Modal Specifications for Probabilistic Timed Systems

Tingting Han[1] Christian Krause[2] Marta Kwiatkowska[1] Holger Giese[2]

1. Department of Computer Science,
University of Oxford
{firstname.lastname}@cs.ox.ac.uk

2. Hasso Plattner Institute,
University of Potsdam
{firstname.lastname}@hpi.uni-potsdam.de

Modal automata are a classic formal model for component-based systems that comes equipped with a rich specification theory supporting abstraction, refinement and compositional reasoning. In recent years, quantitative variants of modal automata were introduced for specifying and reasoning about component-based designs for embedded and mobile systems. These respectively generalize modal specification theories for timed and probabilistic systems. In this paper, we define a modal specification language for combined probabilistic timed systems, called *abstract probabilistic timed automata*, which generalizes existing formalisms. We introduce appropriate syntactic and semantic refinement notions and discuss consistency of our specification language, also with respect to time-divergence. We identify a subclass of our models for which we define the fundamental operations for abstraction, conjunction and parallel composition, and show several compositionality results.

1 Introduction

The design of complex embedded systems can be supported by component-based design methodologies, which can take the form of specification theories that provide the notions of abstraction and refinement, as well as a rich collection of compositional operators. A classical and widely used specification theory for component-based design is that of modal automata [16]. A modal automaton is essentially a deterministic automaton equipped with *may*- and *must*-transitions, which are respectively used to specify allowed and required behavior. Several technical aspects of modal automata have been studied in the literature, including modal vs. thorough refinement, consistency, abstraction, as well as operators for parallel composition of components and conjunction, where the latter supports independent development. These notions enjoy a number of important properties, e.g., that conjunction is the greatest lower bound w.r.t. modal refinement and that abstraction is compositional. In this way, modal automata provide mathematical foundations for designing and reasoning about component-based systems at the abstract level of interfaces and to derive properties on the implementation level of the global system.

In recent years, much attention has been dedicated to formulating quantitative extensions of modal automata, for example to support the development of component-based systems which feature real-time and/or probabilistic behavior. An example of these developments are a modal specification language for timed systems called *Modal Event-Clock Specifications* (MECS) [4]. MECS are essentially a modal extension of *Event-Clock Automata* (ECAs) [2], which form a strict subclass of the classical *Timed Automata* [1] model. Restricting to this model allows Bertrand et al. in [4] to lift a number of compositionality properties known for modal automata to the timed setting, i.e., to the model of MECS. Another recent quantitative variant are *Abstract Probabilistic Automata* [11] (APAs). While MECS are used to specify timed behavior, APAs enable the specification of abstract probabilistic behavior using probability constraints. Probabilistic behavior is commonly required for quantifying the likelihood of events, such as message loss in unreliable channels, or is exploited in the design of randomized protocols. Similarly to MECS, APAs are equipped with notions for conjunction and parallel composition, as well as a number of compositionality results that can be used for compositional reasoning and abstraction. However, in many settings, such as in embedded and mobile systems, a combination of probabilistic and timed behavior is required, which is not supported by APAs and MECS. Although the specification of combined probabilistic and timed behavior is possible with *Probabilistic Timed Automata* [14] (PTAs), there are no corresponding notions of modalities and abstract probabilistic behavior for PTAs.

In this paper, we introduce a modal specification language for probabilistic timed systems, called *Abstract Probabilistic Timed Automata* (APTAs), and a subclass of them, called *Abstract Probabilistic Event-Clock Automata* (APECAs). APTAs serve as a modal specification language for systems with nondeterministic, probabilistic and timed behavior and support the abstract definition of underspecified probabilistic behavior using constraints (as in APAs). Modalities in the form of may- and must-edges are used to distinguish between allowed and required behavior. APTAs are regarded as specifications which are *implemented* by PTAs. In terms of expressiveness, APTAs subsume PTAs, MECS, and APAs. Applications of APTAs can be found in the area of component-based systems with real-time and probabilistic behavior, e.g., in communication and network protocols for embedded and multimedia systems. As a specific example, Stoelinga et al. considered PTAs for modeling the root contention protocol of the IEEE 1394 standard [20]. The authors defined several intermediate automata in between the implementation and the specification automaton which are related by simple refinement notions. This case study could benefit from modeling using APTAs that we introduce here because of their support for abstraction, refinement and compositional operations.

We show the following important results for our models. For APTAs, we define several appropriate refinement notions and establish a hierarchy among them. For deterministic APTAs, we show that three of these refinement notions coincide. We provide a consistency check for APTAs based on a reduction to stochastic two-player games. Both probabilistic and strict time-divergence are considered in consistency and refinement checking. We introduce APECAs as a subclass of APTAs and develop abstraction techniques and a compositional theory for this model. In particular, we show that an APECA is related with its abstractions by means of modal refinements. We define conjunction and parallel composition for APECAs, and show that they interact well with modal refinement and abstraction. Specifically, we show that conjunction is the greatest lower bound after pruning, and that modal refinement is a precongruence with respect to parallel composition. We further show that component-wise abstraction is as powerful as applying the combination of the local abstractions to the entire model. To the best of our knowledge, this is the first compositional modal specification and abstraction theory for probabilistic timed systems. Besides the integration of abstract probabilistic behavior, the work in this paper extends [4] by including a consistency check and refinement relations that consider time divergence. Moreover, our notion for abstraction non-trivially extends the corresponding APA concept in [11] by taking into account the guards of transitions.

Related work. This paper is part of an effort to develop a compositional specification theory and assume-guarantee reasoning for component-based systems. Previously, we have developed a linear-time specification theory for components [6] and its timed extension [9]. We have also formulated the corresponding sound and complete compositional assume-guarantee rules [8], demonstrating their application on examples of component-based systems from the networking domain. Linear-time refinement for probabilistic systems is known not to be compositional, and hence we focus on modal specifications. Modal specification theories for probabilistic systems include APAs [11] and Constraint Markov Chains [5]. A specification theory for real-time systems is defined in [10] including a set of operators supporting stepwise design of timed systems. A general approach for quantitative specification theories with modalities is presented in [3]. A robust specification theory for Modal Event-Clock Automata is discussed in [12]. And aggressive abstraction techniques for probabilistic automata are explored in [18].

Structure. In Section 2, we recall relevant notions for APAs and PTAs. Section 3 introduces our new model of Abstract Probabilistic Timed Automata. In Section 4, refinement notions for APTAs are defined and compared. Section 5 is devoted to abstraction for APTAs. In Section 6, we define conjunction and parallel composition, and present compositionality results for APECAs (a strict subclass of APTA). Section 7 concludes and discusses future work.

2 Preliminaries

In this section, we recall important definitions for PTAs [14] and refer the reader to [11] and [13] for APAs. We first recall some elementary notions. A *discrete probability distribution* over a denumerable set S is a function $\mu : S \to [0,1]$ with $\sum_{s\in S}\mu(s) = 1$. The set of all discrete probability distributions over S is denoted by $Dist(S)$. For a given $s \in S$, the *point distribution* μ_s is the unique distribution on S with $\mu_s(s) = 1$. We denote by $\mathbb{R}_+$ the set of non-negative reals. Let $\mathbb{B}_2 = \{\bot,\top\}$ and $\mathbb{B}_3 = \{\bot,?,\top\}$ be the complete lattices with the respective orderings $\bot < \top$ and $\bot < ? < \top$, and meet ($\sqcap$) and join ($\sqcup$) operators.

2.1 Probabilistic Timed Automata

We now recall the standard timed automata notions of clock valuations and guards. For a finite set X of *clocks*, a *clock valuation* is a function $v : X \to \mathbb{R}_+$. The set of all clock valuations over X is denoted by $\mathbb{R}_+^X$. For any $v \in \mathbb{R}_+^X$ and $t \in \mathbb{R}_+$, we use $v+t$ to denote the clock valuation defined as $(v+t)(x) = v(x)+t$ for all $x \in X$. We use $v[Y := 0]$ to denote the clock valuation obtained from v by resetting all of the clocks in $Y \subseteq X$ to 0, and leaving the values of all other clocks unchanged; formally, $v[Y := 0](x) = 0$ if $x \in Y$ and $v[Y := 0](x) = v(x)$ otherwise. We write $\overline{0}$ for the clock valuation that assign 0 to all clocks.

Let $X = \{x_1,\dots,x_n\}$ be a set of clocks. A *clock constraint* or *guard* g on X is an expression of the form $x \sim c$ such that $x,y \in X$, $c \in \mathbb{R}_+$ and $\sim\, \in \{\leq,<,>,\geq\}$, or a conjunction of guards. A clock valuation v satisfies g, written as $v \triangleright g$, iff g evaluates to true when all clocks $x \in X$ are substituted with their clock value $v(x)$. Let $CC(X)$ denote the set of all guards over X, and let $CC_N(X)$ denote the set of guards on X involving expressions with constants less or equal to N, where N is the maximal constant in all guards.

Definition 1 [PTA [14]] A *probabilistic timed automaton* is a tuple $\mathcal{M} = (L,A,X,AP,V,T,l_0)$ where L is a finite set of locations with initial location $l_0 \in L$; A is a finite set of actions; X is a finite set of clocks; AP is a finite set of atomic propositions; $V : L \to 2^{AP}$ assigns atomic propositions to locations; and $T : L \times CC(X) \times A \times Dist(2^X \times L) \to \mathbb{B}_2$ is a probabilistic edge function.

We write $l \xrightarrow{g,a} \mu$ iff $T(l,g,a,\mu) = \top$, which comprises a source location l, a guard g, and a probability distribution μ which assigns probabilities to pairs of the form (Y,l'), where $Y \subseteq X$ is a set of clocks to be reset and l' is a target location. The behavior of a PTA is as follows: in any location a probabilistic edge can be taken if its guard is satisfied by the current values of the clocks. Once a probabilistic edge is nondeterministically selected, the choice for a particular target location and set of clocks to be reset is made probabilistically using μ.

We define the semantics of a PTA $\mathcal{M}$ by mapping it to a probabilistic automaton M by employing the classical region equivalence for timed automata [1]. A *probabilistic automaton* (PA) [17, 11] $\mathsf{M} = (S,A,AP,V,T,s_0)$ consists of a set of states S with initial state s_0, a set of actions A, a set of atomic propositions AP, a valuation function $V : S \to 2^{AP}$ and a probabilistic transition function $T : S \times A \times Dist(S) \to \mathbb{B}_2$. A *region* θ is the set of clock valuations which satisfy exactly the same guards of $CC_N(X)$.

Given a region θ, we write $Succ(\theta)$ for the union of all regions that can be obtained from θ by letting time elapse. Given a guard $g \in CC(X)$, we write $\theta \subseteq g$ iff for all valuations $v \in \theta$ it holds that $v \triangleright g$. We denote the set of all regions by $\Theta_N(X)$ and simply write Θ if clear from context. We now define the semantics of a PTA in terms of a PA.

Definition 2 [Region PA] For a given PTA $\mathcal{M} = (L, A, X, AP, V, T, l_0)$, the associated *region PA* is given by $\mathsf{R}(\mathcal{M}) = (S, A', AP, V', T', s_0)$ where $S = L \times \Theta$ and $s_0 = (l_0, \overline{0})$, $A' = \Theta \times A \times (2^X)^S$, $V'(l,\theta) = V(l)$, and T' is induced by T in the following way: for any $l \in L$ and any $\theta \in \Theta$ such that (l,θ) is reachable from $(l_0, \overline{0})$, if $l \xrightarrow{g,a}_{\mathcal{M}} \mu$ then for each region $\theta'' \in Succ(\theta) \cap g$ there exists $\zeta : S \to 2^X$ and $\mu' \in Dist(S)$ such that $(l,\theta) \xrightarrow{\theta'',a,\zeta}_{\mathsf{R}(\mathcal{M})} \mu'$ and:

$$\zeta(l',\theta') = Y \text{ and } \mu'(l',\theta') = \mu(l',Y) \text{ if } \theta' = \theta''[Y := 0]; \text{ and } \zeta(l',\theta') = \emptyset \text{ and } \mu'(l',\theta') = 0 \text{ o.w.}$$

In the derived region PA, the transition labels $(\theta'', a, \zeta) \in A'$ consist of the region θ'' that represents the time window in which the transition is taken, the fired action $a \in A$, and for each target state $s \in S$ the set of clocks $\zeta(s) \subseteq X$ that are being reset when s is probabilistically chosen. W.l.o.g., we assume that $\mathsf{R}(\mathcal{M})$ is always pruned, i.e., all its states are reachable.

Similarly as shown for timed specifications in [4], any PA that is defined over the alphabet $\Theta \times A \times (2^X)^S$ can be interpreted as a PTA again. Intuitively, the states of the PA are interpreted as the locations of the corresponding PTA and the information about the guards, actions and clock resets for edges is derived from the transition labels of the PA. We introduce the operator $\mathcal{T}$ which translates any given PA M over the alphabet $\Theta \times A \times (2^X)^S$ into the PTA $\mathcal{T}(\mathsf{M})$. The application of $\mathcal{T} \circ \mathsf{R}$ allows us, moreover, to define a normal form for PTAs.

Definition 3 [Normal form] A PTA $\mathcal{M}$ is in *normal form* iff it is isomorphic to (the reachable part) of $(\mathcal{T} \circ \mathsf{R})(\mathcal{M})$.

Note that, if a PTA in normal form, every location is associated with a unique region. Moreover, $(\mathcal{T} \circ \mathsf{R})(\mathcal{M})$ is isomorphic to $(\mathcal{T} \circ \mathsf{R})^2(\mathcal{M})$ for any PTA $\mathcal{M}$. The PTA in normal form in needed later for technical reasons (e.g., Proposition 1 or Theorem 1).

3 Abstract Probabilistic Timed Automata

We now define *Abstract Probabilistic Timed Automata* (APTAs) as the central model of this paper. APTAs extend PTAs in three ways: (1) probability distributions are generalized to probability constraints, (2) may- and must-transitions are distinguished, and (3) locations are labeled with sets of admissible atomic propositions. All three modeling concepts are borrowed from APAs [11]. We use satisfaction relations to relate APTAs with PTAs that implement them. Let a *probability constraint* φ be a symbolic representation of a set of probability distributions over a set S. As in [11], we do not fix the language for probability constraints. The set of probability distributions that satisfy φ is denoted by $Sat(\varphi) \subseteq Dist(S)$. We define the constraints *true* and *false*, for which we require $Sat(true) = Dist(S)$ and $Sat(false) = \emptyset$. The set of probability constraints over S is denoted by $PC(S)$.

Definition 4 [APTA] An *abstract probabilistic timed automaton* is a tuple $\mathcal{A} = (L, A, X, AP, V, T, l_0)$, where L, A, X, AP, l_0 are defined as for PTAs; $V : L \to 2^{2^{AP}}$ assigns sets of admissible atomic propositions to locations; and $T : L \times CC(X) \times A \times PC(2^X \times L) \to \mathbb{B}_3$ is a three-valued probabilistic edge function.
We use the notation $l \overset{g,a}{\dashrightarrow} \varphi$ to denote may-edges (formally if $T(l,g,a,\varphi) = ?$), $l \xrightarrow{g,a} \varphi$ for must-edges (if $T(l,g,a,\varphi) = \top$), and $l \overset{g,a}{\rightsquigarrow} \varphi$ for may- or must-edges, where $g \in CC(X)$ is a guard, $a \in A$ is an action, and $\varphi \in PC(2^X \times L)$ is a probability constraint.

Example 1 Fig. 1 depicts an example APTA modeling a scheduler component. In the initial location l_0, tasks can be submitted to the scheduler and will be started within 1 time unit. Two types of tasks can occur: short- and long-running ones. The choice between them is probabilistic according to the probability constraint $\varphi_p = (0.25 \le p_1 \le 0.75) \wedge (0.25 \le p_2 \le 0.75) \wedge (p_1 + p_2 = 1)$. Tasks either finish in the expected time frame or can be canceled at any point. The canceling of tasks is modeled using may-edges, and thus is not required to be realized by implementations.

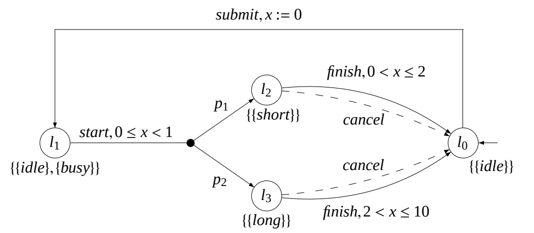


Figure 1: An example APTA specification for a scheduler component.

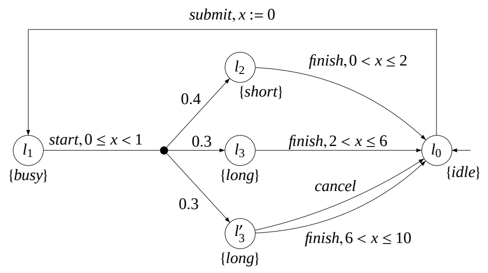


Figure 2: An example PTA implementing the APTA in Fig. 1.

Definition 5 [APTA Satisfaction] Let $\mathcal{M} = (L, A, X, AP, V, T, l_0)$ be a PTA in normal form and $\mathcal{A} = (L', A, X, AP, V', T', l'_0)$ be an APTA. $R \subseteq L \times L'$ is called a *satisfaction relation* iff, for all $(l, l') \in R$, these conditions hold:

1. $\forall a \in A$, $\forall \varphi' \in PC(2^X \times L')$, $\forall g \in CC(X)$ and $\forall \theta \in \Theta$: if $l' \xrightarrow{g,a}_{\mathcal{A}} \varphi'$ and both (l, θ) and (l', θ) are reachable in $\mathcal{M}$ and $\mathcal{A}$ respectively, then $\exists n \in \mathbb{N}$, $\exists g_1, \ldots, g_n \in CC(X)$ and $\exists \mu_1, \ldots, \mu_n \in Dist(2^X \times L)$ with: (i) $Succ(\theta) \cap g \subseteq Succ(\theta) \cap \bigcup_{i=1}^{n} g_i$; and (ii) $\forall 1 \le i \le n$: $l \xrightarrow{g_i,a}_{\mathcal{M}} \mu_i$ and $\exists \mu'_i \in Sat(\varphi')$ s.t. $\mu_i \Subset_R \mu'_i$ (see the definition of $\Subset_R$ in [13]);
2. $\forall a \in A$, $\forall \mu \in Dist(2^X \times L)$, $\forall g \in CC(X)$: if $l \xrightarrow{g,a}_{\mathcal{M}} \mu$ then $\exists g' \in CC(X)$ and $\exists \varphi' \in PC(2^X \times L')$: $l' \overset{g',a}{\rightsquigarrow}_{\mathcal{A}} \varphi'$, $g \subseteq g'$ and $\exists \mu' \in Sat(\varphi')$ with $\mu \Subset_R \mu'$;
3. $V(l) \in V'(l')$.

We say that $\mathcal{M}$ satisfies $\mathcal{A}$, denoted $\mathcal{M} \models \mathcal{A}$, iff there exists a satisfaction relation relating l_0 and l'_0. If $\mathcal{M} \models \mathcal{A}$, $\mathcal{M}$ is called an *implementation* of $\mathcal{A}$.

Condition 1 states that any must-edge in the specification is required to be realized in an implementation (possibly split up into several edges emitting from one location). Condition 2 ensures that any edge in the implementation is allowed by the specification (as a may- or a must-edge). Note that, since $\mathcal{M}$ is in normal form, the guard in the edge $l \xrightarrow{g,a}_{\mathcal{M}} \mu$ is necessarily a region. The set of all implementations of $\mathcal{A}$ is given by $[\![\mathcal{A}]\!] = \{\mathcal{M} | \mathcal{M} \models \mathcal{A}\}$.

Example 2 Fig. 2 depicts an implementation of the APTA for a scheduler component in Fig. 1. We indicate the satisfaction relation by using equal location indices, e.g., the locations l_3 and l'_3 in the implementation are in relation with l_3 in the specification. The implementation differs from the specification in the following aspects. After a task has been submitted, the scheduler becomes busy, i.e., the set of atomic propositions $\{busy\}$ is chosen for location l_1. Two types of long-running tasks are distinguished in the implementation: ones that finish in the interval $(2, 6]$ and ones that finish in the interval $(6, 10]$. Only tasks of the latter type can be canceled. The probability constraint φ_p is realized by the probability distribution assigning 0.4 to l_2, and 0.3 to l_3 and l'_3, respectively. Note, however, that this PTA is not in normal form, because the location l_0 can be reached within three different regions: $(0, 2]$, $(2, 6]$ and $(6, 10]$. Thus, by splitting up location l_0, the normal form can be obtained and the satisfaction relation is constructed.

3.1 The Region-Based Interpretation

We show now that the check for the existence of a satisfaction relation between a PTA and an APTA can be reduced to a check for a satisfaction relation between their corresponding probabilistic region automata. Analogously to the mapping of a PTA to a PA using the region construction (cf. Def. 2), we can transform any APTA $\mathcal{A}$ into an APA $\mathsf{R}(\mathcal{A})$. In the resulting APA $\mathsf{R}(\mathcal{A})$, the types of the three-valued edge function are inherited from $\mathcal{A}$. The transition relation $T_{\mathsf{R}(\mathcal{A})} : S \times \Theta \times A \times (2^X)^S \times PC(S) \to \mathbb{B}_3$ is lifted from distributions to constraints by:

- for any $l \in L$, $\theta \in \Theta$ such that (l,θ) is reachable from $(l_0, \overline{0})$: if $l \xrightarrow{g,a}_{\mathcal{A}} \varphi$ then for each $\theta'' \in Succ(\theta) \cap g$ there exists $\zeta : S \to 2^X$ and $\varphi' \in PC(S)$ such that $(l,\theta) \xrightarrow{\theta'',a,\zeta}_{\mathsf{R}(\mathcal{A})} \varphi'$, and $\exists \mu \in Sat(\varphi)$ iff $\exists \mu' \in Sat(\varphi')$ with:

  $$\zeta(l',\theta') = Y \text{ and } \mu'(l',\theta') = \mu(l',Y) \text{ if } \theta' = \theta''[Y := 0]; \text{ and } \zeta(l',\theta') = \emptyset \text{ and } \mu'(l',\theta') = 0 \text{ o.w.}$$

  and analogously for all may-edges.

Proposition 1 Consider an APTA $\mathcal{A} = (L, A, X, AP, V, T, l_0)$ and a PA $\mathsf{M} = (S, A', AP, V', T', s_0)$ where $A' = \Theta(X) \times A \times (2^X)^S$. If $\mathsf{M} \models \mathsf{R}(\mathcal{A})$ then $\mathcal{T}(\mathsf{M})$ is in normal form and $\mathcal{T}(\mathsf{M}) \models \mathcal{A}$.

This proposition will be used later in Section 3.2.

Theorem 1 Given a PTA $\mathcal{M} = (L, A, X, AP, V, T, l_0)$ in normal form and APTA $\mathcal{A} = (L', A, X, AP, V', T', l_0')$, and let $\mathsf{R}(\mathcal{M}) = (S, A_{\mathsf{R}}, AP, V_{\mathsf{R}}, T_{\mathsf{R}}, s_0)$ and $\mathsf{R}(\mathcal{A}) = (S', A_{\mathsf{R}}, AP, V'_{\mathsf{R}}, T'_{\mathsf{R}}, s'_0)$ be the respective region automata. Then $\mathcal{M} \models \mathcal{A}$ if and only if $\mathsf{R}(\mathcal{M}) \models \mathsf{R}(\mathcal{A})$.

Theorem 1 does not hold for arbitrary PTAs, since the *if*-part only holds for PTAs in normal form. That is to say, there exist a PTA $\mathcal{M}$ and an APTA $\mathcal{A}$ such that $\mathcal{M} \not\models \mathcal{A}$, while $\mathsf{R}(\mathcal{M}) \models \mathsf{R}(\mathcal{A})$. A similar example for (non-probabilistic) timed modal specifications can be found in [4].

Definition 6 [Deterministic APTA] Given an APTA $\mathcal{A} = (L, A, X, AP, V, T, l_0)$ and its region automaton $\mathsf{R}(\mathcal{A}) = (S, A', AP, V', T', s_0)$. $\mathcal{A}$ is called:

- *action-deterministic*, iff for all reachable states s in $\mathsf{R}(\mathcal{A})$ it holds: if there exist $s \overset{\theta_1,a,\zeta_1}{\rightsquigarrow}_{\mathsf{R}(\mathcal{A})} \varphi_1$ and $s \overset{\theta_2,a,\zeta_2}{\rightsquigarrow}_{\mathsf{R}(\mathcal{A})} \varphi_2$ such that $\varphi_1 \neq \varphi_2$, then $\theta_1 \cap \theta_2 = \emptyset$;
- *AP-deterministic*, iff $s \overset{\theta,a,\zeta}{\rightsquigarrow}_{\mathsf{R}(\mathcal{A})} \varphi$ implies that for all $\mu', \mu'' \in Sat(\varphi)$, and $s' \neq s'' \in S$ it holds: $(\mu'(s') > 0 \wedge \mu''(s'') > 0) \Longrightarrow V(s') \cap V(s'') = \emptyset$.

$\mathcal{A}$ is called *deterministic* iff it is action-deterministic and AP-deterministic.

Note that Def. 6 is inspired by [11]. Action-determinacy can also be enforced on the syntactical level. However, such a definition would only be a sufficient, but not a necessary condition.

3.2 Consistency

Consistency of a specification refers to the property that there exists at least one model for this specification. In our setting, an APTA $\mathcal{A}$ is said to be consistent if it admits at least one implementation, hence formally iff $[\![\mathcal{A}]\!] \neq \emptyset$. For any given APTA $\mathcal{A}$, we can decide whether the APA $\mathsf{R}(\mathcal{A})$ is consistent and, if so, derive a PA M, such that $\mathsf{M} \models \mathsf{R}(\mathcal{A})$ [11]. $\mathcal{T}(\mathsf{M})$ is then a PTA with finitely many states, and, by Proposition 1, a model of $\mathcal{A}$.

In order to deal with consistency also on the syntactic level, we further define a location $l \in L_{\mathcal{A}}$ in an APTA $\mathcal{A} = (L_{\mathcal{A}}, A, X, AP, V_{\mathcal{A}}, T_{\mathcal{A}}, l^0_{\mathcal{A}})$ to be consistent if $V_{\mathcal{A}}(l) \neq \emptyset$ and for all guards $g \in CC(X)$, actions

$a \in A$ and probability constraints $\varphi \in PC(2^X \times L_{\mathcal{A}})$ it holds: if $T_{\mathcal{A}}(l,g,a,\varphi) = \top$ then $Sat(\varphi) \neq \emptyset$. Note that inconsistency of a location does not imply inconsistency of the whole APTA. In order to decide whether an APTA is consistent, we follow the usual approach and use a *pruning operator* β that filters out distributions leading to inconsistent locations [11]. The detailed definition and properties of a pruning operator can be found in [13].

The following theorem shows that the application of β operator does not change the set of implementation. And it also implies that if $\beta^*(\mathcal{A})$ is empty, then $\mathcal{A}$ is inconsistent.

Theorem 2 For any APTA $\mathcal{A}$, it holds that $[\![\mathcal{A}]\!] = [\![\beta(\mathcal{A})]\!] = [\![\beta^*(\mathcal{A})]\!]$.

The above definition of consistency, however, places no restrictions on the derived implementations. In particular, the derived PTA could show unrealistic behaviors by preventing time from diverging. Therefore, we also aim at checking consistency in such a way that only divergent implementations are considered. Note that a divergent consistent APTA must be consistent, therefore we assume that the APTAs that we deal with are already consistent.

We consider the set of *strict* and *probabilistic divergent* (Sd and Pd, for short) implementations of $\mathcal{A}$ given by $[\![\mathcal{A}]\!]^{\mathsf{Sd}} = \{\mathcal{M} \mid \mathcal{M} \models \mathcal{A}$ and $\mathcal{M}$ is strict divergent$\}$ and $[\![\mathcal{A}]\!]^{\mathsf{Pd}} = \{\mathcal{M} \mid \mathcal{M} \models \mathcal{A}$ and $\mathcal{M}$ is probabilistic divergent$\}$, respectively. The formal definitions of probabilistic and strict divergency can be found in [19] and [13]. Now we define an APTA to be Sd- or Pd-*consistent* if it admits at least one Sd or Pd implementation, i.e., $[\![\mathcal{A}]\!]^{\mathsf{Sd}} \neq \emptyset$ or $[\![\mathcal{A}]\!]^{\mathsf{Pd}} \neq \emptyset$.

Theorem 3 shows that a time-divergence sensitive consistency check for APTAs can be defined based on a reduction to APAs and stochastic two-player games. The details of this technique can be found in [13]. This result effectively allows us to check whether an APTA has at least one strict or probabilistic divergent implementation.

Theorem 3 An APTA $\mathcal{A}$ is Pd (resp. Sd) consistent if and only if in the game $\mathcal{G}(\mathcal{A})$, the $\bullet$-player has a winning strategy for the objective $\mathbb{P}_{=1}(\Box\Diamond tick)$ (resp. objective $\Box\Diamond tick$).

3.3 Abstract Probabilistic Event-Clock Automata

We now introduce *Abstract Probabilistic Event-Clock Automata* (APECAs), which form a strict subclass of APTA, where clock resets are not arbitrary: each action a is associated with a clock x_a which is reset exactly when the action a occurs. This kind of clock resets originated from *Event-Clock Automata* (ECAs) [2]: they form a strict subclass of TA, but they enjoy nice properties, e.g., they are closed under union and intersection, and can be determinized.

Definition 7 [APECA] A (complete) *abstract probabilistic event-clock automaton* (APECA) is a tuple $\mathcal{E} = (L, A, X_A, AP, V, T, l_0)$, where L, A, AP, l_0, and V are defined as for APTAs;

- X_A is a set of clocks where every $x_a \in X_A$ corresponds to an action $a \in A$;
- $T : L \times CC(X_A) \times A \times PC(L) \to \mathbb{B}_3$ is a three-valued probabilistic transition function, s.t. for all $l \in L, a \in A$: $\bigvee_i \{g_i \mid \exists \varphi_i : T(l, g_i, a, \varphi_i) \neq \bot\} = true$.

Example 3 [APECA] Fig. 3 depicts two APECAs Cl and Acc. Cl models a clients requesting access to a given resource. It can either invoke *get* to request the resource; or *grant* to access it. The action *extra* is used when a privileged access with extended time is needed. We use ! and ? to indicate whether an action comes from the designed component or from its environment. The clock corresponding to the action *get* is x_{get}. The client sends a second *get*-request at most one time unit after the first request. With a probability satisfying constraint φ_1, the client terminates and stops requesting resources (state 2). The client can also request extended time at any moment while it is still active. Let the probability from state 1 to state 0 be p_1 and from state 1 to state 2 be p_2 in Cl. Then φ_1 could be defined as $0 \le p_1 \le 1/3$ and $p_1 + p_2 = 1$.

The APECA *Acc* specifies the behavior of an access controller. If the access to the resource is granted, then it should happen within 2 time units after reception of a *get* request. In case of a privileged access with extra time, this duration will be extended to at most 4 time units. However, with a certain probability (satisfying φ_2), the access controller will switch back to the default access time of 2 time units. The probability constraint φ_2 can be defined in a similar way as φ_1. The use of probability constraints is explained in more detail in Examples 5 and 6.

The main difference to APTAs is that the probability constraints in APECAs are defined on L instead on $2^X_A \times L$, and that we require completeness for the edge function. However, completeness is not a restriction, e.g., we model the case where in location l there exists no outgoing a-edge by setting $T(l, true, a, false) = ?$. Completing an APECA in this way does not modify its set of implementations. Note that a similar approach is also used in [4] to obtain completeness for timed modal specifications. An implementation of an APECA is a *Probabilistic Event-Clock Automaton* (PECA), which is a probabilistic variant of ECAs. PECAs form a strict subclass of PTAs. Formally, a PECA is a tuple $C = (L, A, X_A, AP, V, T, l_0)$, where L, A, AP, V and l_0 are as in PTAs, X_A is as in the APECA, and $T : L \times CC(X_A) \times A \times Dist(L) \to \mathbb{B}_2$ is a two-valued probabilistic edge function.

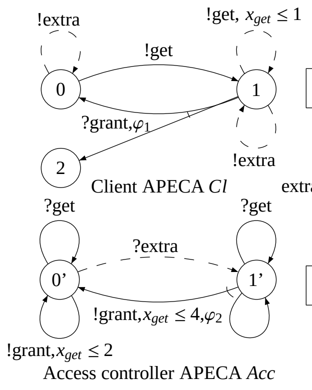


Figure 3: The two APECAs Cl and Acc.

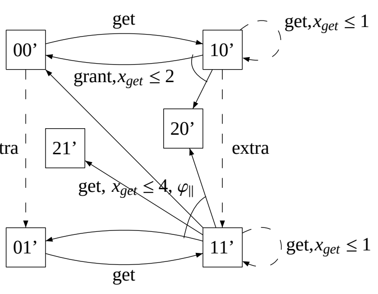


Figure 4: The parallel composition $Cl \,||\, Acc$.

4 Refinement

In this section, we define various refinement notions for APTAs and discuss their relationships. More specifically, we define syntactical refinements based on simulation relations and investigate their relationship to semantical refinement (also referred to as *thorough* refinement), i.e., inclusion of sets of implementations. Since our refinement notions for APTAs are based on the refinement notions for APAs [11], we recall relevant definitions now.

Definition 8 [Weak APA refinement [11]] Let $\mathsf{A}_1 = (S_1, A, AP, V_1, T_1, s_1^0)$ and $\mathsf{A}_2 = (S_2, A, AP, V_2, T_2, s_2^0)$ be two APAs. A relation $R \subseteq S_1 \times S_2$ is called a *weak refinement relation* iff, for all $(s_1, s_2) \in R$, the following conditions hold:

1. $\forall a \in A, \forall \varphi_2 \in PC(S_2) : s_2 \xrightarrow{a}_2 \varphi_2 \Longrightarrow \exists \varphi_1 \in PC(S_1) : s_1 \xrightarrow{a}_1 \varphi_1$ and $\forall \mu_1 \in Sat(\varphi_1) : \exists \mu_2 \in Sat(\varphi_2)$ with $\mu_1 \Subset_R \mu_2$ (see the definition of $\Subset_R$ in [13]);
2. $\forall a \in A, \forall \varphi_1 \in PC(S_1) : s_1 \overset{a}{\rightsquigarrow}_1 \varphi_1 \Longrightarrow \exists \varphi_2 \in PC(S_2) : s_2 \overset{a}{\rightsquigarrow}_2 \varphi_2$ and $\forall \mu_1 \in Sat(\varphi_1) : \exists \mu_2 \in Sat(\varphi_2)$ with $\mu_1 \Subset_R \mu_2$; and
3. $V_1(s_1) \subseteq V_2(s_2)$.

We write $\mathsf{A}_1 \preceq_W \mathsf{A}_2$ iff there exists a weak refinement relation relating s_1^0 and s_2^0.

Note that the correspondence function (see [13]) is not fixed in advance in weak refinements. This is the case in *strong* APA refinements [11], which we denote by $\preceq_S$.

We are now in a position to define our refinement notions for APTAs. While *thorough* refinement is a *semantical* inclusion between sets of implementations, *strong* and *weak* refinements are its *syntactical* counterparts. For the latter two, we apply the refinement notions for APAs to the induced region automata.

Definition 9 [APTA refinements] Let $\mathcal{A}_1 = (L_1, A, X, AP, V_1, T_1, l_1^0)$ and $\mathcal{A}_2 = (L_2, A, X, AP, V_2, T_2, l_2^0)$ be two APTAs. We say that

1. $\mathcal{A}_1$ *thoroughly refines* $\mathcal{A}_2$, denoted as $\mathcal{A}_1 \preceq_T \mathcal{A}_2$, iff $[\![\mathcal{A}_1]\!] \subseteq [\![\mathcal{A}_2]\!]$;
2. $\mathcal{A}_1$ *Sd-thoroughly refines* $\mathcal{A}_2$, denoted as $\mathcal{A}_1 \preceq_T^{\mathsf{Sd}} \mathcal{A}_2$, iff $[\![\mathcal{A}_1]\!]^{\mathsf{Sd}} \subseteq [\![\mathcal{A}_2]\!]^{\mathsf{Sd}}$;
3. $\mathcal{A}_1$ *Pd-thoroughly refines* $\mathcal{A}_2$, denoted as $\mathcal{A}_1 \preceq_T^{\mathsf{Pd}} \mathcal{A}_2$, iff $[\![\mathcal{A}_1]\!]^{\mathsf{Pd}} \subseteq [\![\mathcal{A}_2]\!]^{\mathsf{Pd}}$;
4. $\mathcal{A}_1$ *strongly refines* $\mathcal{A}_2$, denoted as $\mathcal{A}_1 \preceq_S \mathcal{A}_2$, iff $\mathsf{R}(\mathcal{A}_1) \preceq_S \mathsf{R}(\mathcal{A}_2)$;
5. $\mathcal{A}_1$ *weakly refines* $\mathcal{A}_2$, denoted as $\mathcal{A}_1 \preceq_W \mathcal{A}_2$, iff $\mathsf{R}(\mathcal{A}_1) \preceq_W \mathsf{R}(\mathcal{A}_2)$.

By Theorem 1, we can directly obtain that, for any APTAs $\mathcal{A}_1$ and $\mathcal{A}_2$, it also holds that:

$$\mathcal{A}_1 \preceq_T \mathcal{A}_2 \quad \textit{iff} \quad \mathsf{R}(\mathcal{A}_1) \preceq_T \mathsf{R}(\mathcal{A}_2) \tag{1}$$

where $\mathsf{R}(\mathcal{A}_1) \preceq_T \mathsf{R}(\mathcal{A}_2)$ refers to thorough refinement for APAs, which is also defined as inclusion of implementation sets [11] (analogously for $\preceq_T^{\mathsf{Sd}}$ and $\preceq_T^{\mathsf{Pd}}$). An example of a strong refinement can be found in [13].

The following theorem establishes a hierarchy among the different notions of refinement. We use $RF_1 \supset RF_2$ to indicate that the refinement RF_1 is strictly finer than the refinement RF_2.

Theorem 4 APTA refinements form the following hierarchy: $\preceq_T^{\mathsf{Sd}} \supset \preceq_T^{\mathsf{Pd}} \supset \preceq_T \supset \preceq_W \supset \preceq_S$.

In Proposition 2, we relate weak and strong refinement with probabilistic time-abstracting bisimulation [7] for PTAs. In Proposition 3 we further show that strong, weak and thorough refinements coincide for deterministic APTAs.

Proposition 2 Let $\sim$ denote probabilistic time-abstracting bisimilarity [7] on PTAs. If $\mathcal{A}_1$ and $\mathcal{A}_2$ are implementations, then *(a)* $\mathcal{A}_1 \preceq_W \mathcal{A}_2$ iff $\mathcal{A}_1 \sim \mathcal{A}_2$; and *(b)* $\mathcal{A}_1 \preceq_S \mathcal{A}_2$ only if $\mathcal{A}_1 \sim \mathcal{A}_2$.

Proposition 3 For deterministic APTAs, where the sets of admissible atomic propositions in the initial locations are singletons, thorough, strong and weak refinement coincide.

Remark 1 *The counterexamples in Fig. 5 also show that the Sd- and Pd-thorough refinements do not coincide with each other or with thorough refinement even for deterministic APTAs.*

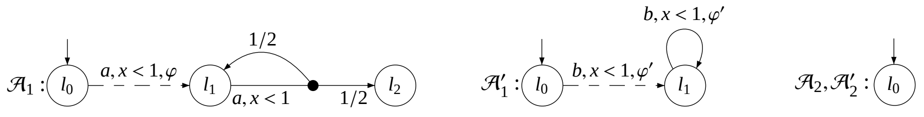


Figure 5: Counterexamples for showing the *strictly* finer relations

5 Abstraction

The goal of abstraction is to hide internal details of a specification and thereby to obtain a simpler and usually smaller specification. In the setting of automata, an abstraction can be defined by partitioning the state space, i.e., by forming disjoint groups of states (or locations) where each of these groups is mapped to one abstract state (or location).

Given a set of locations L, an *abstraction function* for L is a surjective function $\alpha : L \to \tilde{L}$. Its inverse $\gamma : \tilde{L} \to 2^L$ is called a *concretization function*. The abstraction of $\mu \in Dist(2^X \times L)$, denoted $\alpha(\mu) \in Dist(2^X \times \tilde{L})$ is uniquely defined by $\alpha(\mu)(\tilde{l}) = \mu(\gamma(\tilde{l}))$, for all $\tilde{l} \in \tilde{L}$. Abstraction is lifted to sets of states, sets of distributions, and sets of probability constraints in a pointwise manner. It follows that $\tilde{\varphi} = \alpha(\varphi)$ iff $Sat(\tilde{\varphi}) = \alpha(Sat(\varphi))$. The abstraction of the product of constraint function φ and φ' is given as $\alpha(\varphi \cdot \varphi') = \alpha(\varphi) \cdot \alpha(\varphi')$.

A technical challenge in defining abstraction for APTAs is the handling of guards. For this purpose, we introduce a pre-processing step that syntactically transforms an APTA into an equivalent APTA such that an abstraction function can be applied.

Definition 10 Let $\mathcal{A} = (L,A,X,AP,V,T,l_0)$ be an APTA, $\alpha : L \to \tilde{L}$ be an abstraction function, $\gamma : \tilde{L} \to 2^L$ its concretization function. We define the function $\mathsf{g} : \tilde{L} \times A \to CC(X)$ s.t. $\mathsf{g}(\tilde{l},a) = \bigwedge g_i$, if $\forall l_i \in \gamma(\tilde{l}) : \exists \varphi_i \in PC(2^X \times L), g_i \in CC(X) : T(l_i,g_i,a,\varphi_i) = \top$; and $\mathsf{g}(\tilde{l},a) = \mathit{false}$, otherwise.

Here, g calculates the common guards of all must-transitions emitting from $\tilde{l}$ with action a. Given the function g, we define a pre-processing step for APTAs that splits some of the must-transitions such that abstraction has the intended meaning.

Given a guard $g \in CC(X)$, we define the negation of g, denoted $\overline{g} \subseteq CC(X)$, as the set of guards, such that for any valuation $v \in \mathbb{R}^X$ it holds that $v \triangleright g$ if and only if there exists no $g' \in \overline{g}$ such that $v \triangleright g'$. The negation of a guard can be effectively computed by splitting the guard into its atomic comparisons and inverting the comparisons.

Definition 11 [Pre-processing] Let $\mathcal{A} = (L,A,X,AP,V,T,l_0)$ be an APTA, $\alpha : L \to \tilde{L}$ be an abstraction function, $\gamma : \tilde{L} \to 2^L$ be its concretization function. Let $\mathsf{g} : \tilde{L} \times A \to CC(X)$ be defined as before. The pre-processing function $\mathcal{P}_\alpha$ maps $\mathcal{A}$ to the APTA $\mathcal{P}_\alpha(\mathcal{A}) = (L,A,X,AP,V,T',l_0)$ such that for any $l \in L$, $g \in CC(X)$ and $\varphi \in PC(2^X \times L)$, if $T(l,g,a,\varphi) = \top$, then $T'(l,\mathsf{g}(\alpha(l),a),a,\varphi) = \top$ and $\forall g' \in \overline{\mathsf{g}}(\alpha(l),a)$: $T'(l, g \wedge g', a, \varphi_i) = \top$; and $T'(l,g,a,\varphi) = T(l,g,a,\varphi)$, otherwise.

As a result of the pre-processing function, the guards on a must-transition are either the common guard determined by g, or are disjoint with the common guard. Since $\mathsf{g}(\alpha(l),a)$ and $g \wedge g'$ for all $g' \in \overline{\mathsf{g}}(\alpha(l_i),a)$ form a partition of g, it is easy to see that $\mathsf{R}(\mathcal{A}) = \mathsf{R}(\mathcal{P}_\alpha(\mathcal{A}))$. We are now in a position to define the abstraction.

Definition 12 [APTA Abstraction] Given an abstraction function $\alpha : L \to \tilde{L}$ and its concretization function $\gamma : \tilde{L} \to 2^L$, a pre-processed APTA $\mathcal{P}_\alpha(\mathcal{A}) = (L,A,X,AP,V,T,l_0)$ and a guard function $\mathsf{g} : \tilde{L} \times A \to CC(X)$. Let $\alpha(\mathcal{P}_\alpha(\mathcal{A})) = (\tilde{L},A,X,AP,\tilde{V},\tilde{T},\alpha(l_0))$ be the APTA defined by: $\tilde{V}(\tilde{l}) = \bigcup_{l \in \gamma(\tilde{l})} V(l)$ and

$$\tilde{T}(\tilde{l},\tilde{g},a,\tilde{\varphi}) = \begin{cases} \top & \text{if } \tilde{g} = \mathsf{g}(\tilde{l},a), \text{and } Sat(\tilde{\varphi}) = \alpha(\bigcup_{\langle l,\varphi\rangle \in \gamma(\tilde{l}) \times PC(2^X \times L) : T(l,\tilde{g},a,\varphi) = \top} Sat(\varphi)) \\ ? & \text{if } \tilde{g} \neq \mathsf{g}(\tilde{l},a), \text{and } \exists l \in \gamma(\tilde{l}), \varphi \in PC(2^X \times L) : T(l,\tilde{g},a,\varphi) \neq \bot, \text{and} \\ & \quad Sat(\tilde{\varphi}) = \alpha(\bigcup_{\langle l,\varphi\rangle \in \gamma(\tilde{l}) \times PC(2^X \times L) : T(l,\tilde{g},a,\varphi) \neq \bot} Sat(\varphi)) \\ \bot & \text{otherwise} \end{cases}$$

Lemma 1 Let $\alpha(\mathcal{P}_\alpha(\mathcal{A}))$ be an abstraction of $\mathcal{A}$. Then there exists an APA abstraction function (cf. [11]) α' on $\mathsf{R}(\mathcal{A})$, such that $\mathsf{R}(\alpha(\mathcal{P}_\alpha(\mathcal{A}))) = \alpha'(\mathcal{P}_{\alpha'}(\mathsf{R}(\mathcal{A})))$.

Proposition 4 For any APTA $\mathcal{A}$ and abstraction function α, $\mathcal{A} \leq_W \alpha(\mathcal{P}_\alpha(\mathcal{A}))$.

Lemma 1 states that an abstraction function for an APTA $\mathcal{A}$ induces an abstraction function on $\mathsf{R}(\mathcal{A})$. Proposition 4 follows directly from Lemma 1 and a similar result known for APAs [11]. This result is important in order to ensure that applying an abstraction yields a generalized specification, i.e., formally that the original specification always weakly refines its abstraction. Note also that we show in Section 6 that abstraction interacts well with parallel composition.

Example 4 Another client specification Cl_1 is depicted in Fig. 6(a), where state 1 in Cl is split into 1' and 1" in Cl_1. State 1' can be seen as a "quick phase", since a *get* request should be sent in less than 2 time units, while state 1" is the "slow phase" due to the guard $x_{get} \geq 2$. Furthermore, in state 0' and 1', an extended time slot will be granted whether needed or not. However, in state 0', the additional time will only be granted after 1 time unit. From state 1", it is possible to either start from 0' again (quick phase), to move to state 1' (slow phase), or to state 2' (termination) after the access to the resource has been granted.

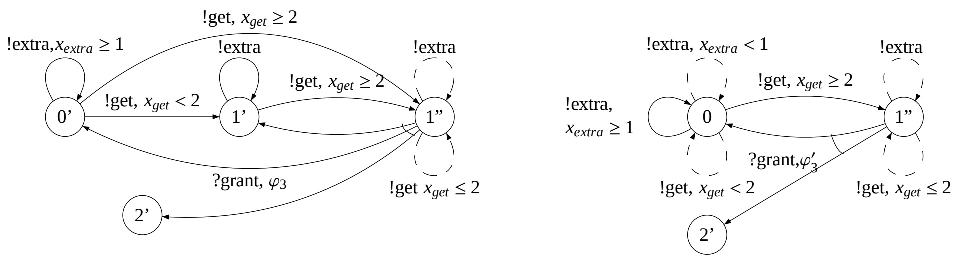
Let the abstraction function be defined as $\alpha(l) = 0$ for $l \in \{0', 1'\}$ and $\alpha(l) = l$ for $l \in \{1'', 2'\}$. The pre-processing splits the edge (!*extra*,*true*) from state 1' into (!*extra*, $x_{extra} \geq 1$) and (!*extra*, $x_{extra} < 1$). The abstraction $\alpha(Cl_1)$ is shown in Fig. 6(b). Any distribution μ' satisfying the constraint φ'_3 in $\alpha'(Cl_1)$ is defined such that $\mu'(0) = \mu(0') + \mu(1')$, for any $\mu \in Sat(\varphi_3)$ in Cl_1.

6 Conjunction and Parallel Composition

In this section, we define two composition operators for APECAs, i.e., conjunction and parallel composition. These two operators are intentionally defined only for APECAs, and not for general APTAs, in order to be able to ensure compositionality properties. Conjunction and parallel composition form the cornerstones for a specification theory supporting independent development and structural composition.

6.1 Conjunction

Given two APECAs $\mathcal{E}_1$ and $\mathcal{E}_1$, their conjunction (or *logical composition*), denoted as $\mathcal{E}_1 \wedge \mathcal{E}_2$, is the specification that realizes the conjunctive behavior of $\mathcal{E}_1$ and $\mathcal{E}_2$. Specifically, the set of implementations of the conjunction approximates the intersection of the sets of implementations of $\mathcal{E}_1$ and $\mathcal{E}_2$. We define conjunction here only for action-deterministic APECAs over the same set of actions. For two automata with different action sets, we add the missing actions in the respective automaton, and complete the edge function as explained in Section 3.3. We leave the generalization to nondeterministic APECAs for future work.



(a) APECA Cl_1. (b) The abstraction $\alpha'(Cl_1)$.

Figure 6: APECA abstraction.

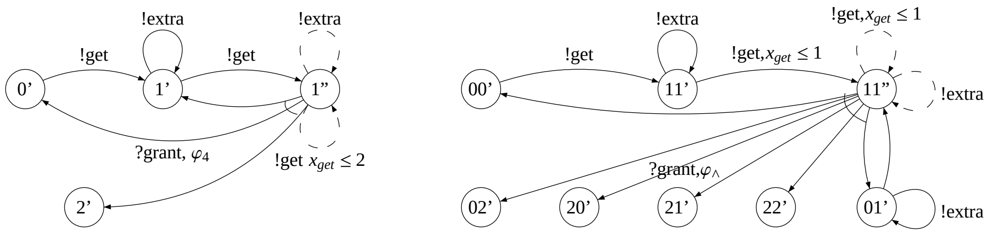
Definition 13 [APECA Conjunction] Let $\mathcal{E}_1 = (L_1, A, X_A, AP, V_1, T_1, l_0^1)$ and $\mathcal{E}_2 = (L_2, A, X_A, AP, V_2, T_2, l_0^2)$ be two action-deterministic APECAs over the same sets of actions and atomic propositions. Their *conjunction* is

$$\mathcal{E}_1 \wedge \mathcal{E}_2 = (L_1 \times L_2, A, X_A, AP, V_\wedge, T_\wedge, \langle l_0^1, l_0^2 \rangle)$$

where $V_\wedge(\langle l_1, l_2 \rangle) = V_1(l_1) \cap V_2(l_2)$ for all $l_1 \in L_1, l_2 \in L_2$ and $T_\wedge$ is defined as follows: for all $a \in A$, $g_1, g_2 \in CC(X_A)$, and $l_1 \in L_1$, $l_2 \in L_2$:

1. $\forall \varphi_1 \in PC(L_1)$, $\forall \varphi_2 \in PC(L_2)$ such that $T_1(l_1, g_1, a, \varphi_1) \neq \bot$ and $T_2(l_2, g_2, a, \varphi_2) \neq \bot$, let $T_\wedge(\langle l_1, l_2 \rangle, g_1 \wedge g_2, a, \varphi_\wedge) = T_1(l_1, g_1, a, \varphi_1) \sqcup T_2(l_2, g_2, a, \varphi_2)$ with $\varphi_\wedge$ the new constraint in $PC(L_1 \times L_2)$ such that $\mu_\wedge \in Sat(\varphi_\wedge)$ iff
   - the distribution $\mu_1 = \{\, k_1 \mapsto \sum_{k_2 \in L_2} \mu_\wedge(\langle k_1, k_2 \rangle) \,\}$ is in $Sat(\varphi_1)$, and
   - the distribution $\mu_2 = \{\, k_2 \mapsto \sum_{k_1 \in L_1} \mu_\wedge(\langle k_1, k_2 \rangle) \,\}$ is in $Sat(\varphi_2)$.
2. For all other $\varphi'_\wedge \in PC(L_1 \times L_2)$ and $g' \in CC(X_A)$ we define $T_\wedge(\langle l_1, l_2 \rangle, g', a, \varphi'_\wedge) = \bot$.

Informally, the conjunction $\mathcal{E}_1 \wedge \mathcal{E}_2$ can be regarded as the *largest* specification that refines $\mathcal{E}_1$ *and* $\mathcal{E}_2$. Note that we rely on the completeness of the edge functions. For example, assume that in l_1 there is a must-edge via the action a, and in l_2 the action a is not enabled. As we require completeness, the latter means that there is an edge $l_2 \overset{g,a}{\dashrightarrow}$ *false*. In the conjunction, this edge is combined with the must-edge from l_1 using Rule 1. Thus, in the conjunction we obtain the edge $\langle l_1, l_2 \rangle \xrightarrow{g,a}$ *false*, which means that $\langle l_1, l_2 \rangle$ is inconsistent. On the other hand, for may-edges no inconsistencies are produced.



(a) APECA Cl_2. (b) APECA $Cl \wedge Cl_2$.

Figure 7: APECA conjunction.

Example 5 Fig. 7(a) depicts Cl_2 as another version of the client. The conjunction of Cl and Cl_2 is shown in Fig. 7(b). Let φ_1 be defined as in Example 3 and let the probability from state 1" to state 0' be q_1 and from state 1" to state 1' be q_2 and from state 1" to 2' be q_3 in Cl_2. We set φ_4 such that $0 \le q_1 \le 1/5$, $1/3 \le q_2 \le 1$ and $q_1 + q_2 + q_3 = 1$. Now let $\mu_\wedge$ be any distribution satisfying $\varphi_\wedge$. Let $\mu_\wedge(00') = r_1$, $\mu_\wedge(01') = r_2$, $\mu_\wedge(02') = r_3$, $\mu_\wedge(20') = r_4$, $\mu_\wedge(21') = r_5$, $\mu_\wedge(22') = r_6$. Those r's should satisfy the following constraints, given the constraints on the p's and the q's:

$$r_1 + r_2 + r_3 = p_1 \quad r_4 + r_5 + r_6 = p_2 \quad r_1 + r_4 = q_1 \quad r_2 + r_5 = q_2 \quad r_3 + r_6 = q_3.$$

We use the notation $\mathsf{A}_1 \equiv \mathsf{A}_2$ to denote that both $\mathsf{A}_1 \preceq_W \mathsf{A}_2$ and $\mathsf{A}_2 \preceq_W \mathsf{A}_1$. Next, we show that conjunction is preserved by the region construction.

Lemma 2 For any APECAs $\mathcal{E}_1, \mathcal{E}_2$, it holds that $\mathsf{R}(\mathcal{E}_1 \wedge \mathcal{E}_2) \equiv \mathsf{R}(\mathcal{E}_1) \wedge \mathsf{R}(\mathcal{E}_2)$.

We now show that conjunction is the greatest lower bound of APECAs w.r.t. weak refinement, after pruning the conjunction. Note that $\beta^*(\mathcal{E})$ means applying the pruning operator β on $\mathcal{E}$ for finitely many times. Theorem 6 relates the sets of implementations of the conjunction with the implementation sets of its components.

Theorem 5 Let $\mathcal{E}_1$, $\mathcal{E}_2$ and $\mathcal{E}_3$ be action-deterministic consistent APECAs. Then the following properties hold: (1) $\beta^*(\mathcal{E}_1 \wedge \mathcal{E}_2) \leq \mathcal{E}_1$; (2) if $\mathcal{E}_3 \leq \mathcal{E}_1$ and $\mathcal{E}_3 \leq \mathcal{E}_2$, then $\mathcal{E}_3 \leq \beta^*(\mathcal{E}_1 \wedge \mathcal{E}_2)$.

Theorem 6 For any APECAs $\mathcal{E}_1$, $\mathcal{E}_2$ it holds: $[\![\mathcal{E}_1 \wedge \mathcal{E}_2]\!] \subseteq [\![\mathcal{E}_1]\!] \cap [\![\mathcal{E}_2]\!]$.

6.2 Parallel Composition

We now define a parallel composition operator for APECAs which enables modular specifications of systems. The formal definition is similar to the one for conjunction and requires again that all actions are shared. To enable interleaving of non-shared actions, we realize completeness of the edges differently. Let a be an action of $\mathcal{E}_1$ that does not occur in $\mathcal{E}_2$. We now add a to the set of actions of $\mathcal{E}_2$ and for every location l in $\mathcal{E}_2$ we add a self-loop must-edge such that $T_2(l, true, a, \varphi)$ with $Sat(\varphi) = \{\mu_l\}$, where μ_l is the point distribution on l. This is repeated for all non-shared actions of $\mathcal{E}_2$ (symmetrically for $\mathcal{E}_1$) until $\mathcal{E}_1$ and $\mathcal{E}_2$ have the same set of actions. This simplifies the definition of the parallel composition (and our proofs for related theorems) since interleaved and synchronized behavior can be uniformly treated using a single composition rule.

Definition 14 [APECA Parallel Composition] Given two APECAs $\mathcal{E}_1 = (L_1, A, X_A, AP_1, V_1, T_1, l_0^1)$, $\mathcal{E}_2 = (L_2, A, X_A, AP_2, V_2, T_2, l_0^2)$ with $AP_1 \cap AP_2 = \emptyset$. The *parallel composition* of $\mathcal{E}_1$ and $\mathcal{E}_2$, written as $\mathcal{E}_1 \parallel \mathcal{E}_2$, is defined as

$$\mathcal{E}_1 \parallel \mathcal{E}_2 = (L_1 \times L_2, A, X_A, AP_1 \cup AP_2, V_{\parallel}, T_{\parallel}, \langle l_0^1, l_0^2 \rangle) \quad \text{where}$$

- $T_{\parallel}$ is defined as follows. For all $l_1 \in L_1$, $l_2 \in L_2$, $g_1, g_2 \in CC(X_A)$, $a \in A$:
  - $\forall \varphi_1 \in PC(L_1)$ and $\forall \varphi_2 \in PC(L_2)$ such that $T_1(l_1, g_1, a, \varphi_1) \neq \bot$ and $T_2(l_2, g_2, a, \varphi_2) \neq \bot$, let $T_{\parallel}(\langle l_1, l_2 \rangle, g_1 \wedge g_2, a, \varphi_{\parallel}) = T_1(l_1, g_1, a, \varphi_1) \sqcap T_2(l_2, g_2, a, \varphi_2)$ with $\varphi_{\parallel}$ the new constraint on $\mathcal{A}_1 \parallel \mathcal{A}_2$ defined as follows: $\mu_{\parallel} \in Sat(\varphi_{\parallel})$ if and only if there exist $\mu_1 \in Sat(\varphi_1)$ and $\mu_2 \in Sat(\varphi_2)$ s.t. $\mu_{\parallel}(\langle k_1, k_2 \rangle) = \mu_1(k_1) \cdot \mu_2(k_2)$ for all $k_1 \in L_1, k_2 \in L_2$.
  - For all other $\varphi'_{\parallel} \in PC(L_1 \times L_2)$ and $g' \in CC(X_A)$ we define $T_{\parallel}(\langle l_1, l_2 \rangle, g', a, \varphi'_{\parallel}) = \bot$.
- $V_{\parallel}(\langle l_1, l_2 \rangle) = \{E_1 \cup E_2 \mid E_1 \in V_1(l_1) \wedge E_2 \in V_2(l_2)\}$ for all $l_1 \in L_1, l_2 \in L_2$.

Example 6 Fig. 4 shows the parallel composition $Cl \parallel Acc$ of the client and access controller in Fig. 3. Let φ_1 be defined as in Example 5. Let the probability from state 1' to state 0' be p_3 and from state 1' to state 1' be p_4 in Acc. Set φ_2 be $0 \leq p_3 \leq 1/2$ and $p_3 + p_4 = 1$. In $Cl \parallel Acc$, let the probability from state 11' to states 00', 01', 20' and 21' be q_1, q_2, q_3 and q_4, respectively. The resulting $\varphi_{\parallel}$ is $q_1 + q_2 + q_3 + q_4 = 1$ and $q_1 = p_1 \cdot p_3$, $q_2 = p_1 \cdot p_4$, $q_3 = p_2 \cdot p_3$ and $q_4 = p_2 \cdot p_4$.

Similarly to conjunction, parallel composition is preserved by the region construction (Lemma 3). Moreover, parallel composition interacts well with refinement (Theorem 7) and abstraction (Theorem 8). The latter result allows us to avoid state space explosion by applying abstraction component-wise instead of computing the complete system specification and then applying the abstraction. By Theorem 7 we also know that $\equiv$ is a congruence w.r.t. parallel composition.

Lemma 3 For any APECAs $\mathcal{E}_1, \mathcal{E}_2$ it holds: $\mathsf{R}(\mathcal{E}_1 \parallel \mathcal{E}_2) \equiv \mathsf{R}(\mathcal{E}_1) \parallel \mathsf{R}(\mathcal{E}_2)$.

Theorem 7 Weak refinement is a precongruence w.r.t. parallel composition.

Theorem 8 Let $\mathcal{E}_1$ and $\mathcal{E}_2$ be APECAs and α_1, α_2 be abstraction functions. The following equalities hold up to isomorphism:

1. $\mathcal{P}_{\alpha_1}(\mathcal{E}_1) \parallel \mathcal{P}_{\alpha_2}(\mathcal{E}_2) = \mathcal{P}_{\alpha_1 \times \alpha_2}(\mathcal{E}_1 \parallel \mathcal{E}_2)$;
2. $\alpha_1(\mathcal{P}_{\alpha_1}(\mathcal{E}_1)) \parallel \alpha_2(\mathcal{P}_{\alpha_2}(\mathcal{E}_2)) = (\alpha_1 \times \alpha_2)(\mathcal{P}_{\alpha_1}(\mathcal{E}_1) \parallel \alpha_2(\mathcal{P}_{\alpha_2}(\mathcal{E}_2)))$;
3. $(\alpha_1 \circ \mathcal{P}_{\alpha_1})(\mathcal{E}_1) \parallel (\alpha_2 \circ \mathcal{P}_{\alpha_2})(\mathcal{E}_2) = (\alpha_1 \times \alpha_2) \circ (\mathcal{P}_{\alpha_1 \times \alpha_2})(\mathcal{E}_1 \parallel \mathcal{E}_2)$.

7 Conclusions

We introduced a modal specification theory of abstract probabilistic timed automata (APTAs) that capture systems which contain nondeterminism, probability and time. The theory supports refinement, abstraction, and the operations of parallel composition and conjunction, the latter for independent development. The main challenge in combining probabilistic and timed behaviour was due to the need to consider guards of transitions, as well as a time-divergence sensitive consistency check. In order to obtain a compositional theory, we had to restrict to the class of abstract probabilistic event clock automata (APECAs). As future work, we aim at finding a class of modal specifications for probabilistic timed systems that, in terms of expressiveness, lies between APECAs and APTAs, but still enjoys the compositionality properties of APECAs. Moreover, we plan to extend our work on abstraction to support counterexample generation.

Acknowledgement. This work is supported by the ERC Advanced Grant VERIWARE and EU project CONNECT.

References

[1] Rajeev Alur & David L. Dill (1994): *A theory of timed automata*. *Theor. Comput. Sci.* 126(2), pp. 183–235, doi:10.1016/0304-3975(94)90010-8.

[2] Rajeev Alur, Limor Fix & Thomas A. Henzinger (1999): *Event-Clock Automata: A Determinizable Class of Timed Automata*. *Theor. Comput. Sci.* 211(1-2), pp. 253–273, doi:10.1016/S0304-3975(97)00173-4.

[3] Sebastian Bauer, Uli Fahrenberg, Axel Legay & Claus Thrane (2012): *General Quantitative Specification Theories with Modalities*. In: *Computer Science – Theory and Applications*, LNCS 7353, Springer, pp. 18–30, doi:10.1007/978-3-642-30642-6_3.

[4] Nathalie Bertrand, Axel Legay, Sophie Pinchinat & Jean-Baptiste Raclet (2012): *Modal event-clock specifications for timed component-based design*. *Science of Computer Programming* 77(12), pp. 1212–1234, doi:10.1016/j.scico.2011.01.007.

[5] Benoît Caillaud, Benoît Delahaye, Kim G. Larsen, Axel Legay, Mikkel L. Pedersen & Andrzej Wąsowski (2011): *Constraint Markov Chains*. *Theoretical Computer Science* 412(34), pp. 4373–4404, doi:10.1016/j.tcs.2011.05.010.

[6] Taolue Chen, Chris Chilton, Bengt Jonsson & Marta Z. Kwiatkowska (2012): *A Compositional Specification Theory for Component Behaviours*. In: *ESOP'12*, LNCS 7211, Springer, pp. 148–168, doi:10.1007/978-3-642-28869-2_8.

[7] Taolue Chen, Tingting Han & Joost-Pieter Katoen (2008): *Time-Abstracting Bisimulation for Probabilistic Timed Automata*. In: *TASE'08*, IEEE Computer Society, pp. 177–184, doi:10.1109/TASE.2008.29.

[8] Chris Chilton, Bengt Jonsson & Marta Kwiatkowska (2012): *Assume-Guarantee Reasoning for Safe Component Behaviours*. In: *FACS'12*, LNCS 7684, Springer, pp. 92–109, doi:10.1007/978-3-642-35861-6_6.

[9] Chris Chilton, Marta Z. Kwiatkowska & Xu Wang (2012): *Revisiting Timed Specification Theories: A Linear-Time Perspective*. In: *FORMATS'12*, LNCS 7595, Springer, pp. 75–90, doi:10.1007/978-3-642-33365-1_7.

[10] Alexandre David, Kim G. Larsen, Axel Legay, Ulrik Nyman & Andrzej Wasowski (2010): *Timed I/O automata: a complete specification theory for real-time systems*. In: *HSCC'10*, ACM, pp. 91–100, doi:10.1145/1755952.1755967.

[11] Benoît Delahaye, Joost-Pieter Katoen, Kim G. Larsen, Axel Legay, Mikkel L. Pedersen, Falak Sher & Andrzej Wasowski (2011): *Abstract Probabilistic Automata*. In: *VMCAI'11*, LNCS 6538, Springer, pp. 324–339, doi:10.1007/978-3-642-18275-4_23.

[12] Uli Fahrenberg & Axel Legay (2012): *A Robust Specification Theory for Modal Event-Clock Automata*. In: *FIT*, EPTCS 87, pp. 5–16, doi:10.4204/EPTCS.87.2.

[13] Tingting Han, Christian Krause, Marta Kwiatkowska & Holger Giese (2013): *Modal Specifications for Probabilistic Timed Systems*. Technical Report CS-RR-13-03, University of Oxford, Department of Computer Science.

[14] M. Kwiatkowska, G. Norman, R. Segala & J. Sproston (2002): *Automatic verification of real-time systems with discrete probability distributions*. *Theoretical Computer Science* 282, pp. 101–150, doi:10.1016/S0304-3975(01)00046-9.

[15] Kim Guldstrand Larsen, Ulrik Nyman & Andrzej Wasowski (2007): *On Modal Refinement and Consistency*. In: *CONCUR'07*, LNCS 4703, Springer, pp. 105–119, doi:10.1007/978-3-540-74407-8_8.

[16] Kim Guldstrand Larsen & Bent Thomsen (1988): *A Modal Process Logic*. In: *LICS'88*, IEEE Computer Society, pp. 203–210, doi:10.1109/LICS.1988.5119.

[17] Roberto Segala (1995): *Modeling and Verification of Randomized Distributed Real-Time Systems*. Ph.D. thesis, Massachusetts Institute of Technology.

[18] Falak Sher & Joost-Pieter Katoen (2012): *Compositional Abstraction Techniques for Probabilistic Automata*. In: *IFIP TCS*, LNCS 7604, Springer, pp. 325–341, doi:10.1007/978-3-642-33475-7_23.

[19] Jeremy Sproston (2009): *Strict Divergence for Probabilistic Timed Automata*. In: *CONCUR'09*, LNCS 5710, Springer, pp. 620–636, doi:10.1007/978-3-642-04081-8_41.

[20] M. Stoelinga & F. Vaandrager (1999): *Root Contention in IEEE 1394*. In: *ARTS'99*, LNCS 1601, Springer, pp. 53–74, doi:10.1007/3-540-48778-6_4.