Ira}} \equiv \text{val}_{\text{H}, \text{Rew}(H), \text{Ira}}^{\text{H}, \text{Rew}(H), \text{Ira}}$ and define the accumulated rewards accordingly as $\text{val}_{\text{M}, \text{Acc}}^{\text{H}, \text{Acc}} \equiv \text{val}_{\text{H}, \text{Acc}}^{\text{H}, \text{Rew}(H), \text{Acc}}$ and $\text{val}_{\text{M}, \text{Acc}}^{\text{H}, \text{Acc}} \equiv \text{val}_{\text{H}, \text{Rew}(H), \text{Acc}}^{\text{H}, \text{Rew}(H), \text{Acc}}$. We only optimise over fair schedulers, because otherwise, we could assign a relevant probability mass to paths which are time convergent [9] Chapter 9, that is their trace will end in a sequence $\tau \tau \tau \ldots$ corresponding to time durations $t_0 t_1 t_2 \ldots$ with $\sum_{i=0}^{\infty} t_i < \infty$. This way, time effectively stops, which means that only the reward up to this point of time will be taken into account, which is unrealistic. Now assume that infinitely many commands are executed, and that the automaton is structurally (strongly) nonzero [89]. Definition 6, that is the guards are defined so that they cannot be executed without a minimal fixed delay. Then this ensures the time divergence of the path.

One point is worth noting. One might be interested in models in which it is a legal behaviour to eventually reside in a mode $m$ of the model without further executing any commands. However, the definition above requires that infinitely many commands are executed on each legal path. Because of this, such models have to be adapted accordingly. This can be done, e.g. by adding a new auxiliary command $e_m$ which can be executed infinitely often after a given delay whenever residing in $m$.

### 3.1 Expressing Properties

Table 1 provides an overview of common properties that are expressible using the mechanisms described. Here, $F$ denotes the set of failed states of the PHA, and $T$ describes states in which operation has terminated. The availability [10] of a system can then be expressed as a time-average reward value, by specifying the reward of the PHA under consideration so that in each mode in which the system is available the reward increases with rate 1 per unit time and zero else. Survivability [10] is the ability of a system to recover a certain quality of service level in a timely manner after a disaster. Here, we consider the maximal expected time needed to recover from an error condition. This can be expressed using expected total rewards, by maximising over all states of the system (or, an abstraction of the system) in which it is not available.

### 4. ABSTRACTION

In this section, we develop the necessary tools for the abstraction for PHAs and their reward structures. For the theoretical justification of our abstraction method in its entirety we use simulation relations. The relations are actually never constructed during the verification process, just like the full semantics of PHAs, which is our reference semantics, but its construction is prohibitive.

To prove the validity of abstractions of PHAs for reward-based properties, we extend the definition of simulation relations [34, 33] to take into account reward structures. A simulation relation requires that every successor distribution of a state of a simulated PA $M$ is related to a successor distribution of its corresponding state of a simulating PA $M_{\text{sim}}$ using a weight function [25] Definition 4.3.

**Definition 16. Let $\mu \in \text{Distr}(S)$ and $\mu_{\text{sim}} \in \text{Distr}(S_{\text{sim}})$ be two distributions. For a relation $R \subseteq S \times S_{\text{sim}}$, a weight function for $(\mu, \mu_{\text{sim}})$ with respect to $R$ is a function $w : (S \times S_{\text{sim}}) \rightarrow [0, 1]$ with

1. $w(s, s_{\text{sim}}) > 0$ implies $(s, s_{\text{sim}}) \in R$,
2. $\mu(s) = \sum_{s_{\text{sim}} \in S_{\text{sim}}} w(s, s_{\text{sim}})$ for $s \in S$, and
3. $\mu_{\text{sim}}(s_{\text{sim}}) = \sum_{s \in S} w(s, s_{\text{sim}})$ for $s_{\text{sim}} \in S_{\text{sim}}$.

We write $\mu \subseteq_R \mu_{\text{sim}}$ if and only if there exists a weight function for $(\mu, \mu_{\text{sim}})$ with respect to $R$.

Using weight functions, we define simulations.

**Definition 17. Given the two PAs $M = (S, \pi, Act, T)$ and $M_{\text{sim}} = (S_{\text{sim}}, \pi_{\text{sim}}, Act, T_{\text{sim}})$, we say that $M_{\text{sim}}$ simulates $M$, denoted by $M \preceq M_{\text{sim}}$, if and only if there exists a relation $R \subseteq S \times S_{\text{sim}}$, which we will call simulation relation from now on, where
(a) time until failure
(b) cost until termination
(c) long-run cost of operation
(d) system availability
(e) survivability

<table>
<thead>
<tr>
<th>property</th>
<th>Rew</th>
<th>(\dot{r})</th>
<th>type</th>
<th>remark</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>{0 \ s \in \mathbf{F} } \begin{cases} 1 \text{ else} \ any \text{ else} \end{cases}</td>
<td>minimum accumulated reward</td>
<td>(\mathbf{F}) absorbing</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>{0 \ s \in \mathbf{T} } \begin{cases} any \text{ else} \ any \text{ else} \end{cases}</td>
<td>maximum accumulated reward</td>
<td>(\mathbf{T}) absorbing</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>{1 \ s \in \mathbf{F} } \begin{cases} any \text{ else} \end{cases}</td>
<td>maximum long-run average reward</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>{1 \ s \in \mathbf{T} } \begin{cases} any \text{ else} \end{cases}</td>
<td>minimum long-run average reward</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>{1 \ s \in \mathbf{F} } \begin{cases} any \text{ else} \end{cases}</td>
<td>maximum accumulated reward</td>
<td>maximum over (s \in \mathbf{F})</td>
</tr>
</tbody>
</table>

Table 1: Overview of expressible reward properties.

| Figure 4: (a) PA simulating the one of Figure 3 (b) PA affinely simulating the one of Figure 3 |

1. For each \((s, s_{\text{sim}}) \in R, a \in \text{Act}, \mu \in T(s, a)\), there is a distribution \(\mu_{\text{sim}} \in \text{Distr}(s_{\text{sim}})\) with \(\mu_{\text{sim}} \in T_{\text{sim}}(s_{\text{sim}}, a)\) and \(\mu \subseteq_{R} \mu_{\text{sim}}\).

For our purposes, we must consider rewards of PAs.

DEFINITION 18. Consider two PAs \(\mathcal{M} = (S, s, \text{Act}, T)\) and \(\mathcal{M}_{\text{sim}} = (S_{\text{sim}}, s_{\text{sim}}, \text{Act}, T_{\text{sim}})\) with reward structures \(\text{rew}_{\text{num}}, \text{rew}_{\text{den}}\), \(\text{rew}_{\text{num}} = (\text{rew}_{\text{num}}(s, a))\), \(\text{rew}_{\text{num}}(s, a)\), and a simulation relation \(R\) between the PAs. We say that \(R\) is upper-bound compatible, if in case we have \((s, s_{\text{sim}}) \in R\) then for all \(a \in \text{Act}\) it is

\[
\text{rew}_{\text{num}}(s, a) \leq \text{rew}_{\text{num}}(s_{\text{sim}}, a),
\]

\[
\text{rew}_{\text{den}}(s, a) \geq \text{rew}_{\text{den}}(s_{\text{sim}}, a).
\]

If there exists such a relation \(R\), we write

\[(\mathcal{M}, \text{rew}) \uparrow \mathcal{M}_{\text{sim}}, \text{rew}_{\text{sim}}.\]

We define lower-bound compatible simulations \(R\) accordingly by swapping \(\leq\) and \(\geq\) above and write

\[(\mathcal{M}, \text{rew}) \downarrow \mathcal{M}_{\text{sim}}, \text{rew}_{\text{sim}}.\]

With simulations, we can establish upper and lower bounds on the reward properties of simulated models by considering the corresponding property in the simulating model.

LEMMA 1. For PAs \(\mathcal{M}\) and \(\mathcal{M}_{\text{sim}}\) with reward structures \(\text{rew}\) and \(\text{rew}_{\text{sim}}\), if \((\mathcal{M}, \text{rew}) \uparrow (\mathcal{M}_{\text{sim}}, \text{rew}_{\text{sim}})\) then

\[
\text{val}_{\mathcal{M}, \text{rew}, \text{tra}} \leq \text{val}_{\mathcal{M}_{\text{sim}}, \text{rew}_{\text{sim}}, \text{tra}}.
\]

Accordingly for the accumulated rewards. In case \((\mathcal{M}, \text{rew}) \downarrow (\mathcal{M}_{\text{sim}}, \text{rew}_{\text{sim}})\), we obtain lower bounds for minimal values.

We can thus bound the maximal (minimal) reward values from above (below). The principle idea of this simulation is, that the simulating automaton can mimic the behaviour of the simulated one, while overapproximating or underapproximating respectively \(\text{rew}_{\text{num}}\) and \(\text{rew}_{\text{den}}\).

EXAMPLE 9. In Figure 4 (a) we give a PA with corresponding reward structures which simulates the one of Figure 3 by an upper-bound compatible simulation relation. The maximal fractional long-run average reward of the latter is indeed much higher than the former, namely, 7 rather than \(\frac{12}{7} = 2.4\). To obtain a lower-bound compatible simulation relation, we would replace the rewards \((7, 1)\) by \((3, 3)\), thus to obtain a reward of 1 which is considerably lower than the minimal reward \(\frac{7}{7} \approx 2.333\) of the original model.

In the case of affine reward structures, we can define a different simulation relation to obtain more precise results.

DEFINITION 19. Consider two PAs \(\mathcal{M} = (S, s, \text{Act}, T)\) and \(\mathcal{M}_{\text{sim}} = (S_{\text{sim}}, s_{\text{sim}}, \text{Act}, T_{\text{sim}})\) between which there is a simulation relation \(R\). Consider affine reward structures

\[\text{rew} = (\text{rew}_{\text{num}}, \text{rew}_{\text{den}}),\]

\[\text{rew}_{\text{num}} = (\text{rew}_{\text{num}}(s, a))\]

\[\text{rew}_{\text{den}} = (\text{rew}_{\text{den}}(s, a))\]

We require that \(\text{rew}, \text{rew}_{\text{num}}\) and \(\text{rew}_{\text{den}}\) are affine with the same factors \(\mu_{\text{num}}, \mu_{\text{den}}\) (cf. Definition 11) for each action \(a\), that is for \(s \in S\) and \(a \in \text{Act}\) it is

\[\text{rew}_{\text{num}}(s, a) = \mu_{\text{num}} \text{rew}_{\text{num}}(s, a) + \mu_{\text{num}} \text{rew}_{\text{num}}(s, a),\]

\[\text{rew}_{\text{den}}(s, a) = \mu_{\text{den}} \text{rew}_{\text{den}}(s, a) + \mu_{\text{den}} \text{rew}_{\text{den}}(s, a).\]

Then, we define \(R\) as affine compatible if for all \((s, s_{\text{sim}}) \in R\) and \(a \in \text{Act}\) it is

\[\text{rew}_{\text{num}}(s, a) \leq \text{rew}_{\text{num}}(s_{\text{sim}}, a),\]

\[\text{rew}_{\text{den}}(s, a) \geq \text{rew}_{\text{den}}(s_{\text{sim}}, a).\]

If there exists such a relation, we write

\[(\mathcal{M}, \text{rew}) \overset{\text{def}}{=} (\mathcal{M}_{\text{sim}}, \text{rew}_{\text{sim}}, \text{rew}_{\text{sim}}).\]

As before, affine simulations maintain reward properties.

LEMMA 2. Consider the PAs \(\mathcal{M} = (S, s, \text{Act}, T)\) with the reward structure \(\text{rew}\) and \(\mathcal{M}_{\text{sim}} = (S_{\text{sim}}, s_{\text{sim}}, \text{Act}, T_{\text{sim}})\) with reward structures \(\text{rew}_{\text{num}}\) and \(\text{rew}_{\text{sim}}\) with \((\mathcal{M}, \text{rew}) \overset{\text{def}}{=} (\mathcal{M}_{\text{sim}}, \text{rew}_{\text{num}}, \text{rew}_{\text{desim}}).\) We define

\[\mathcal{M}_{\text{aff}} \overset{\text{def}}{=} (S_{\text{sim}}, s_{\text{sim}}, \text{Act} \times \{\text{up, lo}\}, \text{aff}),\]

\[\text{rew}_{\text{aff}} \overset{\text{def}}{=} (\text{rew}_{\text{aff}}, \text{rew}_{\text{num}}, \text{rew}_{\text{den}}),\]

\[\text{val}_{\mathcal{M}, \text{aff}, \text{tra}} \leq \text{val}_{\mathcal{M}_{\text{aff}}, \text{aff}, \text{tra}}.\]

\[\text{val}_{\mathcal{M}, \text{aff}, \text{tra}} \leq \text{val}_{\mathcal{M}_{\text{aff}}, \text{aff}, \text{tra}}.\]

\[\text{val}_{\mathcal{M}, \text{aff}, \text{tra}} \leq \text{val}_{\mathcal{M}_{\text{aff}}, \text{aff}, \text{tra}}.\]
and accordingly for the accumulated rewards and the minimising cases.

Similar to upper-bound and lower-bound compatible simulations, the affinely simulating automaton \( M_{\text{aff}} \) can mimic the behaviours of the simulated one. In \( M_{\text{aff}} \) then, it also mimics the behaviours of the original model, but can use randomised choices over \((a, up)\) and \((a, lo)\) to obtain exactly the same reward as when choosing \( a \) in the original model. The reason that we will obtain results which are more precise is, intuitively, that for nonaffine reward structures we had to bound \( rew_{\text{num}} \) and \( rew_{\text{den}} \) from opposite directions.

**Example 10.** In Figure 4 (b) we give a PA which affinely simulates the one of Example 3. Maximal and minimal long-run averages are \( 3 \) and \( \frac{7}{2} \approx 3.33 \), which is more precise than the values obtained from Figure 4 (a) in Example 9.

We describe abstract state spaces to subsume uncountably many states of the infinite semantics of PHAs.

**Definition 20.** An abstract state space of dimension \( k \) for a set of modes \( M \) is a finite set \( A = \{z_1, \ldots, z_n\} \) where \( z_i = (m_i, c_i) \in M \times \mathbb{R}^k \) and \( \mu = \bigcup_{(m, c) \in A} c = \mathbb{R}^k \) for all \( m \in M \). We identify \( (m, c) \) with the set \( \{m\} \times c \) which allows us to apply the usual set operations on abstract states, and we will for instance write \( s \in (m, c) \).

We do not require \( A \) to be a partitioning of \( M \times \mathbb{R}^k \), that is we do allow overlapping states. This way, one concrete state may be contained in several abstract states. We need to allow this, because in several hybrid system solvers from which we obtain these abstractions, these cases indeed happen. For instance, in the tool HSolver \[1\] we may have overlapping borders, whereas for PHAVer \[9\] we may also have common interiors of abstract states.

An abstraction of a PHA is defined as follows. There, we will need to transfer probability distributions over the states of the PHA semantics to the states of abstractions.

**Definition 21.** Consider an arbitrary PHA \( H = (M, k, m, \langle Post_m \rangle_{m \in M}, Cmds, Rew) \), and abstract state space \( A = \{z_1, \ldots, z_n\} \) of corresponding dimension and modes. We say that \( M = (A, Z, Cmds \uplus \{\tau, T\}) \) is an abstraction of \( H \) using \( A \) if

- \((m, 0, \ldots, 0) \in Z \),
- for all \( z \in A \), \( s \in z \), \( c = (g \rightarrow [u_1 \mapsto p_1, \ldots, u_n \mapsto p_n]) \in Cmds \), if \( s \in z \cap g \), then for all \( (s_1', \ldots, s_n') \in u_1(s) \times \cdots \times u_n(s) \)
  - there are \((z_1', \ldots, z_n') \in A^n \) with \( s_i \in z_i, 1 \leq i \leq n \)
  - so that there is \( \mu \in \text{Distr}(A) \) with \( \mu(z') = \sum_{z'' \in z'} p_i \) and \( \mu \in \bigcap_{z \in Z} c \),
- for all \( z \in A \), \( s = (m, v) \in z \) and all \( s' = (m, v') \in \{m\} \times Post_m(v) \), we require that there is \( z' \in A \) with \( s' \in z' \) and \([z' \mapsto 1] \in T(z, \tau) \).

By \( Abs(H, A) \) we denote the set of all such abstractions.

Next, we equip PHA abstractions with rewards.

**Definition 22.** Let \( H \) be a PHA with rewards \( Rew \) and consider \( M = (A, Z, Cmds \uplus \{\tau, T\}) \in Abs(H, A) \). The abstract upper-bound reward structure is defined as

\[
\text{absup}(H, M) \triangleq (rew_{\text{num}}, rew_{\text{den}}),
\]

- for all \( z \in A \) it is \( rew_{\text{num}}(z, \tau) \triangleq rew_{\text{den}}(z, \tau) \triangleq 0 \),
- for all \( z \in A \) and \( c = (g \rightarrow [u_1 \mapsto p_1, \ldots, u_n \mapsto p_n]) \in Cmds \),

\[
rew_{\text{num}}(z, (c, up)) \triangleq c_{\text{sup}} + Rew(c),
\]

\[
rew_{\text{den}}(z, (c, up)) \triangleq c_{\text{inf}}.
\]

The abstract lower-bound reward structure abslo is defined accordingly by swapping sup and inf.

We can use these reward structures to safely bound the reward values of PHAs semantics.

**Theorem 1.** Consider a PHA \( H \) with reward structure \( Rew \) and \( M = (A, Z, Cmds \uplus \{\tau, T\}) \in \text{Abs}(H, A) \) with \( rew_{\text{up}} \triangleq \text{absup}(H, M) \). Then

\[
\text{val}_{H, \text{abslo}}^+ \leq \text{val}_{H, \text{abslo}}^+ M_{\text{aff}} \leq \text{val}_{H, \text{abslo}}^+ M_{\text{aff}} \leq \text{val}_{H, \text{abslo}}^+ M_{\text{aff}} ^{\text{num}},
\]

and accordingly for accumulated rewards and minimal values.

The theorem follows by **Lemma 1** because abstractions simulate the semantics of PHAs.

**Example 11.** In Figure 5 we sketch an abstraction of Figure 2 with an abstract upper-bound reward structure for the PHA reward structure \( Rew_{\text{abslo}} \) of Example 9. Thus, we have a reward of 1 per time in Error and 0 else. In the abstract states, we left out constraints to restrict to states which are actually reachable. Consider the abstract state \( z_4 \). As the mode of this state is Check, we obtain a reward of 0 when executing \( cc_h \), according to the guard of this command (which we slightly modify when using this reward structure), we have to wait at least until \( t \geq 0.25 \) to execute it. Now consider \( z_2 \). We can leave this state at point \( t = 0.25 \), and we thus obtain a reward and time of 0.25.

In case we are given reward affine PHAs, we can use more precise reward structures in the abstraction.

**Definition 23.** Consider a reward affine PHA \( H \) with commands \( Cmds \) and the reward structure \( Rew \) and \( M = (A, Z, Cmds \uplus \{\tau, T\}) \in \text{Abs}(H, A) \). We define the affine abstraction

\[
M_{\text{aff}} \triangleq (A, Z, Act_{\text{aff}}, T_{\text{aff}}),
\]

where \( Act_{\text{aff}} \triangleq (\{\tau, up\}, \{\tau, lo\}) \uplus \{(c, up) \mid c \in Cmds \} \uplus \{(c, lo) \mid c \in Cmds \} \), and define \( T_{\text{aff}} \) as \( T_{\text{aff}}(z, (a, up)) \text{ def } = T_{\text{aff}}(z, (a, lo)) = T(z, a) \). Then, the abstract affine reward structure is defined as \( rew_{\text{aff}} = (rew_{\text{num}}, rew_{\text{den}}) \) where

- for all \( z \in A \), let
  \[
  rew_{\text{num}}(z, (\tau, up)) \text{ def } = rew_{\text{num}}(z, (\tau, lo))
  \]
  \[
  rew_{\text{den}}(z, (\tau, up)) \text{ def } = rew_{\text{den}}(z, (\tau, lo)) \leq 0,
  \]
- for all \( z \in A \) with mode \( m \) and \( c = (g \rightarrow [u_1 \mapsto p_1, \ldots, u_n \mapsto p_n]) \in Cmds \), let
  \[
  rew_{\text{num}}(z, (c, up)) \text{ def } = c_{\text{up}} + Rew(c),
  \]
  \[
  rew_{\text{den}}(z, (c, up)) \text{ def } = c_{\text{inf}}.
  \]
Theorem follows by Lemma 2.

We then define abstracts \( \mathbb{H}, M \) as \( (M, \text{rew} \cup \text{inf}) \).

\[ \text{rew} = \text{rew}(z, (c, v)) \equiv w_{\text{inf}} \]
\[ \text{inf} = \sup \{t \mid (m, r, t, v) \in z\} \]
\[ \text{vsup} = \inf \{t \mid (m, r, t, v) \in z\} \]

We accordingly for accumulated rewards and minimal values.

The theorem follows by Lemma 2.

**Example 12**. If using an affine abstraction, we replace the actions of Figure 5 by:

- \( (c_{\text{HH}}, \uparrow) : (0, 0), (c_{\text{HH}}, \downarrow) : (0, 0) \)
- \( (c_{\text{CCH}}, \uparrow) : (0, 2), (c_{\text{CCH}}, \downarrow) : (0, 0.5) \)
- \( (c_{\text{CHC}}, \uparrow) : (0.5, 0.5), (c_{\text{CHC}}, \downarrow) : (0.5, 0.5) \)

4.1 Computing Abstractions

A practical recipe to compute approximations of PHAs has been developed in earlier work [39], and it is implemented in the tool ProHVer. The abstraction process is piggybacked on solvers for HAs (as in Definition 2). Concretely, the tool applies PHAVer for this purpose. Thus far however there was no means to compute reward structures for the abstraction at hand.

According to Definition 2 and Definition 3, we have to find suprema and infima of the variables for rewards and time. How this can be done depends on the hybrid systems solver used. PHAVer uses polyhedra to represent abstract states, which can be represented as sets of linear equations. Because of this, we can use a linear programming tool to find the minimal and maximal values of reward and time variables.

In some cases, this construction can be simplified. If we do not have time-dependent rewards and only a constant reward value for each command, and only want to consider the expected accumulated reward, we do not need to compute maxima or minima at all. For affine abstractions, we only need a variable to remember the time since a mode change, because we can compute the rewards from these values; in Definition 3, the time since a mode change is not used.

In usual abstractions of HAs, the information about the time which a continuous transition takes is lost. This is the main reason why we encode reward and time into the first dimension of a PHA, rather than assigning them directly to the timed transitions.

4.2 Algorithmic Considerations

After we have obtained a finite abstraction and have computed the according reward structure using linear programming, it remains to compute expected accumulated or long-run average rewards in the abstraction. There are several algorithms which we can apply for this purpose. For accumulated rewards we can use algorithms based on policy iteration or linear programming [39]. In case we want to compute time-average average values, there are also algorithms using linear programming [15] or policy iteration [16].

**Table 2**: Accumulated rewards in thermostat.

<table>
<thead>
<tr>
<th>len.</th>
<th>PHAVer</th>
<th>( S )</th>
<th>uptime</th>
<th>constr. ana.</th>
<th>res.</th>
<th>constr. ana.</th>
<th>res.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>42.0</td>
</tr>
<tr>
<td>0.05</td>
<td>151</td>
<td>24593</td>
<td>6</td>
<td>65</td>
<td>62.02</td>
<td>1</td>
<td>103</td>
</tr>
<tr>
<td>0.03</td>
<td>2566</td>
<td>93079</td>
<td>18</td>
<td>584</td>
<td>62.72</td>
<td>2</td>
<td>722</td>
</tr>
</tbody>
</table>

**Table 3**: Long-run average rewards in thermostat.

<table>
<thead>
<tr>
<th>len.</th>
<th>PHAVer</th>
<th>( S )</th>
<th>\text{time in error}</th>
<th>constr. ana.</th>
<th>res.</th>
<th>time in error (lin)</th>
<th>constr. ana.</th>
<th>res.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>126</td>
<td>0</td>
<td>0.013</td>
<td>0</td>
<td>0</td>
<td>0.013</td>
<td>0</td>
</tr>
<tr>
<td>0.5</td>
<td>1</td>
<td>382</td>
<td>0</td>
<td>0.011</td>
<td>1</td>
<td>0</td>
<td>0.011</td>
<td>0</td>
</tr>
<tr>
<td>0.1</td>
<td>28</td>
<td>7072</td>
<td>4</td>
<td>0.009</td>
<td>6</td>
<td>14</td>
<td>0.009</td>
<td>0</td>
</tr>
<tr>
<td>0.05</td>
<td>185</td>
<td>29376</td>
<td>8</td>
<td>0.309</td>
<td>25</td>
<td>315</td>
<td>0.309</td>
<td>0</td>
</tr>
</tbody>
</table>
5.2 Water Level Control

We consider a model of a water level control system (extended from the one of Alur et al. \cite{alur1994model}) which uses wireless sensors. Values submitted are thus subject to probabilistic delays, due to the unreliable transport medium. A sketch of the model is given in Figure 6. The water level W of a tank is controlled by a monitor. Its change is specified by an affine function. Initially, the water level is \( W = 1 \). When no pump is turned on (Fill), the tank is filled by a constant stream of water \( W \). When a water level of \( W = 10 \) 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 (FillD2), but with a probability of 0.05 it takes 3 time units (FillD3). The delay is modelled by the timer \( t \). After the delay has passed, the water is pumped out with a higher speed than it is filled into the tank \( W = -2 \) in Drain). 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 (DrainD2 and DrainD3). Similar to the thermostat case, we considered the minimal expected time and number of command executions until the Error mode is reached. As this model features only affine continuous dy-

\begin{figure}[ht]
\centering
\includegraphics[width=\textwidth]{water_level_control_automaton.png}
\caption{Water level control automaton.}
\end{figure}

\begin{table}[ht]
\centering
\begin{tabular}{|c|c|c|c|c|c|c|}
\hline
\textbf{len.} & \textbf{PHAVER} & \textbf{S} & \textbf{uptime} & \textbf{commands} & \textbf{constr. ana. result} & \textbf{constr. ana. result} \\
\hline
\hline
- & 1 & 0 & 0 & 0 & 0 & 40 \\
1 & 0 & 53 & 0 & 0 & 137 & 0 & 0 & 40 \\
0.05 & 5 & 814 & 1 & 1 & 164 & 0 & 2 & 40 \\
0.01 & 31 & 4294 & 1 & 42 & 166 & 1 & 46 & 40 \\
\hline
\end{tabular}
\caption{Accumulated rewards in water control.}
\end{table}

\begin{table}[ht]
\centering
\begin{tabular}{|c|c|c|c|c|}
\hline
\textbf{len. PHAVER} & \textbf{S} & \textbf{avg. energy} & \textbf{avg. energy (lin.)} \\
\hline
\hline
1 & 0 & 51 & 0 & 1.985 & 0 & 0 & 1.814 \\
0.1 & 2 & 409 & 0 & 0 & 1.656 & 0 & 0 & 1.640 \\
0.02 & 11 & 2149 & 0 & 0 & 1.630 & 1 & 0 & 1.627 \\
0.01 & 24 & 4292 & 0 & 1 & 1.627 & 2 & 1 & 1.625 \\
\hline
\end{tabular}
\caption{Long-run average rewards in water control.}
\end{table}

...
7. REFERENCES


