A Framework for Verification of Software with Time and Probabilities

Marta Kwiatkowska, Gethin Norman, David Parker

A Framework for Verification of Software with Time and Probabilities

Marta Kwiatkowska[1], Gethin Norman[2], and David Parker[1]

[1] Oxford University Computing Laboratory, Parks Road, Oxford, OX1 3QD, UK
[2] Department of Computing Science, University of Glasgow, Glasgow, G12 8RZ, UK

Abstract. Quantitative verification techniques are able to establish system properties such as "the probability of an airbag failing to deploy on demand" or "the expected time for a network protocol to successfully send a message packet". In this paper, we describe a framework for quantitative verification of software that exhibits both real-time and probabilistic behaviour. The complexity of real software, combined with the need to capture precise timing information, necessitates the use of abstraction techniques. We outline a quantitative abstraction refinement approach, which can be used to automatically construct and analyse abstractions of probabilistic, real-time programs. As a concrete example of the potential applicability of our framework, we discuss the challenges involved in applying it to the quantitative verification of SystemC, an increasingly popular system-level modelling language.

1 Introduction

Computerised systems pervade all aspects of modern society, including safety-critical application domains such as the automotive and avionics industries. This, combined with the growing complexity of such devices, necessitates the development of rigorous techniques to verify their correctness. Furthermore, this analysis must often take into account the *quantitative* aspects of the systems that are being verified. This includes both *real-time* characteristics and *probabilistic* behaviour. Embedded devices, in safety-critical applications for example, will often have strict timing requirements. Similarly, it is important to quantify the effect of inherently stochastic behaviour, such as component failures or message loss in communication between networked devices. Another source of probabilistic behaviour is the use of randomisation, an essential ingredient of many network protocols such as FireWire or Ethernet and wireless technologies including Bluetooth and ZigBee.

Quantitative verification is a formal method for the analysis of timed and probabilistic systems. It is based on the construction of a mathematical model capturing the system's behaviour, followed by the analysis of formally specified quantitative properties. These might include, for example, "the probability of an airbag failing to deploy within 0.02 seconds", "the expected time for a network protocol to send a packet" or "the expected power consumption of a sensor

network during 1 hour of operation". Notice that this permits an analysis not just of a system's correctness, but also its performance and reliability.

Quantitative verification techniques have seen a great deal of progress in recent years. For real-time systems, a prominent modelling formalism is timed automata, for which mature verification tools such as UPPAAL [38] exist. For probabilistic systems, the most commonly used models are Markov chains or Markov decision processes (MDPs). Probabilistic model checking tools such as PRISM [21] and MRMC [25] are widely used and have been successfully applied to the verification of a range of systems. Recent work [45,32,17,4] has seen progress in the development of tools for systems with time *and* probabilities.

A weakness of all these tools, however, is that they require the user to specify the system model in a custom modelling language. In order to minimise the chance of errors introduced in the modelling phase and to encourage the use of these tools, there is a need to extend quantitative verifications techniques to the languages used by real system designers. In the context of non-probabilistic verification, progress has been made in this direction. In particular, software model checking tools and techniques can now be applied directly to mainstream programming languages such as C and Java.

In this paper, we develop the underlying theory for quantitative verification of software with both probabilistic and real-time characteristics. We formalise the notion of *probabilistic timed programs*, whose semantics are defined in terms of infinite-state MDPs. The complexity of real software means that the use of *abstraction* is typically essential. Building on our previous work on tools and techniques for verifying a probabilistic extension of ANSI-C [26], we propose a *quantitative abstraction refinement* approach [27,31,32]. This builds successive, increasingly precise abstractions of an MDP, represented as two-player stochastic games [31]. At each step, the process is driven by *refinement* techniques which construct a new abstraction using information derived from quantitative verification of the stochastic game. This technique has already proven to be an efficient approach to the verification of probabilistic timed automata [32].

As a concrete illustration of the potential applicability of our verification framework, we discuss the challenges involved in applying it to the quantitative verification of SystemC, a C++-based system-level modelling language. SystemC is becoming increasingly prominent in the embedded systems domain, for example in the development of System-on-Chips (SoCs). Building formal verification techniques for SystemC has already been identified as an important but challenging direction of research [46]. Clearly, *quantitative* verification of SystemC will be even more demanding. We describe how some of the existing approaches and tools might be combined with our framework and identify some of the more important directions of future work.

This is an extended and improved version of [33], including additional details and proofs. All proofs are located in the Appendix.

2 Background Material

A distribution over Q is a function $\lambda : Q \to [0,1]$ where the support $\{q \in Q \mid \lambda(q) > 0\}$ is countable and $\sum_{q \in Q} \lambda(q) = 1$, let $\mathsf{Dist}(Q)$ denote the set of such distributions.

2.1 MDPs and Stochastic Games

Markov decision processes (MDPs) are used to model systems that exhibit both nondeterministic and probabilistic behaviour.

Definition 1. *An MDP* M *is a tuple* $(S, \overline{S}, Act, Steps_{\mathsf{M}})$ *where* S *is a set of* states, $\overline{S} \subseteq S$ *is a set of* initial states, Act *is a set of* actions *and* $Steps_{\mathsf{M}} : S \times Act \to \mathsf{Dist}(S)$ *is a partial* probabilistic transition function.

For a state s of an MDP M, we write $Act(s)$ for the set of actions available in s, i.e., the actions $a \in Act$ for which $Steps_{\mathsf{M}}(s,a)$ is defined. The behaviour in state s is both probabilistic and nondeterministic: first an available action (i.e. an action in $Act(s)$) is selected nondeterministically, then a successor state is chosen according to the distribution $Steps_{\mathsf{M}}(s,a)$. A *path* is a sequence of such choices and a state is *reachable* if there is a path to it from an initial state. Under an *adversary* A, which resolves all nondeterminism, we can define a probability measure over paths [28]. The fundamental quantitative property for MDPs is that of *probabilistic reachability* which concerns the minimum or maximum probability of reaching a set of states F. Formally, we have:

$$p^{\min}_{\mathsf{M}}(F) \stackrel{\text{def}}{=} \inf_{s \in \overline{S}} \inf_A p^A_s(F) \quad \text{and} \quad p^{\max}_{\mathsf{M}}(F) \stackrel{\text{def}}{=} \sup_{s \in \overline{S}} \sup_A p^A_s(F)$$

where $p^A_s(F)$ denotes the probability of reaching target F, starting from s, when the MDP behaves according to adversary A.

Stochastic two-player games [41,8] extend MDPs by allowing two types of nondeterministic choice, controlled by separate players.

Definition 2. *A* stochastic game G *is a tuple* $(S, \overline{S}, Act, Steps_{\mathsf{G}})$ *where* S *is a set of* states, $\overline{S} \subseteq S$ *is a set of* initial states Act *is a set of* actions *and* $Steps_{\mathsf{G}} : S \times Act \to 2^{\mathsf{Dist}(S)}$ *is a partial* probabilistic transition function.

The behaviour in a state s of a game G includes two successive nondeterministic choices: first player 1 selects an available action, then a distribution $\lambda \in Steps_{\mathsf{G}}(s,a)$ is selected by player 2. Finally, the successor state is then chosen according to the distribution λ. A pair of *strategies* (σ_1, σ_2) for players 1 and 2, resolve all the nondeterminism present in the game, and this induces a probability measure over the paths of the game.

2.2 Quantitative Abstraction Refinement

As proposed in [31], we use stochastic games to represent *abstractions* of MDPs. The key idea is to separate the two forms of nondeterminism: using player 1

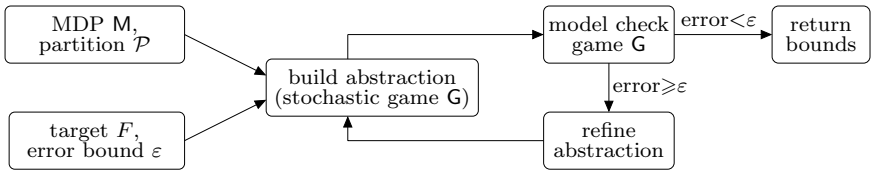

Fig. 1. Quantitative abstraction refinement for MDPs

choices to represent the nondeterminism caused by abstraction; and player 2 choices for the nondeterminism of the MDP. For an MDP M, the construction of an abstraction is based on a *partition* $\mathcal{P}=\{S_1,\ldots,S_n\}$ of its state space. For $\lambda \in \mathsf{Dist}(S)$, we let $\lambda_{\mathcal{P}} \in \mathsf{Dist}(\mathcal{P})$ denote the distribution where $\lambda_{\mathcal{P}}(S')=\sum_{s\in S'}\lambda(s)$ for all $S' \in \mathcal{P}$. Formally, we define the abstraction of an MDP as follows.

Definition 3. *Let* $\mathsf{M}=(S,\overline{S},Act,Steps_{\mathsf{M}})$ *be an MDP and* $\mathcal{P}$ *a partition of* S. *The* abstraction of M with respect to $\mathcal{P}$ *is given by the stochastic game* $\mathsf{G}=(\mathcal{P},\overline{\mathcal{P}},2^{Act\times\mathsf{Dist}(\mathcal{P})},Steps_{\mathsf{G}})$ *where* $\overline{\mathcal{P}}=\{S'\in\mathcal{P}\mid S'\cap\overline{S}\neq\emptyset\}$ *and for* $S'\in\mathcal{P}$ *and* $\Theta\in 2^{Act\times\mathsf{Dist}(\mathcal{P})}$*:* $Steps_{\mathsf{G}}(S',\Theta)$ *is defined and equals* $\{\lambda\mid(a,\lambda)\in\Theta\wedge a\in Act\}$ *if and only if there exists* $s\in S'$ *such that* $\Theta=\{(a,Steps_{\mathsf{M}}(s,a)_{\mathcal{P}})\mid a\in Act(s)\}$.

If G is an abstraction of M, then the reachability probabilities on G yield lower and upper bounds on the reachability probabilities of M:

$$p_{\mathsf{G}}^{lb,\min}(F)\leqslant p_{\mathsf{M}}^{\min}(F)\leqslant p_{\mathsf{G}}^{ub,\min}(F) \quad\text{and}\quad p_{\mathsf{G}}^{lb,\max}(F)\leqslant p_{\mathsf{M}}^{\max}(F)\leqslant p_{\mathsf{G}}^{ub,\max}(F) \qquad (1)$$

where, for example, in the stochastic game G:

$$p_{\mathsf{G}}^{lb,\max}(F) \stackrel{\text{def}}{=} \sup_{S'\in\overline{\mathcal{P}}}\inf_{\sigma_1}\sup_{\sigma_2} p_{S'}^{\sigma_1,\sigma_2}(F)$$
$$p_{\mathsf{G}}^{ub,\max}(F) \stackrel{\text{def}}{=} \sup_{S'\in\overline{\mathcal{P}}}\sup_{\sigma_1}\sup_{\sigma_2} p_{S'}^{\sigma_1,\sigma_2}(F)$$

and $p_{S'}^{\sigma_1,\sigma_2}(F)$ denotes the probability of reaching the target $\alpha(F)$ under the pair of strategies (σ_1,σ_2) when starting in the state S'. These reachability values can be determined efficiently using *value iteration* [40] together with the corresponding adversary or strategy-pair which achieves the value. In [47], the game-based abstraction of Definition 3 above is phrased in terms of abstract interpretation [9] and the resulting bounds obtained are shown to coincide with the "best" values obtainable for a fixed abstraction of the MDP.

Quantitative abstraction refinement [27,32,26] is an approach for automatically constructing abstractions of probabilistic models. Using the notion of abstracting MDPs with stochastic games describe above, it has been successfully applied to the verification of a probabilistic extension of ANSI-C [26], probabilistic timed automata [32] and concurrent probabilistic systems [47].

Illustrated in Figure 1, the technique starts with an MDP M (or, in practice a high-level model with MDP semantics) and coarse partition $\mathcal{P}$ of its state space. It then constructs and analyses the resulting abstraction of M, yielding lower and upper bounds on a property of interest (e.g. the probability of reaching a

set of target states F as in (1) above). If the difference between these bounds (the "error") is below a pre-specified bound ε, the process terminates, producing suitably tight lower and upper bounds. If not, the abstraction is refined, based on information (strategies) from the analysis of the game. The abstraction, analysis and refinement loop is repeated until the error drops below ε.

2.3 Clocks, Zones, Variables and Predicates

Clocks. Let $\mathcal{X}$ be a finite set of clocks. A function $v : \mathcal{X} \to \mathbb{R}$ is referred to as a *clock valuation* and the set of all clock valuations is denoted by $\mathbb{R}^{\mathcal{X}}$. For $v \in \mathbb{R}^{\mathcal{X}}$, $t \in \mathbb{R}$ and $X \subseteq \mathcal{X}$, we use $v{+}t$ to denote the valuation which increments all clocks by t and $v[X{:=}0]$ for the valuation in which clocks in X are reset to 0.

Zones. The set of zones of $\mathcal{X}$, written $\mathit{Zones}(\mathcal{X})$, is defined by the syntax:

$$\zeta ::= \texttt{true} \mid x \leqslant d \mid c \leqslant x \mid x{+}c \leqslant y{+}d \mid \neg\zeta \mid \zeta \vee \zeta$$

where $x, y \in \mathcal{X}$ and $c, d \in \mathbb{N}$. A zone ζ represents the set of clock valuations v which *satisfy* ζ, denoted $v \triangleleft \zeta$, i.e. those where ζ resolves to `true` by substituting each clock x with $v(x)$. We will use several classical operations on zones [18,44]:

- $\nearrow\zeta$ contains all valuations that can be reached from ζ by letting time pass;
- $\swarrow\zeta$ contains all valuations that can reach ζ by letting time pass;
- $[X{:=}0]\zeta$ contains the valuations which are in ζ after resetting the clocks X;
- $\zeta[X{:=}0]$ contains the valuations obtained from ζ by resetting the clocks X.

In standard fashion, we restrict our attention to *c-closed* zones, in which constraints with bounds greater than c are removed, and where c is the largest such bound appearing in the description of the model under study.

Data Variables and Predicates. Let $\mathcal{D}$ be a finite set of data variables. We denote by $\mathit{Val}(\mathcal{D})$ the set of *data valuations* over $\mathcal{D}$ and by $\mathit{Up}(\mathcal{D})$ the set of *updates*, i.e. the set of functions $\mathit{up} : \mathit{Val}(\mathcal{D}) \to \mathit{Val}(\mathcal{D})$. Let $\mathit{Pred}(\mathcal{D})$ be the set of predicates over the data variables $\mathcal{D}$. For a data predicate ϕ and valuation u, we say $u \triangleleft \phi$ if ϕ holds after substituting each variable d with the value $u(d)$.

3 Probabilistic Timed Programs

We now introduce the formal model of *probabilistic timed programs* (PTPs), on which the techniques in this paper are based. These combine:

- timed behaviour, through real-valued clocks in the style of timed automata;
- stochastic behaviour, through discrete probabilistic choice;
- nondeterminism and concurrency through parallel composition;
- control flow and discrete data variables to capture program behaviour.

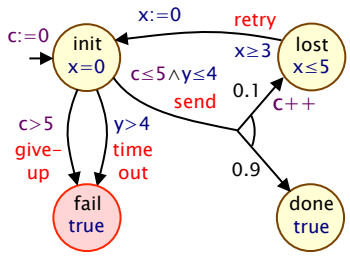

Fig. 2. Example of a PTP

PTPs are essentially probabilistic timed automata (PTAs) [23,35,2] with the addition of discrete-valued variables. For timed automata formalisms, discrete variables are typically considered to be a straightforward syntactic extension since their values can simply be encoded into locations. Our focus in this paper is how to use abstraction when such an encoding is not feasible in practice.

Definition 4. *A* probabilistic timed program *(PTP) is a tuple of the form* $\mathsf{P}=(L,\bar{l},\mathcal{D},\overline{u},\mathcal{X},Act,inv,enab,prob)$ *where:*

- L *is a finite set of* locations *and* $\bar{l}\in L$ *is an* initial location*;*
- $\mathcal{D}$ *is a finite set of* data variables *and* $\overline{u}\in Val(\mathcal{D})$ *is an initial data valuation;*
- $\mathcal{X}$ *is a finite set of* clocks*;*
- Act *is a finite set of actions;*
- $inv : L \to Zones(\mathcal{X})$ *is an* invariant condition*;*
- $enab : L\times Act \to Pred(\mathcal{D})\times Zones(\mathcal{X})$ *is an* enabling condition*;*
- $prob : L\times Act \to \mathsf{Dist}(Up(\mathcal{D})\times 2^{\mathcal{X}}\times L)$ *is a* probabilistic transition function*.*

A *state* of a PTP is a tuple $(l,u,v)\in L\times Val(\mathcal{D})\times\mathbb{R}^{\mathcal{X}}$ such that $v\triangleleft inv(l)$. In a state (l,u,v), a certain amount of time $t\in\mathbb{R}$ can elapse, after which an action $a\in Act$ is performed. The choice of t requires that, while time passes, the invariant $inv(l)$ remains continuously satisfied. An action a can be chosen only if it is *enabled*. We use $enab_{\mathcal{D}}$ and $enab_{\mathcal{X}}$ to denote the data and time components of the enabling function, i.e. $enab(l,a) = (enab_{\mathcal{D}}(l,a), enab_{\mathcal{X}}(l,a))$. Action a can be chosen if u satisfies $enab_{\mathcal{D}}(l,a)$ *and* $v{+}t$ satisfies $enab_{\mathcal{X}}(l,a)$). Once an action a is chosen, the update of the data variables, clocks to reset and successor location are selected at random, according to the distribution $prob(l,a)$. We call each element (l,a,up,X,l') such that $(up,X,l')\in Up(\mathcal{D})\times 2^{\mathcal{X}}\times L$ is in the support of $prob(l,a)$ an *edge* and, for convenience, assume that the set of such edges, denoted $edges(l,a)$, is an ordered list $\langle e_1,\ldots,e_n\rangle$.

Example 1. Figure 2 shows an example of a PTP modelling a simple communication protocol that aims to send a message over an unreliable channel. It tries to send the message up to 5 times or until a time-out of 4 seconds has elapsed. Between each attempt, there is a delay of between 3 and 5 seconds. The PTP has a single discrete variable c and two clocks x and y. The variable c is used to keep track of the number of attempts to send, clock x is used to time each delay and y stores the total time elapsed.

Definition 5. *Let* $\mathsf{P}=(L,\bar{l},\mathcal{D},\overline{u},\mathcal{X},Act,inv,enab,prob)$ *be a PTP. The semantics of* P *is an (infinite-state) MDP* $[\![\mathsf{P}]\!]=(S,\overline{S},\mathbb{R}\times Act,Steps_{\mathsf{P}})$ *where:*

- $S=\{(l,u,v)\in L\times Val(\mathcal{D})\times\mathbb{R}^{\mathcal{X}} \mid v\triangleleft inv(l)\}$ *and* $\overline{S}=\{(\bar{l},\overline{u},\mathbf{0})\}$*;*
- $Steps_{\mathsf{P}}((l,u,v),(t,a))=\lambda$ *if and only if*
  - $v+t'\triangleleft inv(l)$ *for all* $0\leqslant t'\leqslant t$*;*
  - $(u,v+t)\triangleleft enab(l,a)$*;*
  - *for any* $(l',u',v')\in S$*:*

$$\lambda(l',u',v')=\textstyle\sum\{\!|\, prob(l,a)(up,X,l') \mid (up,X)\in Up_{u\mapsto u'}\times\mathcal{X}_{v+t\mapsto v'} \,|\!\}$$

  where the set of updates $Up_{u\mapsto u'}$ *equals* $\{up\in Up(U) \mid up(u)=u'\}$ *and the set of clock resets* $\mathcal{X}_{v+t\mapsto v'}$ *is given by* $\{X\subseteq\mathcal{X} \mid (v+t)[X:=0]=v'\}$*.*

Each transition of the semantics of a PTP is a time-action pair (t,a), representing t time units elapsing, followed by a discrete a-labelled transition. For any state (l,u,v) and time-action pair (t,a), if $Steps_{\mathsf{P}}((l,u,v),(t,a))$ is defined and $edges(l,a)=\langle(l,a,up_1,X_1,l_1),\ldots,(l,a,up_n,X_n,l_n)\rangle$, then we write $(l,u,v)\xrightarrow{t,a}\langle(l_1,up_1(u),(v+t)[X_1:=0]),\ldots,(l_n,up_n(u),(v+t)[X_n:=0])\rangle$.

The definition of parallel composition for PTPs is a straightforward extension of that for probabilistic timed automata [34].

Definition 6. *Let* $\mathsf{P}_i=(L_i,\bar{l}_i,\mathcal{D}_i,\overline{u}_i,\mathcal{X}_i,Act_i,inv_i,enab_i,prob)$ *for* $i=1,2$ *be PTPS such that* $\mathcal{D}_1\cap\mathcal{D}_2=\emptyset$ *and* $\mathcal{X}_1\cap\mathcal{X}_2=\emptyset$*. The* parallel composition *of* P_1 *and* P_2*, denoted* $\mathsf{P}_1\|\mathsf{P}_2$*, is given by the PTP:*

$$\mathsf{P}_1\|\mathsf{P}_2=(L_1\times L_2,(\bar{l}_1,\bar{l}_2),\mathcal{D}_1\cup\mathcal{D}_2,(\overline{u}_1,\overline{u}_2),\mathcal{X}_1\cup\mathcal{X}_2,Act_1\cup Act_2,inv,enab,prob)$$

such that for any location pair $(l_1,l_2)\in L_1\times L_2$ *and action* $a\in Act_1\cup Act_2$*:*

- *the invariant condition* $inv((l_1,l_2))$ *equals* $inv_1(l_1)\wedge inv_2(l_2)$*;*
- *the enabling condition is given by:*

$$enab((l_1,l_2),a)=\begin{cases} enab_1(l_1,a)\wedge enab_2(l_2,a) & \text{if } a\in Act_1\cap Act_2\\ enab_1(l_1,a) & \text{if } a\in Act_1\backslash Act_2\\ enab_2(l_2,a) & \text{if } a\in Act_2\backslash Act_1;\end{cases}$$

- *the probabilistic transition function* $prob((l_1,l_2),a)$ *equals* λ *where for any* $(up,X,(l_1',l_2'))\in Up(\mathcal{D}_1\cup\mathcal{D}_2)\times 2^{\mathcal{X}_1\cup\mathcal{X}_2}\times(L_1\times L_2)$*:*
  - *if* $a\in Act_1\cap Act_2$*, then*

$$\lambda(up,X,(l_1',l_2'))=\lambda_1(up\lceil_{\mathcal{D}_1},X\backslash\mathcal{X}_2,l_1')\cdot\lambda_2(up\lceil_{\mathcal{D}_2},X\backslash\mathcal{X}_1,l_2')$$

  - *if* $a\in Act_1\backslash Act_2$*, then*

$$\lambda(up,X,(l_1',l_2'))=\begin{cases}\lambda_1(up\lceil_{\mathcal{D}_1},X,l_1') & \text{if } up\lceil_{\mathcal{D}_2}=id_{\mathcal{D}_2},\ X\backslash\mathcal{X}_1=\emptyset \text{ and } l_2'=l_2\\ 0 & \text{otherwise}\end{cases}$$

  - *if* $a\in Act_2\backslash Act_1$*, then*

$$\lambda(up,X,(l_1',l_2'))=\begin{cases}\lambda_2(up\lceil_{\mathcal{D}_2},X,l_2') & \text{if } up\lceil_{\mathcal{D}_1}=id_{\mathcal{D}_1},\ X_1\backslash\mathcal{X}_2=\emptyset \text{ and } l_1'=l_1\\ 0 & \text{otherwise}\end{cases}$$

  where $\lambda_1=prob_1(l_1,a)$*,* $\lambda_2=prob_2(l_2,a)$ *and, for* $\mathcal{D}'\subseteq\mathcal{D}_1\cup\mathcal{D}_2$ *and* $up: Val(\mathcal{D})\to Val(\mathcal{D})$*, the function* $up\lceil_{\mathcal{D}'}: Val(\mathcal{D}')\to Val(\mathcal{D}')$ *is obtained by restricting up to the domain* $\mathcal{D}'$*, while* $id_{\mathcal{D}'}$ *is the identity function over* $Val(\mathcal{D}')$*.*

4 Abstraction of PTPs

We now consider the problem of constructing *abstractions* of PTPs, that is to say building a stochastic game abstraction [31] for its underlying MDP semantics. For this, we combine several different techniques. For the data part of PTPs, we use *predicate abstraction* [15]. For the time aspect, we consider two possibilities: first, we again use predicates (over clock, rather than data, variables); secondly we use *zones*, which can be efficiently stored and manipulated using difference-bound matrices (DBMs) [18,44]. The latter is better suited to the *forwards reachability* based techniques commonly used for timed automata (and proposed for the construction of abstractions of PTAs in [32]).

We thus have two abstractions to consider: (i) using predicates for both data and time; (ii) using predicates for data and zones for time. A crucial difference between the two is that (i) induces a *partition* of the PTP states S (in which case, Definition 3 applies directly), whereas (ii) gives instead a *covering* of S.

4.1 Abstract Domains for PTPs

We formally define the two different abstractions discussed above using the notion of *abstract domains* from the abstract interpretation framework of [9].

Definition 7. *For a given set of concrete states S, an* abstract domain $\mathtt{A}$ *is a tuple* $((\mathtt{Z}, \sqcup, \sqcap, \sqsubseteq), \alpha, \gamma)$ *where:*

- $(\mathtt{Z}, \sqcup, \sqcap, \sqsubseteq)$ *is a lattice of* abstract states*;*
- $\alpha : 2^S \to \mathtt{Z}$ *and* $\gamma : \mathtt{Z} \to 2^S$ *are* abstraction *and* concretisation *functions;*

such that (α, γ) *form a Galois connection.*

From this point on, we assume sets of data and clock predicates $\Phi = \{\phi_1, \ldots, \phi_n\} \subseteq \mathit{Pred}(\mathcal{D})$ and $\Psi = \{\psi_1, \ldots, \psi_m\} \subseteq \mathit{Pred}(\mathcal{X})$. For a predicate φ and valuation w (over $\mathcal{D}$ or $\mathcal{X}$), let $\varphi(w)$ denote the value of φ evaluated against w. For predicates $\Upsilon = \{\varphi_1, \ldots, \varphi_k\}$, let $\Upsilon(w)$ denote the predicate valuation $(\varphi_1(w), \ldots, \varphi_{|\Upsilon|}(w)) \in \mathbb{B}^\Upsilon$ and, for $b \in \mathbb{B}^\Upsilon$, let $\Upsilon[b]$ denote the predicate $\tilde{\varphi}_1 \wedge \ldots \wedge \tilde{\varphi}_k$ where $\tilde{\varphi}_i = \varphi_i$ if $b_i = \mathtt{true}$ and $\neg\varphi_i$ otherwise. Note that $\Psi[b]$ can be considered as a zone.

The Abstract Domain $\mathtt{A}^{\Phi,\Psi}$*.* The atoms of the lattice $(\mathtt{Z}^{\Phi,\Psi}, \sqcup, \sqcap, \sqsubseteq)$ are the tuples $\mathtt{z} = (l, b_1, b_2) \in L \times \mathbb{B}^\Phi \times \mathbb{B}^\Psi$, comprising a location l and predicate valuations b_1 and b_2. Since $L \times \mathbb{B}^\Phi \times \mathbb{B}^\Psi$ form the atoms of the lattice, the operations $\sqcup$, $\sqcap$ and $\sqsubseteq$ can be considered as the standard set operators over $2^{L \times \mathbb{B}^\Phi \times \mathbb{B}^\Psi}$. For any $S' \subseteq S$ and $(l, b_1, b_2) \in L \times \mathbb{B}^\Phi \times \mathbb{B}^\Psi$ the abstraction and concretisation functions are defined as follows:

$$\alpha(S') = \sqcup_{(l,u,v) \in S'} (l, \Phi(u), \Psi(v)) \text{ and } \gamma(l, b_1, b_2) = \{(l, u, v) \mid \Phi(u) = b_1 \wedge \Psi(v) = b_2\}.$$

The Abstract Domain $\mathtt{A}^{\Phi,\mathcal{X}}$*.* A basis for the lattice $(\mathtt{Z}^{\Phi,\mathcal{X}}, \sqcup, \sqcap, \sqsubseteq)$ are the tuples $\mathtt{z} = (l, b, \zeta) \in L \times \mathbb{B}^\Phi \times \mathit{Zones}(\mathcal{X})$ comprising a location l, predicate valuation b and zone ζ. For $(l, b, \zeta), (l', b', \zeta') \in L \times \mathbb{B}^\Phi \times \mathit{Zones}(\mathcal{X})$:

$$(l, b, \zeta) \sqcup (l', b', \zeta') = \textbf{if } (l, b) = (l, b') \textbf{ then } \{(l, b, \zeta \vee \zeta')\} \textbf{ else } \{(l, b, \zeta), (l', b', \zeta')\}$$
$$(l, b, \zeta) \sqcap (l', b', \zeta') = \textbf{if } (l, b) = (l, b') \textbf{ then } \{(l, b, \zeta \wedge \zeta')\} \textbf{ else } \emptyset$$

and $(l,b,\zeta) \sqsubseteq (l',b',\zeta')$ if and only if $(l,b)=(l',b')$ and $\zeta \Rightarrow \zeta' \equiv \texttt{true}$, while $\texttt{Z}' \sqsubseteq \texttt{Z}''$ if and only if for all $\texttt{z}' \in \texttt{Z}'$ there exists $\texttt{z}'' \in \texttt{Z}''$ such that $\texttt{z}' \sqsubseteq \texttt{z}''$. We illustrate these operations with the following examples:

$$\begin{aligned}\{(l,b,x\leqslant 4)\} \sqcup \{(l,b,y\geqslant 1),(l',b,x\geqslant 4)\} &= \{(l,b,(x\leqslant 4)\vee(y\geqslant 1)),(l',b,x\geqslant 4)\}\\ \{(l,b,x\leqslant 4)\} \sqcap \{(l,b,x\geqslant 2)\} &= \{(l,b,2\leqslant x\leqslant 4)\}\\ \{(l,b,x\leqslant 1),(l',b',y\geqslant 3)\} &\sqsubseteq \{(l,b,(x\leqslant 4),(l',b',y\geqslant 2)),(l'',b'',z\leqslant 1)\}\,.\end{aligned}$$

With regards to the abstraction and concretisation functions, we have:

$$\alpha(S')=\sqcup_{(l,u,v)\in S'}\,(l,\Phi(u),[v]) \text{ and } \gamma(l,b,\zeta)=\{(l,u,v) \mid \Phi(u)=b \ \wedge \ v \triangleleft \zeta\}\,.$$

where $[v]$ is the *clock equivalence class* (or clock region) of v [1].

For both abstract domains we use $loc(\texttt{z})$, $data(\texttt{z})$ and $time(\texttt{z})$ to denote the different components of the tuple $\texttt{z}$ representing an abstract state.

4.2 Abstract Post Operators

We now describe how to construct an *abstract post operator* for each abstract domain, i.e. the "best" abstraction [9] of the concrete PTP transition semantics. For convenience, we split the *post* operator into two parts, representing the elapse of time in the current location l and the subsequent discrete transition along edge $e=(l,a,up,X,l')$. For any set $S' \subseteq S$ of concrete states, we have:

$$\begin{aligned}\mathsf{tpost}[l](S') &= \{(l,u,v+t) \mid (l,u,v)\in S' \ \wedge \ t\in\mathbb{R} \ \wedge \ \forall t'\leqslant t.\ (v+t')\triangleleft inv(l)\}\\ \mathsf{dpost}[e](S') &= \{(l,up(u),v[X:=0]) \mid (l,u,v)\in S' \wedge (u,v)\triangleleft enab(l,a)\}\end{aligned}$$

For the abstract domain $\texttt{A}=((\texttt{Z},\sqcup,\sqcap,\sqsubseteq),\alpha,\gamma)$, the corresponding "best" *abstract time-post* and *discrete-post* operators are given by:

$$\mathsf{tpost}^{\texttt{A}}[l](\texttt{z}) = \alpha(\mathsf{tpost}[l](\gamma(\texttt{z}))) \quad \text{and} \quad \mathsf{dpost}^{\texttt{A}}[e](\texttt{z}) = \alpha(\mathsf{dpost}[e](\gamma(\texttt{z})))\,.$$

For both of the abstract domains introduced in the previous section, these operators can be efficiently computed, using DBMs to manipulate zones [18,44] and SAT or SMT based techniques for predicates over data variables [37].

When representing clock valuations as zones, the elapse of time in l is captured by the function $\mathsf{tpost}_{\mathcal{X}}[l] : \mathit{Zones}(\mathcal{X}) \to \mathit{Zones}(\mathcal{X})$ and the effect of edge e by $\mathsf{dpost}_{\mathcal{X}}[e](\zeta) : \mathit{Zones}(\mathcal{X}) \to \mathit{Zones}(\mathcal{X})$. Both can computed with simple and efficient zone operations that can be implemented with DBMs. For $\zeta \in \mathit{Zones}(\mathcal{X})$:

$$\begin{aligned}\mathsf{tpost}_{\mathcal{X}}[l](\zeta) &= inv(l) \wedge \nearrow \zeta\\ \mathsf{dpost}_{\mathcal{X}}[e](\zeta) &= (\zeta \wedge enab_{\mathcal{X}}(l,a))[X:=0] \wedge inv(l')\,.\end{aligned}$$

For the representation of data using predicates, we define the function $\mathsf{dpost}^{\Phi}_{\mathcal{D}}[e] : \mathbb{B}^{\Phi} \to 2^{\mathbb{B}^{\Phi}}$ giving the set of possible predicate valuations of the variables in successor states when taking edge e. Let $p_1,\ldots,p_{|\Phi|}$ be Boolean variables corresponding to the predicates $\phi_1,\ldots,\phi_{|\Phi|}$ and $b \in \mathbb{B}^{\Phi}$ be a predicate valuation. Then $\mathsf{dpost}^{\Phi}_{\mathcal{D}}[e](b)$ contains all satisfying instances of $p_1,\ldots,p_{|\Phi|}$ such that:

$$\exists u,u' \in \mathit{Val}(\mathcal{D}).\ up(u)=u' \wedge \Phi(u)=b \wedge \big(p_1 \Leftrightarrow \phi_1(u') \wedge \cdots \wedge p_{|\Phi|} \Leftrightarrow \phi_{|\Phi|}(u')\big)$$

which we use an SMT solver to enumerate. This assumes that the types of variables in $\mathcal{D}$ and the operations occurring in up and predicates ϕ_i match the underlying theory of the SMT solver used. Alternatively, a bit-level encoding of data variables can be employed, and a SAT solver used for enumeration [7].

On the other hand, $\mathsf{tpost}^{\Psi}_{\mathcal{X}}[l] : \mathbb{B}^{\Psi} \to 2^{\mathbb{B}^{\Psi}}$ and $\mathsf{dpost}^{\Psi}_{\mathcal{X}}[e] : \mathbb{B}^{\Psi} \to 2^{\mathbb{B}^{\Psi}}$, given b_2 as input, return the satisfiable instances b_2' of the predicates:

$$\Psi[b_2'] \wedge \mathsf{tpost}_{\mathcal{X}}[l](\Psi[b_2]) \quad \text{and} \quad \Psi[b_2'] \wedge \mathsf{dpost}_{\mathcal{X}}[e](\Psi[b_2])\,.$$

Using these functions, we are able to efficiently compute the abstract post operators for both domains. More precisely, for any $(l, b_1, b_2) \in \mathsf{Z}^{\Phi,\Psi}$:

$$\begin{aligned}\mathsf{tpost}^{\Phi,\Psi}[l](l, b_1, b_2) &= \{(l, b_1, b_2') \mid b_2' \in \mathsf{tpost}^{\Psi}_{\mathcal{X}}[l](b_2)\}\\ \mathsf{dpost}^{\Phi,\Psi}[e](l, b_1, b_2) &= \{(l', b_1', b_2') \mid b_1' \in \mathsf{dpost}^{\Phi}_{\mathcal{D}}[e](b) \wedge b_2' \in \mathsf{dpost}^{\Psi}_{\mathcal{X}}[e](b_2)\}\end{aligned}$$

and for any $(l, b, \zeta) \in \mathsf{Z}^{\Phi,\mathcal{X}}$:

$$\begin{aligned}\mathsf{tpost}^{\Phi,\mathcal{X}}[l](l, b, \zeta) &= \{(l, b, \mathsf{tpost}_{\mathcal{X}}[l](\zeta))\}\\ \mathsf{dpost}^{\Phi,\mathcal{X}}[e](l, b, \zeta) &= \{(l', b', \mathsf{dpost}_{\mathcal{X}}[e](\zeta)) \mid b' \in \mathsf{dpost}^{\Phi}_{\mathcal{D}}[e](b)\}\,.\end{aligned}$$

4.3 Abstract Reachability Graphs

The abstract post operators of the previous section can be used to construct an *abstract reachability graph*. This generalises the approach taken in [32] for PTAs. In this section, for a given abstract domain, we formally define the concept of an abstract reachability graph, describe how it can be used to construct a stochastic game abstraction of a PTP, and then how to build such a graph. We fix a PTP P and abstract domain $\mathtt{A}=((\mathsf{Z}, \sqcup, \sqcap, \sqsubseteq), \alpha, \gamma)$ over P.

We begin by introducing the concept of *abstract transitions*. An abstract transition of P with respect to the abstract domain $\mathtt{A}$ takes the form:

$$\theta = \big(\mathsf{z}, a, \langle \mathsf{z}_1, \ldots, \mathsf{z}_n \rangle\big) \in \mathsf{Z} \times \mathit{Act} \times \mathsf{Z}^{+}$$

where $n = |\mathit{edges}(\mathit{loc}(\mathsf{z}), a)|$. Intuitively, θ represents the possibility of, from a PTP state in $\gamma(\mathsf{z})$, letting time pass, then taking action a and, for each edge $e_i = (l, a, up_i, X_i, l_i) \in \mathit{edges}(\mathit{loc}(\mathsf{z}), a)$, reaching a state in $\gamma(\mathsf{z}_i)$. The notion of *validity* for an abstract transition expresses the fact that a corresponding concrete transition actually exists. More precisely, we define:

$$\mathit{valid}(\theta) \stackrel{\text{def}}{=} \Big\{ s \in \gamma(\mathsf{z}) \mid \exists t \in \mathbb{R}.\ \Big(s \xrightarrow{t,a} \langle s_1, \ldots, s_n \rangle \wedge \forall 1 \leqslant i \leqslant n.\ s_i \in \gamma(\mathsf{z}_i) \Big) \Big\}$$

as the set of PTP states from which such a transition is possible, and we say that θ is *valid* if the set $\mathit{valid}(\theta)$ is non-empty.

We now explain how validity can be checked for the abstract domains $\mathtt{A}^{\Phi,\Psi}$ and $\mathtt{A}^{\Phi,\mathcal{X}}$. For simplicity, we will split the computation by considering validity with respect to the data and time components of an abstract transition $\theta =$

$(\mathtt{z}, a, \langle \mathtt{z}_1, \ldots, \mathtt{z}_n \rangle)$. More precisely, we let $\mathit{valid}_{\mathcal{D}}(\theta) \subseteq \mathit{Val}(\mathcal{D})$ and $\mathit{valid}_{\mathcal{X}}(\theta) \subseteq \mathbb{R}^{\mathcal{X}}$ denote the sets of data valuations satisfying $\mathit{data}(\mathtt{z})$ and clock valuations satisfying $\mathit{time}(\mathtt{z})$ from which it is possible to let time pass and perform the action a such that taking the ith edge e_i gives a state in $\mathtt{z}_i$. An abstract transition is then *valid* if the valid sets for both data and time are nonempty and moreover:

$$\mathit{valid}(\theta) = \{(l, u, v) \in \gamma(\mathtt{z}) \mid l = \mathit{loc}(\mathtt{z}) \wedge u \in \mathit{valid}_{\mathcal{D}}(\theta) \wedge v \in \mathit{valid}_{\mathcal{X}}(\theta)\}\,.$$

For the data component, $\mathit{valid}_{\mathcal{D}}(\theta)$ is characterised by the formula:

$$\Phi[\mathit{data}(\mathtt{z})] \wedge \big(\mathit{enab}_{\mathcal{D}}(\mathit{loc}(\mathtt{z}), a) \wedge (\wedge_{i=1}^{n} (\mathtt{wp}[\mathit{up}_i](\Phi[\mathit{data}(\mathtt{z}_i)]))\big)$$

where $\mathtt{wp}[\mathit{up}_i]$ denotes the weakest precondition for update up_i. Thus, like in the previous section, we can check emptiness of $\mathit{valid}_{\mathcal{D}}(\theta)$ via a satisfiability check of the above formula using an SMT/SAT solver.

For the time component, we can compute $\mathit{valid}_{\mathcal{X}}(\theta)$ as a zone. For abstract domain $\mathtt{A}^{\Phi,\mathcal{X}}$, where abstract states contain zones, $\mathit{valid}_{\mathcal{X}}(\theta)$ is the zone:

$$\mathit{time}(\mathtt{z}) \wedge \swarrow \big(\mathit{enab}_{\mathcal{X}}(\mathit{loc}(\mathtt{z}), a) \wedge (\wedge_{i=1}^{n} ([X_i{:=}0]\mathit{time}(\mathtt{z}_i)))\big)\,.$$

Checking its emptiness is a simple DBM operation. For abstract domain $\mathtt{A}^{\Phi,\Psi}$, we can perform the same computation after first converting predicate valuations to zones, i.e. replacing $\mathit{time}(\cdot)$ with $\Psi[\mathit{time}(\cdot)]$ in the above.

We are now in a position to define abstract reachability graphs for PTPs.

Definition 8. *An* abstract reachability graph *for PTP* $\mathtt{P}$ *with respect to target locations* F *and abstract domain* $\mathtt{A}{=}((\mathtt{Z}, \sqcup, \sqcap, \sqsubseteq), \alpha, \gamma)$*, is a tuple* $(\mathtt{Y}, \mathtt{R})$ *where:*

- $\mathtt{Y} \subseteq \mathtt{Z}$ *is a covering multiset of abstract states, i.e.* $S \subseteq \cup_{\mathtt{z} \in \mathtt{Y}} \gamma(\mathtt{z})$*;*
- $\mathtt{R} \subseteq \mathtt{Y} \times \mathit{Act} \times \mathtt{Y}^{+}$ *is a set of* valid *abstract transitions;*

such that, if $\mathtt{z} \in \mathtt{Y}$*,* $\mathit{loc}(\mathtt{z}) \notin F$*,* $s \in \gamma(\mathtt{z})$ *and* $s \xrightarrow{t,a} \langle s_1, \ldots, s_n \rangle$*, then* $\mathtt{R}$ *contains an abstract transition* $(\mathtt{z}, a, \langle \mathtt{z}_1, \ldots, \mathtt{z}_n \rangle)$ *such that* $s_i \in \gamma(\mathtt{z}_i)$ *for all* $1 \leqslant i \leqslant n$*.*

An abstract transition $\theta = (\mathtt{z}, a, \langle \mathtt{z}_1, \ldots, \mathtt{z}_n \rangle)$ induces the probability distribution λ_θ over the abstract states $\mathtt{Z}$ where for any $\mathtt{z}' \in \mathtt{Z}$:

$$\lambda_\theta(\mathtt{z}') \stackrel{\text{def}}{=} \textstyle\sum_{i=1}^{n} \{\!|\, \mathit{prob}(l, a)(e_i) \mid \mathtt{z}_i{=}\mathtt{z}' \,|\!\}\ .$$

We now extend the notion of *validity* for abstract transitions to *sets* of abstract transitions with the same source. For abstract state $\mathtt{z} \in \mathtt{Z}$, we let $\mathtt{R}(\mathtt{z})$ denote the set of abstract transitions in $\mathtt{R}$ with source $\mathtt{z}$. Then, for a set of abstract transitions $\Theta \subseteq \mathtt{R}(\mathtt{z})$, we define:

$$\mathit{valid}(\Theta) \stackrel{\text{def}}{=} (\cap_{\theta \in \Theta} \mathit{valid}(\theta)) \setminus \big(\cup_{\theta \in \mathtt{R}(\mathtt{z}) \setminus \Theta} \mathit{valid}(\theta)\big) \tag{2}$$

and say that Θ is *valid* if $\mathit{valid}(\Theta)$ is non-empty. Thus, Θ is valid if there exists a state $s \in \gamma(\mathtt{z})$ such that it is possible to perform the transition encoded by any abstract transition $\theta \in \Theta$, but it is not possible to perform a transition encoded by any other abstract transition of $\mathtt{R}(\mathtt{z})$. To verify the validity of sets of abstract transitions, we use the following result, which reduces the problem to a satisfiability check for which we can employ a SMT/SAT solver.

$\mathsf{BuildGame}(\mathsf{P}, (\mathtt{Y}, \mathtt{R}), \mathtt{A})$

1 $\overline{\mathtt{Y}} = \{\mathtt{y} \mid (\bar{l}, \overline{u}, \mathbf{0}) \in \gamma_{\mathtt{A}}(\mathtt{y})\}$
2 **for** $\mathtt{y} \in \mathtt{Y}$
3 **for** $\Theta \subseteq \mathtt{R}(\mathtt{y})$ **such that** Θ is valid
4 $Steps_{\mathsf{G}}(\mathtt{y}, \Theta) := \{\lambda_\theta \mid \theta \in \Theta\}$
5 **return** $\mathsf{G} = (\mathtt{Y}, \overline{\mathtt{Y}}, 2^{\mathtt{R}}, Steps_{\mathsf{G}})$

Fig. 3. Algorithm for building abstraction from reachability graph

Lemma 1. *Consider any PTP* P*, target set* F*, abstract domain* $\mathtt{A}$ *and a reachability graph* $(\mathtt{Y}, \mathtt{R})$ *for* P *with respect to* F *and* $\mathtt{A}$*. If* $\mathtt{z} \in \mathtt{Y}$ *and* $\Theta \subseteq \mathtt{R}(\mathtt{z})$*, then* Θ *is valid if and only if there exists* $s \in \gamma(\mathtt{z})$ *such that:*

- $data(s) \in \cap_{\theta \in \Theta} valid_{\mathcal{D}}(\theta)$ *and* $time(s) \in \cap_{\theta \in \Theta} valid_{\mathcal{X}}(\theta)$*;*
- *for any* $\theta \in \mathtt{R}(\mathtt{z}) \backslash \Theta$ *either* $data(s) \notin valid_{\mathcal{D}}(\theta)$ *or* $time(s) \notin valid_{\mathcal{X}}(\theta)$*.*

We use the approach of [31] to represent an abstraction of an MDP as a stochastic two-player game. The basic idea is that the two players in the game represent nondeterminism introduced by the abstraction and nondeterminism from the original model. In an abstract state $\mathtt{z}$ of the game abstraction of a PTP, player 1 first picks a PTP state $(l, u, v) \in \gamma(\mathtt{z})$ and then player 2 makes a choice over the actions that become enabled after letting time pass from (l, u, v).

The algorithm $\mathsf{BuildGame}$ in Figure 3 describes how to construct for a PTP P, from a reachability graph $(\mathtt{Y}, \mathtt{R})$ over an abstract domain $\mathtt{A}$, a stochastic game. In a state $\mathtt{y}$ of the game, player 1 chooses between any *valid* set of abstract transitions $\Theta \subseteq \mathtt{R}(\mathtt{y})$. Player 2 then selects an abstract transition $\theta \in \Theta$. As the following result demonstrates, this game yields lower and upper bounds on both the minimum and maximum reachability probabilities of the PTP.

Theorem 1. *Let* P *be a PTP with target locations* F *and* $\mathtt{A}$ *an abstract domain over* P*. If* $(\mathtt{Y}, \mathtt{R})$ *is a reachability graph for* P *with respect to* F *and* $\mathtt{A}$*, then*

$$p^{lb,*}_{\mathsf{G}}\{\mathtt{y} \in \mathtt{Y} \mid loc(\mathtt{y}) \in F\} \;\leqslant\; p^{*}_{\mathsf{P}}(F) \;\leqslant\; p^{ub,*}_{\mathsf{G}}\{\mathtt{y} \in \mathtt{Y} \mid loc(\mathtt{y}) \in F\}$$

where G *is returned by* $\mathsf{BuildGame}(\mathsf{P},(\mathtt{Y},\mathtt{R}),\mathtt{A})$ *(see Figure 3) and* $* \in \{\min, \max\}$*.*

Finally, we describe the process of constructing the reachability graph for a PTP. A generic reachability graph generation algorithm is given in Figure 4 which takes as input a PTP, target set of locations and abstract domain. The following theorem states the correctness of this algorithm.

Theorem 2. *Let* P *be a PTP with target locations* F *and* $\mathtt{A}$ *an abstract domain over* P*. If* $(\mathtt{Y}, \mathtt{R})$ *is returned by* $\mathsf{BuildReachGraph}(\mathsf{P}, F, \mathtt{A})$*, then* $(\mathtt{Y}, \mathtt{R})$ *is a reachability graph of* P *with respect to* F *and* $\mathtt{A}$*.*

The following proposition demonstrates that, for the abstract domain $\mathtt{A}^{\Phi,\Psi}$, the resulting abstraction corresponds to the one generated using the approach of [31], applied to the (infinite-state) MDP semantics of a PTP based on the partition of the state space induced from the predicates Φ and Ψ.

BuildReachGraph($\mathsf{P}, F, \mathtt{A}$)

```
1  Y := ∅
2  X := α(l̄, ū, 0)
3  while X ≠ ∅
4    choose x ∈ X
5    l := loc(x)
6    X := X \ {x}
7    Y := Y ∪ {x}
8    for z ∈ tpost^A[l](x)
9      for a ∈ Act such that enab(l, a) ≠ false
10       for e_i = (l, a, up_i, X_i, l_i) ∈ edges(l, a) = ⟨e_1, ..., e_n⟩
11         Z_i := dpost^A[e_i](z)
12         if l_i ∉ F then X := X ∪ (Z_i \ Y)
13       for ⟨z_1, ..., z_n⟩ ∈ Z_1 × ··· × Z_n
14         R := R ∪ {(x, a, ⟨z_1, ..., z_n⟩)}
15 return (Y, R)
```

Fig. 4. Algorithm for reachability graph construction

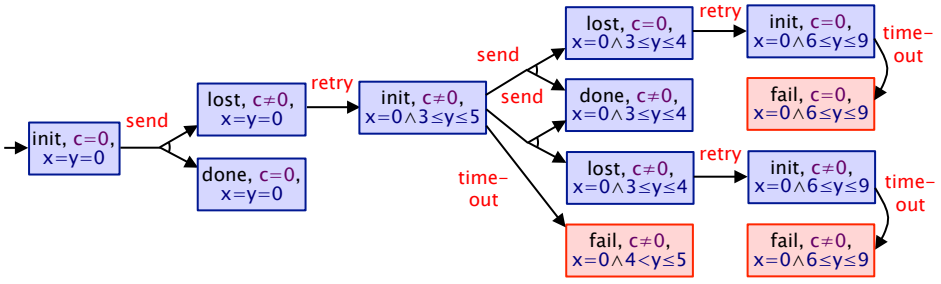

Fig. 5. Abstract reachability graph for the PTP in Example 1 (Figure 2)

Proposition 1. *Let* P *be a PTP with target locations* F *and abstract domain* $\mathtt{A}^{\Phi,\Psi}$*. If* $(\mathtt{Y}, \mathtt{R})$ *is the reachability graph returned by* BuildReachGraph$(\mathsf{P}, F, \mathtt{A}^{\Phi,\Psi})$ *(see Figure 4), then the game* BuildGame$(\mathsf{P}, (\mathtt{Z}, \mathtt{R}), \mathtt{A}^{\Phi,\Psi})$ *equals that constructed by Definition 3 for the MDP* $[\![\mathsf{P}]\!]$ *(after the states with locations in* F *are made absorbing) when using the partition* $\mathcal{P} = \{\gamma(\mathtt{z}) \mid \mathtt{z} \in \mathtt{Z}^{\Phi,\Psi}\}$.

Example 2. We now return to the PTP from Example 1, shown in Figure 2, and consider the construction of an abstraction using the abstract domain $\mathtt{A}^{\Phi,\mathcal{X}}$ with predicate set $\Phi = \{c = 0\}$. Figure 5 shows the abstract reachability graph for this PTP that is constructed by the algorithm BuildReachGraph using $\mathtt{A}^{\Phi,\mathcal{X}}$ and set of target locations $F = \{\mathit{fail}\}$.

The corresponding stochastic game abstraction G of the PTP, under abstract domain $\mathtt{A}^{\Phi,\mathcal{X}}$, is shown in Figure 6. We have $p_{\mathsf{G}}^{lb,\max}(F') = 0.01$ and $p_{\mathsf{G}}^{ub,\max}(F') = 0.1$ where F' contains the states of G with location *fail*. The

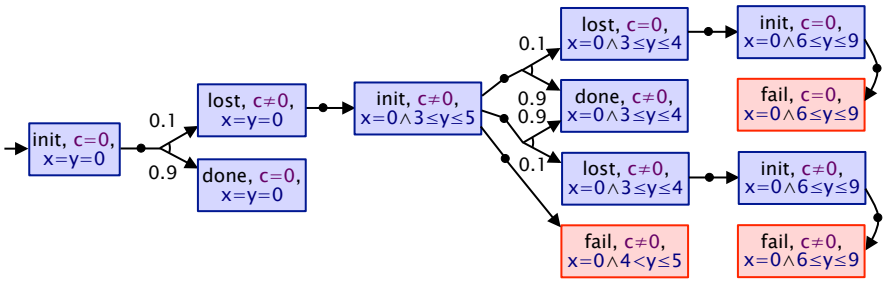

Fig. 6. Stochastic game abstraction for the PTP in Example 1 (Figure 2) and reachability graph in Figure 5

interval $[0.01, 0.1]$ represents lower and upper bounds on the actual maximum probability of reaching the *fail* location in the PTP, which is 0.1.

5 Abstraction Refinement for PTPs

The previous section described how to compute the abstraction of a PTP for two types of abstract domains, $\mathtt{A}^{\Phi,\Psi}$ and $\mathtt{A}^{\Phi,\mathcal{X}}$. To implement a quantitative abstraction refinement scheme similar to that described in Section 2.2, we also require *refinement* techniques for automatically constructing more precise abstractions. In practice, this means splitting one or more abstract states. There are two forms of refinement, either in terms of zones or predicates. In both cases, the refinement process works by modifying the abstract reachability graph. However, in the former, we will split abstract states by breaking up the zone component of an abstract state, while for the latter, we modify the abstract domain by adding a new data or clock predicate.

Choosing a State to Refine. Given an abstraction for a PTP P with target locations F, i.e. a reachability graph $(\mathtt{Y}, \mathtt{R})$ with respect to some abstract domain $\mathtt{A}$, the refinement approach is guided by the analysis of the corresponding stochastic game, i.e. the one generated by $\mathsf{BuildGame}(\mathsf{P}, (\mathtt{Y}, \mathtt{R}), \mathtt{A})$. More precisely, given the bounds for the probability of reaching F and player 1 strategies that attain these bounds, we look at a single abstract state $\mathtt{z}$ for which the bounds differ and for which distinct player 1 strategies yield each bound.[3] In state $\mathtt{z}$, a player 1 strategy chooses an action available in $\mathtt{z}$, which, by construction, is a valid set of abstract transitions from $\mathtt{R}(\mathtt{z})$. Therefore, let $\Theta_{lb}, \Theta_{ub} \subseteq \mathtt{R}(\mathtt{z})$ denote the distinct player 1 strategy choices for the lower and upper bound respectively.

By construction, $\Theta_{lb} \neq \Theta_{ub}$ and hence there exists an abstract transition $\theta^{ref} \in \mathtt{R}(\mathtt{z})$ such that either $\theta^{ref} \in \Theta_{lb} \backslash \Theta_{ub}$ or $\Theta_{ub} \backslash \Theta_{lb}$. We will show that if we refine by adding a predicate for $\mathit{valid}_{\mathcal{D}}(\theta^{ref})$ and, depending on whether the abstract domain is of the form $\mathtt{A}^{\Phi,\Psi}$ or $\mathtt{A}^{\Phi,\mathcal{X}}$, either adding a predicate for

[3] From the results of [31,27] such a state exists when the bounds differ in some state.

RefineDataPredicate($\mathsf{P}, (\mathtt{Y}, \mathtt{R}), \mathtt{A}^{\Phi,\star}, \phi'$)

1 $\Phi^{ref} := \Phi \cup \{\phi'\}$
2 $\mathtt{A}^{ref} := \mathtt{A}^{\Phi^{ref},\star}$
3 $\mathtt{Y}^{ref} := \{\mathtt{y}^{ref} \in \mathtt{Z}^{\Phi^{ref},\star} \mid \exists \mathtt{y} \in \mathtt{Y}.\ \gamma_{\mathtt{A}^{ref}}(\mathtt{y}^{ref}) \subseteq \gamma_{\mathtt{A}}(\mathtt{y})\}$
4 $\mathtt{R}^{ref} := \emptyset$
5 **for** $\theta = (\mathtt{z}_0, a, \langle \mathtt{z}_1, \ldots, \mathtt{z}_n \rangle) \in \mathtt{R}$
6 $\quad \Theta^{new} := \{(\mathtt{z}'_0, a, \langle \mathtt{z}'_1, \ldots, \mathtt{z}'_n \rangle) \mid \gamma_{\mathtt{A}^{ref}}(\mathtt{z}'_i) \subseteq \gamma_{\mathtt{A}}(\mathtt{z}_i)$ for all $0 \leqslant i \leqslant n\}$
7 $\quad$ **for** $\theta^{new} \in \Theta^{new}$ **such that** $valid(\theta^{new}) \neq \emptyset$
8 $\quad\quad \mathtt{R}^{ref} := \mathtt{R}^{ref} \cup \{\theta^{new}\}$
9 **return** $(\mathtt{Z}^{ref}, \mathtt{R}^{ref}, \mathtt{A}^{ref})$

Fig. 7. Algorithm for refinement in terms of data predicates

$valid_{\mathcal{X}}(\theta^{ref})$ or splitting the zone $time(\mathtt{z})$ using $valid_{\mathcal{X}}(\theta^{ref})$, then we eliminate the player 1 choice between Θ_{lb} and Θ_{ub}.

For example, consider the case where $\theta^{ref} \in \Theta_{lb} \backslash \Theta_{ub}$ and the abstract domain is of the form $A^{\Phi,\Psi}$. Now by adding predicates for $valid_{\mathcal{D}}(\theta^{ref})$ and $valid_{\mathcal{X}}(\theta^{ref})$ to Φ and Ψ, it follows that Θ_{lb} can only be valid when both predicates hold. Furthermore, if both predicates hold, then θ^{ref} must be an element of any valid set, since by definition for all s in the concretisation of the abstract state we have $data(s) \in valid_{\mathcal{D}}(\theta^{ref})$ and $time(s) \in valid_{\mathcal{X}}(\theta^{ref})$ since $valid_{\mathcal{D}}(\theta^{ref})$ and $valid_{\mathcal{X}}(\theta^{ref})$ hold respectively.

Notice that we add both data and clock predicates and adding only one of the predicates is insufficient to remove the choice between Θ_{lb} and Θ_{ub}. Similarly, for the abstract domain $\mathtt{A}^{\Phi,\mathcal{X}}$, to ensure the choice is eliminated we must both add a predicate for $valid_{\mathcal{D}}(\theta^{ref})$ and split the zone $time(\mathtt{z})$ using $valid_{\mathcal{X}}(\theta^{ref})$. In the next sections we explain the predicate and zone refinement procedures in more detail.

Data Predicate Refinement. As outlined in the previous section, we refine by adding a predicate $\phi_{\theta^{ref}}$ representing $valid_{\mathcal{D}}(\theta^{ref})$. Notice that if predicate is such that it represents the set $data(\mathtt{z})$, then refinement will not change the abstract model. However, in such a case, by definition of validity it follows that the refinement using $valid_{\mathcal{X}}(\theta^{ref})$ will yield a strict refinement. Similarly, there will be cases where the refinement using $valid_{\mathcal{X}}(\theta^{ref})$ will not change the abstraction, however in these cases, again from definition of validity, it follows that the data predicate refinement presented in this section will yield a strict refinement.

The data predicate refinement algorithm is given in Figure 7 where ϕ' is a predicate over $\mathcal{D}$ (which in practice would correspond to the predicate $\phi_{\theta^{ref}}$). The lines 1–3 create the new abstract domain, while lines 4–8 create an abstract reachability graph over the new abstract domain, for which the corresponding stochastic game is a refined abstraction of the PTP. The following demonstrates the correctness of the data predicate refinement algorithm.

Proposition 2. *Let* P *be a PTP with target locations* F, $\mathtt{A}^{\Phi,\star}$ *an abstract domain for* P, $(\mathtt{Y}, \mathtt{R})$ *a reachability graph for* P *with respect to* F *and* $\mathtt{A}^{\Phi,\star}$, *and* G *the*

RefineClockPredicate$(\mathsf{P}, (\mathtt{Y}, \mathtt{R}), \mathtt{A}^{\Phi,\Psi}, \psi')$

$\Psi^{ref} := \Psi \cup \{\psi'\}$
$\mathtt{A}^{ref} := \mathtt{A}^{\Phi,\Psi^{ref}}$
$\mathtt{Y}^{ref} := \{\mathtt{y}^{ref} \in \mathtt{Z}^{\Phi,\Psi^{ref}} \mid \exists \mathtt{y} \in \mathtt{Y}.\ \gamma_{\mathtt{A}^{ref}}(\mathtt{y}^{ref}) \subseteq \gamma_{\mathtt{A}}(\mathtt{y})\}$
$\mathtt{R}^{ref} := \emptyset$
for $\theta = (\mathtt{z}_0, a, \langle \mathtt{z}_1, \ldots, \mathtt{z}_n \rangle) \in \mathtt{R}$
  $\Theta^{new} := \{(\mathtt{z}'_0, a, \langle \mathtt{z}'_1, \ldots, \mathtt{z}'_n \rangle) \mid \gamma_{\mathtt{A}^{ref}}(\mathtt{z}'_i) \subseteq \gamma_{\mathtt{A}}(\mathtt{z}_i) \text{ for all } 0 \leqslant i \leqslant n\}$
  for $\theta^{new} \in \Theta^{new}$ **such that** $valid(\theta^{new}) \neq \emptyset$
    $\mathtt{R}^{ref} := \mathtt{R}^{ref} \cup \{\theta^{new}\}$
return $(\mathtt{Z}^{ref}, \mathtt{R}^{ref}, \mathtt{A}^{ref})$

Fig. 8. Algorithm for refinement in terms of clock predicates

game. If $(\mathtt{Y}^{ref}, \mathtt{R}^{ref}, \mathtt{A}^{ref})$ *is returned by* RefineDataPredicate$(\mathsf{P}, (\mathtt{Y}, \mathtt{R}), \mathtt{A}^{\Phi,\star}, \varphi')$ *for any data predicate* φ'*, and* G^{ref} *is the resulting game, then:*

(i) $(\mathtt{Z}^{ref}, \mathtt{R}^{ref})$ *is a reachability graph for* P *with respect to* F *and* $\mathtt{A}^{ref}$*;*
(ii) $p^{lb,*}_{\mathsf{G}}(\mathtt{Y}_F) \leqslant p^{lb,*}_{\mathsf{G}^{ref}}(\mathtt{Y}^{ref}_F)$ *and* $p^{ub,*}_{\mathsf{G}^{ref}}(\mathtt{Y}^{ref}_F) \leqslant p^{ub,*}_{\mathsf{G}}(\mathtt{Y}_F)$ *for* $* \in \{\min, \max\}$

where $\mathtt{Y}_F = \{\mathtt{y} \in \mathtt{Y} \mid loc(\mathtt{y}) \in F\}$ *and* $\mathtt{Y}^{ref}_F = \{\mathtt{y} \in \mathtt{Y}^{ref} \mid loc(\mathtt{y}) \in F\}$.

Clock Predicate Refinement. Similarly to the data case, we refine by adding a predicate $\psi_{\theta^{ref}}$ representing $valid_{\mathcal{X}}(\theta^{ref})$. The clock predicate refinement is similar to the data predicate case given in the previous section and the algorithm is presented in Figure 8 where ψ' is a predicate over $\mathcal{X}$ (which in practice would correspond to the predicate $\psi_{\theta^{ref}}$). The following demonstrates the correctness of the clock predicate refinement.

Proposition 3. *Let* P *be a PTP with target locations* F*,* $\mathtt{A}^{\Phi,\Psi}$ *an abstract domain for* P*,* $(\mathtt{Y}, \mathtt{R})$ *a reachability graph for* P *with respect to* F *and* $\mathtt{A}^{\Phi,\Psi}$*, and* G *the game. If* $(\mathtt{Y}^{ref}, \mathtt{R}^{ref}, \mathtt{A}^{ref})$ *is returned by* RefineDataPredicate$(\mathsf{P}, (\mathtt{Y}, \mathtt{R}), \mathtt{A}^{\Phi,\star}, \varphi')$ *for any clock predicate* ψ*, and* G^{ref} *is the resulting game, then:*

(i) $(\mathtt{Z}^{ref}, \mathtt{R}^{ref})$ *is a reachability graph for* P *with respect to* F *and* $\mathtt{A}^{ref}$*;*
(ii) $p^{lb,*}_{\mathsf{G}}(\mathtt{Y}_F) \leqslant p^{lb,*}_{\mathsf{G}^{ref}}(\mathtt{Y}^{ref}_F)$ *and* $p^{ub,*}_{\mathsf{G}^{ref}}(\mathtt{Y}^{ref}_F) \leqslant p^{ub,*}_{\mathsf{G}}(\mathtt{Y}_F)$ *for* $* \in \{\min, \max\}$

where $\mathtt{Y}_F = \{\mathtt{y} \in \mathtt{Y} \mid loc(\mathtt{y}) \in F\}$ *and* $\mathtt{Y}^{ref}_F = \{\mathtt{y} \in \mathtt{Y}^{ref} \mid loc(\mathtt{y}) \in F\}$.

Zone Refinement. In this case, the abstract domain is of the form $\mathtt{A}^{\Phi,\mathcal{X}}$ and we split the zone component of $\mathtt{z}$ into:

$$valid_{\mathcal{X}}(\theta^{ref}) \text{ and } time(\mathtt{z}) \wedge \neg(valid_{\mathcal{X}}(\theta^{ref})) \,. \tag{3}$$

Clearly, if $valid_{\mathcal{X}}(\theta^{ref}) = time(\mathtt{z})$, then such a splitting will not change the abstraction, however, as explained above, in such a situation the data (predicate) refinement yields a strict refinement.

$\mathsf{RefineZone}(\mathsf{P}, (\mathtt{Y}, \mathtt{R}), \mathtt{A}^{\Phi,\mathcal{X}}, \mathtt{z}, (\zeta_1, \ldots, \zeta_k))$

1 $\mathtt{Y}^{new} := \{(loc(\mathtt{z}), data(\mathtt{z}), \zeta_1), \ldots, (loc(\mathtt{z}), data(\mathtt{z}), \zeta_k))\}$
2 $\mathtt{Y}^{ref} := (\mathtt{Y} \setminus \{\mathtt{z}\}) \uplus \mathtt{Y}^{new}$
3 $\mathtt{R}^{ref} := \emptyset$
4 **for** $\theta = (\mathtt{z}_0, a, \langle \mathtt{z}_1, \ldots, \mathtt{z}_n \rangle) \in \mathtt{R}$
5 **if** $\mathtt{z} \notin \{\mathtt{z}_0, \mathtt{z}_1, \ldots, \mathtt{z}_n\}$ **then**
6 $\mathtt{R}^{ref} := \mathtt{R}^{ref} \cup \{\theta\}$
7 **else**
8 $\Theta^{new} := \{(\mathtt{z}'_0, a, \langle \mathtt{z}'_1, \ldots, \mathtt{z}'_n \rangle) \mid \mathtt{z}'_i \in \mathtt{Y}^{new}$ if $\mathtt{z}_i = \mathtt{z}$ and $\mathtt{z}'_i = \mathtt{z}_i$ o/wise$\}$
9 **for** $\theta^{new} \in \Theta^{new}$ **such that** $valid(\theta^{new}) \neq \emptyset$
10 $\mathtt{R}^{ref} := \mathtt{R}^{ref} \cup \{\theta^{new}\}$
11 **return** $(\mathtt{Y}^{ref}, \mathtt{R}^{ref})$

Fig. 9. Algorithm to perform zone refinement in abstract state $\mathtt{z}$

The refinement algorithm is shown in Figure 9. It takes as input a PTP, abstract domain and set of zones and returns a new reachability graph for the PTP with respect to the same abstract domain. When using the algorithm the set $\{\zeta_1, \ldots, \zeta_k\}$ is given by the non-empty zones in (3). Lines 1–2 split the abstract state $\mathtt{z}$ based on the set of zones given as input, then, based on this splitting, lines 3–10 update the set of abstract transitions $\mathtt{R}$ resulting in a new reachability graph, for which the corresponding stochastic game is a refined abstraction of the PTP. The following result states the correctness of the zone refinement scheme.

Proposition 4. *Let* P *be a PTP with target locations* F*,* $\mathtt{A}^{\Phi,\mathcal{X}}$ *an abstract domain for* P*,* $(\mathtt{Y}, \mathtt{R})$ *a reachability graph for* P *with respect to* F *and* $\mathtt{A}^{\Phi,\mathcal{X}}$ *and* G *the game. If* $(\mathtt{Y}^{ref}, \mathtt{R}^{ref})$ *is returned by* $\mathsf{RefineZone}(\mathsf{P}, (\mathtt{Y}, \mathtt{R}), \mathtt{A}^{\Phi,\mathcal{X}}, \mathtt{z}, \{\zeta_1, \ldots, \zeta_k\})$ *for* $\mathtt{z} \in \mathtt{Y}$ *and* $\zeta_1, \ldots, \zeta_k \in Zones(\mathcal{X})$ *and* $\mathsf{G}^{ref} = \mathsf{BuildGame}(\mathsf{P}, (\mathtt{Z}^{ref}, \mathtt{R}^{ref}), A^{\Phi,\mathcal{X}})$*, then:*

(i) $(\mathtt{Z}^{ref}, \mathtt{R}^{ref})$ *is a reachability graph for* P *with respect to* F *and* $\mathtt{A}^{\Phi,\mathcal{X}}$*;*
(ii) $p^{lb,*}_{\mathsf{G}}(\mathtt{Y}_F) \leqslant p^{lb,*}_{\mathsf{G}^{ref}}(\mathtt{Y}^{ref}_F)$ *and* $p^{ub,*}_{\mathsf{G}^{ref}}(\mathtt{Y}^{ref}_F) \leqslant p^{ub,*}_{\mathsf{G}}(\mathtt{Y}_F)$ *for* $* \in \{\min, \max\}$

where $\mathtt{Y}_F = \{\mathtt{y} \in \mathtt{Y} \mid loc(\mathtt{y}) \in F\}$ *and* $\mathtt{Y}^{ref}_F = \{\mathtt{y} \in \mathtt{Y}^{ref} \mid loc(\mathtt{y}) \in F\}$.

Quantitative Abstraction Refinement. The refinement schemes presented above, combined with the techniques for abstraction given in the previous section, can be combined into a quantitative abstraction refinement loop. This provides fully automatic construction of abstractions for PTPs.

This refinement scheme, applied in a iterative manner, provides a way of computing exact values for minimum or maximum reachability probabilities of a PTP. This algorithm, outlined in Figure 10, starts with some abstract domain, then constructs the reachability graph and repeatedly: (i) builds a stochastic game; (ii) solves the game to obtain lower and upper bounds; and (iii) refines the abstract domain reachability graph, based on an analysis of the game. The iterative process terminates when the difference between the bounds falls below

AbstractRefine($\mathsf{P}, F, \mathtt{A}, \star, \varepsilon$)

1 $(\mathtt{Z}, \mathtt{R}) :=$ BuildReachGraph($\mathsf{P}, F, \mathtt{A}$)
2 $\mathsf{G} :=$ BuildGame($(\mathtt{Z}, \mathtt{R}), \mathtt{A}$)
3 $(p_{\mathsf{G}}^{lb,\star}, p_{\mathsf{G}}^{ub,\star}, \sigma_1^{lb}, \sigma_1^{ub}) :=$ AnalyseGame($\mathsf{G}, F, \star$)
4 **while** $p_{\mathsf{G}}^{ub,\star} - p_{\mathsf{G}}^{lb,\star} > \varepsilon$
5 $\quad(\mathtt{Z}, \mathtt{R}, \mathtt{A}) :=$ Refine($\mathtt{Z}, \mathtt{R}, \mathtt{A}, \sigma_1^{lb}, \sigma_1^{ub}$)
6 $\quad\mathsf{G} :=$ BuildGame($(\mathtt{Z}, \mathtt{R}), \mathtt{A}$)
7 $\quad(p_{\mathsf{G}}^{lb,\star}, p_{\mathsf{G}}^{ub,\star}, \sigma_1^{lb}, \sigma_1^{ub}) :=$ AnalyseGame($\mathsf{G}, F, \star$)
8 **return** $[p_{\mathsf{G}}^{lb,\star}, p_{\mathsf{G}}^{ub,\star}]$

Fig. 10. Abstraction-refinement loop to compute reachability probabilities

a given level of precision ε. In fact, as the following result states, this process is guaranteed to terminate, in a finite number of steps, with the precise answer.

6 Quantitative Verification of SystemC

We now describe a specific potential instantiation of the verification framework we have presented. In particular, we discuss its applicability to the quantitative verification of SystemC, a system-level modelling language that is becoming increasingly prominent in the development of embedded systems, e.g. for System-on-Chip (SoC) designs. Currently, analysis of SystemC designs is primarily performed using simulation; however, there is growing interest in applying verification techniques [14,30,16,24,43,19].

SystemC is appealing to designers because it is close enough to the hardware level to support synthesis to RTL (register transfer level) descriptions, but allows modelling of complex designs at a higher level of abstraction. Based on C++, it combines an imperative programming style, low-level data-types for hardware, an object-oriented approach to design and convenient high-level abstractions of concurrent communicating processes. Furthermore, systems can be efficiently simulated at the design stage.

System-on-Chip designs typically include many different components, including for example support for radio communications. Furthermore, these devices may then become integrated into larger, networked embedded systems. In these instances, reasoning about the behaviour of a SystemC design may need to take account of the inherently stochastic characteristics of the unreliable or unpredictable components that it interacts with. Another source of probabilistic behaviour is the use of randomisation. This is a key feature of, for example communication technologies like ZigBee, which are increasingly found in today's embedded devices.

Considered in its entirety, a quantitative analysis of SystemC requires all of the basic ingredients that we have proposed for probabilistic timed programs:

– **software**: basic process behaviour is defined in terms of C++ code, using a rich array of data types;

- **concurrency**: designs comprise multiple concurrent processes, communicating through message-passing primitives;
- **timing**: processes can be subjected to precisely timed delays, through interaction with the SystemC scheduler;
- **probability**: SystemC components may link to unpredictable devices, due to communication failures, unreliable components or randomisation.

The development of (non-quantitative) verification techniques for SystemC has already been identified as an important, but difficult, challenge [46]. Applying quantitative verification offers more powerful analysis techniques but promises to be even more demanding. In the remainder of this section, we outline how some of the existing approaches and tools for SystemC verification might be built upon to implement our framework. We then conclude by identifying some useful directions and challenges for future work.

Translating SystemC to PTPs. A SystemC design is decomposed into *modules*, representing the separate components within a design. Modules are connected, through *ports*, to *channels*, which model interactions between components. Built-in "primitive" channels such as signals, FIFOs and mutexes are provided. The behaviour of each module is described by a set of *threads* or *processes*, specified as C++ class methods.

In [19], a translation from SystemC to the timed automata based input language of UPPAAL is proposed and implemented. Our probabilistic timed program formalism is a superset of timed automata so the basic ideas can be used directly.[4] The approach of [19] is to translate each C++ method, representing a process or thread into a timed automaton. This is based on an extraction of the control flow graph: control vertices becomes locations, control flow edges become transitions and branching conditions (e.g. on `if` statements or `while` loops) are incorporated into the enabling conditions of transitions. The process of generating the control flow graph is facilitated by model extraction tools for SystemC like Scoot [5] and PINAPA [39]. These have been designed with a variety of applications in mind, including verification.

In order to ensure that the predicate abstraction techniques described in this paper can be applied to SystemC C++ code, the underlying SMT/SAT solvers used need to support the data-types and operations allowed in the language. Since SystemC offers low-level datatypes aimed at hardware designs, a SAT-based approach using a bit-level semantics is likely to be needed [7]. This approach was already applied to abstraction-based software model checking of SystemC in [30] and to a probabilistic extension of ANSI-C in [26].

Scheduling and Timed Behaviour. Concurrency between SystemC threads is controlled by the *scheduler*, whose behaviour is precisely defined in the language standard [22]. The SystemC scheduler is *co-operative* and *non-preemptive*: threads suspend themselves explicitly by calling a `wait()` or `wait(t)` function.

[4] In practice, we need to add various syntactic niceties such as urgent and committed locations, and communication over channels, but this is relatively straightforward.

The latter is an example of *timed* behaviour: the calling thread is suspended until it receives a timed notification from the scheduler after delay `t`.

The translation scheme of [19] captures the behaviour of the SystemC scheduler as a network of timed automata. This keeps track of which processes should be run in each part of each scheduler cycle. Uncertainty between the order in which multiple ready processes are executed is modelled as a nondeterministic choice. Delays in each process are handled by local clocks in each automaton.

Probabilistic Behaviour. As outlined above, it is often desirable to incorporate either randomisation or failures into SystemC models. For randomisation, this is likely to appear as calls to a C/C++ `rand()` function. Alternatively, as is done in [26], custom randomised functions could be added. In either case, these can be intercepted and converted to a probabilistic branch in the PTP. For failures, as in [26], SystemC code that corresponds to connections or communications with unreliable components can be replaced by a stub that captures the stochastic behaviour (e.g. using `rand()` as above). Probabilistic timed automata (PTAs) have already been applied to a large number of realistic case studies in which randomisation or failures are modelled in this fashion [35,11,36,34].

Directions & Challenges. Implementing the verification techniques sketched in this section represents a considerable challenge. As ever, the most immediate difficulty is scalability: extending existing tools and techniques to handle the size and complexity of real SystemC designs. In this respect, it may be beneficial to consider state-of-the-art techniques for software model checking, which are not currently applied to the probabilistic case because they are non-trivial to adapt. These include, for example, the use of approximate abstractions and "lazy" construction of abstractions. One particular source of complexity in SystemC models is concurrency between processes. Development of software model checking techniques for concurrent programs is another very active field of research, that may yield gains in this area.

In a different direction, we may also aim to improve the expressivity of the PTP formalism proposed in this paper. In the current version, probability and time are largely orthogonal which, in many cases, is not a serious restriction for system modelling. However, it would be interesting to explore to what extent this can generalised. An extension with costs and rewards would be relatively straightforward. Looking further ahead, the use of languages like SystemC in embedded systems means that the digital components of the design will often interact with analogue devices. This would necessitate the use of more general, but less tractable, probabilistic models such as stochastic hybrid automata.

7 Related work

For quantitative verification of probabilistic timed automata (PTAs), a variety of techniques [23,35,11,36,34,32] and tools [45,17,4] have been produced. In [17], an extension of PTAs with discrete-valued variables, called VPTAs, is handled

via a translation to the basic case. Another interesting extension is *priced* PTAs [34,3], which add a notion of prices (or weights) to locations and actions.

The development of abstraction and refinement techniques for probabilistic models such as MDPs is also an active area of research. This was first proposed in [10], using MDPs as abstractions, rather than stochastic games as in [27,26,32] (and this paper). In [20], MDP abstractions based on predicates are used to form a probabilistic CEGAR (counterexample-guided abstraction refinement) technique. Later work [47] adapted this to stochastic games. Other abstraction refinement frameworks for MDPs are put forward in [12] and [6].

Abstraction-refinement approaches have been proposed for non-probabilistic timed automata, e.g. [29], which uses bounded model checking and SAT-based techniques, [42], which is based on the region graph construction, and [13], for verifying PLC automata using UPPAAL [38]. Also related is [48], which applies SAT-based techniques to timed automata with data.

Finally, as highlighted in the previous section, there is an increasing amount of interest in developing verification techniques for SystemC [46]. A variety of existing verification techniques have been explored, including BDD-based model exploration [14], bounded model checking [16] and abstraction-refinement [30]. Another approach is to translate SystemC into other formalisms and languages for which tool support exists. This includes translations to Petri nets [24], Promela [43] and UPPAAL [19]. With the exception of [19], which models timing information, none of the above consider quantitative properties of SystemC.

8 Conclusions

We have outlined a theoretical framework for the verification of programs that exhibit both probabilistic and timed behaviour, based on quantitative abstraction refinement techniques. This represents the first steps towards quantitative verification of complex software systems, such as those found in the domain of embedded systems. We discussed some of the ongoing work and the challenges in this direction of research, using the SystemC language as an illustrative example.

Acknowledgements. The authors are supported in part by EPSRC grants EP/D07956X, EP/D076625 and EP/F001096, EU FP7 project CONNECT and ERC Advanced Grant VERIWARE.

References

1. R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.
2. D. Beauquier. Probabilistic timed automata. *Theoretical Computer Science*, 292(1):65–84, 2003.
3. J. Berendsen, D. Jansen, and J.-P. Katoen. Probably on time and within budget on reachability in priced probabilistic timed automata. In *Proc. 3rd International Conference on Quantitative Evaluation of Systems (QEST'06)*, pages 311–322. IEEE CS Press, 2006.

4. J. Berendsen, D. Jansen, and F. Vaandrager. Fortuna: Model checking priced probabilistic timed automata. In *Proc. 7th International Conference on Quantitative Evaluation of Systems (QEST'10)*. IEEE CS Press, 2010. To appear.
5. N. Blanc, D. Kroening, and N. Sharygina. Scoot: A tool for the analysis of SystemC models. In C. Ramakrishnan and J. Rehof, editors, *Proc. TACAS'08*, volume 4963 of *LNCS*, pages 467–470. Springer, 2008.
6. R. Chadha and M. Viswanathan. A counterexample guided abstraction-refinement framework for Markov decision processes. *ACM Transactions on Computational Logic*, 2010. To appear.
7. E. Clarke, D. Kroening, N. Sharygina, and K. Yorav. Predicate abstraction of ANSI-C programs using SAT. *Formal Methods in System Design*, 25(2-3):105–127, 2004.
8. A. Condon. The complexity of stochastic games. *Information and Computation*, 96(2):203–224, 1992.
9. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. 4th ACM Symposium on Principles of Programming Languages (POPL'77)*, pages 238–252. ACM Press, 1977.
10. P. D'Argenio, B. Jeannet, H. Jensen, and K. Larsen. Reachability analysis of probabilistic systems by successive refinements. In L. de Alfaro and S. Gilmore, editors, *Proc. 1st Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modelling and Verification (PAPM/PROBMIV'01)*, volume 2165 of *LNCS*, pages 39–56. Springer, 2001.
11. C. Daws, M. Kwiatkowska, and G. Norman. Automatic verification of the IEEE 1394 root contention protocol with KRONOS and PRISM. *Int. Journal on Software Tools for Technology Transfer (STTT)*, 5(2–3):221–236, 2004.
12. L. de Alfaro and P. Roy. Magnifying-lens abstraction for Markov decision processes. In *Proc. 19th International Conference on Computer Aided Verification (CAV'07)*, volume 4590 of *LNCS*, pages 325–338. Springer, 2007.
13. H. Dierks, S. Kupferschmid, and K. Larsen. Automatic abstraction refinement for timed automata. In J.-F. Raskin and P. Thiagarajan, editors, *Proc. 5th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'07)*, volume 4763 of *LNCS*, pages 114–129. Springer, 2007.
14. R. Drechsler and D. Grosse. Reachability analysis for formal verification of SystemC. In *Proc. Euromicro Symposium on Digital Systems Design (DSD'02)*, page 337. IEEE, 2002.
15. S. Graf and H. Saidi. Construction of abstract state graphs with PVS. In O. Grumberg, editor, *Proc. 9th International Conference on Computer Aided Verification (CAV'97)*, volume 1254 of *LNCS*, pages 72–83. Springer, 1997.
16. D. Grosse, U. Kühne, and R. Drechsler. HW/SW co-verification of embedded systems using bounded model checking. In *Proc. 16th ACM Great Lakes Symposium on VLSI*, pages 43–48. ACM Press, 2006.
17. A. Hartmanns and H. Hermanns. A Modest approach to checking probabilistic timed automata. In *Proc. 6th International Conference on Quantitative Evaluation of Systems (QEST'09)*, pages 187–196, 2009.
18. T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.
19. P. Herber, J. Fellmuth, and S. Glesner. Model checking SystemC designs using timed automata. In *Proc. 6th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, pages 131–136, 2008.

20. H. Hermanns, B. Wachter, and L. Zhang. Probabilistic CEGAR. In A. Gupta and S. Malik, editors, *Proc. 20th International Conference on Computer Aided Verification (CAV'08)*, volume 5123 of *LNCS*, pages 162–175. Springer, 2008.
21. A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In H. Hermanns and J. Palsberg, editors, *Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*, volume 3920 of *LNCS*, pages 441–444. Springer, 2006.
22. IEEE Standards Association. IEEE standard 1666: SystemC language reference manual. `http://www.systemc.org/`, 2005.
23. H. Jensen. Model checking probabilistic real time systems. In B. Bjerner, M. Larsson, and B. Nordström, editors, *Proc. 7th Nordic Workshop on Programming Theory*, pages 247–261, 1996.
24. D. Karlsson, P. Eles, and Z. Peng. Formal verification of SystemC designs using a Petri-net based representation. In *Proc. Design Automation & Test in Europe Conference (DATE'06)*, pages 1228–1233. EDAA, 2006.
25. J.-P. Katoen, E. Hahn, H. Hermanns, D. Jansen, and I. Zapreev. The ins and outs of the probabilistic model checker MRMC. In *Proc. 6th International Conference on Quantitative Evaluation of Systems (QEST'09)*. IEEE CS Press, 2009.
26. M. Kattenbelt, M. Kwiatkowska, G. Norman, and D. Parker. Abstraction refinement for probabilistic software. In N. Jones and M. Muller-Olm, editors, *Proc. Proc. 10th International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'09)*, volume 5403 of *LNCS*, pages 182–197. Springer, 2009.
27. M. Kattenbelt, M. Kwiatkowska, G. Norman, and D. Parker. A game-based abstraction-refinement framework for Markov decision processes. *Formal Methods in System Design*, 36(3):246–280, 2010.
28. J. Kemeny, J. Snell, and A. Knapp. *Denumerable Markov Chains*. Springer, 1976.
29. S. Kemper and A. Platzer. SAT-based abstraction refinement for real-time systems. In *Proc. FACS 2006*, volume 182 of *ENTCS*, pages 107–122, 2007.
30. D. Kroening and N. Sharygina. Formal verification of SystemC by automatic hardware/software partitioning. In *Proc. 3rd ACM & IEEE International Conference on Formal Methods and Models for Co-Design (MEMOCODE'05)*, pages 101–110. IEEE, 2005.
31. M. Kwiatkowska, G. Norman, and D. Parker. Game-based abstraction for Markov decision processes. In *Proc. 3th International Conference on Quantitative Evaluation of Systems (QEST'06)*, pages 157–166. IEEE CS Press, 2006.
32. M. Kwiatkowska, G. Norman, and D. Parker. Stochastic games for verification of probabilistic timed automata. In J. Ouaknine and F. Vaandrager, editors, *Proc. 7th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'09)*, volume 5813 of *LNCS*, pages 212–227. Springer, 2009.
33. M. Kwiatkowska, G. Norman, and D. Parker. A framework for verification of software with time and probabilities. In K. Chatterjee and T. Henzinger, editors, *Proc. 8th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'10)*, LNCS. Springer, 2010. To appear.
34. M. Kwiatkowska, G. Norman, D. Parker, and J. Sproston. Performance analysis of probabilistic timed automata using digital clocks. *Formal Methods in System Design*, 29:33–78, 2006.
35. M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, 282:101–150, 2002.

36. M. Kwiatkowska, G. Norman, J. Sproston, and F. Wang. Symbolic model checking for probabilistic timed automata. *Information and Computation*, 205(7):1027–1077, 2007.
37. S. Lahiri, T. Ball, and B. Cook. Predicate abstraction via symbolic decision procedures. In K. Etessami and S. Rajamani, editors, *Proc. 17th International Conference on Computer Aided Verification (CAV'05)*, volume 3576 of *LNCS*, pages 24–38. Springer, 2005.
38. K. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.
39. M. Moy, F. Maraninchi, and L. Maillet-Contoz. Pinapa: An extraction tool for SystemC descriptions of Systems-on-a-Chip. In *Proc. 5th ACM International Conference on Embedded Software (EMSOFT05)*, pages 317–324, 2005.
40. M. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, 1994.
41. L. Shapley. Stochastic games. In *Proc. National Academy of Science*, volume 39, pages 1095–1100, 1953.
42. M. Sorea. Lazy approximation for dense real-time systems. In Y. Lakhnech and S. Yovine, editors, *Proc. Joint International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS) and Formal Techniques in Real-Time and Fault Tolerant Systems (FTRTFT)*, volume 3253 of *LNCS*, pages 363–378. Springer, 2004.
43. C. Traulsen, J. Cornet, M. Moy, and F. Maraninchi. A SystemC/TLM semantics in Promela and its possible applications. In D. Bošnacki and S. Edelkamp, editors, *Proc. 14th Int. SPIN Workshop on Model Checking of Software (SPIN'07)*, volume 4595 of *LNCS*, pages 204–222. Springer, 2007.
44. S. Tripakis. *The formal analysis of timed systems in practice*. PhD thesis, Université Joseph Fourier, 1998.
45. Uppaal-PRO. `http://www.cs.aau.dk/~arild/uppaal-probabilistic/`.
46. M. Vardi. Formal techniques for SystemC verification. In *Proc. 44th Design Automation Conference (DAC'07)*, pages 188–192, 2007. Position Paper.
47. B. Wachter and L. Zhang. Best probabilistic transformers. In G. Barthe and M. Hermenegildo, editors, *Proc. 11th International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'10)*, volume 5944 of *LNCS*, pages 362–379. Springer, 2010.
48. A. Zbrzezny and Półrola. SAT-based reachability checking for timed automata with discrete data. *Fundamenta Informaticae*, 79(3-4):579–593, 2008.

Appendix

We include below proofs of the main results stated in the paper.

A Proof of Theorem 1

Consider any PTP P with target locations F and abstract domain $\mathtt{A}$. Let $[\![\mathsf{P}]\!] = (S, \overline{S}, \mathbb{R}\times Act, Steps_{\mathsf{P}})$, $(\mathtt{Y}, \mathtt{R})$ be a reachability graph for P with respect to F and $\mathtt{A}$ and $\mathsf{G} = (\mathtt{Y}, \overline{\mathtt{y}}, 2^{\mathtt{R}}, Steps_{\mathsf{G}})$ the game returned by $\mathsf{BuildGame}(\mathsf{P},(\mathtt{Y},\mathtt{R}),\mathtt{A})$. Before we present the proof of Theorem 1 we require the following lemmas. For the remainder of the section let $\mathtt{Y}_F = \{\mathtt{y} \in \mathtt{Y} \mid loc(\mathtt{y}) \in F\}$.

Lemma 2. *For any adversary A of $[\![\mathsf{P}]\!]$ and $s \in S$ there exists a strategy pair (σ_1, σ_2) of G where $p^A_s(F) = p^{\sigma_1,\sigma_2}_{\mathtt{z}}(\mathtt{Y}_F)$ for all $\mathtt{z} \in \mathtt{Y}$ such that $s \in \gamma(\mathtt{z})$.*

Proof. Consider any adversary A of $[\![\mathsf{P}]\!]$, concrete state $s \in S$ and abstract state $\mathtt{z} \in \mathtt{Y}$ such that $s \in \gamma(\mathtt{z})$. The proof follows similarly to [32], by constructing a strategy pair of the game G which matches the choices made by A. More precisely, we match the transitions chosen by A in $[\![\mathsf{P}]\!]$ with transitions in G chosen by the strategy pair (σ_1, σ_2) such that the targets of $[\![\mathsf{P}]\!]$ are concretisations of the targets in G. Supposing, in state s, under A the action $(t, a) \in \mathbb{R}\times Act$ is chosen, that is the transition:

$$s \xrightarrow{(t,a)} \langle s_1, \ldots, s_n \rangle$$

is performed, then since $(\mathtt{Y}, \mathtt{R})$ is a reachability graph (see Definition 8) there exists a symbolic transition $\theta{=}(\mathtt{z}, a, \langle \mathtt{z}_1, \ldots, \mathtt{z}_n \rangle) \in \mathtt{R}$ such that $s_i \in \gamma(\mathtt{z}_i)$ for all $1{\leqslant}i{\leqslant}n$ and we construct the strategy pair (σ_1, σ_2) such that the player one strategy σ_1 choose some $\Theta \subseteq \mathtt{R}(\mathtt{z})$ with $\theta \in \Theta$ and let σ_2 choose θ from Θ. The fact that the reachability probabilities are the same for A and (σ_1, σ_2) then follows from the fact that the corresponding transitions are constructed from the same distribution, namely $prob(l, a)$ when s is of the form (l, u, v) for some $u \in Val(\mathcal{D})$ and $v \in \mathbb{R}^{\mathcal{X}}$. □

Lemma 3. *for any abstract state $\mathtt{z}$ and player 2 strategy σ_2 of G there exists an adversary A of $[\![\mathsf{P}]\!]$ such that:*

$$\inf_{\sigma_1} p^{\sigma_1,\sigma_2}_{\mathtt{z}}(\mathtt{Y}_F) \leqslant p^A_s(F) \;\; \textit{and} \;\; \sup_{\sigma_1} p^{\sigma_1,\sigma_2}_{\mathtt{z}}(\mathtt{Y}_F) \geqslant p^A_s(F)$$

for all $s \in S$ such that $s \in \gamma(\mathtt{z})$.

Proof. Consider any player 2 strategy σ_2 of G, abstract state $\mathtt{z} \in \mathtt{Y}$ and concrete state $s \in \gamma(\mathtt{z})$. By the assumptions we make on PTPs, $s \xrightarrow{(t,a)} \langle s_1, \ldots, s_n \rangle$ for some $(t, a) \in \mathbb{R}\times Act$. Now, since $(\mathtt{Y}, \mathtt{R})$ is a reachability graph, there exists $\theta{=}(\mathtt{z}, a, \langle \mathtt{z}_1, \ldots, \mathtt{z}_n \rangle) \in \mathtt{R}$ such that $s_i \in \gamma(z_i)$ for all $1{\leqslant}i{\leqslant}n$. It then follows by definition that $s \in valid(\theta)$, and hence there exists $\Theta \subseteq \mathtt{R}(\mathtt{z})$ such that:

- $data(s) \in \cap_{\theta\in\Theta} valid_{\mathcal{D}}(\theta)$ and $time(s) \in \cap_{\theta\in\Theta} valid_{\mathcal{X}}(\theta)$;

– for any $\theta \in \mathtt{R}(\mathtt{z}) \backslash \Theta$ either $data(s) \notin valid_{\mathcal{D}}(\theta)$ or $time(s) \notin valid_{\mathcal{X}}(\theta)$.

Now, using Lemma 1 it follows that Θ is a valid set of abstract transitions, and we let σ_1 be the player 1 strategy which chooses Θ. Supposing σ_2 chooses some $\theta'=(\mathtt{z}, a', \langle \mathtt{z}'_1, \ldots, \mathtt{z}'_m \rangle) \in \Theta$ then, from properties of Θ we have that $s \in valid(\theta')$, and hence, since $(\mathtt{Y}, \mathtt{R})$ is a reachability graph, there exists $t' \in \mathbb{R}$ such that $s \xrightarrow{(t',a')} \langle s'_1, \ldots, s'_m \rangle$ and $s'_i \in \gamma(\mathtt{z}'_i)$ for all $1 \leqslant i \leqslant m$. Now we construct A such thatin state s the time-action pair (a', t') is chosen. Repeating this process inductively on the paths of the game we arrive at a player 1 strategy σ_1 and adversary A of $[\![\mathsf{P}]\!]$ such that

$$p_{\mathtt{z}}^{\sigma_1,\sigma_2}(\mathtt{Y}_F) = p_s^A(F)$$

which is sufficient to complete the proof. □

Proof (of Theorem 1). From Lemma 2 it follows that for any $s \in S$:

$$\begin{aligned} \inf_{\sigma_1,\sigma_2} p_{\mathtt{z}}^{\sigma_1,\sigma_2}(\mathtt{Y}_F) &\leqslant \inf_A P_s^A(F) \\ \sup_A p_s^A(F) &\leqslant \sup_{\sigma_1,\sigma_2} p_{\mathtt{z}}^{\sigma_1,\sigma_2}(\mathtt{Y}_F) \end{aligned}$$

for all $\mathtt{z} \in \mathtt{Y}$ such that $s \in \gamma(z)$, and hence $p_{\mathsf{G}}^{lb,\min}(\mathtt{Y}_F) \leqslant p_{\mathsf{P}}^{\min}(F)$ and $p_{\mathsf{P}}^{\max}(F) \leqslant p_{\mathsf{G}}^{ub,\max}(\mathtt{Y}_F)$. On the other hand, using Lemma 3, we have for any $s \in S$ and $\mathtt{z} \in \mathtt{Y}$ such that $s \in \gamma(z)$:

$$\inf_A p_s^A(F) \leqslant \inf_{\sigma_2} \sup_{\sigma_1} p_{\mathtt{z}}^{\sigma_1,\sigma_2}(\mathtt{Y}_F) = \sup_{\sigma_1} \inf_{\sigma_2} p_{\mathtt{z}}^{\sigma_1,\sigma_2}(\mathtt{Y}_F)$$

where the second step follows from properties of stochastic games [8]. Similarly, we can show that:

$$\inf_{\sigma_1} \sup_{\sigma_2} p_{\mathtt{z}}^{\sigma_1,\sigma_2}(\mathtt{Y}_F) \leqslant \sup_A p_s^A(F)$$

and therefore $p_{\mathsf{P}}^{\min}(\mathtt{Y}_F) \leqslant p_{\mathsf{G}}^{ub,\min}(\mathtt{Y}_F)$ and $p_{\mathsf{G}}^{lb,\max}(\mathtt{Y}_F) \leqslant p_{\mathsf{P}}^{\max}(F)$ which completes the proof. □

B Proof of Theorem 2

Proof (of Theorem 2). Consider any PTP P with target locations F and abstract domain $\mathtt{A}$, and suppose $(\mathtt{Y}, \mathtt{R})$ is returned by $\mathsf{BuildReachGraph}(\mathsf{P}, F, \mathtt{A})$. First, considering the set of abstract states $\mathtt{Y}$, from the definition of $\mathsf{tpost}^{\mathtt{A}}$ and $\mathsf{dpost}^{\mathtt{A}}$ it follows that $\mathsf{BuildReachGraph}(\mathsf{P}, F, \mathtt{A})$ includes a forwards search of the concrete state space $[\![P]\!]$ and hence $\mathtt{Y}$ will form a covering of abstract states.

It therefore remains to show that $\mathtt{R}$ is a set of valid abstract transitions such that if $\mathtt{z} \in \mathtt{Y}$, $loc(\mathtt{z}) \notin F$, $s \in \gamma(\mathtt{z})$ and $s \xrightarrow{t,a} \langle s_1, \ldots, s_n \rangle$, then $\mathtt{R}$ contains an abstract transition $(\mathtt{z}, a, \langle \mathtt{z}_1, \ldots, \mathtt{z}_n \rangle)$ such that $s_i \in \gamma(\mathtt{z}_i)$ for all $1 \leqslant i \leqslant n$. These properties follow from the definitions of $[\![\mathsf{P}]\!]$, $\mathsf{tpost}^{\mathtt{A}}$ and $\mathsf{dpost}^{\mathtt{A}}$ completing the proof. □

C Proof of Proposition 1

To present the proof of this proposition we first require the following lemma.

Lemma 4. *Let* P *be a PTP with target locations* F *and abstract domain* $\mathtt{A}$*. If* $(\mathtt{Y}, \mathtt{R})$ *is the reachability graph returned by* $\mathsf{BuildReachGraph}(\mathsf{P}, F, \mathtt{A})$*, then for any abstract state* $z \in \mathtt{Y}$ *and concrete state* $s \in \gamma(\mathtt{z})$ *the set* $\Theta_{\mathtt{z},s} = \{\theta \in \mathtt{R}(\mathtt{z}) \mid s \in \mathit{valid}(\theta)\}$ *is non-empty and:*

- $\mathit{data}(s) \in \cap_{\theta \in \Theta_{\mathtt{z},s}} \mathit{valid}_{\mathcal{D}}(\theta)$ *and* $\mathit{time}(s) \in \cap_{\theta \in \Theta_{\mathtt{z},s}} \mathit{valid}_{\mathcal{X}}(\theta)$*;*
- *for any* $\theta \in \mathtt{R}(\mathtt{z}) \backslash \Theta_{\mathtt{z},s}$ *either* $\mathit{data}(s) \notin \mathit{valid}_{\mathcal{D}}(\theta)$ *or* $\mathit{time}(s \notin \mathit{valid}_{\mathcal{X}}(\theta)$*.*

Furthermore, for any abstract state $\mathtt{z} \in \mathtt{Y}$ *and valid set* $\Theta \subseteq \mathtt{R}(\mathtt{z})$*, there exists a state* $s \in \gamma(\mathtt{z})$ *such that* $\Theta_s = \Theta$*.*

Proof. Consider any PTP P with target locations F and abstract domain $\mathtt{A}$ and let $(\mathtt{Y}, \mathtt{R})$ be the reachability graph constructed by $\mathsf{BuildReachGraph}(\mathsf{P}, F, \mathtt{A})$. Now if we consider any abstract state $\mathtt{z} \in \mathtt{Y}$ and concrete state $s \in \gamma(\mathtt{z})$, the fact $\Theta_{\mathtt{z},s}$ is non-empty follows by the assumptions we have made about PTPs. Next, by definition of $\mathit{valid}_{\mathcal{D}}$ and $\mathit{valid}_{\mathcal{X}}$, we have $\mathit{data}(s) \in \cap_{\theta \in \Theta_s} \mathit{valid}_{\mathcal{D}}(\theta)$ and $\mathit{time}(s) \in \cap_{\theta \in \Theta_s} \mathit{valid}_{\mathcal{X}}(\theta)$. To complete the proof of the first half, suppose for a contradiction that there exists $\theta \in \mathtt{R}(\mathtt{z}) \backslash \Theta_{\mathtt{z},s}$ such that $\mathit{data}(s) \in \mathit{valid}_{\mathcal{D}}(\theta)$ and $\mathit{time}(s) \in \mathit{valid}_{\mathcal{X}}(\theta)$. It then follows by definition of *valid* that $s \in \mathit{valid}(\theta)$ which is a contradiction of $\theta \in \mathtt{R}(\mathtt{z}) \backslash \Theta_{\mathtt{z},s}$.

To prove the second half of the lemma, consider any abstract state $\mathtt{z} \in \mathtt{Y}$ and valid set $\Theta \subseteq \mathtt{R}(\mathtt{z})$. By Lemma 1 there exists $s \in \gamma(\mathtt{z})$ such that:

- $\mathit{data}(s) \in \cap_{\theta \in \Theta} \mathit{valid}_{\mathcal{D}}(\theta)$ and $\mathit{time}(s) \in \cap_{\theta \in \Theta} \mathit{valid}_{\mathcal{X}}(\theta)$;
- for any $\theta \in \mathtt{R}(\mathtt{z}) \setminus \Theta$ either $\mathit{data}(s) \notin \mathit{valid}_{\mathcal{D}}(\theta)$ or $\mathit{time}(s) \notin \mathit{valid}_{\mathcal{X}}(\theta)$.

It then follows that $\Theta_{\mathtt{z},s} = \Theta$ completing the proof. □

Proof (Proof of Proposition 1). Consider any PTP P with target locations F and abstract domain $\mathtt{A}^{\Phi,\Psi}$. Let G_1 be the game constructed by Definition 3 for the MDP $[\![\mathsf{P}]\!]$ (after the states with locations in F are made absorbing) when using the partition $\mathcal{P} = \{\gamma(\mathtt{z}) \mid \mathtt{z} \in \mathtt{Z}^{\Phi,\Psi}\}$. Furthermore, let $(\mathtt{Y}, \mathtt{R})$ be the reachability graph returned by $\mathsf{BuildReachGraph}(\mathsf{P}, F, \mathtt{A}^{\Phi,\Psi})$ and G_2 the stochastic game returned by $\mathsf{BuildGame}(\mathsf{P}, (\mathtt{Y}, \mathtt{R}), \mathtt{A}^{\Phi,\Psi})$.

The fact that the states $\mathtt{Y}$ and initial states $\overline{\mathtt{Y}}$ of the games G_1 and G_2 are the same is a direct result of the construction of the games (see Definition 3 and Figures 3 and 4). It therefore remains to show that the probabilistic transition relations of the two games coincide. More precisely, we will show that for any abstract state $\mathtt{z} \in \mathtt{Y}$:

1. if $\Lambda \in 2^{Act \times \mathsf{Dist}(\mathtt{Y})}$ and $\mathit{Steps}_{\mathsf{G}_2}(\mathtt{z}, \Lambda)$ is defined, then there exists $\Theta \in \mathtt{R}(\mathtt{z})$ such that $\mathit{Steps}_{\mathsf{G}_1}(\mathtt{z}, \Theta) = \mathit{Steps}_{\mathsf{G}_2}(\mathtt{z}, \Lambda)$;
2. if $\Theta \in \mathtt{R}(\mathtt{z})$ and $\mathit{Steps}_{\mathsf{G}_1}(\mathtt{z}, \Theta)$ is defined, then there exists $\Lambda \in 2^{Act \times \mathsf{Dist}(\mathtt{Y})}$ such that $\mathit{Steps}_{\mathsf{G}_2}(\mathtt{z}, \Lambda) = \mathit{Steps}_{\mathsf{G}_1}(\mathtt{z}, \Theta)$.

To prove the correctness of these properties we first require the following result, which is a direct consequence of the definitions of $\mathsf{tpost}^{\Phi,\Psi}$ and $\mathsf{dpost}^{\Phi,\Psi}$, for any concrete state s of $[\![\mathsf{P}]\!]$, action $a \in Act$ and distribution $\lambda \in \mathsf{Dist}(\mathtt{Y})$:

$$\begin{aligned} &s \xrightarrow{t,a} \lambda' \text{ for some } t \in \mathbb{R} \text{ and } \lambda' \in \mathsf{Dist}(S) \text{ such that } \lambda'_{\mathcal{P}}{=}\lambda \Leftrightarrow \\ &\quad \exists \theta = (\alpha(s), a, \langle \mathtt{z}_1, \ldots, \mathtt{z}_n \rangle) \in \mathtt{R}(\alpha(s)) \text{ such that } s \in \mathit{valid}(\theta) \text{ and } \lambda_\theta{=}\lambda\,. \quad (4) \end{aligned}$$

Notice, in the case of the abstract domain $\mathtt{A}^{\Phi,\Psi}$, $\alpha(s)$ is a well defined element of $\mathtt{Y}$. To prove the first property, consider any $\mathtt{z} \in \mathtt{Y}$ and $\Lambda \in 2^{Act \times \mathsf{Dist}(\mathtt{Y})}$ such that $\mathit{Steps}_{\mathsf{G}_1}(\mathtt{z}, \Lambda)$ is defined. By Definition 3 there exists $s \in \gamma(s)$ such that

$$\Lambda = \{((a,t), \mathit{Steps}_{[\![\mathsf{P}]\!]}(s,a)_{\mathcal{P}}) \,|\, (a,t) \in \mathit{Act}(s)\}\,.$$

Now, again using Definition 3, we have:

$$\begin{aligned} \mathit{Steps}_{\mathsf{G}_1}(\mathtt{z}, \Lambda) &= \{\mathit{Steps}_{[\![\mathsf{P}]\!]}(s,(t,a))_{\mathcal{P}} \,|\, (t,a) \in \mathit{Act}(s)\} \\ &= \{\lambda_{\mathcal{P}} \,|\, s \xrightarrow{t,a} \mu \text{ and } (t,a) \in A{\times}\mathbb{R}\} && \text{by Definition 5} \\ &= \{\lambda_\theta \,|\, \theta \in \mathtt{R}(\alpha(s)) \text{ and } s \in \mathit{valid}(\theta)\} && \text{by (4)} \\ &= \{\lambda_\theta \,|\, \theta \in \Theta_{\mathtt{z},s}\} && \text{by definition of } \Theta_{\mathtt{z},s} \\ &= \mathit{Steps}_{\mathsf{G}_2}(z, \Theta_{\mathtt{z},s}) && \text{by Lemma 4,} \end{aligned}$$

and hence the first property holds. The second property follows similarly, using the second half of Lemma 4 which completes the proof. □

D Proof of Proposition 2

Consider any PTP P with target locations F and abstract domain $\mathtt{A}^{\Phi,\star}$. Let $(\mathtt{Y}, \mathtt{R})$ be a reachability graph for P with respect to F and $\mathtt{A}^{\Phi,\star}$ with corresponding game G. Furthermore, let φ' be a data predicate and $(\mathtt{Y}^{ref}, \mathtt{R}^{ref}, \mathtt{A}^{ref})$ be returned by $\mathsf{RefineDataPredicate}(\mathsf{P}, (\mathtt{Y}, \mathtt{R}), \mathtt{A}^{\Phi,\star}, \varphi')$ with corresponding game G^{ref}. Before we give the proof we require the following lemmas and for the remainder of the section we let $\mathtt{Y}_F = \{\mathtt{y} \in \mathtt{Y} \mid \mathit{loc}(\mathtt{y}) \in F\}$.

Lemma 5. *If* $\mathtt{z}^{ref} \in \mathtt{Y}^{ref}$, $(\mathtt{z}^{ref}, a, \langle \mathtt{z}_1^{ref}, \ldots, \mathtt{z}_n^{ref} \rangle) \in \mathtt{R}(\mathtt{z}^{ref})$ *and* $\mathtt{z} \in \mathtt{Y}$ *such that* $\gamma(\mathtt{z}^{ref}) \subseteq \gamma(\mathtt{z})$, *then there exists* $(\mathtt{z}, a, \langle \mathtt{z}_1, \ldots, \mathtt{z}_n \rangle) \in \mathtt{R}$ *such that* $\gamma(\mathtt{z}_i^{ref}) \subseteq \gamma(\mathtt{z}_i)$ *for all* $1 \leqslant i \leqslant n$.

Proof. The proof follows by construction of $(\mathtt{Y}^{ref}, \mathtt{R}^{ref})$ see Figure 7. □

Lemma 6. *For any strategy pair* $(\sigma_1^{ref}, \sigma_2^{ref})$ *of* G^{ref} *and* $\mathtt{z}^{ref} \in \mathtt{Z}^{ref}$ *there exists a strategy pair* (σ_1, σ_2) *of* G *where* $p_{\mathtt{z}}^{\sigma_1,\sigma_2}(\mathtt{Y}_F) = p_{\mathtt{z}^{ref}}^{\sigma_1^{ref},\sigma_2^{ref}}(\mathtt{Y}_F)$ *for all* $\mathtt{z} \in \mathtt{Y}$ *such that* $\gamma(\mathtt{z}^{ref}) \subseteq \gamma(\mathtt{z})$.

Proof. Consider any any strategy pair $(\sigma_1^{ref}, \sigma_2^{ref})$ of G^{ref}, $\mathtt{z}^{ref} \in \mathtt{Z}^{ref}$ and $\mathtt{z} \in \mathtt{Y}$ such that $\gamma(\mathtt{z}^{ref}) \subseteq \gamma(\mathtt{z})$. We construct the strategy pair (σ_1, σ_2) of G so that

in state $\mathtt{z}$ they match the choice made by the pair $(\sigma_1^{ref}, \sigma_2^{ref})$ in $\mathtt{z}^{ref}$. If in $\mathtt{z}^{ref}$ the choice of $(\sigma_1^{ref}, \sigma_2^{ref})$ corresponds to the symbolic transition $(\mathtt{z}^{ref}, a, \langle \mathtt{z}_1^{ref}, \ldots, \mathtt{z}_n^{ref} \rangle)$, then, using Lemma 5, there exists $(\mathtt{z}, a, \langle \mathtt{z}_1, \ldots, \mathtt{z}_n \rangle)$ of G such that $\gamma(\mathtt{z}_i^{ref}) \subseteq \gamma(\mathtt{z}_i)$ for all $1 \leqslant i \leqslant n$ and we construct (σ_1, σ_2) such that their choice corresponds to this symbolic transition. The remainder of the proof then follows in an identical fashion to Lemma 2. □

Lemma 7. *For any $\mathtt{z} \in \mathtt{Z}$ and player 2 strategy σ_2 of G there exists a strategy pair $(\sigma_1^{ref}, \sigma_2^{ref})$ of G^{ref} where*

$$\inf_{\sigma_1} p_{\mathtt{z}}^{\sigma_1,\sigma_2}(\mathtt{Y}_F) \leqslant p_{\mathtt{z}^{ref}}^{\sigma_1^{ref},\sigma_2^{ref}}(\mathtt{Y}_F) \;\; \textit{and} \;\; p_{\mathtt{z}^{ref}}^{\sigma_1^{ref},\sigma_2^{ref}}(\mathtt{Y}_F) \leqslant \sup_{\sigma_1} p_{\mathtt{z}}^{\sigma_1,\sigma_2}(\mathtt{Y}_F)$$

for all $\mathtt{z}^{ref}$ such that $\gamma(\mathtt{z}^{ref}) \subseteq \gamma(\mathtt{z})$.

Proof. Given a player 2 strategy σ_2^{ref} of G^{ref} the proof follows by constructing a player 1 strategy σ_1^{ref} of G^{ref} and strategy pair (σ_1, σ_2) of G such that:

$$p_{\mathtt{z}}^{\sigma_1,\sigma_2}(\mathtt{Y}_F) = p_{\mathtt{z}^{ref}}^{\sigma_1^{ref},\sigma_2^{ref}}(\mathtt{Y}_F)$$

for all $\mathtt{z}^{ref}$ such that $\gamma(\mathtt{z}^{ref}) \subseteq \gamma(\mathtt{z})$. This follows similarly to Lemma 3 using Lemma 5 to construct the choices of σ_1^{ref} and (σ_1, σ_2). □

Proof (of Proposition 2). The fact that $(\mathtt{Y}^{ref}, \mathtt{R}^{ref})$ is a reachability graph follows by construction of $(\mathtt{Y}^{ref}, \mathtt{R}^{ref})$ see Figure 7 The second part of the proof follows similarly to the proof of Theorem 1 using Lemmas 6 and 7 instead of Lemmas 2 and 3. □

E Proof of Proposition 3

Proof (of Proposition 3). The proof follows similarly to Proposition 2. □

F Proof of Proposition 4

Proof (of Proposition 4). The proof is a straightforward extension of that presented in [32] for PTAs. □