PRTS: Specification and Model Checking

Jun Sun[1], Songzheng Song[2], Yang Liu[2] and Jin Song Dong[2]

[1] Singapore University of Technology and Design
sunjun@sutd.edu.sg
[2] National University of Singapore
{songsongzheng,liuyang.comp,dongjs.comp}@nus.edu.sg

1 Introduction

With the development of computing and sensing technology, information process and control software are integrated into everyday objects and activities. Design and development of control software for safety-critical systems are notoriously difficult problems. Real-life systems often have complex data components or complicated hierarchical control flows. Furthermore, control software often interacts with physical environment and therefore depends on quantitative timing. It is a challenging task to verify hierarchical complex real-time systems. In addition, probability exhibits itself commonly in the form of statistical estimates regarding the environment in which control software is embedded. Requiring a system always function perfectly within any environment is often overwhelming.

In last 3 years, we aim to develop a useful tool for verifying hierarchical complex probabilistic real-time systems. In the first phase, we proposed a language called CSP# for system modeling. CSP# is an expressive language, combining Hoare's CSP [7] and data structures; after that, we extended this language to support probabilistic choices and we named it as PCSP#. It extended previous work on combining CSP with probabilistic choice [8] or on combining CSP with data structures [11]. Now we integrate real-time into this language and get PRTS. PRTS combines low-level programs, e.g., sequence programs defined in a simple imperative language or any C# program, with high-level specifications (with process constructs like parallel, choice, hiding, etc.), as well as timed transitions and probabilistic choices. It supports shared variables as well as abstract events, making it both state-based and event-based. Its underlying semantics is based on Markov Decision Processes (MDP) [3].

In summary, our own model checker PAT now could model and verify complex systems which include concurrency, real-time and probabilistic choices. These features guarantee users to analyze complicated real life systems such as Biology system, communication protocol and lift system. In this report, we introduce all the syntax and semantics used in PRTS and present some basic algorithms applied in PAT to handle this kind of model.

2 Background

In this section, we give some concepts and definitions that will be used throughout the rest of the report.

When modeling probabilistic systems (particularly, discrete-time stochastic control processes), Markov Decision Process (MDP) is one of the most widely used models. An MDP is a directed graph whose transitions are labeled with events or probability, and whose states are labeled with atomic propositions. The following notations are used to denote different transition labels. $\mathbb{R}_+$ denotes the set of non-negative real numbers; $\epsilon \in \mathbb{R}_+$ denotes the event of idling for exactly ϵ time units; τ denotes an unobservable event; Act denotes the set of observable events such that $\tau \notin Act$; Act_τ denotes $Act \cup \{\tau\}$. Given a set of states S, a distribution is a function $\mu : S \to [0, 1]$ such that $\Sigma\, \mu(s) = 1$. We say μ is a trivial distribution or is trivial if and only if there exists a state s such that $\mu(s) = 1$. Let $Distr(S)$ be the set of all distributions over S. Formally, we have the following definition.

Definition 1. *An MDP is a tuple $\mathcal{D} = (S, init, Act, Pr)$ where S is a set of system states which are related to variables and process; $init \in S$ is the initial state; $Pr : S \times (Act_\tau \cup \mathbb{R}_+) \times Distr(S)$ is a transition relation*[3].

For simplicity, a transition is written as: $s \xrightarrow{x} \mu$ such that $s \in S$; $x \in Act_\tau \cup \mathbb{R}_+$ and $\mu \in Distr(S)$. If μ is trivial, i.e., $\mu(s') = 1$, then we write $s \xrightarrow{x} s'$. There are three kinds of transitions. A time-transition is labeled with a real-valued constant $\epsilon \in \mathbb{R}_+$. An observable transition is labeled with an event in Act. An un-observable transition is labeled with τ. We state that $\mathcal{D}$ is finite if and only if S is finite. Throughout this report, we have following assumptions.

- *MDPs are assumed to be deadlock-free following the standard practice*. A deadlocking MDP can be made deadlock-free by adding self loops labeled with τ with probability 1 to the deadlocking states, without affecting the result of probabilistic verification.
- Without loss of generality, *we assume that all variables have finite domains and every process reachable from the initial configuration is finite as in [9].*

A run of $\mathcal{D}$ is $\pi = \langle s_0, a_0, \mu_0, s_1, a_1, \mu_1, \cdots \rangle$ such that $s_0 = init$ and $(s_i, a_i, \mu_i) \in Pr$ and $\mu_i(s_{i+1}) > 0$ for all i. The sequence of observable events in π, denoted as $trace(\pi)$, is a trace of $\mathcal{D}$. Let $runs(\mathcal{D})$ denote the set of runs of $\mathcal{D}$. Let $traces(\mathcal{D})$ denote the set of traces of $\mathcal{D}$. The probability of exhibiting π in $\mathcal{D}$, denoted as $\mathcal{P}_\mathcal{D}(\pi)$, is $\mu_0(s_1) * \mu_1(s_2) * \cdots$. Let X be a set of runs, we define $\mathcal{P}_\mathcal{D}(X)$ to be the accumulated probability of exhibiting any run in X, i.e., $\mathcal{P}_\mathcal{D}(X) = \Sigma_{x \in X} \mathcal{P}_\mathcal{D}(x)$. Intuitively speaking, given a state s, firstly an enabled event and a distribution is selected nondeterministically by a *scheduler*, and then one of the successor states is reached according to the probability distribution. A scheduler is a function deciding which event and distribution to choose based on the run history. A Markov Chain [1] can be defined given an MDP $\mathcal{D}$ and a scheduler δ, which is denoted as $\mathcal{D}^\delta$. Intuitively, a Markov Chain is an MDP where only one event and distribution is enabled at every state.

3 Syntax

Processes in Probabilistic Timed Stateful CSP# can be defined by the grammar presented in Figure 1.

[3] This definition is slightly different from the standard one [1].

$$
\begin{array}{lll}
P = \mathit{Stop} & & \text{– in-action} \\
\quad \mid \mathit{Skip} & & \text{– termination} \\
\quad \mid e \rightarrow P & & \text{– event prefixing} \\
\quad \mid a\{\mathit{program}\} \rightarrow P & & \text{– data operation prefixing} \\
\quad \mid \textbf{if } (b)\ \{P\} \textbf{ else } \{Q\} & & \text{– conditional choice} \\
\quad \mid P \mid Q & & \text{– general choice} \\
\quad \mid P \setminus X & & \text{– hiding} \\
\quad \mid P;\ Q & & \text{– sequential composition} \\
\quad \mid P \parallel Q & & \text{– parallel composition} \\
\quad \mid Q & & \text{– process referencing} \\
\quad \mid \mathit{Wait}[d] & & \text{– delay} \\
\quad \mid P\ \mathit{timeout}[d]\ Q & & \text{– timeout} \\
\quad \mid P\ \mathit{interrupt}[d]\ Q & & \text{– timed interrupt} \\
\quad \mid P\ \mathit{within}[d] & & \text{– timed responsiveness} \\
\quad \mid P\ \mathit{deadline}[d] & & \text{– deadline} \\
\quad \mid \textbf{pcase } \{pr_0 : P_0;\ pr_1 : P_1;\ \cdots;\ pr_k : P_k\} & & \text{– probabilistic multi-choices}
\end{array}
$$

Fig. 1. Process constructs

Process Stop does nothing but idling. Process Skip terminates, possibly after idling for some time. Process $e \rightarrow P$ engages in event e first and then behaves as P. Note that e may serve as a synchronization barrier, if combined with parallel composition. In order to seamlessly integrate data operations, we allow sequential programs to be attached with events. Process $a\{\mathit{program}\} \rightarrow P$ performs data operation a (i.e., executing the sequential $\mathit{program}$ whilst generating event a) and then behaves as P. The $\mathit{program}$ may be a simple procedure updating data variables (written in the form of $a\{x := 5;\ y := 3\}$) or a complicated sequential program. A conditional choice is written as **if** (b) $\{P\}$ **else** $\{Q\}$. If b is true, then it behaves as P, else it behaves as Q. Process $P \parallel Q$ offers an (unconditional) choice between P and Q[4]. Process $P;\ Q$ behaves as P until P terminates and then behaves as Q immediately. $P \setminus X$ hides occurrences of events in X. Parallel composition of two processes is written as $P \parallel Q$, where P and Q may communicate via multi-party event synchronization (following CSP rules [7]) or shared variables. A process may be given a name, written as $P \mathrel{\widehat{=}} Q$, and then referenced through its name.

A number of timed process constructs can be used to capture common real-time system behavior patterns. Let d denote an integer constant. Process $\mathit{Wait}[d]$ idles for exactly d time units. In process $P\ \mathit{timeout}[d]\ Q$, the first observable event of P shall occur before d time units elapse (since process $P\ \mathit{timeout}[d]\ Q$ is activated). Otherwise, Q takes over control after exactly d time units. Process $P\ \mathit{interrupt}[d]\ Q$ behaves exactly as P (so that it may engage in multiple observable events) until d time units, and then Q takes over. Process $P\ \mathit{within}[d]$ must react within d time units, i.e., an observable event must be engaged by process P within d time units. Note that $P\ \mathit{within}[d]$

[4] For simplicity, we leave out external and internal choices in the classic CSP [7].

puts a constraint on P. Urgent event prefixing [5], written as $e \xrightarrow{!} P$, is defined as $(e \rightarrow P)\ within[0]$, i.e., e must occur as soon as it is enabled. Process $P\ deadline[d]$ constrains P to terminate (possibly engaging in multiple observable events) before d time units. In the following, d is referred to as the parameter of the timed process. Given a model, the maximum parameter of the timed processes is called the clock ceiling.

Lastly, probabilistic choice is written in the form of **pcase** $\{pr_0 : P_0;\ pr_1 : P_1;\ \cdots;\ pr_k : P_k\}$. where pr_i is an integer constant to express the probability weight. Intuitively, it means that with $\frac{pr_i}{pr_0+pr_1+\cdots+pr_k}$ probability, the system behaves as P_i.

4 Operational Semantics

In this part we introduce operational semantics for PRTS in PAT. According to if we use state abstraction or not, two kinds of system configurations are used.

4.1 Concrete Configuration

A concrete system configuration is a pair (σ, P) where σ is a variable valuation and $P \in \mathcal{P}$ is a process. For simplicity, an empty valuation is written as $\varnothing$. A transition of the system is written in the form $(\sigma, P) \xrightarrow{x} (\sigma', P')$ such that $x \in \Sigma_\tau \cup \mathbb{R}_+$. There are two kinds of transitions, i.e., a transition labeled with an event in Σ_τ or a time transition labeled with a number in $\mathbb{R}_+$. The operational semantics is defined by associating a set of firing rules with each and every process construct. The firing rules associated with processes are presented as follows.

$$\frac{}{(\sigma, Stop) \xrightarrow{\epsilon} (\sigma, Stop)}\ [\ st\ ]$$

$$\frac{}{(\sigma, Skip) \xrightarrow{\checkmark} (\sigma, Stop)}\ [\ sk1\ ] \qquad \frac{}{(\sigma, Skip) \xrightarrow{\epsilon} (\sigma, Skip)}\ [\ sk2\ ]$$

– The semantics of $Stop$ is captured by rule st which means $Stop$ can do nothing but just idle(no event is executed); Rule $sk1,\ sk2$ indicate $Skip$ could either execute a terminating event or just keep idling.

$$\frac{}{(\sigma, e \rightarrow P) \xrightarrow{\epsilon} (\sigma, e \rightarrow P)}\ [\ ev1\ ] \qquad \frac{}{(\sigma, e \rightarrow P) \xrightarrow{e} (\sigma, P)}\ [\ ev2\ ]$$

– The semantics of $e \rightarrow P$ is captured by these two rules which means this process could keep stable if it just idles ϵ time units($ev1$) or execute e and P takes control($ev2$).

$$\frac{}{(\sigma, a\{pr\} \rightarrow P) \xrightarrow{\epsilon} (\sigma, a\{pr\} \rightarrow P)}\ [\ as1\ ]$$

$$\frac{}{(\sigma, a\{pr\} \rightarrow P) \xrightarrow{a} (pr(\sigma), P)}\ [\ as2\ ]$$

– The semantics of $a\{pr\} \rightarrow P$ is defined as above which means this process could keep unchanged if it just idles ϵ time units($as1$) or execute a and then behaves as P, meanwhile, σ is updated by program pr ($as2$).

$$\frac{}{(\sigma, \textbf{if } b \textbf{ then } \{P\} \textbf{ else } \{Q\}) \xrightarrow{\epsilon} (\sigma, \textbf{if } b \textbf{ then } \{P\} \textbf{ else } \{Q\})} [\ co1\ ]$$

$$\frac{\sigma \vDash b}{(\sigma, \textbf{if } b \textbf{ then } \{P\} \textbf{ else } \{Q\}) \xrightarrow{\tau} (\sigma, P)} [\ co2\ ]$$

$$\frac{\sigma \nvDash b}{(\sigma, \textbf{if } b \textbf{ then } \{P\} \textbf{ else } \{Q\}) \xrightarrow{\tau} (\sigma, Q)} [\ co3\ ]$$

– The semantics of **if** (b) $\{P\}$ **else** $\{Q\}$ is defined by rules $co1, co2, co3$. $co1$ is straightforward; $co2$ means if σ satisfies b, then through an invisible event, P will take control; $co3$ means if σ doesn't satisfy b, then through τ event, Q will take control.

$$\frac{(\sigma, P) \xrightarrow{a} (\sigma', P')}{(\sigma, P \mid Q) \xrightarrow{a} (\sigma', P')} [\ ch1\ ] \qquad \frac{(\sigma, Q) \xrightarrow{a} (\sigma', Q')}{(\sigma, P \mid Q) \xrightarrow{a} (\sigma', Q')} [\ ch2\ ]$$

$$\frac{(\sigma, P) \xrightarrow{\epsilon} (\sigma, P'), (\sigma, Q) \xrightarrow{\epsilon} (\sigma, Q')}{(\sigma, P \mid Q) \xrightarrow{\epsilon} (\sigma, P' \mid Q')} [\ ch3\ ]$$

– The semantics of $P \mid Q$ is defined by above rules. $co1(co2)$ indicates if $P(Q)$ could engage event a and then becomes $P'(Q')$, $P \mid Q$ could also engage this event and becomes $P'(Q')$. $co3$ says if P and Q both could idle for ϵ time units, so does $P \mid Q$. And $choice$ is still there after idling.

$$\frac{(\sigma, P) \xrightarrow{x} (\sigma', P'), x \notin X}{(\sigma, P \setminus X) \xrightarrow{x} (\sigma', P' \setminus X)} [\ hi1\ ] \qquad \frac{(\sigma, P) \xrightarrow{e} (\sigma', P'), e \in X}{(\sigma, P \setminus X) \xrightarrow{\tau} (\sigma', P' \setminus X)} [\ hi2\ ]$$

– The semantics of $P \setminus X$ is defined by $hi1, hi2$. The first rule presents that if event x is not included in X and after executing it (σ, P) becomes (σ', P'), then hiding X has no affect on this event. However, if a visible event which is in X occurs, then after hiding X, this event becomes invisible.

$$\frac{(\sigma, P) \xrightarrow{x} (\sigma', P'), \checkmark \notin init(\sigma, P)}{(\sigma, P;\ Q) \xrightarrow{x} (\sigma', P';\ Q)} [\ se1\ ] \qquad \frac{(\sigma, P) \xrightarrow{\checkmark} (\sigma', P')}{(\sigma, P;\ Q) \xrightarrow{\checkmark} (\sigma', Q)} [\ se2\ ]$$

– $se1$ and $se2$ define the semantics of $P;\ Q$. $se1$ says if P could engage event x which is not terminating event, then $P;\ Q$ could also engage this event and $sequencial$ symbol retains. However, if after terminating event, (σ, P) becomes (σ', P'), then after this event, Q will take control in $P;\ Q$.

$$\frac{(\sigma, P) \xrightarrow{a} (\sigma', P'), a \notin X}{(\sigma, P \,|[\, X \,]|\, Q) \xrightarrow{a} (\sigma', P' \,|[\, X \,]|\, Q)} [\ pa1\ ]$$

$$\frac{(\sigma, Q) \xrightarrow{a} (\sigma', Q'), a \notin X}{(\sigma, P \,|[\, X \,]|\, Q) \xrightarrow{a} (\sigma', P \,|[\, X \,]|\, Q')} [\ pa2\ ]$$

$$\frac{(\sigma, P) \xrightarrow{x} (\sigma', P'), (\sigma, Q) \xrightarrow{x} (\sigma', Q'), x \in X \cup \mathbb{R}_+}{(\sigma, P \,|[\, X \,]|\, Q) \xrightarrow{x} (\sigma', P' \,|[\, X \,]|\, Q')} [\ pa3\ ]$$

– The semantics of $P \,|[\, X \,]|\, Q$ is defined by above rules. X means the set of events on which P and Q could synchronize. $pa1(pa2)$ says if $P(Q)$ could engage event a which is not in X, then $P \,|[\, X \,]|\, Q$ could engage this event and $Q(P)$ keeps unchanged. If P and Q engage a common event $x \in X \cup \mathbb{R}_+$, then $P \,|[\, X \,]|\, Q$ could synchronize on x and make one step.

$$\frac{(\sigma, Q) \xrightarrow{x} (\sigma', Q'), P \mathrel{\widehat{=}} Q}{(\sigma, P) \xrightarrow{x} (\sigma', Q')} [\ def\ ]$$

– The semantics of $P \mathrel{\widehat{=}} Q$ is defined by rule def which indicates if Q engages event x and changes to Q', so does P, which is a reference of Q.

$$\frac{\epsilon \leq d}{(\sigma, Wait[d]) \xrightarrow{\epsilon} (\sigma, Wait[d - \epsilon])} [\ wait1\ ]$$

$$\frac{}{(\sigma, Wait[0]) \xrightarrow{\tau} (\sigma, Skip)} [\ wait2\ ]$$

– The semantics of $Wait[d]$ is captured by rule $wait1$ and $wait2$. Rule $wait1$ states that the process may idle for an arbitrary amount of time ϵ such that $\epsilon \leq d$. Afterwards, $Wait[d]$ becomes $Wait[d - \epsilon]$. The valuation of the variables is unchanged. Rule $wait2$ states that the process becomes $Skip$ via a τ-transition whenever d is 0.

$$\frac{(\sigma, P) \xrightarrow{e} (\sigma', P')}{(\sigma, P\ timeout[d]\ Q) \xrightarrow{e} (\sigma', P')} [\ to1\ ]$$

$$\frac{}{(\sigma, P\ timeout[0]\ Q) \xrightarrow{\tau} (\sigma, Q)}\ [\ to2\ ]$$

$$\frac{(\sigma, P) \xrightarrow{\tau} (\sigma', P')}{(\sigma, P\ timeout[d]\ Q) \xrightarrow{\tau} (\sigma', P'\ timeout[d]\ Q)}\ [\ to3\ ]$$

$$\frac{(\sigma, P) \xrightarrow{\epsilon} (\sigma, P'), \epsilon \leq d}{(\sigma, P\ timeout[d]\ Q) \xrightarrow{\epsilon} (\sigma, P'\ timeout[d - \epsilon]\ Q)}\ [\ to4\ ]$$

– The semantics of $P\ timeout[d]\ Q$ is captured by rule $to1$ to $to4$. Rule $to1$ states that if an observable event e can be engaged by P, changing (σ, P) to (σ', P'), then $(\sigma, P\ timeout[d]\ Q)$ becomes (σ', P') so that Q is discharged. That is, P has performed an observable event before timeout occurs. Rule $to2$ states that if P instead performs a τ-transition, then Q and $timeout$ operator remain (since an observable event is yet to be performed). Rule $to3$ states that if P may idle for less than or equal to d time units, so does $P\ timeout[d]\ Q$. Rule $to4$ states that if d is 0, Q takes over control by a τ-transition.

$$\frac{(\sigma, P) \xrightarrow{a} (\sigma', P')}{(\sigma, P\ interrupt[d]\ Q) \xrightarrow{a} (\sigma', P'\ interrupt[d]\ Q)}\ [\ ti1\ ]$$

$$\frac{(\sigma, P) \xrightarrow{\epsilon} (\sigma, P'), \epsilon < d}{(\sigma, P\ interrupt[d]\ Q) \xrightarrow{e} (\sigma, P'\ interrupt[d - \epsilon]\ Q)}\ [\ ti2\ ]$$

$$\frac{}{(\sigma, P\ interrupt[0]\ Q) \xrightarrow{\tau} (\sigma, Q)}\ [\ ti3\ ]$$

– The semantics of $P\ interrupt[d]\ Q$ is defined by rule $ti1$ to $ti3$. Rule $ti1$ states that if event a (which may be observable or τ) can be engaged by P, changing (σ, P) to (σ', P'), then $(\sigma, P\ interrupt[d]\ Q)$ can perform a as well. In contrast to rule $to1$, the $interrupt$ operator remains. Intuitively, it states that before P is interrupted, P can do whatever it can. Rule $ti2$ states that if P may idle for less than or equal to d time units, so does $P\ interrupt[d]\ Q$. Rule $ti3$ states that if d is 0, Q takes over control by a τ-transition.

$$\frac{(\sigma, P) \xrightarrow{\tau} (\sigma', P')}{(\sigma, P\ within[d]) \xrightarrow{\tau} (\sigma', P\ within[d])}\ [\ wi2\ ]$$

$$\frac{(\sigma, P) \xrightarrow{e} (\sigma', P')}{(\sigma, P\ within[d]) \xrightarrow{e} (\sigma', P')}\ [\ wi1\ ]$$

$$\frac{(\sigma, P) \xrightarrow{\epsilon} (\sigma, P'), \epsilon < d}{(\sigma, P\ within[d]) \xrightarrow{\epsilon} (\sigma, P'\ within[d-\epsilon])}\ [\ wi3\ ]$$

– The semantics of $P\ within[d]$ is defined by rule $wi1$ to rule $wi3$. Rule $wi1$ states that if an observable event e occurs, then $within$ is discharged, as the requirement is fulfilled. In contrast, rule $wi2$ states that if instead event τ occurs, then $within$ remains. Rule $wi3$ state if P can idle for ϵ time units, then so does $P\ within[d]$ as long as $\epsilon \leq d$.

$$\frac{(\sigma, P) \xrightarrow{a} (\sigma', P')}{(\sigma, P\ deadline[d]) \xrightarrow{a} (\sigma', P'\ deadline[d])}\ [\ dl1\ ]$$

$$\frac{(\sigma, P) \xrightarrow{\checkmark} (\sigma', P')}{(\sigma, P\ deadline[d]) \xrightarrow{\checkmark} (\sigma', P')}\ [\ dl2\ ]$$

$$\frac{(\sigma, P) \xrightarrow{\epsilon} (\sigma, P'), \epsilon < d}{(\sigma, P\ deadline[d]) \xrightarrow{\epsilon} (\sigma, P'\ deadline[d-\epsilon])}\ [\ dl3\ ]$$

– Rule $dl1$, $dl2$ and $dl3$ define the semantics of $P\ deadline[d]$. Different from $P\ within[d]$, $P\ deadline[d]$ requires P to terminate (marked by a special event $\checkmark$) before d time units. Rule $dl1$ states that if P behaves as usual before the deadline is expired. Rule $dl2$ states that if P terminates, then $deadline$ is discharged. Rule $dl3$ state if P can idle for ϵ time units, so does $P\ deadline[d]$ as long as $\epsilon \leq d$.

After defining all the semantics of non-probabilistic operators, the following firing rules define the semantics of probabilistic choices.

$$\frac{(\sigma, P) \xrightarrow{x} (\sigma', P')}{(\sigma, P) \xrightarrow{x} \mu \text{ such that } \mu((\sigma', P')) = 1}\ [\ pb1\ ]$$

$$\frac{}{(\sigma, \textbf{pcase}\ \{pr_0 : P_0;\ \cdots;\ pr_k : P_k\}) \xrightarrow{\tau} \mu \text{ such that } \mu((\sigma, P_i)) = \frac{pr_i}{pr_0+pr_1+\cdots+pr_k} \text{ for all } i}\ [\ pb2\ ]$$

– $pb1$ indicates that if there is P could engage x and changes to P', then the probabilistic distribution on this transition is always trivial distribution. $pb2$ presents that **pcase** $\{pr_0 : P_0;\ \cdots;\ pr_k : P_k\}$ could take an invisible event immediately and keep the distribution.

4.2 Abstract Configuration

For many processes, especially real-time processes, having infinite states is a problem. PAT adopts *Zone Abstraction* to handle this issue. This technic is proved useful in model checking [6, 2, 4], and we amend it for more suitable for CSP#.

An abstract system configuration is a triple (σ, P_T, D), where σ is a variable valuation; P_T is a process associated with active clocks; and D is a zone. A zone D is the conjunction of multiple primitive constraints over a set of clocks and in stateful Timed CSP, amd clocks are implicitly associated with timed process, which means clocks are only activated when the process faces a timed transition.

Given a process P and a clock t, we defined function $\mathcal{A}$ to return the corresponding process in P_T. Figure 2 presents the detailed definition. Intuitively speaking, A1 to A5 state that if a process is un-timed and none of its sub-processes are activated, then it is unchanged. A6 to A10 states that if a process has been associated with clocks (i.e., it is in P_T), then function $\mathcal{A}$ is applied to its activated sub-processes only. Note that $Wait[d]_{t'}$ denotes that it has already been associated with clock t'. A11 to A14 state that if the process itself is un-timed, then function $\mathcal{A}$ is applied to its activated sub-processes. A16 to A19 state that if the process is timed, then it is associated with t and function $\mathcal{A}$ is applied to its activated sub-processes. Given a process P_T, we can obtain the set of clocks associated with P or any sub-process of P. The set, written as $cl(P_T)$, is referred to as the active clocks of P_T.

$$
\begin{array}{lll}
\mathcal{A}(\mathit{Stop}, t) & = \mathit{Stop} & \text{– A1} \\
\mathcal{A}(\mathit{Skip}, t) & = \mathit{Skip} & \text{– A2} \\
\mathcal{A}(e \rightarrow P, t) & = e \rightarrow P & \text{– A3} \\
\mathcal{A}(a\{\mathit{program}\} \rightarrow P, t) & = a\{\mathit{program}\} \rightarrow P & \text{– A4} \\
\mathcal{A}(\textbf{if } (b)\ \{P\} \textbf{ else } \{Q\}, t) & = \textbf{if } (b)\ \{P\} \textbf{ else } \{Q\} & \text{– A5} \\
\mathcal{A}(\mathit{Wait}[d]_{t'}, t) & = \mathit{Wait}[d]_{t'} & \text{– A6} \\
\mathcal{A}(P\ \mathit{timeout}[d]_{t'}\ Q, t) & = \mathcal{A}(P, t)\ \mathit{timeout}[d]_{t'}\ Q & \text{– A7} \\
\mathcal{A}(P\ \mathit{interrupt}[d]_{t'}\ Q, t) & = \mathcal{A}(P, t)\ \mathit{interrupt}[d]_{t'}\ Q & \text{– A8} \\
\mathcal{A}(P\ \mathit{within}[d]_{t'}, t) & = \mathcal{A}(P, t)\ \mathit{within}[d]_{t'} & \text{– A9} \\
\mathcal{A}(P\ \mathit{deadline}[d]_{t'}, t) & = \mathcal{A}(P, t)\ \mathit{deadline}[d]_{t'} & \text{– A10} \\
\mathcal{A}(P \mid Q, t) & = \mathcal{A}(P, t) \mid \mathcal{A}(Q, t) & \text{– A11} \\
\mathcal{A}(P \setminus X, t) & = \mathcal{A}(P, t) \setminus X & \text{– A12} \\
\mathcal{A}(P;\ Q, t) & = \mathcal{A}(P, t);\ Q & \text{– A13} \\
\mathcal{A}(P \parallel Q, t) & = \mathcal{A}(P, t) \parallel \mathcal{A}(Q, t) & \text{– A14} \\
\mathcal{A}(\mathit{Wait}[d], t) & = \mathit{Wait}[d]_t & \text{– A15} \\
\mathcal{A}(P\ \mathit{timeout}[d]\ Q, t) & = \mathcal{A}(P, t)\ \mathit{timeout}[d]_t\ \mathcal{A}(Q, t) & \text{– A16} \\
\mathcal{A}(P\ \mathit{interrupt}[d]\ Q, t) & = \mathcal{A}(P, t)\ \mathit{interrupt}[d]_t\ \mathcal{A}(Q, t) & \text{– A17} \\
\mathcal{A}(P\ \mathit{within}[d]\ Q, t) & = \mathcal{A}(P, t)\ \mathit{within}[d]_t\ \mathcal{A}(Q, t) & \text{– A18} \\
\mathcal{A}(P\ \mathit{deadline}[d]\ Q, t) & = \mathcal{A}(P, t)\ \mathit{deadline}[d]_t\ \mathcal{A}(Q, t) & \text{– A19} \\
\mathcal{A}(P, t) & = \mathcal{A}(Q, t) \quad \text{if } P \mathrel{\widehat{=}} Q & \text{– A20}
\end{array}
$$

Fig. 2. Clock activation

What is more, we define a function $idle$ which, given a process, calculates how long this process could idle. The result is in the form of a predicate over the clocks, i.e., a zone. Figure 3 shows the detailed definition. Rule $idle1$ to $idle5$ state that if the process is un-timed and none of its sub-processes is activated, then the function returns true. It means that the process may idle for arbitrary amount of time. Rule $idle6$ to $rule9$ state that if sub-processes of the process are activated, then function $idle$ is applied to the sub-processes. In particular, if the process is a choice (rule $idle6$) or a parallel composition (rule $idle9$) of P and Q, then the result is $idle(P) \wedge idle(Q)$. Intuitively, this means that process $P \mid Q$ (or $P \parallel Q$) may idle as long as both P and Q can idle. Rule $idle10$ to $idle14$ define the cases when the process is timed. For instance, process $Wait[d]_t$ may idle as long as t is less or equal to d.

$idle(Stop)$	$= true$	– rule $idle1$
$idle(Skip)$	$= true$	– rule $idle2$
$idle(e \rightarrow P)$	$= true$	– rule $idle3$
$idle(a\{program\} \rightarrow P)$	$= true$	– rule $idle4$
$idle(\textbf{if}\ (b)\ \{P\}\ \textbf{else}\ \{Q\})$	$= true$	– rule $idle5$
$idle(P \mid Q)$	$= idle(P) \wedge idle(Q)$	– rule $idle6$
$idle(P \setminus X)$	$= idle(P)$	– rule $idle7$
$idle(P;\ Q)$	$= idle(P);\ Q$	– rule $idle8$
$idle(P \parallel Q)$	$= idle(P) \wedge idle(Q)$	– rule $idle9$
$idle(Wait[d]_t)$	$= t \leq d$	– rule $idle10$
$idle(P\ timeout[d]_t\ Q)$	$= t \leq d \wedge idle(P)$	– rule $idle11$
$idle(P\ interrupt[d]_t\ Q)$	$= t \leq d \wedge idle(P)$	– rule $idle12$
$idle(P\ within[d]_t\ Q)$	$= t \leq d \wedge idle(P)$	– rule $idle13$
$idle(P\ deadline[d]_t\ Q)$	$= t \leq d \wedge idle(P)$	– rule $idle14$
$idle(P)$	$= idle(Q)$ if $P \mathrel{\widehat{=}} Q$	– rule $idle15$

Fig. 3. Idling calculation

In order to systematically apply zone abstraction, we define a set of *abstract* firing rules. The abstract firing rules eliminate concrete ϵ-transitions all together and use zones to ensure a process behaves correctly with respect to timing requirements. To distinguish from concrete firing rules, an abstract firing rule is written in the the form of $(\sigma, P, D) \overset{x}{\rightsquigarrow} (\sigma', P', D')$ where $x \in \Sigma_\tau \cup \mathbb{R}_+$. Following presents the abstract firing rules for the timed processes. Note that $D^\uparrow$ denotes the zone obtained by delaying arbitrary amount of time.

$$\frac{}{(\sigma, Skip, D) \overset{\checkmark}{\rightsquigarrow} (\sigma, Stop, D^\uparrow)}\ [\ aki\ ]$$

$$\frac{}{(\sigma, e \rightarrow P, D) \overset{e}{\rightsquigarrow} (\sigma, P, D^\uparrow)}\ [\ aev\ ]$$

$$\frac{}{(\sigma, a\{pr\} \to P, D) \overset{a}{\rightsquigarrow} (pr(\sigma), P, D^{\uparrow})} [\ aac\ ]$$

$$\frac{\sigma \vDash b}{(\sigma, \textbf{if } b \textbf{ then } \{P\} \textbf{ else } \{Q\}, D) \overset{\tau}{\rightsquigarrow} (\sigma, P, D^{\uparrow})} [\ co2\ ]$$

$$\frac{\sigma \nvDash b}{(\sigma, \textbf{if } b \textbf{ then } \{P\} \textbf{ else } \{Q\}, D) \overset{\tau}{\rightsquigarrow} (\sigma, Q, D^{\uparrow})} [\ co3\ ]$$

$$\frac{(\sigma, P, D) \overset{a}{\rightsquigarrow} (\sigma', P', D')}{(\sigma, P \mid Q, D) \overset{a}{\rightsquigarrow} (\sigma', P', D' \wedge idle(Q))} [\ aex1\ ]$$

$$\frac{(\sigma, Q, D) \overset{a}{\rightsquigarrow} (\sigma', Q', D')}{(\sigma, P \mid Q, D) \overset{a}{\rightsquigarrow} (\sigma', Q', D' \wedge idle(P))} [\ aex2\ ]$$

$$\frac{(\sigma, P, D) \overset{a}{\rightsquigarrow} (\sigma', Q', D'), init(\sigma, P, D) \cap X = \varnothing}{(\sigma, P \setminus X, D) \overset{a}{\rightsquigarrow} (\sigma', Q', D')} [\ ahi1\ ]$$

$$\frac{(\sigma, P, D) \overset{a}{\rightsquigarrow} (\sigma', Q', D'), init(\sigma, P, D) \cap X \neq \varnothing, a \notin X}{(\sigma, P \setminus X, D) \overset{a}{\rightsquigarrow} (\sigma', Q', D' \wedge D)} [\ ahi2\ ]$$

$$\frac{(\sigma, P, D) \overset{a}{\rightsquigarrow} (\sigma', Q', D'), init(\sigma, P, D) \cap X \neq \varnothing, a \in X}{(\sigma, P \setminus X, D) \overset{\tau}{\rightsquigarrow} (\sigma', Q', D' \wedge D)} [\ ahi3\ ]$$

$$\frac{(\sigma, P, D) \overset{a}{\rightsquigarrow} (\sigma', P', D'), \checkmark \notin init(\sigma, P, D)}{(\sigma, P;\ Q, D) \overset{a}{\rightsquigarrow} (\sigma', P';\ Q, D')} [\ ase1\ ]$$

$$\frac{(\sigma, P, D) \overset{\checkmark}{\rightsquigarrow} (\sigma', P', D')}{(\sigma, P;\ Q, D) \overset{\tau}{\rightsquigarrow} (\sigma, Q, D \wedge D')} [\ ase2\ ]$$

$$\frac{(\sigma, P, D) \overset{a}{\rightsquigarrow} (\sigma', P', D'), a \notin X}{(\sigma, P \,|[\, X \,]|\, Q, D) \overset{a}{\rightsquigarrow} (\sigma', P' \,|[\, X \,]|\, Q, D' \wedge idle(Q))} [\ apa1\ ]$$

$$\frac{(\sigma, Q, D) \overset{a}{\rightsquigarrow} (\sigma', Q', D'), a \notin X}{(\sigma, P \,|[\, X \,]|\, Q, D) \overset{a}{\rightsquigarrow} (\sigma', P \,|[\, X \,]|\, Q', D' \wedge idle(P))} [\ apa2\ ]$$

$$\frac{(\sigma, P, D) \overset{e}{\rightsquigarrow} (\sigma, P', D'), (\sigma, Q, D) \overset{e}{\rightsquigarrow} (\sigma, Q', D''), e \in X}{(\sigma, P \,|[\, X \,]|\, Q, D) \overset{e}{\rightsquigarrow} (\sigma, P' \,|[\, X \,]|\, Q', D' \wedge D'')} [\ apa3\ ]$$

$$\frac{(\sigma, Q, D) \overset{a}{\rightsquigarrow} (\sigma', Q', D'), P \mathrel{\widehat{=}} Q}{(\sigma, P, D) \overset{a}{\rightsquigarrow} (\sigma', Q', D')} [\ ade\ ]$$

– Rules above are almost the same as the relative concrete firing rules and $zone$ transformations are quite straightforward since no timed process construct is included. We just take some rules for example. aki indicates $Skip$ could engage the terminating event after delaying arbitrary time units and then changes to $Stop$. $aex1$ defines the abstract semantics of $P \mid Q$. If P could execute a and (σ, P, D) becomes (σ', P', D'), so does $P \mid Q$. Of course this execution should be constrained by the maximum idle time that Q could wait. $ahi3$ defines the abstract semantics of $P \setminus X$. It means if $a \in X$ is enabled, this transition should happen immediately(D keeps unchanged) and meanwhile, clocks should be under the constraint of D'.

$$\frac{}{(\sigma, Wait[d]_c, D) \overset{\tau}{\rightsquigarrow} (\sigma, Skip, D^{\uparrow} \wedge t = d)} [\ await\ ]$$

– Rule $await$ defines the abstract semantics of $Wait[d]$. In contrast to the concrete firing rules, there is only one abstract rule. It states that a τ-transition occurs exactly when clock c reads d. Intuitively, $D^{\uparrow} \wedge c = d$ denotes the exact moment of d time units elapsed since c starts. Afterwards, the process becomes $Skip$.

$$\frac{(\sigma, P, D) \overset{\tau}{\rightsquigarrow} (\sigma', P', D')}{(\sigma, P\ timeout[d]_c\ Q, D) \overset{\tau}{\rightsquigarrow} (\sigma', P'\ timeout[d]_c\ Q, D' \wedge c \leq d)} [\ ato1\ ]$$

$$\frac{(\sigma, P, D) \overset{e}{\rightsquigarrow} (\sigma', P', D')}{(\sigma, P\ timeout[d]_c\ Q, D) \overset{e}{\rightsquigarrow} (\sigma', P', D' \wedge c \leq d)} [\ ato2\ ]$$

$$\frac{}{(\sigma, P\ timeout[d]_c\ Q, D) \overset{\tau}{\rightsquigarrow} (\sigma, Q, c = d \wedge idle(P))} [\ ato3\ ]$$

– Rule $ato1$, $ato2$ and $ato3$ define the abstract semantics of $P\ timeout[d]\ Q$. Rule $ato1$ states that if a τ-transition transforms (σ, P, D) to (σ', P', D'), then a τ-transition may occur given $(\sigma, P\ timeout[d]_c\ Q, D)$ if zone $D' \wedge c \leq d$ is not empty. Intuitively, this means that the τ-transition must occur before d time units since c starts. Similarly, rule $ato2$ ensures that the occurrence of an observable event e from process P may occur only if $c \leq d$, i.e., before timeout occurs. Rule $ato3$ states that timeout results in a τ-transition when the reading of c is exactly d. The constraint $c = d \wedge idle(P)$ ensures that process P may idle all the way until timeout occurs.

$$\frac{(\sigma, P, D) \overset{a}{\rightsquigarrow} (\sigma', P', D')}{(\sigma, P\ interrupt[d]_c\ Q, D) \overset{a}{\rightsquigarrow} (\sigma', P'\ interrupt[d]_c\ Q, D' \wedge c \leq d)}\ [\ ait1\ ]$$

$$\frac{}{(\sigma, P\ interrupt[d]_c\ Q, D) \overset{\tau}{\rightsquigarrow} (\sigma, Q, c = d \wedge idle(P))}\ [\ ait2\ ]$$

– Rule $ait1$ and $ait2$ define the abstract semantics of $P\ interrupt[d]\ Q$. Rule $ait1$ states that a transition originated from P may occur only if $c \leq d$, i.e., before interrupt occurs. Rule $ait2$ states that interrupt results in a τ-transition when the reading of c is exactly d. The constraint $c = d \wedge idle(P)$ ensures that process P may idle until interrupt occurs.

$$\frac{(\sigma, P, D) \overset{\tau}{\rightsquigarrow} (\sigma', P', D')}{(\sigma, P\ within[d]_c, D) \overset{\tau}{\rightsquigarrow} (\sigma', P'\ within[d]_c, D' \wedge c \leq d)}\ [\ awi1\ ]$$

$$\frac{(\sigma, P, D) \overset{e}{\rightsquigarrow} (\sigma', P', D')}{(\sigma, P\ within[d]_c, D) \overset{e}{\rightsquigarrow} (\sigma', P', D' \wedge c \leq d)}\ [\ awi2\ ]$$

– Rule $awi1$ and $awi2$ define the abstract semantics of $P\ within[d]$. Rule $awi1$ states that if a τ-transition occurs within d time units, then the resultant process is of the form $P'\ within[d]$, which means that it remains to perform some observable event before d time units. Rule $awi2$ states that once an observable event occurs, the $within$ construct is removed.

$$\frac{(\sigma, P, D) \overset{a}{\rightsquigarrow} (\sigma', P', D'), a \neq \checkmark}{(\sigma, P\ deadline[d]_c, D) \overset{e}{\rightsquigarrow} (\sigma', P'\ deadline[d]_c, D' \wedge c \leq d)}\ [\ adl1\ ]$$

$$\frac{(\sigma, P, D) \overset{\checkmark}{\rightsquigarrow} (\sigma', P', D')}{(\sigma, P\ deadline[d]_c, D) \overset{\checkmark}{\rightsquigarrow} (\sigma', P', D' \wedge c \leq d)}\ [\ adl2\ ]$$

– Rule $adl1$ and $adl2$ define the abstract semantics of $P\ deadline[d]$. Rule $adl1$ constraints that all transitions of P must occur within d time units. Rule $adl2$ states that if P terminates (by engaging in $\checkmark$), then $dealine$ is removed.

$$\frac{(\sigma, P, D) \overset{x}{\rightsquigarrow} (\sigma', P', D')}{(\sigma, P, D) \overset{x}{\rightsquigarrow} \mu \text{ such that } \mu((\sigma', P', D')) = 1}\ [\ apb1\ ]$$

$$\frac{}{(\sigma, \textbf{pcase}\ \{pr_0 : P_0;\ \cdots;\ pr_k : P_k\}, D) \overset{\tau}{\rightsquigarrow} \mu \text{ such that } \mu(\sigma, P_i, D) = \frac{pr_i}{pr_0 + pr_1 + \cdots + pr_k} \text{ for all } i}\ [\ apb2\ ]$$

– $apb1$ indicates that if there is P could engage x and changes to P', then the probabilistic distribution on this transition is always trivial distribution. $apb2$ presents that **pcase** $\{pr_0 : P_0; \cdots; pr_k : P_k\}$ should take an invisible event immediately(since D keeps unchanged) and keep the distribution.

5 Model Checking PRTS

In this part, we give the specific algorithms we use in PRTS system. If $M = (Var, \sigma_i, P)$ be a model, then $\mathcal{D}_M$ is defined as the concrete MDP transferred from M and $\mathcal{D}^a_M$ is defined as the abstract MDP transferred from M. In the following we will show that after abstraction, $\mathcal{D}^a_M$ has finite states so that it is suitable for model checking method; what is more, $\mathcal{D}^a_M$ is shown to be equivalent to the concrete semantics $\mathcal{D}_M$ so the verification results based on $\mathcal{D}^a_M$, no matter LTL checking or refinement checking, apply to $\mathcal{D}_M$.

Theorem 1. *$\mathcal{D}^a_M$ is finite for any PRTS model M.* □

Proof By definition, the size of $\mathcal{D}^a_M$ is bounded by $\#\sigma \times \#P \times \#D$ where $\#\sigma$ denotes the number of variable valuations; $\#P$ denotes the number of processes; and $\#D$ denotes the number of zones. By assumption in section 2, all variables have finite domains and therefore $\#\sigma$ is finite. Similarly, all reachable processes are finite. Thus, $\#P$ is finite if and only if values for parameters of the timed processes are finite and the number of clocks in $cl(P_T)$ is finite. It can be shown that all abstract firing rules preserve the parameters and therefore values for parameters is finite. By assumption, there can be only finitely many process constructs in any reachable process. Thus, $cl(P_T)$ is always finite. Therefore, $\#P$ is finite. Lastly, it is known that with zone normalization, the number of zones is finite given finitely many clocks and only integer constants [6]. As a result, we conclude that $\#D$ is finite and then $\mathcal{D}^a_M$ is obviously finite. □

5.1 Probability Preserving

Different from zone abstraction used in Probabilistic Timed Automata (PTA), we prove that our automatic zone abstraction approach is probability preserving for several properties such as reachability checking and LTL-X checking.

First of all, we define *probabilistic time-abstract bi-simulation* of DTMC as follows.

Definition 2. *A* probabilistic time-abstract bi-simulation relation *between a DTMC $\mathcal{C} = (S_c, init_c, Act, Pr_c)$ and an abstract DTMC $\mathcal{C}_a = (S_a, init_a, Act, Pr_a)$ is a relation $\mathcal{R} \subseteq S_c \times S_a$ satisfying the following condition.*

C1: If $(s_c, s_a) \in \mathcal{R}$, then s_c and s_a have the same variable valuation.
C2: If $(s_c, s_a) \in \mathcal{R}$ and $(s_c, (\epsilon, e, p), s'_c) \in Pr_c$ for some $\epsilon \geq 0$, $e \in Act_\tau$ and $p \in [0, 1]$, then there exists s'_a such that $(s_a, (e, p), s'_a) \in Pr_a$ and $(s'_c, s'_a) \in \mathcal{R}$;
C3: If $(s_c, s_a) \in \mathcal{R}$ and $(s_a, (e, p), s'_a) \in Pr_a$ for some $e \in Act_\tau$ and $p \in [0, 1]$, then there exists some $\epsilon \geq 0$ and s'_c such that $(s_c, (\epsilon, e, p), s'_c) \in Pr_c$ and $(s'_c, s'_a) \in \mathcal{R}$;

$\mathcal{C}$ and $\mathcal{C}_a$ are called bisimilar, or $\mathcal{C} \approx \mathcal{C}_a$, if they have the bi-simulation relation $\mathcal{R}$. Obviously a specified LTL-X formula has the same probability in two DTMCs which are bisimilar.

Now, given an MDP $\mathcal{D}$ and an LTL-X formula ϕ, let $\mathcal{P}^{max}_{\mathcal{D}}(\phi)$ be the maximum probability of $\mathcal{D}$ satisfying ϕ. Similarly, let $\mathcal{P}^{min}_{\mathcal{D}}(\phi)$ be the minimum probability of $\mathcal{D}$ satisfying ϕ. The following theorem states that it is sound and complete to model-check LTL-X properties against $\mathcal{D}^a_M$.

Theorem 2. *Let M be a PRTS model. $\mathcal{P}^{max}_{\mathcal{D}^a_M}(\phi) = \mathcal{P}^{max}_{\mathcal{D}_M}(\phi)$ and $\mathcal{P}^{min}_{\mathcal{D}^a_M}(\phi) = \mathcal{P}^{min}_{\mathcal{D}_M}(\phi)$.* □

Proof Obviously, this theorem can be proven if 1) for each scheduler σ in $\mathcal{D}_M$, we could find a scheduler η in $\mathcal{D}^a_M$ such that $\mathcal{D}^\sigma_M \approx (\mathcal{D}^a_M)^\eta$; 2) vice versa.

1) Given $\mathcal{D}^\sigma_M$, we first prove that a sequential timed transitions can be treated as one timed transition in verifying LTL-X. In other words, if we have $s_1 \xrightarrow{\epsilon 1} s_2 \xrightarrow{\epsilon 2} s_3$ as a part of a path in $\mathcal{D}^\sigma_M$, one transition $s_1 \xrightarrow{\epsilon 1+\epsilon 2} s_3$ is equivalent to the previous sequence. This is trivial since timed transitions always have probability 1 and do not affect the variable valuation in our setting. So that the probability of the LTL-X property on this path keeps the same. Therefore, here we just consider $\mathcal{D}^\sigma_M$ in which no successive timed transitions exist.

we can build η in $\mathcal{D}^a_M$ as follows. Obviously, the initial states in $\mathcal{D}_M$ and $\mathcal{D}^a_M$ satisfy $C1$. From $init_c$, there is one distribution chosen from the following three kinds of distributions:

$\mathcal{A}$: timed transition ϵ with probability 1 to s_c; that is $init_c \xrightarrow{\epsilon} s_c$.
$\mathcal{B}$: $e \in Act_\tau$ with probability 1 to s'_c; that is $init_c \xrightarrow{e} s'_c$.
$\mathcal{C}$: $pcase$ specifying a real probability distribution to a set of states.

If case $\mathcal{A}$ is chosen, since no successive timed transitions allowed, there must exist $s_c \xrightarrow{e} s'_c$ in $\mathcal{D}^\sigma_M$, where e could be real action in the model, or τ indicating some timed constructors' expiration. $pcase$ is also not allowed from s_c because in our setting $pcase$ must happen immediately when it is enabled. Since timed transition is allowed at $init_c$, $pcase$ is not allowed in the following unless some timed constructors expire, which means a τ transition must happen before $pcase$. Therefore, we have $init_c \xrightarrow{\epsilon} s_c \xrightarrow{e} s'_c$ in $\mathcal{D}^\sigma_M$. In this case, it is trivial to show e is enabled at $init_a$, otherwise e cannot happen after a timed transition in $\mathcal{D}^\sigma_M$ without some timed constructors expiration. So η chooses e at $init_a$ and we assume the successive state is s'_a. Then we have $(init_c, (\epsilon, e, 1), s'_c) \in Pr_c$ in $\mathcal{D}^\sigma_M$ and $(init_a, (e, 1), s'_a) \in Pr_a$ in $(\mathcal{D}^a_M)^\eta$. Therefore $init_c$ and $init_a$ satisfy $C2$ and $C3$ and as a conclusion $(init_c, init_a) \in \mathcal{R}$. Since timed transition does not change the value of variables and e affects the variable equally in both models, we have s'_c and s'_a satisfy $C1$.

If case $\mathcal{B}$ is chosen, it is obvious that e is enabled at both $init_c$ and $init_a$. Let η choose e in $init_a$, so that in $\mathcal{D}^\sigma_M$ we have $(init_c, (0, e, 1), s'_c)$ and in $(\mathcal{D}^a_M)^\eta$ we have $(init_a, (e, 1), s'_a)$. Thus $init_c$ and $init_a$ satisfy $C2$ and $C3$, which indicates $(init_c, init_a) \in \mathcal{R}$. Since e affects the variable equally in both models, we have s'_c and s'_a satisfy $C1$.

If case $\mathcal{C}$ is chosen, then this $pcase$ is also enabled at $init_a$. Let η choose this $pcase$. Assume in $\mathcal{D}^\sigma_M$ we have $(init_c, (0, \tau, p), s'_c)$, then obviously we have $(init_a, (\tau, p), s'_a)$

in $(\mathcal{D}_M^a)^\eta$. Thus $init_c$ and $init_a$ satisfy $C2$ and $C3$, which indicates $(init_c, init_a) \in \mathcal{R}$. Since probabilistic choice does not affect the value of variables, we have s_c' and s_a' satisfy $C1$.

In summary, no matter which case is chosen in $init_c$, scheduler η in $\mathcal{D}_M^a$ can choose a corresponding distribution from $init_a$, such that $(init_c, init_a) \in \mathcal{R}$ and their successive states s_c' and s_a' satisfy $C1$. Step by step, we could build η which guarantees $\mathcal{D}_M^\sigma \approx (\mathcal{D}_M^a)^\eta$ for arbitrary $\mathcal{D}_M^\sigma$. Therefore 1) is true.

2) Given $(\mathcal{D}_M^a)^\eta$, we can build σ in $\mathcal{D}_M$ as follows. Obviously $init_a$ and $init_c$ satisfy $C1$. From $init_a$, there is one distribution chosen from the following two kinds of distributions.

$\mathcal{A}$: $e \in Act_\tau$ with probability 1 to s_a'; that is $init_a \overset{e}{\rightsquigarrow} s_a'$.
$\mathcal{B}$: $pcase$ specifying a real probability distribution to a set of states.

If case $\mathcal{A}$ is chosen, there are two possibilities.

(I) e is τ transition which is generated because of some timed constructors expiration, e.g. rule $await$. If this is true, it means some timed constructors are enabled at $init_a$, and also $init_c$. Let d be the smallest parameter in those activated timed constructors, e.g. $d = 2$ in $(Wait[2];\ a \rightarrow Skip)within[3]$. Then let σ first choose timed transition d from $init_c$, that is $init_c \overset{d}{\rightarrow} s_c$, and second choose τ in s_c, that is $s_c \overset{\tau}{\rightarrow} s_c'$, indicating the expiration of those activated timed constructors having parameter d. Thus in $(\mathcal{D}_M^a)^\eta$ we have $(init_a, (\tau, 1), s_a') \in Pr_a$ and in $\mathcal{D}_M^\sigma$ we have $(init_c, (d, \tau, 1), s_c') \in Pr_c$. It is easy to show $(init_a, init_c) \in \mathcal{R}$ and s_a' and s_c' satisfy $C1$.

(II) e is an ordinary action in $\mathcal{D}_M^a$. In this case, e must also be enabled in $init_c$. So let σ choose e, and we have $(init_a, (e, 1), s_a') \in Pr_a$ in $(\mathcal{D}_M^a)^\eta$ and $(init_c, (0, e, 1), s_c') \in Pr_c$ in $\mathcal{D}_M^\sigma$. It is also obvious that $(init_a, init_c) \in \mathcal{R}$ and s_a' and s_c' satisfy $C1$.

If case $\mathcal{B}$ is chosen, similar to proof in 1), we could get $(init_a, init_c) \in \mathcal{R}$, and s_a' and s_c' satisfy $C1$ if they are the successive states from $init_a$ and $init_c$ respectively following the same probabilistic choice.

In summary, no matter which case is chosen in $init_a$, scheduler σ in $\mathcal{D}_M$ can choose a corresponding distribution from $init_c$, such that $(init_a, init_c) \in \mathcal{R}$ and their successive states s_a' and s_c' satisfy $C1$. Step by step, we could build σ which guarantees $\mathcal{D}_M^\sigma \approx (\mathcal{D}_M^a)^\eta$ for arbitrary $(\mathcal{D}_M^a)^\eta$. Therefore 2) is true.

As a conclusion, all DTMC from $\mathcal{D}_M$ can be covered by $\mathcal{D}_M^a$ and vice versa, so that this theorem is true.

□

5.2 Refinement Checking

Given an MDP D and a (non-probabilistic) MDP E, let $\mathcal{P}^{max}(D \sqsupseteq E)$ denote the maximum probability of D refining E in un-timed trace semantics. Formally, it is defined as follows.

$$\sup_\delta \mathcal{P}_D(\{\pi \in run(D^\delta) \mid trace(\pi) \in trace(E)\})$$

Intuitively, it is the maximum probability of exhibiting any trace of E by D. Similarly, we define $\mathcal{P}^{min}(D \sqsupseteq E)$ to be the minimum probability. The following theorem states that it is sound and complete to refine-check the abstract models.

Theorem 3. *Let M be a model and N be a non-probabilistic model. $\mathcal{P}^{max}(D_M \sqsupseteq D_N) = \mathcal{P}^{max}(D_M^a \sqsupseteq D_N^a)$ and $\mathcal{P}^{min}(D_M \sqsupseteq D_N) = \mathcal{P}^{min}(D_M^a \sqsupseteq D_N^a)$.* □

Proof Based on theorem 2, for each trace ex in D_M, there is a trace ex' in D_M^a which is stutter equivalent to ex, and vice versa. Since we just consider non-timed trace refinement, we could conclude $trace(ex) = trace(ex')$; what is more, ex and ex' have the same probability. Similarly, we could get D_N and D_N^a are also trace equivalent. So the theorem holds obviously. □

For refinement checking between a finite MDP(the abstract model) and a finite non-probabilistic MDP(the property), we could simply it to probability reachability analysis, which is based on the following theorem.

Theorem 4. *Let $\mathcal{M}$ be an MDP; $\mathcal{N}$ be an non-probabilistic MDP; $\mathcal{D} = \mathcal{M} \times de(\mathcal{N})$ in which $de(\mathcal{N})$ means the deterministic MDP got from $\mathcal{N}$ using standard superset method. If $G \subseteq \mathcal{D}$ and for every pair $(s, s') \in G$, $s' = \phi$, then $\mathcal{P}^{max}(\mathcal{M} \sqsupseteq \mathcal{N}) = 1 - \mathcal{P}_{\mathcal{D}}^{min}(G)$ and $\mathcal{P}^{min}(\mathcal{M} \sqsupseteq \mathcal{N}) = 1 - \mathcal{P}_{\mathcal{D}}^{max}(G)$.*

Proof Let δ be any scheduler for $\mathcal{M}$. Note that δ can be extended to be a scheduler for $\mathcal{D}$ straightforwardly. For simplicity, we use δ to denote both of them. Let $X \subseteq paths(\mathcal{M})$. The following shows that the equivalence holds with any scheduler.

$$
\begin{array}{ll}
\mathcal{P}_M^{\delta}(\{\pi \in paths(\mathcal{M}) \mid trace(\pi) \in traces(\mathcal{N})\}) & \\
\equiv 1 - \mathcal{P}_M^{\delta}(\{\pi \in paths(\mathcal{M}) \mid trace(\pi) \notin traces(\mathcal{N})\}) & \text{– by def.} \\
\equiv 1 - \mathcal{P}_{\mathcal{D}}^{\delta}(\{\pi \in paths(\mathcal{D}) \mid trace(\pi) \notin traces(\mathcal{N})\}) & \text{– (1)} \\
\equiv 1 - \mathcal{P}_{\mathcal{D}}^{\delta}(G) & \text{– (2)}
\end{array}
$$

(1) is true because for every path of $\mathcal{M}$, there is a path of $\mathcal{D}$ with the same probability (as $\mathcal{N}$ is non-probabilistic) and the same trace; and vice versa. (2) is true because by [10], a path of $\mathcal{D}$ such that $trace(\pi) \notin traces(\mathcal{N})$ if and only if it visits some state in G. It can be shown then $\mathcal{P}^{max}(\mathcal{M} \sqsupseteq \mathcal{N})$, which is $\mathcal{P}_M^{max}(\{\pi \in paths(\mathcal{M}) \mid trace(\pi) \in traces(\mathcal{N})\})$, is $1 - \mathcal{P}_{\mathcal{D}}^{min}(G)$ and $\mathcal{P}^{min}(\mathcal{M} \sqsupseteq \mathcal{N})$ is $1 - \mathcal{P}_{\mathcal{D}}^{max}(G)$. □

Intuitively, the theorem holds because, with any scheduler, the probability of $\mathcal{M}$ *not* refining $\mathcal{L}$ is exactly the probability of reaching G in $\mathcal{M} \times de(\mathcal{N})$. As a result, refinement checking is reduced to reachability probability in $\mathcal{D}$. There are known approaches to compute $\mathcal{P}_{\mathcal{M}}^{max}(G)$ and $\mathcal{P}_{\mathcal{M}}^{min}(G)$, e.g., using an iterative approximation method or by solving linear programs [1].

References

1. C. Baier and J. Katoen. *Principles of Model Checking*. The MIT Press, 2008.

2. G. Behrmann, K. G. Larsen, J. Pearson, C. Weise, and W. Yi. Efficient Timed Reachability Analysis Using Clock Difference Diagrams. In *CAV*, volume 1633 of *LNCS*, pages 341–353. Springer, 1999.
3. R. Bellman. A Markovian Decision Process. *Journal of Mathematics of Mechanics*, 6, 1957.
4. J. Bengtsson and W. Yi. Timed Automata: Semantics, Algorithms and Tools. In *Lectures on Concurrency and Petri Nets*, volume 3098 of *LNCS*, pages 87–124. Springer, 2003.
5. J. Davies. *Specification and Proof in Real-Time CSP*. Cambridge University Press, 1993.
6. D. L. Dill. Timing Assumptions and Verification of Finite-State Concurrent Systems. In *Automatic Verification Methods for Finite State Systems*, volume 407 of *LNCS*, pages 197–212. Springer, 1989.
7. C. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.
8. C. Morgan, A. McIver, K. Seidel, and J. W. Sanders. Refinement-Oriented Probability for CSP. *Formal Asp. Comput.*, 8(6):617–647, 1996.
9. J. Ouaknine and J. Worrell. Timed CSP = Closed Timed Safety Automata. *Electrical Notes Theoretical Computer Science*, 68(2), 2002.
10. A. W. Roscoe. Model-checking CSP. pages 353–378, 1994.
11. J. Sun, Y. Liu, J. S. Dong, and C. Q. Chen. Integrating Specification and Programs for System Modeling and Verification. In *TASE*, pages 127–135. IEEE Computer Society, 2009.