Local abstraction refinement for probabilistic timed programs

Dräger, Klaus; Kwiatkowska, Marta; Parker, David; Qu, Hongyang

DOI:
10.1016/j.tcs.2013.07.013

Document Version
Publisher's PDF, also known as Version of record

Citation for published version (Harvard):
Dräger, K, Kwiatkowska, M, Parker, D & Qu, H 2014, 'Local abstraction refinement for probabilistic timed programs', *Theoretical Computer Science*, vol. 538, pp. 37-53. https://doi.org/10.1016/j.tcs.2013.07.013

Theoretical Computer Science 538 (2014) 37–53

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Local abstraction refinement for probabilistic timed programs

Klaus Dräger [a,*], Marta Kwiatkowska [a], David Parker [b], Hongyang Qu [a]

[a] Department of Computer Science, University of Oxford, Oxford, UK
[b] School of Computer Science, University of Birmingham, Birmingham, UK

ARTICLE INFO

Keywords:
Probabilistic verification
Abstraction refinement

ABSTRACT

We consider models of programs that incorporate probability, dense real-time and data. We present a new abstraction refinement method for computing minimum and maximum reachability probabilities for such models. Our approach uses strictly local refinement steps to reduce both the size of abstractions generated and the complexity of operations needed, in comparison to previous approaches of this kind. We implement the techniques and evaluate them on a selection of large case studies, including some infinite-state probabilistic real-time models, demonstrating improvements over existing tools in several cases.

1. Introduction

Abstraction refinement is a highly successful approach to the verification of complex infinite-state systems. The basic idea is to construct a sequence of increasingly precise abstractions of the system to be verified, with each abstraction typically over-approximating its behaviour. Successive abstractions are constructed through a process of refinement which terminates once the abstraction is precise enough to verify the desired property of the system under analysis. Abstraction refinement techniques have also been used to verify probabilistic systems [6,11,13,7], including those with real-time characteristics [16,17,8] and continuous variables [25]. Frequently, though, practical implementations of these techniques are hindered by the high complexity of both the abstractions involved and the operations needed to construct and refine them.

In this paper, we target the verification of programs whose behaviour incorporates both probabilistic and real-time aspects, and which include the manipulation of (potentially infinite) data variables. We analyse systems modelled as *probabilistic timed programs* (PTPs) [17], whose semantics are defined as infinite-state Markov decision processes (MDPs). We introduce an abstraction refinement procedure for computing minimum and maximum reachability probabilities in PTPs. As in [6,11], we use an MDP-based abstraction. This provides *outer* bounds on reachability probabilities (i.e., a lower bound on the minimum probability or an upper bound on the maximum). In addition, we compute dual, *inner* bounds, based on a stepwise concretisation of adversaries of this abstract MDP, yielding upper and lower bounds on minimum and maximum probabilities, respectively. Concretisation is also used, for example, in [11], for untimed models. The key difference in our work is that we aim to keep the abstraction small by using *local* refinement and simplification operations, so as to reduce the need for expensive operations such as Craig interpolation.

At the core of our approach is a refinement loop that repeatedly attempts to construct a concrete adversary of the PTP. This is based on the exploration of the part of the state space on which the current abstract adversary can be concretised. In each exploration step, we may encounter an inconsistency, in which case we derive a refinement operation and restart. Otherwise, we numerically solve the constructed adversary, giving inner bounds on the desired probability values. The refinement loop terminates once the difference between upper and lower bounds is smaller than a specified threshold ε.

* Corresponding author.
E-mail address: klaus.draeger@cs.ox.ac.uk (K. Dräger).

0304-3975/$ – see front matter
http://dx.doi.org/10.1016/j.tcs.2013.07.013

We implement our abstraction refinement approach, deploy it on various large case studies, and compare to the probabilistic verification tools PRISM [18], PASS [9] and FORTUNA [3], illustrating improved performance in many cases. We are also able to verify probabilistic timed programs containing both real-time behaviour and infinite data variables, which these tools cannot handle.

1.1. Related work

Abstraction refinement for MDPs and related models is an active research field. In [6], techniques were proposed for abstracting MDPs using the notion of probabilistic simulation. Building on the same approach to abstraction, [11] developed a probabilistic version of the classic counterexample-guided abstraction refinement (CEGAR) method, which was then implemented in the tool PASS [9]. This verifies a probability-bounded reachability property using a refinement scheme based on probabilistic counterexamples and Craig interpolation. In contrast to the implementation of [6], PASS uses predicate abstraction, allowing it to analyse infinite-state models. More recent work [15] proposes an alternative probabilistic CEGAR technique using stochastic tree counterexamples; this applies to finite-state MDPs, on which properties are specified using simulation rather than reachability. However, all three methods [6,11,15] were applied to discrete-time models (MDPs), whereas our approach generalises to models with real-time behaviour. We provide an experimental comparison with PASS, for MDP models, in Section 4.

In [13], a quantitative abstraction refinement technique for MDPs was proposed, using a different form of abstraction based on stochastic games. This computes lower and upper bounds for reachability probabilities, the difference between which determines if further refinement is needed. The framework of [13] was subsequently applied to verification of C programs with probabilistic behaviour [12]. Later extensions to PASS also use game-based abstraction refinement [24]. In recent work [7], the abstraction frameworks of [13,24] were adapted to handle arbitrary abstract domains, illustrating cases where this can outperform predicate abstraction. As for [6,11,15] above, though, these methods [13,12,24,7] all focus on models with a discrete notion of time.

Probabilistic timed automata (PTAs) are a subclass of the probabilistic timed programs (PTPs) that we target in this paper, since only the latter allows arbitrary (infinite) data variables. For PTAs, several verification techniques exist. Most relevant here is [16], which extends the abstraction refinement framework of [13] mentioned above, to PTAs, by using zones to represent abstract states. Other possibilities include the digital clocks discretisation [19] and backwards reachability [21]. The probabilistic model checker PRISM [18] supports verification of PTAs, using either [16] or [19]. In [16], abstraction refinement was shown to outperform the other available techniques. Subsequently, an optimised version of backwards reachability, implemented in the tool FORTUNA [3], was shown to exhibit superior performance on various examples. We compare the performance of our approach to both PRISM and FORTUNA in Section 4.

Several PTA verification tools do support PTAs with data variables, but they are required to be finite. This includes PRISM, discussed above, `mcpta` [10], which translates the modelling language Modest to PRISM using [19], and UPPAAL PRO, which computes maximum reachability probabilities for PTAs by progressively partitioning the state space into sets of zones.

The closest approaches to the one presented here are [17] and [8]. In [17], an extension of the game-based abstraction refinement framework of [13] is defined for PTPs, but not implemented. This defines abstractions as stochastic games, rather than MDPs as in our approach. In recent work [8], PTPs (there called variable-decorated PTAs) are verified using a combination of discretisation via digital clocks [19] and predicate abstraction methods from PASS [24]. Our approach avoids the use of discretisation by using zones and aims to improve efficiency by using local refinement and simplification operations to reduce the size of abstractions. The implementation of [8] is not currently available; we give a brief, indirect comparison of results in Section 4.

2. Preliminaries

We assume a set $\mathcal{V}$ of *variables*, ranging over a domain D defined by a theory T (linear integer arithmetic in our examples). We require satisfiability of quantifier-free formulae in T to be decidable. The set of *assertions* over $\mathcal{V}$, i.e., (conjunctive) quantifier-free formulae in T, is denoted by $Asrt(\mathcal{V})$, and $Val(\mathcal{V})$ is the set of *valuations* of $\mathcal{V}$, i.e., functions $u : \mathcal{V} \to D$. We use $Assn(\mathcal{V})$ for the set of *assignments* over $\mathcal{V}$, given by a term r_x for each $x \in \mathcal{V}$. The result of applying assignment f to a valuation u is $f(u)$, given for each $x \in \mathcal{V}$ by $f(u)(x) = u(r_x)$. Given an assignment f and an assertion φ, the composition $\varphi \circ f$ is defined by $(\varphi \circ f)(u) \equiv \varphi(f(u))$.

For a set S, we use $\mathcal{P}(S)$ to denote the set of subsets of S and $\mathcal{D}(S)$ for the set of discrete probability distributions over S, i.e. finite-support functions $\Delta : S \to [0,1]$ such that $\sum_{s \in S} \Delta(s) = 1$. A distribution $\Delta \in \mathcal{D}(S)$ with support $\{s_1, \ldots, s_n\}$ and $\Delta(s_j) = \lambda_j$ will also be written $\lambda_1 s_1 + \cdots + \lambda_n s_n$.

2.1. Clocks

We use a set $\mathcal{X}$ of *clock* variables to represent the time elapsed since the occurrence of various events. The set of *clock valuations* is $\mathbb{R}^{\mathcal{X}}_{\geqslant 0} = \{v : \mathcal{X} \to \mathbb{R}_{\geqslant 0}\}$. For any clock valuation v and $\delta \in \mathbb{R}_{\geqslant 0}$, the *delayed* valuation $v + \delta$ is defined by $(v + \delta)(x) = v(x) + \delta$ for all $x \in \mathcal{X}$. For a subset $Y \subseteq \mathcal{X}$, the valuation $v[Y{:=}0]$ is obtained by setting all clocks in Y to 0, i.e., $v[Y{:=}0](x)$ is 0 if $x \in Y$ and $v(x)$ otherwise. The valuation $\mathbf{0}$ has all clocks set to 0.

A *clock difference constraint* over $\mathcal{X}$ is an upper or lower bound on either a clock or the difference between two clocks. It is convenient to extend $\mathcal{X}$ with a dedicated *zero clock* x_0 which is always 0, so that all clock difference constraints have the form $x - y \lesssim b$ with $x, y \in \mathcal{X}_0 := \mathcal{X} \cup \{x_0\}$, $\lesssim \in \{<, \leqslant\}$ and $b \in \mathbb{Z} \cup \{\pm\infty\}$. We define the *complement* $\bar{c}$ of a clock difference constraint c as: $\bar{c} := y - x < -b$ if $c \equiv x - y \leqslant b$; and $\bar{c} := y - x \leqslant -b$ if $c \equiv x - y < b$.

A (convex) *zone* is the set of clock valuations satisfying a number of clock difference constraints, and the set of all zones is $Zones(\mathcal{X})$. We use several standard operations on zones:

- future: $\nearrow \rho = \{v + \delta \mid v \in \rho, \delta \in \mathbb{R}_{\geqslant 0}\}$ is the set of clock valuations reachable from ρ by letting time pass;
- past: $\swarrow \rho = \{v \mid v + \delta \in \rho \text{ for some } \delta \in \mathbb{R}_{\geqslant 0}\}$ is the set of clock valuations from which ρ can be reached by letting time pass;
- clock reset: if $Y \subseteq \mathcal{X}$, then $\rho[Y := 0] = \{v[Y := 0] \mid v \in \rho\}$ contains the valuations obtained from ρ by setting the values of all $y \in Y$ to 0;
- inverse reset: if $Y \subseteq \mathcal{X}$, then $[Y := 0]\rho = \{v \mid v[Y := 0] \in \rho\}$ contains the valuations which end up in ρ if the values of all $y \in Y$ are set to 0.

We will use one additional operation on clocks, for which we first require some standard operations on pairs $(b, \lesssim) \in \mathbb{R}_{\geqslant 0} \times \{<, \leqslant\}$ (see [2]). The set $\{<, \leqslant\}$ is ordered by $< \lhd \leqslant$, and the set of pairs $(b, \lesssim)$ by the lexicographic combination $\prec$ of $<_{\mathbb{R}}$ and $\lhd$, i.e. $(b_1, <) \prec (b_1, \leqslant) \prec (b_2, <)$ for all $b_1 < b_2$. The sum of two pairs is then given by $(b_1, \lesssim_1) + (b_2, \lesssim_2) := (b_1 + b_2, \min_{\lhd}(\lesssim_1, \lesssim_2))$.

Definition 1. Let $\rho_1, \ldots, \rho_m$ be zones whose intersection $\rho_1 \cap \cdots \cap \rho_m$ is empty. An *unsatisfiable core* for $\rho_1, \ldots, \rho_m$ is a set C of constraints such that:

(i) each constraint in C is implied by some ρ_k,
(ii) the conjunction of all constraints in C is unsatisfiable, and
(iii) C is minimal in the sense that no proper subset $C' \subsetneq C$ satisfies (ii).

An unsatisfiable core always exists, since the zones are given by finite sets of constraints whose union U is unsatisfiable (otherwise the zones would have a non-empty intersection), and the subsets of U satisfy the descending chain condition. We can compute an unsatisfiable core with the following straightforward generalisation of the Craig interpolation procedure for difference logic from [5].

For all i, j, let $(b_{ij}, \lesssim_{ij})$ be the least pair for which $c_{ij} \equiv x_i - x_j \lesssim_{ij} b_{ij}$ is implied by some ρ_k. Note that the conjunction of the c_{ij} is unsatisfiable. This means that, if we label the edges of the complete directed graph on $\mathcal{X}_0$ with the pairs $(b_{ij}, \lesssim_{ij})$, then there is a cycle with a negative label sum [2]; using the Floyd–Warshall algorithm, we can find a shortest such cycle $(i_1, \ldots, i_k)$.

The constraints along this cycle are $x_{i_{j-1}} - x_{i_j} \lesssim_{i_{j-1} i_j} b_{i_{j-1} i_j}$ for $j = 1, \ldots, k$, where $i_0 = i_k$; the label sum $(b, \lesssim)$ being negative means that summing the constraints results in an unsatisfiable implied constraint $0 \lesssim b$. On the other hand, removing one of the constraints (w.l.o.g., we can assume it is the first one) gives a path whose associated constraints are satisfied by any clock valuation t of the form $v(x_{i_j}) = a + b_{i_1 i_2} + \cdots + b_{i_{j-1} i_j}$ for some large enough a. So the constraints along the cycle form an unsatisfiable core.

2.2. Markov decision processes (MDPs)

The underlying semantics for the models studied in this paper is defined in terms of *Markov decision processes* (MDPs), a standard model for systems with both probability and nondeterminism. An MDP is a tuple (S, s_i, S_e, T), where S is a (possibly infinite) set of states, $s_i \in S$ is the initial state, $S_e \subseteq S$ is a set of error states and $T : S \to \mathcal{P}(\mathcal{D}(S))$ is a probabilistic transition function. In a state $s \in S$, the choice of a successor distribution $\Delta \in T(s)$ is nondeterministic and a successor state is then selected probabilistically according to Δ.

An *adversary* for an MDP resolves the nondeterminism in each state, based on the current history, i.e., it is a function $\sigma : S^+ \to \mathcal{D}(S)$ such that $\sigma(s_1 \ldots s_l) \in T(s_l)$ for any path $s_1 \ldots s_l$. The adversary is *memoryless* if σ depends only on s_l; it can then be written as a function $\sigma : S \to \mathcal{D}(S)$. The behaviour of an MDP M under a particular adversary σ can be viewed as a (possibly infinite-state) Markov chain. This allows us to define, in standard fashion [14], a probability space $\Pr^{\sigma}_{M,s}$ over the set of all (infinite) paths[1] from a given state s of M.

In this paper, we focus on one kind of property: the *error probability* $p^{\sigma}_{M,s} = \Pr^{\sigma}_{M,s}(\{s_1 s_2 \ldots \mid s_1 = s \text{ and } s_j \in S_e \text{ for some } j\})$, i.e., the probability of reaching one of M's designated error states. In particular, we aim to compute the *minimum error probability* $p^{\min}_{M,s} = \inf\{p^{\sigma}_{M,s} \mid \sigma \text{ is an adversary of } M\}$ or the *maximum error probability* $p^{\max}_{M,s} = \sup\{p^{\sigma}_{M,s} \mid$

[1] The MDPs in this paper are infinite state due to the use of both unbounded data and dense real-time; see e.g. [20] for a discussion of how to ensure measurability for such models.

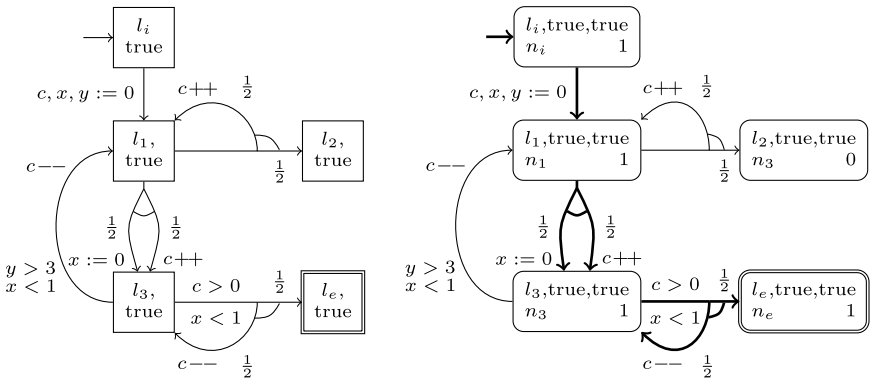

Fig. 1. Left: an example PTP P. Right: initial abstraction $A_i(P)$, labelled with node ids and upper probability bounds, and with abstract adversary σ_a marked in bold.

σ is an adversary of M}. Probabilities from the initial state s_i of the MDP are denoted by omitting the subscript s_i, i.e. $p^{\sigma}_{M} = p^{\sigma}_{M,s_i}$, $p^{\min}_{M} = p^{\min}_{M,s_i}$, and $p^{\max}_{M} = p^{\max}_{M,s_i}$.

2.3. Probabilistic timed programs (PTPs)

The systems we will verify are *probabilistic timed programs* (PTPs) [17], which can be thought of as MDPs extended with state variables and real-valued clocks (or as probabilistic timed automata with state variables). For simplicity, we assume that a PTP contains an *initial* location which must be left immediately and never re-entered, and an *error* location which cannot be left. We also make the common assumption that models are *structurally non-Zeno* [23].

Definition 2 *(PTP).* A PTP is a tuple $P = (L, \mathcal{V}, \mathcal{X}, l_i, u_i, l_e, \mathcal{I}, \mathcal{T})$ where:

- L is a finite set of *locations* and $l_i, l_e \in L$ are *initial* and *error locations*;
- $\mathcal{V}$ is a finite set of *state variables* and $u_i \in \mathit{Val}(\mathcal{V})$ is the *initial valuation*;
- $\mathcal{X}$ is a finite set of *clocks* and $\mathcal{I} : L \to \mathit{Zones}(\mathcal{X})$ is the *invariant condition*, where we assume $\mathcal{I}(l_i) = \{\mathbf{0}\}$;
- $\mathcal{T} : L \to \mathcal{P}(\mathit{Trans}(L, \mathcal{V}, \mathcal{X}))$ is the *probabilistic transition function*, where $\mathit{Trans}(L, \mathcal{V}, \mathcal{X}) = \mathit{Asrt}(\mathcal{V}) \times \mathit{Zones}(\mathcal{X}) \times \mathcal{D}(\mathit{Assn}(\mathcal{V}) \times \mathcal{P}(\mathcal{X}) \times L)$.

A (concrete) *state* of a PTP P is a triple $q = (l, u, v) \in L \times \mathit{Val}(\mathcal{V}) \times \mathbb{R}^{\mathcal{X}}_{\geqslant 0}$ such that $v \in \mathcal{I}(l)$. The set of all states is denoted $Q_c(P)$, or Q_c if P is clear from the context. The initial state is $q_i = (l_i, u_i, \mathbf{0})$, and the set of error states is $Q_e = \{(l_e, u, v) \mid v \in \mathcal{I}(l_e)\}$. A step of the PTP from state (l, u, v) consists of some *delay* $\delta \geqslant 0$ followed by a *transition* $(\mathcal{G}, \mathcal{E}, \Delta) \in \mathcal{T}(l)$. The transition comprises a *guard* $\mathcal{G}$, *enabling condition* $\mathcal{E}$ and probability distribution $\Delta = \lambda_1(f_1, r_1, l_1) + \cdots + \lambda_k(f_k, r_k, l_k)$ over triples containing an *update* $f_j \in \mathit{Assn}(\mathcal{V})$, *clock resets* $r_j \subseteq \mathcal{X}$ and *target location* $l_j \in L$.

The delay δ must be chosen such that the invariant $\mathcal{I}(l)$ remains continuously satisfied; since $\mathcal{I}(l)$ is a (convex) zone, this is equivalent to requiring that both v and $v + \delta$ satisfy $\mathcal{I}(l)$. The chosen transition must be *enabled*, i.e., the guard $\mathcal{G}$ and the enabling condition $\mathcal{E}$ must be satisfied by u and $v + \delta$, respectively. An assignment, set of clocks to reset and successor location are then selected at random, according to the distribution Δ.

Formally, the semantics of PTP P is given by an MDP $[\![P]\!] = (Q_c, q_i, Q_e, T)$ where $\lambda_1(l_1, u_1, v_1) + \cdots + \lambda_k(l_k, u_k, v_k) \in T(l, u, v)$ if and only if there are $\delta \geqslant 0$ and $(\mathcal{G}, \mathcal{E}, \lambda_1(f_1, r_1, l_1) + \cdots + \lambda_k(f_k, r_k, l_k)) \in \mathcal{T}(l)$ such that: (i) $u \models \mathcal{G}$; (ii) $v + \delta \in \mathcal{I}(l) \cap \mathcal{E}$; (iii) $u_j = f_j(u)$ for all j; and (iv) $v_j = (v + \delta)[r_j := 0] \in \mathcal{I}(l_j)$ for all j. Thus, any adversary $\sigma : Q_c^+ \to \mathcal{D}(Q_c)$ of $[\![P]\!]$ is *induced* by two functions $\delta_\sigma : Q_c^+ \to \mathbb{R}_{\geqslant 0}$ and $\tau_\sigma : Q_c^+ \to \mathit{Trans}(L, \mathcal{V}, \mathcal{X})$ such that $\delta_\sigma(wq)$ and $\tau_\sigma(wq) \in \mathcal{T}(l)$ satisfy (i)–(iv) for all $w \in Q_c^*$ and $q = (l, u, v)$.

Our focus in this paper is determining the *minimum* or *maximum probability* of reaching an error state in PTP P, denoted $p^{\min}_P$ and $p^{\max}_P$, respectively. These are defined as the values $p^{\min}_{[\![P]\!]}$ and $p^{\max}_{[\![P]\!]}$ for its MDP semantics $[\![P]\!]$. We determine the desired value up to a given precision $\varepsilon > 0$ by producing lower and upper bounds $p^{lb,\min}_P \leqslant p^{\min}_P \leqslant p^{ub,\min}_P$ or $p^{lb,\max}_P \leqslant p^{\max}_P \leqslant p^{ub,\max}_P$ which differ by at most ε.

Example 1. Fig. 1 (left) shows an example PTP with integer variable c and clocks x, y. Guards (e.g. $c > 0$), enabling conditions (e.g. $x < 1$), resets (e.g. $x := 0$) and probabilities (e.g. $\frac{1}{2}$) label transitions; invariants (true) label locations.

3. Abstraction refinement for PTPs

We now introduce our abstraction refinement approach for PTPs, which we call *local abstraction refinement*. Our abstractions are based on MDPs and yield both lower and upper bounds on the desired probability. The *outer* bound ($p_P^{lb,\min}$ or $p_P^{ub,\max}$) is obtained from an adversary σ_a on the abstract MDP (which overapproximates the choices available to concrete adversaries), while the *inner* bound ($p_P^{ub,\min}$ or $p_P^{lb,\max}$) is based on a partial *concretisation* of σ_a.

In the next section, we describe the basic ideas underlying our abstractions; in subsequent sections, we describe in more detail how to generate suitably precise abstractions using a refinement loop. Throughout, we will assume a fixed PTP $P = (L, \mathcal{V}, \mathcal{X}, l_i, u_i, l_e, \mathcal{I}, \mathcal{T})$ with semantics $[\![P]\!] = (Q_c, q_i, Q_e, T)$.

3.1. MDP abstractions

The abstraction for a PTP P in our approach is an MDP, which, for convenience, we augment with concretisation information.

Definition 3 *(MDP abstraction).* For a probabilistic timed program P, an *abstract state* (or *node*) is a triple $n = (l, \varphi, \rho) \in L \times Asrt(\mathcal{V}) \times Zones(\mathcal{X})$. We use $Q_a(P)$ to denote the set of all abstract states. An *MDP abstraction* is a tuple $A = (N, n_i, N_e, T)$, where:

- $N \subseteq Q_a(P)$ is a finite set of nodes, $n_i \in N$ is the initial node, and $N_e \subseteq N$ the set of error nodes;
- $T : N \to \mathcal{P}(Trans(N, \mathcal{V}, \mathcal{X}))$ maps nodes to finite sets of *abstract transitions* in $Trans(N, \mathcal{V}, \mathcal{X}) = Asrt(\mathcal{V}) \times Zones(\mathcal{X}) \times \mathcal{D}(Assn(\mathcal{V}) \times \mathcal{P}(\mathcal{X}) \times N)$.

The set of *dead nodes* from which N_e is not reachable is denoted by N_d.

In order to formalise the relationship between an MDP abstraction A and its corresponding PTP P, we introduce the notion of *reflections*. Recall from the definition of $[\![P]\!]$ that any adversary σ is induced by functions $\delta_\sigma : Q_c^+ \to \mathbb{R}_{\geqslant 0}$ and $\tau_\sigma : Q_c^+ \to \mathcal{T}$, such that, for $wq \in Q_c^+$ with $q = (l, u, \nu)$ and $\tau_\sigma(wq) = (\mathcal{G}, \mathcal{E}, \lambda_1(f_1, r_1, l_1) + \cdots + \lambda_k(f_k, r_k, l_k))$, we get $\sigma(wq) = \lambda_1 q_1 + \cdots + \lambda_k q_k$ with $q_j = (l_j, f_j(u), (\nu + \delta_\sigma(wq))[r_j := 0])$. A reflection captures the idea that every concrete adversary can be simulated in an abstraction; if, in addition, every transition in the abstraction represents a special case of a concrete transition, we call this abstraction *sound*.

Definition 4 *(Reflection).* Let $A = (N, n_i, N_e, T)$ be an MDP abstraction for P. A *reflection* of adversaries for A is a map $\nabla : Q_c^+ \times \mathbb{R}_{\geqslant 0} \to N$ with the following properties:

- $\nabla(wq, \delta) = n_i$ iff $q = (l_i, \_, \_)$ and $\nabla(wq, \delta) \in N_e$ iff $q \in Q_e$;
- for any adversary σ of $[\![P]\!]$, let $\nabla_\sigma : Q_c^+ \to N$ be the *reflection of* σ, defined as $\nabla_\sigma(w) = \nabla(w, \delta_\sigma(w))$. Then for any path $w \in Q_c^+$, where $\sigma(w) = \lambda_1 q_1 + \cdots + \lambda_k q_k$ is induced by $\delta_\sigma(w)$ and $\tau_\sigma(w) = (\mathcal{G}, \mathcal{E}, \lambda_1(f_1, r_1, l_1) + \cdots + \lambda_k(f_k, r_k, l_k))$, there are $\mathcal{G}'$, $\mathcal{E}'$ such that the set $T(\nabla_\sigma(w))$ contains an abstract transition of the form $(\mathcal{G}', \mathcal{E}', \lambda_1(f_1, r_1, \nabla_\sigma(wq_1)) + \cdots + \lambda_k(f_k, r_k, \nabla_\sigma(wq_k)))$.

The need to have a delay as an extra argument arises from the behaviour of refinement with respect to clock constraints (see Section 3.4). The delay is effectively a prophecy variable [1] representing the next decision of the adversary. This dependency means, in particular, that we do not have a straightforward simulation relation between concrete and abstract states; the reflection allows us, however, to construct a simulation for each concrete adversary (see Theorem 1).

Definition 5 *(Concretisable transitions).* Let $A = (N, n_i, N_e, T)$ be an MDP abstraction for P. A has *concretisable transitions* if, for each node $n = (l, \varphi, \rho) \in N$ and abstract transition $\tau = (\mathcal{G}, \mathcal{E}, \lambda_1(f_1, r_1, n_1) + \cdots + \lambda_k(f_k, r_k, n_k)) \in T(n)$, where $n_j = (l_j, \varphi_j, \rho_j)$ for all j, the PTP P contains a concrete transition $(\mathcal{G}', \mathcal{E}', \lambda_1(f_1, r_1, l_1) + \cdots + \lambda_k(f_k, r_k, l_k)) \in \mathcal{T}(l)$ with $\mathcal{G} \Rightarrow \mathcal{G}'$ and $\mathcal{E} \subseteq \mathcal{E}'$.

Definition 6 *(Sound abstraction).* An MDP abstraction A for PTP P is *sound* if it has a reflection of its adversaries and concretisable transitions.

All our abstractions will be sound and have the following additional property.

Definition 7 *(Tight abstraction).* An MDP abstraction A is *tight* if, for every abstract transition $(\mathcal{G}, \mathcal{E}, \lambda_1(f_1, r_1, n_1) + \cdots + \lambda_k(f_k, r_k, n_k)) \in T(n)$, where $n = (l, \varphi, \rho)$ and $n_j = (l_j, \varphi_j, \rho_j)$ for all j, we have $\mathcal{E} \subseteq \rho$, and $\mathcal{G}, \mathcal{E}$ ensure validity of all successors, i.e. $\mathcal{G} \Rightarrow \varphi_j \circ f_j$ and $\mathcal{E} \subseteq [r_j := 0]\rho_j$ for all j.

The significance of these properties is as follows. Soundness allows us to obtain correct bounds, while tightness ensures progress. Specifically, a reflection of adversaries ensures that the reachability probabilities in the MDP underlying A yield correct *outer* bounds $p_P^{lb,\min} := p_A^{\min}$ and $p_P^{ub,\max} := p_A^{\max}$ (see Theorem 1) while concretisability of transitions means that we obtain correct *inner* bounds by partially concretising an abstract adversary (see Theorem 3). Tightness ensures that each refinement step is a *proper* refinement, in the sense that each node obtained by splitting a node n either satisfies stronger constraints than n or has stronger enabledness conditions on its outgoing transitions (see Theorems 4 and 5).

Theorem 1. *If A is a sound MDP abstraction for PTP P, then $p_A^{\min} \leqslant p_P^{\min}$ and $p_A^{\max} \geqslant p_P^{\max}$.*

Proof. Let $A = (N, n_i, n_e, T)$ be an MDP abstraction for P with a reflection of adversaries $\nabla : Q_c^+ \times \mathbb{R}_{\geqslant 0} \to N$, and let σ be an arbitrary adversary of $[\![P]\!]$. Consider the MDP $M = (Q_c^+, q_i, \{wq \in Q_c^+ \mid q \in Q_e\}, T')$ with the singleton transition sets $T'(w) = \{\lambda_1(wq_1) + \cdots + \lambda_k(wq_k)\}$ obtained from $\sigma(w) = \lambda_1 q_1 + \cdots + \lambda_k q_k$ for all w. This is essentially the DTMC induced by $[\![P]\!]$ and σ, and satisfies $p_M^{\min} = p_M^{\max} = p_{[\![P]\!]}^{\sigma}$.

By the assumption that l_i cannot be revisited, all reachable states of M lie in $q_i Q_r^*$, where $Q_r = (L \setminus \{l_i\}) \times \mathit{Val}(\mathcal{V}) \times \mathbb{R}_{\geqslant 0}^{\mathcal{X}}$. Consider the relation $R := \{(wq, \nabla_\sigma(q)) \mid wq \in q_i Q_r^*\}$. From the definition of ∇, we have that, for all $(w, n) \in R$, $w = q_i$ iff $n = n_i$, $w \in Q_e$ iff $n \in N_e$, and $T(\nabla_\sigma(w))$ contains an abstract transition $(\mathcal{G}', \mathcal{E}', \lambda_1(f_1, r_1, \nabla_\sigma(wq_1)) + \cdots + \lambda_k(f_k, r_k, \nabla_\sigma(wq_k)))$ corresponding to the sole transition $\lambda_1(wq_1) + \cdots + \lambda_k(wq_k)$ in $T'(w)$, such that $(wq_j, \nabla_\sigma(wq_j)) \in R$ for all j. Thus, R is a probabilistic simulation, and we get $p_A^{\min} \leqslant p_M^{\min} \leqslant p_{[\![P]\!]}^{\sigma}$ and $p_A^{\max} \geqslant p_M^{\max} \geqslant p_{[\![P]\!]}^{\sigma}$ [6]. Since the argument works for arbitrary σ, we also get $p_A^{\min} \leqslant p_{[\![P]\!]}^{\min} = p_P^{\min}$ and $p_A^{\max} \geqslant p_{[\![P]\!]}^{\max} = p_P^{\max}$. □

To build abstractions for PTPs, we start with an *initial abstraction*, which is defined as follows.

Definition 8 *(Initial abstraction).* For a location l of PTP P, let n_l denote the abstract state $(l, \mathit{true}, \mathcal{I}(l))$. The *initial abstraction* for P is the MDP abstraction $A_i(P) := (\{n_l \mid l \in L\}, n_{l_i}, \{n_{l_e}\}, T)$ where: $T(n_l) = \{\alpha_l(\tau) \mid \tau \in \mathcal{T}(l)\}$ and $\alpha_l(\mathcal{G}, \mathcal{E}, \lambda_1(f_1, r_1, l_1) + \cdots + \lambda_k(f_k, r_k, l_k)) = (\mathcal{G}, \mathcal{E}', \lambda_1(f_1, r_1, n_{l_1}) + \cdots + \lambda_k(f_k, r_k, n_{l_k}))$ with $\mathcal{E}' = I(l) \cap \mathcal{E} \cap \bigcap_j [r_j := 0]\mathcal{I}(l_j)$.

Theorem 2. *The initial abstraction $A_i(P)$ for PTP P is sound and tight.*

Proof. Let $A_i(P) = (\{n_l \mid l \in L\}, n_{l_i}, \{n_{l_e}\}, T)$. We define $\nabla : Q_c^+ \times \mathbb{R}_{\geqslant 0} \to N$ by $\nabla(wq, \delta) = n_l$ for $q = (l, v, t)$. Clearly, ∇ maps (wq, δ) to n_{l_i} iff $l = l_i$, and to n_{l_e} iff $q \in Q_e$. Let σ be an adversary of $[\![P]\!]$ induced by the delays $\delta_\sigma(q)$ and transitions $\tau_\sigma(q)$ as defined in Definition 4. Then, for each $w \in Q_c^*$ and $q = (l, v, t)$, $\nabla_\sigma(wq) = n_l$ and, by the definition of $A_i(P)$, $T(n_l)$ contains the abstract transition $\alpha_l(\tau_\sigma(wq)) = (\mathcal{G}, \mathcal{E}', \lambda_1(f_1, r_1, n_{l_1}) + \cdots + \lambda_k(f_k, r_k, n_{l_k}))$ for $\tau_\sigma(wq) = (\mathcal{G}, \mathcal{E}, \lambda_1(f_1, r_1, l_1) + \cdots + \lambda_k(f_k, r_k, l_k))$. Altogether, this means that ∇ is a reflection of adversaries and, since concretisability of transitions is ensured by the obvious mapping of $\alpha_l(\tau)$ to τ, the initial abstraction $A_i(P)$ is sound.

Tightness is achieved by using the strengthened enabledness condition $\mathcal{E}'$ in $\alpha_l(\tau)$ (since all state assertions are *true*, $\mathcal{G}$ does not need to be changed). □

To extract the complementary (inner) bound $p_P^{ub,\min}$ or $p_P^{lb,\max}$ from an MDP abstraction A, we will need the notion of a (memoryless) *abstract adversary*, which selects an outgoing transition from each node n of A. In practice, we obtain such an adversary when computing the extremal reachability probabilities $p_{A,n}^{\min}$ or $p_{A,n}^{\max}$ for each node n in A.

Definition 9 *(Abstract adversary).* An *abstract adversary* for an MDP abstraction $A = (N, n_i, N_e, T)$ is a function $\sigma_a : N \to \mathit{Trans}(N, \mathcal{V}, \mathcal{X})$ such that $\sigma_a(n) \in T(n)$ for all $n \in N$.

Example 2. Fig. 1 (right) shows the initial MDP abstraction $A_i(P)$ for PTP P from Example 1 (Fig. 1, left), for which we want to determine the maximum error probability. The top of each box shows the abstract state (node); underneath is (to the left) a node id and (to the right) the computed maximum probability of reaching error node n_e. A corresponding adversary σ_a is indicated in bold.

An abstract adversary resolves the nondeterminism in the MDP, giving a discrete-time Markov chain which will allow us to compute $p_P^{ub,\min}$ or $p_P^{lb,\max}$. More precisely, we build a (partial) *concretisation*, based on a forward exploration through the model. We explore the Markov chain induced by σ_a using *discrete states* $s = (n, u)$ consisting of a node $n = (l, \varphi, \rho)$ of A and a valuation u with $u \vDash \varphi$. Interleaved with these forward exploration steps, we perform backwards propagation of *time constraints*, starting with the invariants of the newly expanded successor states, which are then iteratively strengthened. We formalise this as follows.

Definition 10 *(Concretisation).* A *(partial) concretisation* of an abstract adversary σ_a is a tuple $C = (S, O, E)$, consisting of:

- a set $S \subseteq N \times Val(\mathcal{V})$ of discrete states $s = (n, u)$, with a subset $O \subseteq S$ of *open* states whose successors still have to be explored;
- a set E of *edges* $e = (s, \lambda, r, s')$, representing edges of the adversary transitions, with the associated probability λ and resets r. For a state $s = (n, u)$, $E_s \subseteq E$ is the set of edges with source s. We have:
  - $E_s = \emptyset$ if $s \in O$ or $n \in N_d \cup N_e$ is a dead or error node;
  - otherwise, $E_s = \{(s, \lambda_j, r_j, (n_j, f_j(u))) \mid 1 \leqslant j \leqslant k\}$ based on the edges in $\sigma_a(n) = (\mathcal{G}, \mathcal{E}, \lambda_1(f_1, r_1, n_1) + \cdots + \lambda_k(f_k, r_k, n_k))$.

We denote by $\overline{S} = \{s \in S \mid E_s \neq \emptyset\}$ the set of *expanded* states in C.

Every partial concretisation induces time constraints $\eta : (S \cup E) \to Zones(\mathcal{X})$ and bounds $\pi_0, \pi_1 : S \to [0, 1]$ on reachability probabilities as follows.

- Let the zones $\eta(s), \eta(e)$ for states s and edges e be the greatest solutions to the fixpoint equations:
  - for all $s = (n, u)$ with $n = (l, \varphi, \rho)$: $\eta(s) = \rho \cap \swarrow (\rho \cap \bigcap_{e \in E_s} \eta(e))$,
  - for each $e = (s, \lambda, r, s')$ with $s = (n, u)$, $\sigma_a(n) = (\mathcal{G}, \mathcal{E}, \Delta)$: $\eta(e) = \mathcal{E} \cap [r := 0]\eta(s')$.
- The probability bounds $\pi_b(s)$ for $b \in \{0, 1\}$ are the least solutions to:
  - $\pi_b(n, u) = 1$ for $n \in N_e$ an error node,
  - $\pi_b(n, u) = 0$ for $n \in N_d$ a dead node,
  - $\pi_b(s) = b$ for $s \in O$, and
  - $\pi_b(s) = \sum_{(s,\lambda,r,s') \in E} \lambda \cdot \pi_b(s')$ otherwise (i.e. for $s \in \overline{S}$).

The idea of this construction is that a partial concretisation whose associated time constraints are satisfiable represents a portion of the model on which the abstract adversary's chosen transitions can be consistently followed by a concrete adversary; $\pi_0(n_i, u_i)$ and $\pi_1(n_i, u_i)$ then give a lower bound and an upper bound on p_P^σ for any adversary σ doing so.

We can therefore use the probability bounds π_b from the concretisation of a maximising or minimising abstract adversary to determine inner bounds, i.e., we take $p_P^{lb,\max} := \pi_0(n_i, u_i)$ or $p_P^{ub,\min} := \pi_1(n_i, u_i)$.

In order to formalise this, we first define some auxiliary sets capturing the relationship between concrete and discrete states. For $s = (n, u) \in S$, with $n = (l, \varphi, \rho)$, we let $Q_s = \{(l, u, v) \in Q_c(P) \mid v \in \eta(s)\}$ and define $Q_S = \bigcup_{s \in S} Q_s$ and $Q_{\overline{S}} = \bigcup_{s \in \overline{S}} Q_s$; for $q \in Q_c(P)$, $S_q = \{s \in S \mid q \in Q_s\}$ and $\overline{S}_q = \{s \in \overline{S} \mid q \in Q_s\}$.

We then say that a concrete adversary σ for P *follows* the partial concretisation $C = (S, O, E)$ if there is a map $s : Q_S^+ \to S$ such that, for each path $w = w_1 \dots w_m \in Q_S^+$, if $w_m \in Q_{\overline{S}}$ and $\sigma(w) = \lambda_1 q_1 + \cdots + \lambda_k q_k$, then $E_{s(w)} = \{(s(w), \lambda_j, r_j, s(wq_j)) \mid 1 \leqslant j \leqslant k\}$ for some $r_1, \dots, r_k \subseteq \mathcal{X}$.

Theorem 3. *Let $C = (S, O, E)$ be a partial concretisation for adversary σ_a of A, such that all zones $\eta(s)$ for $s \in S$ and $\eta(e)$ for $e \in E$ are non-empty. Then*:

1. *there exists a concrete adversary σ following C;*
2. *for any such σ, we have $\pi_0(n_i, u_i) \leqslant p_P^\sigma \leqslant \pi_1(n_i, u_i)$.*

Proof. 1. Let $w = w_1 \dots w_m \in Q_{\overline{S}}^+$ with $w_m = (l, u, v)$. Let $s(w)$ be $(n, u) \in \overline{S}_{w_m}$ with $n = (l, \varphi, \rho)$ (chosen arbitrarily if $|w| = 1$) and let the abstract adversary's chosen transition be $\sigma_a(n) = (\mathcal{G}, \mathcal{E}, \lambda_1(f_1, r_1, n_1) + \cdots + \lambda_k(f_k, r_k, n_k))$, where $n_j = (l_j, \varphi_j, \rho_j)$. Since $s(w) \in \overline{S}_{w_m}$, we have that $v \in \eta(s(w)) = \rho \cap \swarrow (\rho \cap \bigcap_{e \in E_{s(w)}} \eta(e))$. Therefore, there exists some $\delta \in \mathbb{R}_{\geqslant 0}$ such that $v + \delta \in \rho \cap \bigcap_{e \in E_{s(w)}} \eta(e)$ (and $v + \epsilon \in \rho$ for all $0 \leqslant \epsilon \leqslant \delta$, since ρ is convex).

By concretisability of transitions, there is a corresponding concrete transition $\tau' = (\mathcal{G}', \mathcal{E}', \lambda_1(f_1, r_1, l_1) + \cdots + \lambda_k(f_k, r_k, l_k))$ with $\mathcal{G} \Rightarrow \mathcal{G}'$ and $\mathcal{E} \subseteq \mathcal{E}'$. Due to the above, a concrete adversary can choose this concrete transition after a suitable delay δ, getting the distribution $\lambda_1 q_1 + \cdots + \lambda_k q_k$, where $q_j = (l_j, f_j(u), v_j)$ with $v_j = (v + \delta)[r_j := 0]$. The edges in E_s are $e_j = (s(w), \lambda_j, r_j, s_j)$, where $s_j = (n_j, f_j(u))$; since δ was chosen such that $v + \delta \in \eta(e_j) = \mathcal{E} \cap [r_j := 0]\eta(s_j)$ for all j, we get $v_j \in \eta(s_j)$, and thus $s_j \in S_{q_j}$, so we can choose $s(wq_j) = s_j$. Iterating this argument, we get a concrete adversary following C.

2. Let σ be a concrete adversary following C, $w = w_1 \dots w_m$ be any path within Q_S in the Markov chain induced by P and σ, and $s_1, \dots, s_m$ be the corresponding path in the Markov chain (S, T_S) represented by C (with transitions $T(s) = \lambda_1 s_1 + \cdots + \lambda_k s_k$ for $E_s = \{(s, \lambda_j, r_j, s_j) \mid 1 \leqslant j \leqslant k\}$). By the above correspondence, the paths have equal probabilities. From the soundness of the abstraction, we get a correspondence regarding reachability of the error:

- $s_m = (n, u)$ is an error state ($n \in N_e$) iff q_m is an error state ($q_m \in Q_e$);
- if $s_m = (n, u)$ with $n \in N_d$, i.e. N_e is not reachable from n, then Q_e is likewise not reachable from q_m;
- if $s_m \in O$, then the error may be reachable from q_m with any probability in $[0, 1]$ (we made no assumption on the behaviour of σ outside $Q_{\overline{S}}$).

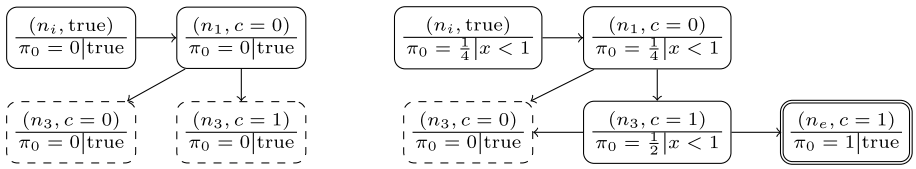

Fig. 2. Two partial concretisations for the MDP abstraction of Fig. 1.

This implies that the reachability probabilities of $S_e = \{(n, u) \in S \mid n \in N_e\}$ and of $S_e \cup O$ are a lower and an upper bound on p^{σ}_P, and (see e.g. Thm. 1.3.2 in [22]) they are exactly the least solutions of:

- $p(s) = 1$ if $s \in S_e$ (resp. $s \in S_e \cup O$), and
- $p(s) = \sum T(s)(s')p(s')$ otherwise,

making them equal to π_0 and π_1. □

Example 3. We return to the MDP abstraction and abstract adversary σ_a of Example 2, shown in Fig. 1 (right). Fig. 2 (left) shows a partial concretisation. The top of each box shows the discrete state; the bottom shows the lower probability bound π_0 (since we are computing maximum probabilities) and time constraint $\eta(s)$. Unexpanded states are drawn with dashed lines. For clarity, edge details are omitted. Fig. 2 (right) shows a subsequent partial concretisation, expanding state $(n_3, c = 1)$. The outgoing transition contains a time constraint $x < 1$, which gets propagated backwards. One successor is error state $(n_e, c = 1)$, leading to an increase in the lower bounds.

3.2. The refinement loop

Our approach to computing reachability probabilities for PTPs is based on an iterative abstraction refinement loop, which generates increasingly precise MDP abstractions. At each iteration, we first compute minimum or maximum reachability probabilities for the MDP, yielding an (outer) bound $p^{lb,\min}$ or $p^{ub,\max}$ and an abstract adversary. Then, we partially concretise this adversary, based on a forward exploration through the model, yielding after each step a complementary (inner) probability bound $p^{ub,\min} = \pi_1(n_i, u_i)$ or $p^{lb,\max} = \pi_0(n_i, u_i)$. If the difference between the lower and upper bounds for the initial state of the model falls below a pre-specified tolerance ε, then abstraction refinement terminates. Otherwise, concretisation continues until an *inconsistency* is identified, which will be used to *refine* the abstraction.

There are two classes of inconsistencies which may occur during concretisation: *state-based* inconsistencies, which, due to tightness, will always manifest themselves as the failure of a valuation to satisfy the guard of the adversary's chosen transition, and *time-based* inconsistencies, which occur when there is no consistent choice of delay, and manifest themselves as the occurrence of an empty zone $\eta(\_) = \emptyset$ in the concretisation. Accordingly, there are two separate refinement operations for an MDP abstraction: *state refinement*, which splits a node with a predicate $\varphi \in Asrt(\mathcal{V})$; and *time refinement*, which splits based on an inconsistent set of clock difference constraints $c_1, \ldots, c_k$. The former is relatively standard, for abstraction refinement techniques; the latter is a novel method that we have developed for the PTP model. In the next two sections, we describe each of these in more detail. The main refinement loop is sketched in Fig. 3. Pseudocode and descriptions of auxiliary functions addState(), expand() and backpropagate() are shown in Figs. 4 and 5. Details of the refinement functions stateRefine() and timeRefine() are given in the next sections. The algorithm maintains an MDP abstraction $A = (N, n_i, N_e, T)$, along with:

- an abstract adversary σ_a and resulting outer bounds $p^{\min}_{A,n}$ or $p^{\max}_{A,n}$;
- a partial concretisation (S, O, E), along with the associated inner bounds $\pi_b(s)$ and time constraints $\eta(s)$ and $\eta(e)$ for all $s \in S$ and $e \in E$;
- a set $bp \subseteq S$ of states whose time constraints still need to be strengthened to satisfy the required fixpoint equations (see p. 43).

The outcome of the algorithm, if it terminates, is a set containing an inner and an outer bound $\{\pi_b(n_i, u_i), p^{dir}_{A,n_i}\}$. Note that termination cannot be guaranteed in general, since the class of PTPs is Turing complete (it contains counter automata as a subclass). If the system is a PTA, then in the worst case the refinement procedure constructs the region graph, since in each iteration we nontrivially split a clock zone; see the proof of Theorem 5.

3.3. State refinement

State refinement is triggered when forward exploration encounters a state (n, u) such that u does not satisfy the guard $\mathcal{G}$ of the transition $\sigma_a(n)$. Since $\mathcal{G}$ is a conjunctive quantifier-free formula, at least one of its atomic constraints is violated by u. We use the first failed constraint g to split n, introducing a case distinction. This makes the refinement *local* in the

```
Input: PTP P, dir ∈ {min, max}, ε ≥ 0
A := A_i(P);
b := if dir = min then 1 else 0;
Loop:
    compute σ_a and p^dir_{A,n} for n ∈ N;
    S, O, E, bp := ∅;
    addState(n_i, u_i); π_b(n_i, u_i) := b;
    while |p^dir_{A,n_i} − π_b(n_i, u_i)| > ε do
        take s = (n, u) from O;
        let (G, E, Δ) = σ_a(n);
        if u ⊭ G then stateRefine(n,u); goto Loop;
        expand(s);
        if backpropagate() then goto Loop;
        update π_b;
    return {π_b(n_i, u_i), p^dir_{A,n_i}};
```

Fig. 3. The refinement loop. In each iteration, the abstract adversary is computed, followed by a partial concretisation. The inner loop uses functions expand(s) to obtain the successors of s and add them to the concretisation, and backpropagate() to ensure consistency of the time constraints. It is aborted if a refinement step occurred.

```
addState(s)
    if s ∉ S then
        let (n, u) = s,
            (l, φ, ρ) = n;
        add s to S;
        η(s) := ρ;
        if n ∉ N_d ∪ N_e then
            add s to O;
```

```
expand(s)
    let (n, u) = s, (G, E, Δ) = σ_a(n);
    foreach λ_j(f_j, r_j, n_j) in Δ do
        s_j := (n_j, f_j(u));
        addState(s_j);
        add e_j := (s, λ_j, r_j, s_j) to E;
        η(e_j) := E ∩ [r_j := 0]η(s_j);
        if η(s) ⊈ η(e_j) then add s to bp;
```

Fig. 4. Procedure addState(s), left, adds a new state s to the partial concretisation, initialises its time constraints $\eta(s)$, and schedules it for expansion (puts it in O) if necessary. Procedure expand(s), right, determines the successors $s_1, \ldots, s_k$ of s, adds the edges from s to s_j, and adds s to bp if its time constraints now need to be strengthened.

```
backpropagate()
    while bp ≠ ∅ do
        take s from bp, where s = (n, u) and n = (l, φ, ρ);
        η' := ℝ^X_{≥0};
        foreach e ∈ E_s do η' := η' ∩ η(e);
        if η' = ∅ then
            C := unsatisfiable core for ρ, {η(e) | e ∈ E_s};
            timeRefine(n, C); return true;
        η(s) := ρ ∩ ↙ η';
        foreach e = (s', λ', r', s) ∈ E do
            let (n', u') = s', (G', E', Δ') = σ(n');
            η(e) := E' ∩ [r' := 0]η(s);
            if η(e) = ∅ then
                C := unsatisfiable core for E'[r' := 0], η(s);
                timeRefine(n, C); return true;
            if η(s') ⊈ η(e) then add s' to bp;
    return false;
```

Fig. 5. backpropagate() strengthens the constraints $\eta(\_)$ in order to make them consistent. A state s is in bp if $\eta(s)$ does not imply $\eta(e)$ for some edge $e = (s, \lambda, r', s')$. The function takes states from bp and strengthens their constraints (potentially causing new additions to bp) until a contradiction occurs (in which case a refinement step is triggered and we return *true*) or bp is empty (in which case a fixpoint was reached, and we return *false*). Since in each iteration one of the finitely many zones $\eta(s)$ shrinks, and zones satisfy the descending chain condition, the procedure always terminates.

sense that we only use information directly related to state (n, u), rather than having to take the entire concretisation into account. The result of this split is a pair of new nodes $n_+ = (l, \varphi \wedge g, \rho)$ and $n_- = (l, \varphi \wedge \neg g, \rho)$. We then modify the sets of transitions:

- first, n_+ and n_- both inherit the outgoing transitions in $T(n)$;
- then, for each $\tau = (\mathcal{G}, \mathcal{E}, \lambda_1(f_1, r_1, n_1) + \cdots + \lambda_k(f_k, r_k, n_k))$ in A:
  - find the indices $I := \{j \mid n_j = n\}$ of edges which need to be redirected;
  - for each possible redirection $\nu : I \to \{n_+, n_-\}$, add a new transition $\tau_\nu := (\mathcal{G}_\nu, \mathcal{E}, \lambda_1(f_1, r_1, n'_1) + \cdots + \lambda_k(f_k, r_k, n'_k))$ with $n'_j = \nu(j)$ for $j \in I$, $n'_j = n_j$ otherwise, and $\mathcal{G}_\nu$ obtained from $\mathcal{G}$ by adding the corresponding preconditions $g \circ f_j$ or $\neg g \circ f_j$ for $j \in I$;
  - for each $n' = (l', \varphi', \rho')$ with $\tau \in T(n')$, replace τ by those τ_ν for which $\varphi' \wedge \mathcal{G}_\nu$ is satisfiable.

```
stateRefine(n, u)
    if n = n_i then remove σ_a(n_i) from T(n_i); return;
    let (l, φ, ρ) = n, (G, E, Δ) = σ_a(n);
    g := first constraint from G with u ⊭ g;
    n_+ := (l_n, φ ∧ g, ρ);
    n_− := (l_n, φ ∧ ¬g, ρ);
    T(n_+) := T(n_−) := T(n);
    split(n, {n_+, n_−});
```

Fig. 6. stateRefine(n,u) splits node n into new nodes n_+, n_- based on a guard constraint violated by u.

```
split(n, N')
    N := (N \ {n}) ∪ N';
    foreach n' = (l, φ, ρ) ∈ N do
        T* := ∅;
        foreach τ ∈ T(n') and τ' = (G', E', Δ') ∈ tsplit(τ, n, N') do
            if φ ∧ G' satisfiable and ρ ∩ E' ≠ ∅ then add τ' to T*;
        T(n') := T*;
    remove unreachable nodes;
```

Fig. 7. split(n, N') replaces the node n with the set N' of nodes into which n has been split, using tsplit (Fig. 8) to obtain the corresponding sets of transitions.

```
tsplit(τ, n, N')
    let (G, E, λ_1(f_1, r_1, n_1) + ··· + λ_k(f_k, r_k, n_k)) = τ;
    S := ∅; I := {j | n_j = n};
    foreach ν : I → N' do
        for j = 1, ..., k do
            n'_j := if j ∈ I then ν(j) else n_j;
            let (l'_j, φ'_j, ρ'_j) = n'_j;
        G_ν := G ∧ ⋀_{j∈I}(φ'_j ∘ f_j);
        E_ν := E ∩ ⋂_{j∈I}[r_j := 0]ρ'_j;
        τ_ν := (G_ν, E_ν, λ_1(f_1, r_1, n'_1) + ··· + λ_k(f_k, r_k, n'_k));
        add τ' to S;
    return S;
```

Fig. 8. tsplit(τ, n, N') is an auxiliary function for the refinement procedures. Given that n is split into a set N' of nodes, it computes all possible variants of the given transition τ where each edge leading to n is redirected to some node in N' instead.

Satisfiability checks needed in the last step of the above are performed using an SMT solver. Pseudocode for the refinement procedure stateRefine(), along with auxiliary functions split() and tsplit() are shown in Figs. 6, 7, and 8.

Note that the strengthening of $\mathcal{G}$ to $\mathcal{G}'$ ensures the refined abstraction is again tight, and this is the step which actually introduces new predicates into the abstraction. Furthermore, the refined abstraction remains sound.

Theorem 4. *Let $A = (N, n_i, N_e, T)$ be an MDP abstraction of PTP P, and let $A' = (N', n'_i, N'_e, T')$ be obtained from A by a state refinement. If A is sound for P, then so is A'. If A is tight, then so is A'. Furthermore, A' is a proper refinement of A in the sense that the state assertions $\varphi \wedge g$ and $\varphi \wedge \neg g$ in the new nodes are both satisfiable.*

Proof. Let ∇ be a reflection of adversaries for A, and let A' be obtained from A by splitting n into n_+, n_- using the constraint g. We can then adapt ∇ by defining for each $wq \in Q_c^+$ with $q = (l, u, v)$ and each $\delta \geqslant 0$:

$$\nabla'(wq, \delta) = \begin{cases} n_+ & \text{if } \nabla(wq, \delta) = n, u \vDash g, \\ n_- & \text{if } \nabla(wq, \delta) = n, u \nvDash g, \\ \nabla(wq, \delta) & \text{otherwise.} \end{cases}$$

Since n_i and error nodes $n \in N_e$ are never split (the former because it only occurs in (n_i, u_i), and inconsistency with u_i means that $\sigma(n_i)$ cannot be taken and can be discarded; the latter because there are no outgoing transitions), we still have $\nabla'(wq, \delta) = n_i$ iff $q = (l_i, \_, \_)$ and $\nabla'(wq, \delta) \in N_e$ iff $q \in Q_e$. Let $w \in Q_c^+$, and σ be a (concrete) adversary for P with $\sigma(w) = \lambda_1 q_1 + \cdots + \lambda_k q_k$ induced by $\delta_\sigma(w) \in \mathbb{R}_{\geqslant 0}$ and $\tau_\sigma(w) = (\mathcal{G}, \mathcal{E}, \lambda(f_1, r_1, l_1) + \cdots + \lambda_k(f_k, r_k, l_k))$. Since ∇ is a reflection of adversaries, $T(\nabla_\sigma(w))$ contains a corresponding abstract transition $\tau_a = (\mathcal{G}', \mathcal{E}', \lambda_1(f_1, r_1, \nabla_\sigma(wq_1)) + \cdots + \lambda_k(f_k, r_k, \nabla_\sigma(wq_k)))$. Let I be the set $\{j \mid \nabla_\sigma(wq_j) = n\}$ and let the function $\nu : I \to \{n_+, n_-\}$ be given by $\nu(j) = n_+$ iff $q_j \vDash g$. The refinement creates (among others) the new abstract transition $\tau_\nu := (\mathcal{G}'', \mathcal{E}', \lambda_1(f_1, r_1, \nabla'_\sigma(wq_1)) + \cdots + \lambda_k(f_k, r_k, \nabla'_\sigma(wq_k)))$ with $\mathcal{G}''$ obtained from $\mathcal{G}'$ by adding the respective preconditions.

Existence of the concrete transition $\sigma(w)$ implies that $\nabla'_\sigma(w)$ satisfies these preconditions, and τ_ν occurs in $T'(\nabla'_\sigma(w))$ after the split. So ∇' is a reflection of adversaries. As for concretisability of transitions, let $\tau' = (\mathcal{G}', \mathcal{E}', \lambda_1(f_1, r_1, n'_1) + \cdots +$

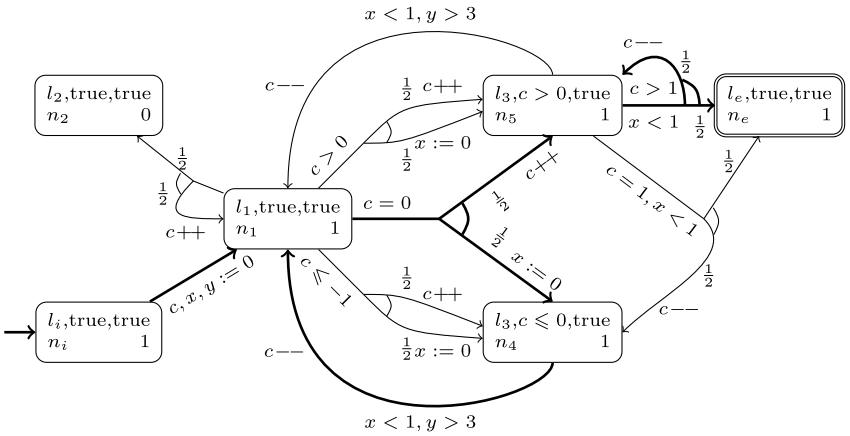

Fig. 9. MDP abstraction for the running example, after state refinement (see Example 4).

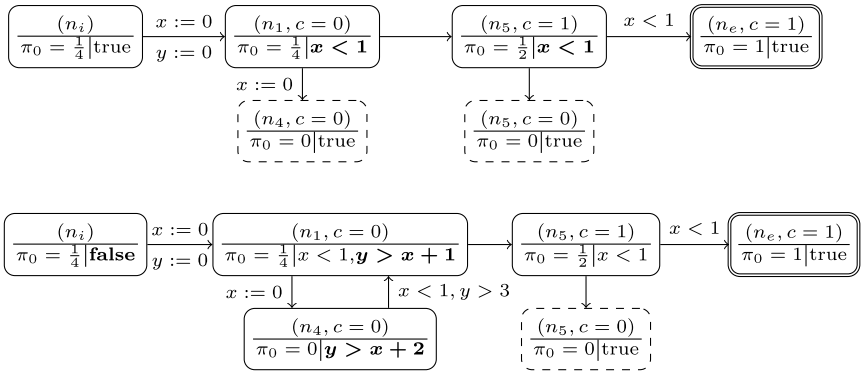

Fig. 10. Partial concretisations for the refined abstraction of Fig. 9 (see Example 4).

$\lambda_k(f_k, r_k, n'_k))$ be a transition in A'. Then τ' was obtained from a transition $\tau = (\mathcal{G}, \mathcal{E}, \lambda_1(f_1, r_1, n_1) + \cdots + \lambda_k(f_k, r_k, n_k))$ in A such that:

(i) $\mathcal{G}' \Rightarrow \mathcal{G}$ and $\mathcal{E}' \subseteq \mathcal{E}$;
(ii) for each j, either $n_j = n'_j$ or $n_j = n, n'_j \in \{n_-, n_+\}$; and
(iii) either $\tau \in T(n'), \tau' \in T'(n')$ for some n', or $\tau \in T(n), \tau' \in T'(n_-) \cup T'(n_+)$.

In each case, the associated locations are the same, so any concrete transition witnessing concretisability of τ also does so for τ', and we are done.

In order to preserve tightness, all transitions have their guard $\mathcal{G}$ strengthened by the preconditions of the newly introduced constraints as needed; note that time constraints are copied from n to n_+, n_- and remain unchanged otherwise, so that $\mathcal{E}$ does not need to be changed.

Finally, $\varphi \wedge g$ is satisfiable since otherwise the transition $\sigma_a(n) = (\mathcal{G}, \mathcal{E}, \Delta)$ would not have been in $T(n)$, and $\varphi \wedge \neg g$ is satisfiable since the valuation u giving rise to the split satisfies both. □

Example 4. We return to the partial concretisation shown in Fig. 2 (right). The next step of forward exploration, from $(n_3, c = 0)$, fails since the guard constraint $c > 0$ is violated. Thus, node n_3 is split using the predicate $c > 0$. Fig. 9 shows the resulting MDP abstraction, where node n_3 has been split into n_4 and n_5, representing the states with $c > 0$ and $c \leqslant 0$, respectively. This leads to a corresponding split of transitions. Note the strengthening of guards with preconditions to ensure tightness. Fig. 10 shows two successive partial concretisations for the new abstraction. The first step encounters the clock constraint $x < 1$ on the transition from n_5 and propagates it backwards. In the second step, we expand $(n_4, c = 0)$, encountering the constraints $x < 1$, $y > 3$. Backpropagation then obtains the constraints:

- $\swarrow (x < 1 \wedge y > 3) \equiv y > x + 2$, added to $(n_4, c = 0)$;
- $[x := 0](y > x + 2) \equiv y > 2$, added to the edge $(n_1, c = 0) \to (n_4, c = 0)$;

timeRefine(n, C)
  let $(l, \varphi, \rho) = n, \{c_1, \ldots, c_m\} = C$, with $c_j \equiv x_{i_{j-1}} - x_{i_j} \lesssim_j b_j$;
  /* where $i_0 = i_m$, and ordered such that only i_0 can be 0 */
  if $i_0 > 0$ **then**
    for $j = 1, \ldots, m$ **do**
      $\rho' := c_1 \cap \cdots \cap c_{j-1} \cap \overline{c_j}$;
      $n_j^* := (l, \varphi, \rho \cap \rho')$;
      $T(n_j^*) := \{(\mathcal{G}, \mathcal{E} \cap \rho', \Delta) \mid (\mathcal{G}, \mathcal{E}, \Delta) \in T(n)\}$;
  else /* $c_1, \overline{c_m}$ are lower bounds */
    $n_1^* := (l, \varphi, \rho \cap \overline{c_1})$;
    $T(n_1^*) := T(n)$;
    for $j = 2, \ldots, m-1$ **do**
      $\rho' := c_2 \cap \cdots \cap c_{j-1} \cap \overline{c_j}$;
      $n_j^* := (l, \varphi, \rho \cap \rho')$;
      $T(n_j^*) := \{(\mathcal{G}, \mathcal{E} \cap c_1 \cap \rho', \Delta) \mid (\mathcal{G}, \mathcal{E}, \Delta) \in T(n)\}$;
    $n_m^* := (l, \varphi, \rho \cap c_2 \cap \cdots \cap c_{m-1})$;
    $T(n_m^*) := \{(\mathcal{G}, \mathcal{E} \cap c_1 \cap \cdots \cap c_{m-1} \cap \overline{c_m}, \Delta) \mid (\mathcal{G}, \mathcal{E}, \Delta) \in T(n)\}$;
  split$(n, \{n_1^*, \ldots, n_m^*\})$;

Fig. 11. timeRefine(n, C) splits a node with time constraints obtained from an inconsistency. Note the strengthening of the enabledness conditions $\mathcal{E}$, which amounts to intersecting them with the corresponding source invariants. We distinguish two cases, depending on whether c contains single-clock bounds; see the proof of Theorem 5 for details.

- $\swarrow (x < 1 \wedge y > 2) \equiv x < 1 \wedge y > x + 1$, added to $(n_1, c = 0)$;
- $[x := 0, y := 0](x < 1 \wedge y > x + 1) \equiv$ false, triggering a *time refinement* step.

3.4. Time refinement

The refinement step for timing inconsistencies is more complex, for several reasons. Firstly, because we treat time symbolically, a failure of concretisability cannot always be localised as it could for the state predicate case. In general, for some discrete state s, we will be dealing with a contradiction between the constraints for some incoming edge (s_0, λ_0, r_0, s) and those for the outgoing edges (s, λ_j, r_j, s_j), $j = 1, \ldots, k$. In particular, we will need to use more than one difference constraint for splitting and obtain more than two new nodes.

Secondly, because of the implicit passing of time between transitions, we must be careful when splitting with difference constraints: suppose one of the constraints is a lower bound like $x \geqslant 5$. Naively splitting $n = (l, \varphi, \rho)$ into $(l, \varphi, \rho \cap (x < 5))$ and $(l, \varphi, \rho \cap (x \geqslant 5))$ wrongly eliminates any path which enters n while $x < 5$, lets time pass until $x \geqslant 5$, and then leaves n. We avoid this problem by not adding the lower bound $x \geqslant 5$ to the *invariant* of the second node; instead, we add it to the *enabledness conditions* of its outgoing transitions, so that it only has to hold when the node is left.

One consequence of this is that the abstraction will not represent a partition of states, in the sense that exactly one abstract state is associated to each concrete state. Instead, we have a partition (captured by the notion of reflection) of *pairs* (q, δ) of a concrete state q and a delay $\delta \geqslant 0$, in the sense that exactly one abstract state is associated to each such pair.

A time refinement step is triggered whenever, during the backpropagation of timing constraints, we find $s = (n, u)$ such that strengthening $\eta(s)$ with the constraints $\eta(e)$ on the outgoing edges $e = (s, \lambda, r, s') \in E_s$, and successively strengthening the constraints $\eta(e')$ on the incoming edges with $\eta(s)$, would encounter an empty zone. Given such a contradiction involving $n = (l, \varphi, \rho)$, the backpropagation algorithm computes an unsatisfiable core $C = \{c_1, \ldots, c_m\}$ (see Section 2.1), and calls timeRefine, which, for each $j = 1, \ldots, m$:

- defines a set of constraints $\psi_j = \{c_1, \ldots, c_{j-1}, \overline{c_j}\}$ – note that since C is unsatisfiable, every clock valuation satisfies at least one of the complements $\overline{c_i}$, so that $\psi_1, \ldots, \psi_m$ define a partition of $\mathbb{R}_{\geqslant 0}^{\mathcal{X}}$ into zones;
- introduces a new state $n_j^* = (l, \varphi, \rho \cap \psi_j')$, where ψ_j' is given by the constraints in ψ_j not of the form $x \gtrsim c$;
- adds to $T(n_j^*)$ all transitions from $T(n)$, strengthening their enabledness conditions by the constraints in ψ_j to take care of lower bounds and ensure that again $\mathcal{E} \subseteq \rho$.

Finally, the transitions in A are split like in the state-based case. For each $\tau = (\mathcal{G}, \mathcal{E}, \lambda_1(f_1, r_1, n_1) + \cdots + \lambda_k(f_k, r_k, n_k))$, we determine the set $I := \{j \mid n_j = n\}$. Then, for each $\nu : I \to \{n_1^*, \ldots, n_m^*\}$, we compute $\tau_\nu := (\mathcal{G}, \mathcal{E}_\nu, \lambda_1(f_1, r_1, n_1') + \cdots + \lambda_k(f_k, r_k, n_k'))$ where $n_j' = \nu(j)$ for $j \in I$, $n_j' = n_j$ otherwise, and $\mathcal{E}_\nu$ is the intersection of $\mathcal{E}$ with the target node preconditions. Lastly, for each $n' = (l', \varphi', \rho')$ with $\tau \in T(n')$, we replace τ by those τ_ν for which $\mathcal{E}_\nu$ is non-empty. Pseudocode for the time refinement procedure timeRefine() is given in Fig. 11.

Theorem 5. *Let $A = (N, n_i, N_e, T)$ be an MDP abstraction for PTP P and $A' = (N', n_i, N_e, T')$ be obtained from A by a time refinement step. If A is sound for P, then so is A'. If A is tight, then so is A'. Furthermore, A' is a proper refinement of A in the sense that, for each of the new nodes n_j^*, either its invariant or the enabledness condition on its outgoing transitions is stronger.*

Proof. Let ∇ be a reflection of adversaries for A and A' be obtained from A by splitting $n = (l, \varphi, \rho)$ into $n_1^*, \ldots, n_m^*$ using an unsatisfiable core $c_1, \ldots, c_m$. Like in the proof of Theorem 3, we can refine ∇ by mapping all w, δ with $\nabla(w, \delta) = n$ to a suitable n_j^* instead. We make one simplifying assumption on the ordering of $c_1, \ldots, c_m$. Recall that any variable occurs in exactly 2 or 0 constraints of an unsatisfiable core; in particular, there is at most one lower bound $x \gtrsim b$ and one upper bound $x' \lesssim b'$. We assume that, if these do occur, they are c_1 and c_m, respectively.

We first treat the case in which each c_j is a proper difference constraint, i.e. not an upper/lower bound. In this case, the new nodes form a partition $n_j^* = (l, \varphi, \rho \cap \eta_j)$, where the η_j are pairwise disjoint and closed under time elapse. This means that, for each $q = (l, u, v) \in Q_c$, there is an index j_q such that $v + \delta \in \rho \cap \eta_{j_q}$ for all delays $\delta \geqslant 0$ with $v + \delta \in \rho$. We thus define:

$$\nabla'(wq, \delta) = \begin{cases} n_{j_q}^* & \text{if } \nabla(wq, \delta) = n, \\ \nabla(wq, \delta) & \text{otherwise.} \end{cases}$$

This again ensures $\nabla'(wq, \delta) = n_i$ iff $q = (l_i, \_, \_)$ and $\nabla'(wq, \delta) \in N_e$ iff $q \in Q_e$.

Let $w \in Q_c(P)$ and $\sigma(w) = \lambda_1 q_1 + \cdots + \lambda_k q_k$, induced by $\delta_\sigma(w) \in \mathbb{R}_{\geqslant 0}$ and $\tau_\sigma(w) = (\mathcal{G}, \mathcal{E}, \lambda(f_1, r_1, l_1) + \cdots + \lambda(f_k, r_k, l_k))$. Since ∇ is a reflection of adversaries, $T(\nabla_\sigma(w))$ contains a transition $\tau_a = (\mathcal{G}', \mathcal{E}', \lambda_1(f_1, r_1, \nabla_\sigma(wq_1)) + \cdots + \lambda_k(f_k, r_k, \nabla_\sigma(wq_k)))$. The refinement produces a corresponding abstract transition $\tau_a' = (\mathcal{G}', \mathcal{E}'', \lambda_1(f_1, r_1, \nabla_\sigma'(wq_1)) + \cdots + \lambda_k(f_k, r_k, \nabla_\sigma'(wq_k)))$, where $\mathcal{E}''$ is obtained by intersecting $\mathcal{E}'$ with the invariant of $\nabla_\sigma'(w)$ and the preconditions of the zones $\nabla_\sigma'(wq_j)$. The existence of the concrete transition $\sigma(w)$ shows that $\mathcal{E}''$ is non-empty, and therefore τ_a' will be included in $T'(\nabla_\sigma'(w))$.

In case we do have lower and upper bounds $c_1 \equiv x_1 \gtrsim b_1, c_m \equiv x_m \lesssim b_{m-1}$, the split of n involves a fragment n_1^* representing all "early" steps, i.e. those taken while c_1 is false, all others requiring the outgoing transitions to occur after it becomes true. We reflect this in ∇' as follows, for $wq \in Q_c^+$ with $q = (l, u, v)$:

$$\nabla'(wq, \delta) = \begin{cases} n_1^* & \nabla(wq, \delta) = n, v + \delta \not\models c_1, \\ n_{j_q}^* & \nabla(wq, \delta) = n, v + \delta \models c_1, \\ \nabla(n, \delta) & \text{otherwise,} \end{cases}$$

where again j_q is the unique index with $v + \delta \in \rho \cap \eta_{j_q}$. Otherwise, the argument works exactly as in the first case.

As in the state refinement case, we again observe that any transition $\tau' \in T'(n')$ is obtained from a transition in A whose source node belongs to the same location as n', each of whose target nodes belongs to the same location as the corresponding target in τ', and whose enabledness conditions are implied by those in τ', so that A' inherits concretisability of transitions from A.

Tightness is preserved by strengthening the enabledness condition $\mathcal{E}$ in each transition with the same constraints as the invariant in its source node (in addition to lower bounds which are only added to $\mathcal{E}$), and also with the pre-images $[r := 0]\rho$ of the invariants in their new target nodes (done in tsplit()). State assertions are copied from n to the n_j^* and remain unchanged otherwise, so that $\mathcal{G}$ does not need to be strengthened.

To show that A' is a proper refinement, it is enough to show that, for each constraint c_j in the unsatisfiable core, $\overline{c_j}$ is not implied by ρ, i.e. c_j is satisfied by some $t \in \rho$. There are two cases, depending on where in the backpropagation the inconsistency giving rise to the split occurred.

If it happened when trying to strengthen $\eta(s)$ for $s = (n, u)$, then:

$$\begin{aligned} \rho \cap \swarrow \left(\rho \cap \bigcap_{e \in E_s} \eta(e)\right) = \emptyset \quad &\Leftrightarrow \quad \nearrow \rho \cap \left(\rho \cap \bigcap_{e \in E_s} \eta(e)\right) = \emptyset \\ &\Leftrightarrow \quad \rho \cap \bigcap_{e \in E_s} \eta(e) = \emptyset \end{aligned}$$

and the split is performed with an unsatisfiable core C for $\rho, \{\eta(e) \mid e \in E_s\}$. By tightness, for each $e \in E_s$, $\eta(e) \subseteq \mathcal{E} \subseteq \rho$, and $\eta(e) \neq \emptyset$ since otherwise a refinement would have been triggered earlier. So each constraint in C is implied by a non-empty subset of ρ, and therefore satisfied by some $v \in \rho$.

If the inconsistency occurred when strengthening $\eta(e)$ for $e = (s', \lambda, r, s)$ with $s = (n, u)$ and $\sigma_a(n) = (\mathcal{G}, \mathcal{E}, \Delta)$, then:

$$\mathcal{E} \cap [r := 0]\eta(s) = \emptyset \quad \Leftrightarrow \quad \mathcal{E}[r := 0] \cap \eta(s) = \emptyset,$$

and the split is performed with an unsatisfiable core C for $\mathcal{E}[r := 0], \eta(s)$. By tightness, both are subsets of ρ. $\mathcal{E}[r := 0]$ is non-empty since $\mathcal{E}$ is non-empty, and $\eta(s)$ is non-empty since otherwise a refinement would have been triggered earlier. Again, each constraint in C is implied by a non-empty subset of ρ, and therefore satisfied by some $v \in \rho$. □

Example 5. For presentational simplicity, we illustrate time refinement on a separate example. Fig. 12 (top left) shows part of an MDP abstraction in which we want to split n_1 with $c = \{x > 2, x < y, y < 1\}$. From c, we first obtain a partition $\{x \leqslant 2, x > 2 \wedge x \geqslant y, x > 2 \wedge x < y \wedge y \geqslant 1\}$. The straightforward analogue to the state-based case, using these zones as invariants in three copies of n_1, falsely makes it impossible for time to progress beyond 2 in l_1. Adding lower bounds such as $x > 2$ to enabledness conditions instead of node invariants (as on the transition $n_5 \to n_2$ in the corrected version) fixes this.

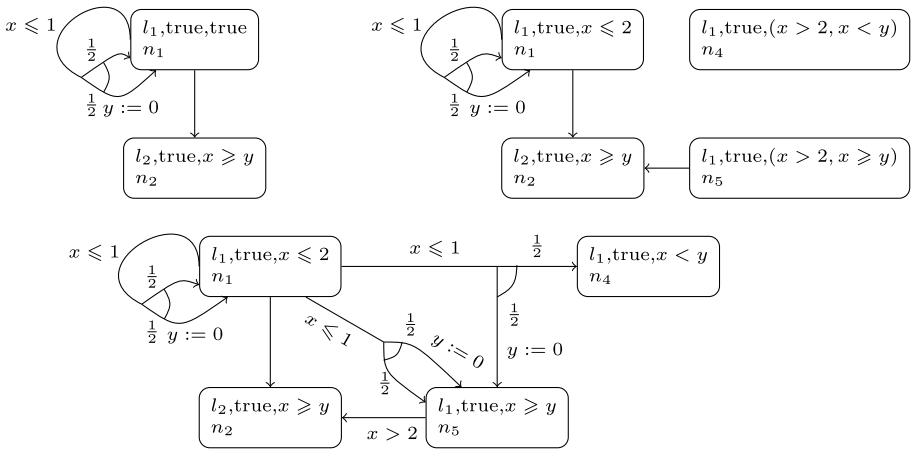

Fig. 12. Time refinement (see Example 5): an MDP abstraction (top left); a naive, faulty refinement, splitting node n_1 (top right); and a correct one (bottom).

4. Experiments

We have built a prototype implementation of our local abstraction refinement approach on top of the probabilistic model checker PRISM [18]. We use an extended version of PRISM's existing DBM library for manipulating zones and the MathSAT 4 SMT solver [4] for satisfiability queries. Experiments were run on a 64-bit PC with an Intel Xeon CPU X5660 2.80 GHz and 32 GB RAM. Our implementation is parameterised by the strategy used to search the state space during concretisation. For models with unbounded data types (see below), we use breadth-first search and set $\varepsilon = 1 \times 10^{-6}$ for termination of refinement; for other models we use depth-first search and set ε to zero.

PRISM and its modelling language already have native support for PTAs, and these can incorporate data variables using PRISM's built-in datatypes (bounded integers and Booleans) so we use this as a basis for modelling PTPs. We evaluated our implementation on 11 case studies. Firstly, we used 6 existing PTA benchmarks [26]: *csma full*, *csma abst* (two models of the CSMA/CD protocol); *nrp malicious*, *nrp honest* (two variants of Markowitch & Roggeman's non-repudiation protocol); and *firewire impl*, *firewire abst* (two models of the FireWire root contention protocol). Secondly, we used real-time versions of two more complex models: the Zeroconf model of [13] (*zeroconf full*), and the sliding window model of [11] (*sliding window real-time*), which uses infinite data types (an unbounded integer storing the round counter). Finally, we tested our implementation on three discrete-time (MDP) models with infinite data types: the (original, discrete-time) sliding window protocol (*sliding window discr.*) and bounded retransmission protocol (*brp*) models from [11]; and the discrete-time model of Zeroconf (*zeroconf discr.*) from [13], modified so that the counter of "probe" messages sent is an unbounded integer. This change does not affect the value of the property we check. For *brp*, we fix the parameter *TIMEOUT* = 16. In order to support infinite data types, we slightly extended the PRISM modelling language in the style of PASS [9]. All models and properties, and our prototype implementation, are available at [27].

We compare our approach to other available tools. For PTPs with only bounded datatypes, we run PRISM [18] (v.4.0.1), using its game-based abstraction refinement [13] engine (which outperforms digital clocks [19] on all but the *zeroconf full* case study). We also compare to FORTUNA [3] (v.0.2), which verifies (priced) PTAs, but we are only able to run a subset of the benchmarks since models are hard-coded into the tool. FORTUNA only handles maximum probabilities, but can compute minimum probabilities for some models through hard-coded model translations. For the three MDP models, we compare to the predicate abstraction tool PASS [9]. None of these tools are applicable for the *sliding window real-time* model, which uses both clocks and unbounded data types. Recent work [8] describes an extension of PASS for real-time models. Presently, we only have access to the experimental results in [8], rather than the tool; we make a brief comparison below.

For the *csma full* benchmark, we pre-process models to reduce the degree of probabilistic branching and avoid transition explosion during refinement (details at [27]). PRISM performs better on the original so we use that for comparison.

4.1. Results

Experimental results for the real-time models are summarised in Tables 1 and 2, covering maximum and minimum probability properties, respectively. The results for discrete-time (MDP) models are in Table 3. For each abstraction refinement method (PRISM, PASS and our local abstraction refinement), we show the number of abstract states (both the peak and final number), the number of iterations of refinement and the total time. For FORTUNA, we just show states and time, since no refinement is performed.

Our prototype consistently outperformed PRISM's game-based abstraction for PTAs except on the *firewire impl* model with $T = 10\,000$. We use a significantly higher number of refinement steps, but these are small, local refinements and, crucially, the size of the abstraction constructed is typically much smaller. It should be noted, though, that the *zeroconf full*

Table 1
Experimental results for real-time models (maximum probabilities).

Model & parameters		PRISM (stoch. games)			Local abstr.-refinement			FORTUNA		Probability
		States (peak/final)	Iter.	Time (s)	States (peak/final)	Iter.	Time (s)	States	Time (s)	
csma full [*bmax*, *K*]	3, 8	69 123/69 123	10	18.6	3215/3215	1714	6.3	1058	**1.2**	2.32e−4
	3, 16	168 525/168 525	0	42.4	8032/8032	5419	28.1	2266	**2.9**	2.11e−9
	4, 8	239 353/239 353	10	151.4	8292/8292	3320	20.0	2315	**4.4**	1.65e−5
	4, 16	646 019/646 019	0	461.8	18 548/18 548	10 728	83.0	4547	**9.1**	7.65e−13
nrp malicious [*T*]	20	30 088/30 088	6	36.8	556/556	540	4.6	635	**1.9**	0.10566
	25	58 494/58 494	6	80.7	490/490	482	**2.1**	805	4.3	0.10566
	30	85 807/85 807	6	124.5	770/770	762	**7.6**	crashed		0.10566
	35	122 182/122 182	6	227.6	968/968	949	**7.4**	1145	10.4	0.10566
zeroconf full [*N*, *M*, *K*]	3, 8, 4	142 627/1690	0	13.4	2273/2273	310	**3.4**	n/a		3.021e−5
	3, 8, 8	231 363/3162	0	21.5	2476/2476	405	**4.7**	n/a		2.351e−9
	4, 8, 4	1 979 351/5481	0	275.7	13 116/434	849	**33.9**	n/a		1.464e−4
	4, 8, 8	3 109 815/8737	0	489.7	13 667/818	1189	**53.5**	n/a		5.013e−8
sliding window real-time [*N*, *p*]	8, 0.5	n/a			4666/4666	3561	**61.8**	n/a		0.78123
	8, 0.98	n/a			4286/4286	3231	**284.1**	n/a		0.05805
	10, 0.5	n/a			5888/5888	4455	**104.9**	n/a		0.78125
	10, 0.98	n/a			5317/5317	4013	**614.1**	n/a		0.05805

Table 2
Experimental results for real-time models (minimum probabilities).

Model & parameters		PRISM (stoch. games)			Local abstr.-refinement			FORTUNA		Probability
		States (peak/final)	Iter.	Time (s)	States (peak/final)	Iter.	Time (s)	States	Time (s)	
csma abst [*bmax*, *T*]	1, 1000	6420/6420	0	1.3	108/108	78	0.42	254	**0.21**	0.0
	1, 2000	24 789/24 789	37	9.5	402/397	471	1.3	437	**0.39**	0.86979
	1, 3000	80 741/80 741	76	158.5	749/749	876	2.95	1178	**2.15**	0.99982
	1, 4000	91 923/91 923	0	69.2	1236/1236	1451	9.26	1900	**5.9**	0.9999997
firewire abst [*delay*, *T*]	360, 5000	206/206	7	0.26	31/31	20	0.09	64	**0.02**	0.78125
	360, 10 000	1020/1020	19	0.86	97/96	91	0.30	181	**0.05**	0.97473
	360, 20 000	9070/9070	40	7.5	347/347	340	0.85	641	**0.36**	0.99963
	360, 30 000	34 682/34 682	46	102.1	1049/1049	1057	5.5	1378	**1.44**	0.99999
firewire impl [*delay*, *T*]	360, 2500	1443/1443	0	0.82	178/178	115	**0.55**	n/a		0.5
	360, 5000	4463/4463	17	3.2	991/990	1230	**2.6**	n/a		0.78125
	360, 7500	10 700/10 700	34	17.8	2030/2030	2196	**9.9**	n/a		0.93164
	360, 10 000	24 449/24 449	56	**94.1**	8434/8434	9143	382.5	n/a		0.97473
nrp honest [*T*]	80	1531/1531	39	4.5	46/46	44	**0.18**	n/a		0.86491
	100	2286/2286	49	9.0	56/56	54	**0.23**	n/a		0.92023
	200	8311/8311	99	147.1	106/106	104	**0.31**	n/a		0.99427
	400	31 611/31 611	199	2893.4	206/206	204	**0.56**	n/a		0.99997

model proves to be expensive for PRISM since the current implementation expands parallel composition in a PTA before constructing an abstraction from it, resulting in a blow-up in the (intermediate) state space. It should be possible to adapt PRISM to bypass this step and avoid the blow-up.

The comparison with FORTUNA is more varied. Our prototype runs slightly faster on the larger *nrp* models, and does not fall far behind FORTUNA on the *csma abst* and *firewire abst* models. But FORTUNA performs much better on the *csma full* examples. This is because abstraction refinement for these models needs a relatively high number of refinement steps, each of which requires numerical computation to be performed. For PTAs, it is possible to avoid abstraction refinement entirely, as done by FORTUNA. On more complex models with many data variables, the benefits of abstraction should be more visible. In particular, for those with unbounded data variables, our abstraction refinement method works where PRISM and FORTUNA do not.

PASS has several different refinement strategies, from which we select the best performing one on each model. The results show that our prototype needs significantly less runtime on the *sliding window discr.* and *zeroconf discr.* case studies (and scales to larger models on the latter). Like for the comparison with PRISM, we use more iterations of refinement, but each one is cheaper; our abstractions also have significantly fewer states,

For the *brp* example, PASS performs slightly better: it benefits from a small number of states, while our implementation suffers from heavy numerical analysis, especially when $p1$ and $p2$ are close to one, which makes value iteration converge very slowly. On this example, we used a simple heuristic that gives significantly improved performance: we discover all

Table 3
Experimental results for discrete-time models.

Model & parameters		PASS			Local abstr.-refinement			Probability
		States (peak/final)	Iter.	Time (s)	States (peak/final)	Iter.	Time (s)	
sliding window [*N*, *p*]	8, 0.5	62 160/3357	9	157.7	1546/1546	1197	**9.0**	0.4999
	8, 0.98	232 489/5009	6	2027.8	1480/1480	1136	**31.6**	0.01999999
	10, 0.5	202 487/1377	12	2244.6	2004/2004	1546	**12.9**	0.49999
	10, 0.98	307 748/4452	6	3490.4	1670/1670	1294	**58.3**	0.01999999
brp [*Max*, *p*1, *p*2]	10, 0.5, 0.5	899/899	10	**1.5**	793/793	480	7.048	4.883e−4
	10, 0.98, 0.99	899/899	10	**4.3**	753/753	474	20.899	2.048e−19
	20, 0.5, 0.5	1679/1679	20	**4.5**	1341/1341	869	11.121	4.768e−7
	20, 0.98, 0.99	1679/1679	20	**11.2**	1821/1821	1048	165.048	2.097e−36
	100, 0.5, 0.5	7919/7919	100	**189.4**	4752/4751	4170	254.181	3.944e−31
zeroconf discr. [*N*, *M*, *K*]	3, 8, 4	4 465 394/10 012	6	283.4	5446/5446	1797	**33.2**	3.021e−5
	3, 8, 8	4 465 394/17 276	8	459.4	7143/7143	2603	**82.8**	2.351e−9
	4, 8, 4	memory overflow			17 209/17 209	2818	**109.6**	1.464e−4
	4, 8, 8	memory overflow			19 411/19 411	3842	**255.3**	5.013e−8

conflict states during the symbolic execution of an adversary and choose the state with the maximum depth for splitting. Interestingly, this heuristic does not improve performance for other case studies, which is worth further investigation. A different numerical solution method (not value iteration) might also help in this case. Further investigation on a wider set of untimed models is required to make a more thorough comparison between local abstraction refinement and the techniques implemented in PASS. At the present time, our focus is primarily on models incorporating both real-time and data aspects.

Finally, we make a brief comparison with the tool PASS-PTA, described in the recent paper [8]. Since this is not available for testing, we give an indirect comparison based on the results in this paper and in [8]. The only model in common is *csma* (property P1 in [8]), on which PASS-PTA is slower than PRISM and local abstraction refinement is faster. For *zeroconf* our results are from a more complex variant of the model than in [8]. We intend to perform a more in-depth comparison when PASS-PTA becomes available.

4.2. Overview

In summary, our approach tends to perform well, generating comparatively small abstract models using a relatively large number of (small, local) refinement steps. When the number of refinements becomes particularly high (e.g., *csma* with $bmax = 4$, $K = 16$), performance is degraded but, in our experiments, the tool still outperformed the game-based abstraction of PRISM. This gain derives from the use of simple operations on smaller abstractions and a reduction in the amount of numerical computation that needs to be performed.

Perhaps unsurprisingly, our approach performs better on real-time models, than on the versions that have been discretised. Compare, for example, the *zeroconf* models in Tables 1 and 3. These models are equivalent (and indeed, the results match), but abstraction is much more efficient when applied to the real-time version. For the *sliding window* model, a direct comparison is not possible since the real-time version of the model contains an additional clock because of limitations in the modelling language. This model does, though, illustrate the main benefit of our approach: the ability to verify models where abstraction of both the timed and data aspects is required, in particular when both real-time clocks and unbounded data types are present.

5. Conclusions

We have presented a novel abstraction refinement approach for the verification of probabilistic, real-time systems with potentially infinite data variables. Our approach uses local refinement steps, which results in more compact abstractions than alternative abstraction refinement techniques.

Future work includes extending it to support probabilistic hybrid automata (where operations to build and refine abstractions are even more costly) and applying it to mainstream programming languages such as C, Java or SystemC.

Acknowledgements

The authors are part supported by EC FP7 projects CONNECT (IST 231167) and HIERATIC, ERC Advanced Grant VERIWARE, and EPSRC project LSCITS (EP/F001096/1).

References

[1] M. Abadi, L. Lamport, The existence of refinement mappings, Theor. Comput. Sci. 82 (2) (1991) 253–284.

[2] J. Bengtsson, W. Yi, Timed automata: Semantics, algorithms and tools, in: Lectures on Concurrency and Petri Nets, 2003, pp. 87–124.
[3] J. Berendsen, D. Jansen, F. Vaandrager, Fortuna: Model checking priced probabilistic timed automata, in: Proc. QEST'10, 2010, pp. 273–281.
[4] R. Bruttomesso, A. Cimatti, A. Franzén, A. Griggio, R. Sebastiani, The MathSAT 4 SMT solver, in: Proc. CAV'08, Springer, 2008, pp. 299–303.
[5] A. Cimatti, A. Griggio, R. Sebastiani, Efficient generation of Craig interpolants in satisfiability modulo theories, ACM Trans. Comput. Log. 12 (1) (2010).
[6] P. D'Argenio, B. Jeannet, H. Jensen, K. Larsen, Reachability analysis of probabilistic systems by successive refinements, in: Proc. PAPM/PROBMIV'01, Springer, 2001, pp. 39–56.
[7] J. Esparza, A. Gaiser, Probabilistic abstractions with arbitrary domains, in: Proc. SAS'11, 2011, pp. 334–350.
[8] L.M. Ferrer Fioriti, H. Hermanns, Heuristics for probabilistic timed automata with abstraction refinement, in: Proc. MMB/DFT'12, 2012, pp. 151–165.
[9] E.M. Hahn, H. Hermanns, B. Wachter, L. Zhang, PASS: Abstraction refinement for infinite probabilistic models, in: Proc. TACAS'10, 2010.
[10] A. Hartmanns, H. Hermanns, A modest approach to checking probabilistic timed automata, in: Proc. QEST'09, IEEE, 2009, pp. 187–196.
[11] H. Hermanns, B. Wachter, L. Zhang, Probabilistic CEGAR, in: Proc. CAV'08, in: LNCS, vol. 5123, Springer, 2008, pp. 162–175.
[12] M. Kattenbelt, M. Kwiatkowska, G. Norman, D. Parker, Abstraction refinement for probabilistic software, in: Proc. VMCAI'09, 2009.
[13] M. Kattenbelt, M. Kwiatkowska, G. Norman, D. Parker, A game-based abstraction-refinement framework for Markov decision processes, Form. Methods Syst. Des. 36 (3) (2010) 246–280.
[14] J. Kemeny, J. Snell, A. Knapp, Denumerable Markov Chains, 2nd edition, Springer-Verlag, 1976.
[15] A. Komuravelli, C. Pasareanu, E. Clarke, Assume-guarantee abstraction refinement for probabilistic systems, in: Proc. CAV'12, 2012, pp. 310–326.
[16] M. Kwiatkowska, G. Norman, D. Parker, Stochastic games for verification of probabilistic timed automata, in: Proc. FORMATS'09, 2009, pp. 212–227.
[17] M. Kwiatkowska, G. Norman, D. Parker, A framework for verification of software with time and probabilities, in: Proc. FORMATS'10, 2010, pp. 25–45.
[18] M. Kwiatkowska, G. Norman, D. Parker, PRISM 4.0: Verification of probabilistic real-time systems, in: Proc. CAV'11, 2011, pp. 585–591.
[19] M. Kwiatkowska, G. Norman, D. Parker, J. Sproston, Performance analysis of probabilistic timed automata using digital clocks, Form. Methods Syst. Des. 29 (2006) 33–78.
[20] M. Kwiatkowska, G. Norman, R. Segala, J. Sproston, Automatic verification of real-time systems with discrete probability distributions, Theor. Comput. Sci. 282 (2002) 101–150.
[21] M. Kwiatkowska, G. Norman, J. Sproston, F. Wang, Symbolic model checking for probabilistic timed automata, Inf. Comput. 205 (7) (2007) 1027–1077.
[22] J. Norris, Markov Chains, Cambridge Series in Statistical and Probabilistic Mathematics, Cambridge University Press, 1998.
[23] S. Tripakis, Verifying progress in timed systems, in: Proc. ARTS'99, in: LNCS, vol. 1601, Springer, 1999, pp. 299–314.
[24] B. Wachter, L. Zhang, Best probabilistic transformers, in: Proc. VMCAI'10, in: LNCS, vol. 5944, Springer, 2010, pp. 362–379.
[25] L. Zhang, Z. She, S. Ratschan, H. Hermanns, E.M. Hahn, Safety verification for probabilistic hybrid systems, in: Proc. CAV'08, 2008.
[26] http://www.prismmodelchecker.org/benchmarks/.
[27] http://www.prismmodelchecker.org/files/tcs-ptps/.