Verifying Refinement of Probabilistic Contracts Using Timed Automata (With Proofs)*

Anton Hampus[0000−0002−3939−3919] (✉) and Mattias Nyberg

KTH Royal Institute of Technology, Stockholm, Sweden
ahampus@kth.se

Abstract. Compositional verification allows a system to be verified indirectly by verifying the individual components of the system. The key step is to ensure that the decomposition of the system specification into component specifications is correct. That is, it needs to be verified that the composition of the component specifications refines the system specification. In many cyber-physical systems, specifications are stochastic in nature. For instance, a specification might state that the probability of reaching an unsafe state within 10.000 hours shall be less than 0.05. Verifying refinement under such assumptions requires techniques beyond traditional theorem proving. The paper presents a solution where specifications are built up by probabilistic contracts based upon timed automata. In particular, the main contribution is an algorithm for verifying refinement between such specifications. The algorithm utilizes a reduction to the language emptiness problem, making the algorithm terminate after a finite number of computations.

Keywords: Probabilistic Contracts · Automata · Algorithms.

1 Introduction

Given a top-level specification ϕ and a system S made up of components $S_1, \ldots S_m$, the goal is to verify that S satisfies ϕ. For a scalable solution to this problem, the principle of *compositional verification* [28] has been proposed. The top-level specification ϕ is first *decomposed* into component specifications $\phi_1, \ldots \phi_m$. Applying the principle compositional verification then consists of verifying that (1) each component S_i implements its specification ϕ_i, and (2) the composition of the component specifications $\phi_1, \ldots \phi_m$ *refines* the top-level specification ϕ. The key difficulty, and the topic of the present paper, is (2).

We consider the scope of general cyber-physical systems, which encapsulates both safety and cyber-security aspects. To accurately represent and reason about these systems, they are assumed to progress in continuous time with continuous state spaces. Based on formal logic, there already exist several frameworks that allow expressing specifications and verifying refinement between them, see e.g. [7, 22, 25, 30]. However, they give no support for reasoning about probabilistic or stochastic behaviors. This is a major limitation considering the stochastic nature of many cyber-physical, in particular within safety where specifications typically set limits

* Supported by Vinnova FFI through the SafeDim project.

on the probability of undesired events to occur within certain time intervals. For instance, a specification might state that the probability of reaching an unsafe state within 10.000 hours shall be less than 0.05. Also of interest are specifications of software using probabilistic algorithms or running in a probabilistic setting.

Recently, a new trace-based specification theory for stochastic systems has been proposed [15]. Here, *behaviors* of components are expressed as probability measures on trace sets, and specifications are represented as sets of such behaviors. As is usual in specification theories [13], the theory in [15] enables *compositions* of behaviors, *compositions* of specifications, reasoning about behaviors *implementing* a specification, and a specification *refining* another. These are all crucial for verifying decompositions. Based on *assume-guarantee contracts* [5, 21, 32], the syntax of these specifications is built up by *probabilitic contracts* $\mathcal{P}_{>p}(\mathcal{A}, \mathcal{G})$ that give a lower bound p for a linear-time property $\mathcal{G}$, called *guarantee*, given a linear-time property $\mathcal{A}$, called *assumption.*

The two major limitations with the theory in [15] is that (1) it assumes a discrete state space, and (2) the procedure for verifying refinement is not guaranteed to terminate due to an explicit enumeration of infinite sets of paths. To solve (1), the present paper proposes, as a first contribution, an extension to the theory in [15] based on predicates enabling support for continuous state spaces. To solve (2), as the main contribution of the paper, a new algorithm is proposed for verifying refinement using reduction to the well-studied language emptiness problem for timed automata. The result is that an answer can be found in finite time. Furthermore, the present paper augments the syntax of specifications to facilitate more general formulae, and is now based on the well-studied timed Muller automata [2], allowing the reuse of theories and algorithms applying to that class.

The paper is organized as follows. Section 2 recalls the specification theory of [15]. In Section 3, the new type of *predicates* are defined and connected to deterministic timed Muller automata (DTMAs). Section 4 presents the syntax of specifications based on DTMAs. Section 5 presents the new algorithm for verifying refinement between specifications and Section 6 demonstrates the algorithm on an example. Lastly, Section 7 surveys related work and Section 8 concludes the paper.

2 Stochastic Behaviors

In the present paper, we will represent the current state of any given system using a set of *variables.* As a system changes state over time, the variables change values accordingly. Assume a fixed non-empty finite set X of variables, each $x \in X$ ranging over a (possibly uncountable) non-empty set $\texttt{range}(x)$. As part of the first contribution of the paper, we allow these ranges to be arbitrary sets. For instance, variables can range over the integers or reals, thereby permitting both discrete and continuous state spaces. Besides introducing *continuous traces* that capture this generality, the rest of the section recalls important definitions from [15].

2.1 Continuous Traces

The intuition behind a *continuous trace* is to represent how the variables in X change values over time, in a way similar to *signals* of [18, 19]. A continuous trace is a function mapping each point in time to the current values assigned to the variables. These value assignments are called *valuations.* Formally, given a set $E \subseteq X$ of variables, a *valuation for* E is a function $\nu^E : E \to \bigcup_{x_i \in E} \texttt{range}(x_i)$ mapping each $x_i \in E$ to a value in $\texttt{range}(x_i)$. Let $\texttt{val}(E)$ denote the set of all possible valuations for E.

Definition 1 (Continuous Trace). *Given a set $E \subseteq X$ of variables, a* continuous trace for E *is a function $\theta : \mathbb{R}_{\geq 0} \to \texttt{val}(E)$ mapping each time-point t to a valuation for E.*

In what follows, continuous traces will be referred to simply as *traces*, for short. Let $\texttt{tr}(E)$ denote the set of all possible traces for E. By convention, let $\texttt{tr}(\emptyset) = \emptyset$. Furthermore, for any trace $\theta \in \texttt{tr}(E)$ and set $E' \subseteq E$ of variables, let $\theta|_{E'}$ denote the projection $\theta' : \mathbb{R}_{\geq 0} \to \texttt{val}(E')$ such that $\forall t \in \mathbb{R}_{\geq 0} \,.\, \forall x \in E' \,.\, \theta'(t)(x) = \theta(t)(x)$. To define *trace composition*, consider two traces θ_1 and θ_2 for disjoint sets of variables E_1 and E_2, respectively. The *composition of* θ_1 *and* θ_2 is the trace $\theta_1 \parallel \theta_2 : \mathbb{R}_{\geq 0} \to \texttt{val}(E_1 \cup E_2)$ such that $(\theta_1 \parallel \theta_2)(t)(x)$ equals $\theta_i(t)(x)$ whenever $x \in E_i$, $i \in \{1, 2\}$.

2.2 Input/Output Behaviors

Before we can define *input/output behaviors*, we must first define "ordinary" *behaviors.* In short, these are just probability measures on traces, similar to *trace distributions* of [29, 31] but over continuous time and general state spaces. For the following definition of behavior, we assume, for each set $E \subseteq X$, a fixed σ-algebra σ_E such that all trace sets considered in the rest of the paper are measurable. Fore more details, see [15].

Definition 2 (Behavior). *Given a non-empty set of variables $E \subseteq X$, a* behavior over E *is a probability measure defined on σ_E.*

Let $\texttt{beh}(E)$ denote the set of all possible behaviors over a non-empty $E \subseteq X$. Since system components often have control over some variables while being dependent on others, a more general notion of behavior is needed to capture this. These are the so-called *input/output behaviors.* As the name suggests, they dependent on traces over the input variables and describe the probabilities of the output traces. More precisely, they are functions mapping each possible input trace to an "ordinary" behavior over the output variables.

Definition 3 (Input/Output Behavior). *Given two disjoint sets of variables $I \subseteq X$ and $O \subseteq X$, where O is non-empty, an* input/output behavior from I to O *is a function $\beta : \texttt{tr}(I) \to \texttt{beh}(O)$.*

Let $\texttt{beh}(I, O)$ denote set of all possible I/O behaviors from a set I to a non-empty set O. Given an I/O behavior β from I to O, we call I the set of *input variables*, denoted $\texttt{in}(\beta)$, and O the set of *output variables*, denoted $\texttt{out}(\beta)$.

To be able to compose behaviors, they are assumed to be *compatible.* Formally, two I/O behaviors β_1 and β_2 are said to be *compatible* if $\texttt{in}(\beta_1) = \emptyset$ and $\texttt{in}(\beta_2) = \texttt{out}(\beta_1)$. Note that, although it is possible to create a more general definition of compatibility, these assumptions are made for simplicity.

Definition 4 (Composition of I/O Behaviors). *The composition of two compatible I/O behaviors β_1 and β_2, denoted $\beta_1 \parallel \beta_2$, is the unique I/O behavior from $\emptyset$ to $\texttt{out}(\beta_1) \cup \texttt{out}(\beta_2)$, i.e. the probability measure*

$$\beta_1 \parallel \beta_2 \in \texttt{beh}(\texttt{out}(\beta_1) \cup \texttt{out}(\beta_2)) \ ,$$

induced by

$$\beta_1 \parallel \beta_2(\Theta_1 \times \Theta_2) = \int_{\Theta_1} \beta_2(\theta_1)(\Theta_2)\beta_1(d\theta_1) \ .$$

Note that composing two compatible behaviors results in yet another behavior with no input variables.

2.3 Specifications

Intuitively, we may think of a specification as a set of "allowed" I/O behaviors. However, instead of saying that an I/O behavior β is allowed by a specification Σ, we use the terminology that β *implements* Σ.

Definition 5 (Specification). *Given two disjoint sets of variables $I \subseteq X$ and $O \subseteq X$ such that O is non-empty, a* specification Σ from I to O *is a subset of the I/O behaviors $\texttt{beh}(I, O)$, i.e. $\Sigma \subseteq \texttt{beh}(I, O)$.*

An I/O behavior β from I to O is said to *implement* a specification Σ from I to O if $\beta \in \Sigma$. The key notion of the present paper is *refinement* between specifications. A specification Σ refines a specification Σ' if each behavior implementing Σ also implements Σ'. This is captured by the following definition.

Definition 6 (Refines). *A specification Σ_1 from I to O* refines *a specification Σ_2 from I to O if $\Sigma_1 \subseteq \Sigma_2$.*

Not unlike I/O behaviors, specifications for two different components can be composed to form a single specification for the component composition. This is done on the behavior level, by composing each possible pair of behaviors taken from the two specifications. Extending the notion of compatible behaviors, two specifications Σ_1 and Σ_2 are said to be *compatible* if $\texttt{in}(\Sigma_1) = \emptyset$ and $\texttt{in}(\Sigma_2) = \texttt{out}(\Sigma_1)$.

Definition 7 (Parallel Composition of Specifications). *Given two compatible specifications Σ_1 and Σ_2, the* parallel composition of Σ_1 and Σ_2, *denoted $\Sigma_1 \parallel \Sigma_2$, is the specification $\Sigma_1 \parallel \Sigma_2 = \{\beta_1 \parallel \beta_2 \mid \beta_1 \in \Sigma_1, \beta_2 \in \Sigma_2\}$.*

3 Timed Automata

In the present paper, timed automata as defined by [2, 3] are used as a way to represent sets of traces. As we will see later, putting probability bounds on the trace sets of such automata serves as a representation for the type of specifications defined in Section 2.3.

3.1 Timed Predicate Sequences

In the present paper, we represent time using a *continuous semantics* [3, 4, 19] as opposed to the *point-wise semantics* of e.g. timed words used in [1, 2]. For a comparison between the two, see e.g. [12, 16, 26]. Runs of an automaton thus generate *timed predicate sequences*, which are a simple extension of the timed state sequences of [3]. More precisely, instead of atomic propositions, we will utilize a more general notion of *predicates* over variables in X.

In accordance with [3], a *time interval* $I \subseteq \mathbb{R}_{\geq 0}$ is a subset of the non-negative real numbers, taking the form (a, b), $(a, b]$, $[a, b)$ or $[a, b]$ where $a, b \geq 0$ such that $a \leq b$ if $I = [a, b]$ and $a < b$ otherwise. For any interval $I \subseteq \mathbb{R}_{\geq 0}$, let $l(I)$ denote the left endpoint of I and $r(I)$ denote the right endpoint of I. Two intervals I_1 and I_2 are said to be *adjacent* if $r(I_1) = l(I_2)$, and either I_1 is right-closed and I_2 is left-open or I_1 is right-open and I_2 is left-closed.

Also in accordance with [3], we now define *interval sequences* partitioning the real timeline. Let $\mathbb{N}$ denote the non-negative integers.

Definition 8 (Interval Sequence [3]). *An* interval sequence *is an infinite sequence of intervals $I_0 I_1 I_2 \dots$ partitioning the non-negative real line such that (1) $0 \in I_0$, (2) for each $i \in \mathbb{N}$, I_i and I_{i+1} are adjacent, and (3) the intervals cover all of $\mathbb{R}_{\geq 0}$, i.e. $\bigcup_{i \in \mathbb{N}} I_i = \mathbb{R}_{\geq 0}$.*

To express boolean statements about the values of variables, for instance that x_1 lies within some interval or x_2 is a singleton set, we now define *predicates* over subsets of X. These facilitate the first contribution of the paper, allowing truth statements about both discrete and continuous state spaces. As will be presented later, these predicates can be coupled with timed automata to represent sets of continuous traces.

Definition 9 (Predicate). *Given a set $E \subseteq X$ of variables, a* predicate over E *is a function $q : \mathtt{val}(E) \to \{\mathtt{true}, \mathtt{false}\}$ mapping each valuation for E to a truth value.*

Given a predicate q over some set E of variables, let $\mathtt{var}(q)$ denote the set E. Throughout the rest of the paper, consider a fixed non-empty finite predicate set $\boldsymbol{\mathcal{Q}}$. We lift the above notation also to predicate sets, so that whenever $Q \subseteq \boldsymbol{\mathcal{Q}}$, $\mathtt{var}(Q) = \bigcup_{q \in Q} \mathtt{var}(q)$.

Definition 10 (Timed Predicate Sequence). *A* timed predicate sequence *is a pair $\tau = (\bar{Q}, \bar{I})$ where $\bar{Q} = Q_0 Q_1 Q_2 \dots$ is an infinite sequence of predicate sets $Q_i \subseteq \boldsymbol{\mathcal{Q}}$ and $\bar{I} = I_0 I_1 I_2 \dots$ is an interval sequence.*

Intuitively, in any timed predicate sequence $(Q_0 Q_1 Q_2 \dots, I_0 I_1 I_2 \dots)$, each set Q_i represents the predicates that are true throughout the corresponding interval I_i. Given a timed predicate sequence $\tau = (\bar{Q}, \bar{I})$, let $\tau^* : \mathbb{R}_{\geq 0} \to 2^{\boldsymbol{\mathcal{Q}}}$ be the function such that $\tau^*(t) = Q_i$ for each $i \in \mathbb{N}$ and $t \in I_i$.

3.2 Deterministic Timed Muller Automata

Throughout the rest of the paper, we make use of a subclass of timed automata that suits our specific needs. These are the so-called *deterministic timed Muller automata* (DTMAs). Although strictly less expressive than their non-deterministic counterpart, DTMAs are closed under both intersection and complement, rather than just intersection [2]. Since these are crucial operations in the present paper, DTMAs are a good candidate. The paper also relies heavily on the fact that the language emptiness problem for DTMAs is decidable. This, however, is the case also for the non-deterministic version [2].

Definition 11 (Clock Constraint). *Given a set C of clocks, a* clock constraint δ on C *is defined inductively by the grammar*

$$\delta ::= c \sim k \mid \delta \wedge \delta \mid \delta \vee \delta \mid \neg\delta$$

where c ranges over clocks C, $\sim \in \{<, \leq, \geq, >\}$, and k ranges over rationals $\mathbb{Q}$.

Note that a constraint `true` can be defined as an abbreviation. Given a set C of clocks, let Δ^C denote the set of all possible clock constraints on C. The reason that k ranges over rationals instead of e.g. reals is that the decidability result of [2] for the language emptiness problem relies on this fact. It is worth noting that this is not an assumption made on the nature of traces, but rather a restriction on the timing constraints that the automata can specify. This is a reasonable restriction since any real number can be approximated to arbitrary precision by a rational number.

Given a set C of clocks, a *clock valuation for C* is a function $\nu^C : C \to \mathbb{R}_{\geq 0}$ mapping each clock $c \in C$ to a non-negative real number $\nu^C(c)$. For any number $t \in \mathbb{R}_{\geq 0}$, let $\nu^C + t$ be the clock valuation for C assigning each clock $c \in C$ to $\nu^C(c) + t$. Given a function f and two values a and v, let $f[a \mapsto v]$ denote the function mapping a to v and agreeing with f on all other values. Extending this to sets $A = \{a_1, \dots, a_k\}$, let $f[A \mapsto v]$ be shorthand for $f[a_1 \mapsto v] \dots [a_k \mapsto v]$. For example, if f is the constant function mapping each element to 0, then $f[A \mapsto 1]$ is the indicator function $\mathbf{1}_A$.

Definition 12 (Satisfy Clock Constraint). *Given a set $C = \{c_1, \dots, c_m\}$ of clocks, a clock constraint δ on C and a clock valuation ν^C for C, the valuation ν^C* satisfies δ *if $\delta[c_1 \mapsto \nu^C(c_1)] \dots [c_m \mapsto \nu^C(c_m)]$ interprets to* `true` *in the usual logic sense.*

The following definition introduces *predicate constraints*. These are formulae built up by predicates and logic connectives, and serve as a more flexible way to express complex predicates from simpler ones.

Definition 13 (Predicate Constraint). *A predicate constraint ψ is defined inductively by the grammar*

$$\psi ::= q \mid \psi \wedge \psi \mid \psi \vee \psi \mid \neg\psi$$

where q ranges over predicates $\boldsymbol{Q}$.

Once again, a constraint `true` can be defined as an abbreviation. Let Ψ denote the set of all possible predicate constraints.

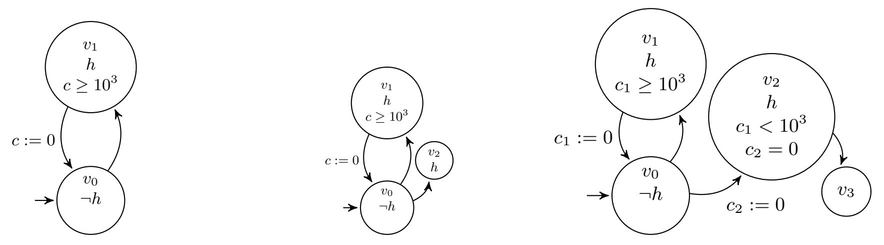
Definition 14 (Satisfy Predicate Constraint). *A predicate set Q* satisfies *a predicate constraint ψ if the formula obtained from ψ by substituting* `true` *for each predicate $q \in Q$ and* `false` *for each predicate $q \in \mathfrak{Q} \setminus Q$ evaluates to* `true` *in the usual logic sense.*

To define timed Muller automata and their semantics, we will closely follow [3], which also use a continuous semantics. However, besides requiring a singleton initial location for determinism reasons, the difference is that we replace propositional constraints by predicate constraints.

Definition 15 (Timed Muller Automaton). *A* timed Muller automaton (TMA) *is a tuple $\mathcal{M} = \langle V, v_0, C, \alpha, \beta, \rightarrow, \mathcal{F} \rangle$ where*

- *V is a non-empty finite set of* locations*,*
- *$v_0 \in V$ is the* initial location*,*
- *C is a non-empty finite set of clocks,*
- *$\alpha : V \rightarrow \Psi$ is a location labeling function associating each $v \in V$ to a* predicate constraint *$\psi \in \Psi$,*
- *$\beta : V \rightarrow \Delta^C$ is a location labeling function associatibng each $v \in V$ to a clock constraint $\delta \in \Delta^C$,*
- *$\rightarrow \subseteq V \times 2^C \times V$ is a set of* transitions *where each $(v, R, v') \in \rightarrow$ consists of a* source *v, a set R of* reset clocks*, and a* destination *v',*
- *and $\mathcal{F} \subseteq 2^V$ is the* acceptance family*.*

For notational convenience, unless otherwise stated, each location v is assumed to have a self-transition of the form $(v, \emptyset, v)$. This is done to enable the automaton to generate infinite runs even when no other outgoing transitions exist. An example TMA is given in Figure 1a. It has two locations, a clock c, two predicate constraints h and $\neg h$, a clock constraint $c \geq 10^3$, two drawn transitions, and two implicit self-transitions.



(a) Deterministic (b) Complete (c) Deterministic and complete

Fig. 1: Timed Muller automata describing an overheating motor. Omitted constraints are to be understood as constraints that always evaluate to true.

We extend the notation `var`$(\cdot)$ such that, given a TMA $\mathcal{M} = \langle V, v_0, C, \alpha, \beta, \rightarrow, \mathcal{F} \rangle$, `var`$(M)$ denotes the set of all variables E such that there appears some predicate

q in α for which $\texttt{var}(q) \subseteq E$. The semantics of TMAs is defined in terms of *runs*, which represent stepping through the transitions of an automaton subject to the clock constraints. The definition presented here follows closely that of [3].

Definition 16 (Run). *Given a timed Muller automaton $\mathcal{M} = \langle V, v_0, C, \alpha, \beta, \rightarrow, \mathcal{F}\rangle$, a run ρ of $\mathcal{M}$ is an infinite sequence*

$$\rho = \xrightarrow[\nu_0^C]{} (v_0, I_0) \xrightarrow[\nu_1^C]{R_1} (v_1, I_1) \xrightarrow[\nu_2^C]{R_2} (v_2, I_2) \xrightarrow[\nu_3^C]{R_3} \cdots$$

where $\nu_i^C \in \texttt{val}(C)$ are clock valuations, $v_i \in V$ are locations, $I_0 I_1 I_2 \ldots$ is an interval sequence, and $R_i \subseteq C$ are clock sets, such that

- *for each clock $c \in C$, $\nu_0^C(c) = 0$,*
- *for each $i \in \mathbb{N}$, there exists a transition (v_i, R_{i+1}, v_{i+1}) in $\rightarrow$ that resets the clocks R_{i+1}, i.e. such that $\nu_{i+1}^C = (\nu_i^C + (r(I_i) - l(I_i)))[R_{i+1} \mapsto 0]$,*
- *for each $i \in \mathbb{N}$ and $t \in I_i$, $\nu_i^C + (t - l(I_i))$ satisfies $\beta(v_i)$.*

Given a run ρ, let $\texttt{inf}(\rho)$ denote the set of locations visited an infinite number of times in ρ, i.e. the locations $v \in V$ such that $v = v_i$ for infinitely many i. Using similar notation as for timed predicate sequences, let ρ^* denote the function associating each time with the current location in the run ρ. That is, given a run $\rho = \xrightarrow[\nu_0^C]{} (v_0, I_0) \xrightarrow[\nu_1^C]{R_1} (v_1, I_1) \xrightarrow[\nu_2^C]{R_2} (v_2, I_2) \xrightarrow[\nu_3^C]{R_3} \cdots$ of a TMA $\mathcal{M} = \langle V, v_0, C, \alpha, \beta, \rightarrow, \mathcal{F}\rangle$, $\rho^* : \mathbb{R}_{\geq 0} \rightarrow V$ is the function such that $\rho^*(t) = v_i$ for each $i \in \mathbb{N}$ and $t \in I_i$.

In the following definitions, we connect automata to corresponding sets of *accepted* timed predicate sequences. In short, a timed predicate sequence τ is accepted by an automaton $\mathcal{M}$ if it is *consistent with* some run ρ of $\mathcal{M}$ with respect to the acceptance family and the constraints of the visited locations. Formally, a timed predicate sequence τ is *consistent with* a run ρ if for each $i \in \mathbb{N}$ and time $t \in I_i$, the predicate set $\tau^*(t)$ satisfies $\alpha(\rho^*(t))$.

Definition 17 (Accepts). *A timed Muller automaton $\mathcal{M} = \langle V, v_0, C, \alpha, \beta, \rightarrow, \mathcal{F}\rangle$* accepts *a timed predicate sequence τ if τ is consistent with some run ρ of $\mathcal{M}$ such that $\texttt{inf}(\rho) \in \mathcal{F}$.*

Through this notion of acceptance, we connect automata to the variables X, and, as a further step, in Section 4, we will extend acceptance also to traces, completing the goal of using automata to represent trace sets. Given a TMA $\mathcal{M}$, let $L(\mathcal{M})$ denote the set of all timed predicate sequences accepted by $\mathcal{M}$.

Example 1 (Overheating Motor). Consider a motor that may *overheat*, exceeding 200 °C. A possible requirement is that the motor shall not overheat within the first 10^3 s and, whenever overheated, shall not overheat again within 10^3 s. Let x be a variable representing the current temperature and h be a predicate evaluating to true if and only if $x \geq 200$. A TMA describing this requirement is shown in Figure 1a. It has two locations and a single clock c keeping track of the continuous duration of being below 200 °C. The acceptance family is $\mathcal{F} = \{\{v_0\}, \{v_0, v_1\}\}$, accepting exactly the timed predicate sequences for which either (a) there exists a point after which the motor is never overheated, or (b) the motor alternates between overheating and cooling, and is never overheated more often than every 10^3 s. Note that omitting the set $\{v_0\}$ from $\mathcal{F}$ neglects case (a).

3.3 Operations on Timed Automata

To support the later presented algorithm for the refinement problem, we now introduce the two operations *intersection* and *complement* for timed automata. As the name suggests, the language accepted by the intersection of two automata is precisely the intersection of their respective languages. Likewise, the language of the complement of an automaton is the complement of its language. Since the languages accepted by non-deterministic TMAs are not closed under complement, we first define *deterministic* TMAs, or *DTMAs* for short.

Definition 18 (Deterministic TMA). *A timed Muller automaton $\mathcal{M} = \langle V, v_0, C, \alpha, \beta, \rightarrow, \mathcal{F}\rangle$ is* deterministic *if for each location $v \in V$, predicate set $Q \subseteq \mathfrak{Q}$ and clock valuation $\nu^C \in \texttt{val}(C)$, whenever there exist locations $v', v'' \in V$ such that*

- *Q satisfies both $\alpha(v')$ and $\alpha(v'')$,*
- *there exist transitions (v, R', v') and (v, R'', v'') for some sets R' and R'',*
- *$\nu^C[R' \mapsto 0]$ satisfies $\beta(v')$, and $\nu^C[R'' \mapsto 0]$ satisfies $\beta(v'')$,*

then $v' = v''$.

Figure 1a and 1c are two examples of DTMAs. The automaton of Figure 1b is not deterministic since whenever the automaton is at location v_0, $c > 10^3$, and h turns true, there are two possible locations to enter nondeterministically.

In the present paper, the viewpoint is that runs *generate* timed predicate sequences. However, it is also possible to view it in the other direction, so that an automaton instead *reads* a timed predicate sequence and generates a corresponding run. Taking this view, it is possible that a given DTMA $\mathcal{M}$, while reading a timed predicate sequence and constructing a corresponding run, gets "stuck" when arriving at a predicate for which there exists no corresponding neighboring location. This is not a problem semantically, and simply means, in our context, that there exists no run generateing the sequence. However, while not necessarily impossible, such "stuck" sequences make it cumbersome to define the complement and intersection operators for automata. It is moreover always possible to construct an equivalent automaton, said to be *complete*, and preserving determinism, in which sequences never get stuck. For such a construction, see e.g. [2]. In short, it is done by adding new "trap" locations and corresponding transitions, through which all otherwise stuck sequences are lead. Thus, without loss of generality, the automata of the present paper are henceforth assumed to be complete. However, we usually omit to explicitly represent trap locations and corresponding transitions to make the illustrations more compact. In the present context, completeness is captured by the following definition.

Definition 19 (Complete DTMA). *A DTMA $\mathcal{M} = \langle V, v_0, C, \alpha, \beta, \rightarrow, \mathcal{F}\rangle$ is* complete *if for each timed predicate sequence τ, there exists a run ρ of $\mathcal{M}$ such that τ is consistent with ρ.*

Figure 1b and 1c are two examples of complete timed Muller automata. The automaton of Figure 1a is not complete because whenever the automaton is at location v_0, $c < 10^3$, and h turns true, there is no location to enter.

Now, the intersection operator for complete DTMAs is based on parallel composition as defined in [3] and the intersection construction presented in [2].

Definition 20 (Intersection of Complete DTMA). *Let $\mathcal{M}_1 = \langle V_1, v_{01}, C_1, \alpha_1, \beta_1, \rightarrow_1, \mathcal{F}_1\rangle$ and $\mathcal{M}_2 = \langle V_2, v_{02}, C_2, \alpha_2, \beta_2, \rightarrow_2, \mathcal{F}_2\rangle$ be two complete DTMAs such that $V_1 \cap V_2 = \emptyset$ and $C_1 \cap C_2 = \emptyset$.* The intersection of $\mathcal{M}_1$ and $\mathcal{M}_2$, *denoted $\mathcal{M}_1 \sqcap \mathcal{M}_2$, is the TMA $\mathcal{M}_1 \sqcap \mathcal{M}_2 = \langle V_1 \times V_2, (v_{01}, v_{02}), C_1 \cup C_2, \alpha, \beta, \rightarrow, \mathcal{F}\rangle$ where*

- *for each $(v_1, v_2) \in V_1 \times V_2$, $\alpha((v_1, v_2)) = \alpha_1(v_1) \wedge \alpha_2(v_2)$,*
- *for each $(v_1, v_2 \in V_1 \times V_2)$, $\beta((v_1, v_2)) = \beta_1(v_1) \wedge \beta_2(v_2)$,*
- *$\rightarrow$ is the smallest set that contains, for each pair of transitions $(v_1, R_1, v_1') \in \rightarrow_1$ and $(v_2, R_2, v_2') \in \rightarrow_2$, the three transitions $((v_1, v_2), R_1 \cup R_2, (v_1', v_2'))$, $((v_1, v_2), R_1, (v_1', v_2))$ and $((v_1, v_2), R_2, (v_1, v_2'))$,*
- *the acceptance family $\mathcal{F}$ is the set $\boldsymbol{\mathcal{F}}^1 \cap \boldsymbol{\mathcal{F}}^2$, where*

$$\boldsymbol{\mathcal{F}}^1 = \{F \subseteq V_1 \times V_2 \mid \{v_1 \in V_1 \mid \exists v_2 \in V_2 \ .\ (v_1, v_2) \in F\} \in \mathcal{F}_1\},$$

$$\boldsymbol{\mathcal{F}}^2 = \{F \subseteq V_1 \times V_2 \mid \{v_2 \in V_2 \mid \exists v_1 \in V_1 \ .\ (v_1, v_2) \in F\} \in \mathcal{F}_2\}.$$

Definition 21 (Complement of Complete DTMA [2]). *Let $\mathcal{M} = \langle V, v_0, C, \alpha, \beta, \rightarrow, \mathcal{F}\rangle$ be a DTMA.* The complement of $\mathcal{M}$, *denoted $\mathcal{M}^c$, is the DTMA $\mathcal{M}^c = \langle V, v_0, C, \alpha, \beta, \rightarrow, 2^L \setminus \mathcal{F}\rangle$.*

Proposition 1. *The class of complete DTMAs is closed under intersection and complement.*

Proof. Starting with complement. Since $\mathcal{M}$ and $\mathcal{M}^c$ only differ in the acceptance family and not in the structure, completeness and determinism are obviously preserved.

Now for intersection. Assume that $\mathcal{M}_1$ and $\mathcal{M}_2$ are complete DTMAs. Since $\mathcal{M}_1$ and $\mathcal{M}_2$ are both complete and deterministic, then any timed predicate sequence τ has exactly one run ρ_1 of $\mathcal{M}_1$ and one run ρ_2 of $\mathcal{M}_2$. Since $\mathcal{M}_1 \sqcap \mathcal{M}_2$ has exactly one transition for every possible combination of transitions from $\mathcal{M}_1$ and $\mathcal{M}_2$, it follows that there exists a run ρ of $\mathcal{M}$ with $\rho^*(t) = (\rho_1^*(t), \rho_2^*(t))$. The timed predicate sequence τ is therefore also consistent with ρ, implying that $\mathcal{M}_1 \sqcap \mathcal{M}_2$ is complete.

We must now prove that, given that $\mathcal{M}_1$ and $\mathcal{M}_2$ are deterministic, then so is $\mathcal{M}_1 \sqcap \mathcal{M}_2$. Let $(v_1, v_2) \in V_1 \times V_2$, $Q \subseteq \boldsymbol{\mathcal{Q}}$, $\nu^C \in \mathtt{val}(C)$, $(v_1', v_2') \in V_1 \times V_2$, and $(v_1'', v_2'') \in V_1 \times V_2$ such that

- Q satisfies both $\alpha((v_1', v_2'))$ and $\alpha((v_1'', v_2''))$. It must then be the case that Q satisfies v_1', v_2', v_1'', v_2''.
- There exist transitions $((v_1, v_2), R', (v_1', v_2'))$ and $((v_1, v_2), R', (v_1'', v_2''))$. These transitions must have come from transitions (including self-transitions) (v_1, R', v_1') and (v_2, R', v_2') of M_1 and M_2, respectively.
- $\nu^C[R' \mapsto 0]$ satisfies $\beta((v_1', v_2'))$ and $\nu^C[R'' \mapsto 0]$ satisfies $\beta((v_1'', v_2''))$. It must then be the case that $\nu^{C_1}[R' \mapsto 0]$ satisfies $\beta_1(v_1')$, $\nu^{C_1}[R'' \mapsto 0]$ satisfies $\beta_1(v_1'')$, $\nu^{C_2}[R' \mapsto 0]$ satisfies $\beta_2(v_2')$, and $\nu^{C_2}[R'' \mapsto 0]$ satisfies $\beta_2(v_2'')$.

Due to these implications, it follows from Definition 18 that $v_1' = v_1''$ and $v_2' = v_2''$. Thus, $(v_1', v_2') = (v_1'', v_2'')$, and once again from Definition 18, $\mathcal{M}_1 \sqcap \mathcal{M}_2$ is deterministic. □

Proposition 2. *Suppose that $\mathcal{M}_1$ and $\mathcal{M}_2$ are complete DTMAs. Then $L(\mathcal{M}_1 \sqcap \mathcal{M}_2) = L(\mathcal{M}_1) \cap L(\mathcal{M}_2)$.*

Proof. Consider arbitrary complete DTMAs $\mathcal{M}_1$ and $\mathcal{M}_2$ and an arbitrary timed predicate sequence τ. We will show that τ is accepted by $\mathcal{M}_1$ and $\mathcal{M}_2$ if and only if τ is accepted by $\mathcal{M}_1 \sqcap \mathcal{M}_2$.

Beginning with right implication, assume that τ accepted by both $\mathcal{M}_1$ and $\mathcal{M}_2$. That means that there exists a run ρ_1 of $\mathcal{M}_1$ and a run ρ_2 of $\mathcal{M}_2$ such that τ is consistent with ρ_1 and ρ_2 and $\texttt{inf}(\rho_1) \in \mathcal{F}_1$ and $\texttt{inf}(\rho_2) \in \mathcal{F}_2$. Because also $\mathcal{M}_1 \sqcap \mathcal{M}_2$ is complete, there exists a run ρ of $\mathcal{M}_1 \sqcap \mathcal{M}_2$ corresponding to the individual runs ρ_1 and ρ_2. That is, for each time t, $\rho^*(t) = (\rho_1^*(t), \rho_2^*(t))$. The last step is to show that $\texttt{inf}(\rho) \in \boldsymbol{\mathcal{F}^1} \cap \boldsymbol{\mathcal{F}^2}$. Since $\texttt{inf}(\rho_i) \in \mathcal{F}_i$, $i = 1, 2$, it follows that $\texttt{inf}(\rho) \in \boldsymbol{\mathcal{F}^i}$, $i = 1, 2$. Thus, $\texttt{inf}(\rho) \in \boldsymbol{\mathcal{F}^1} \cap \boldsymbol{\mathcal{F}^2}$.

Now for left implication. Assume that τ accepted by $\mathcal{M}_1 \sqcap \mathcal{M}_2$. Then there exists a run ρ with $\texttt{inf}(\rho) \in \boldsymbol{\mathcal{F}^1} \cap \boldsymbol{\mathcal{F}^2}$. Because $\mathcal{M}_1$ and $\mathcal{M}_2$ are complete, there exist corresponding runs ρ_1 and ρ_2 with $\rho^*(t) = (\rho_1^*(t), \rho_2^*(t))$. We must now show that $\texttt{inf}(\rho_1) \in \mathcal{F}_1$ and $\texttt{inf}(\rho_2) \in \mathcal{F}_2$. Since $\texttt{inf}(\rho) \in \boldsymbol{\mathcal{F}^1} \cap \boldsymbol{\mathcal{F}^2}$, it follows that $\texttt{inf}(\rho) \in \boldsymbol{\mathcal{F}^i}$ for $i = 1, 2$, and furthermore that $\texttt{inf}(\rho_i) \in \mathcal{F}_i$ for $i = 1, 2$. □

Proposition 3. *Suppose that $\mathcal{M}$ is a complete DTMA. Then $L(\mathcal{M}^c) = L(\mathcal{M})^c$.*

Proof. Consider an arbitrary complete DTMA $\mathcal{M}$ and an arbitrary timed predicate sequence τ. We must show that τ is accepted by exactly one of $\mathcal{M}$ and $\mathcal{M}^c$.

Since the set $\texttt{inf}(\tau)$ is an element of 2^L, which is spanned by $\mathcal{F}$ and $2^L \setminus \mathcal{F}$ together, $\texttt{inf}(\tau)$ must be an element of at least one of $\mathcal{F}$ and $2^L \setminus \mathcal{F}$, implying that τ is accepted by at least one of $\mathcal{M}$ and $\mathcal{M}^c$. Furthermore, since the sets $\mathcal{F}$ and $2^L \setminus \mathcal{F}$ are disjoint, τ is accepted by at most one of $\mathcal{M}$ and $\mathcal{M}^c$. □

4 Specifying Behaviors Using Timed Automata

In this section, we will define a specification language and accompanying semantics that allows us to instantiate the type of specification from Definition 5. This language, defined using a formal grammar, consists of so-called *Probabilistic Contract Formulae* (PCFs). The basic building-block is a probabilistic assume-guarantee contract, or *probabilistic contract* for short, of the form $\mathcal{P}_{\sim p}(\mathcal{A}, \mathcal{G})$, consisting of an *assumption* $\mathcal{A}$, *guarantee* $\mathcal{G}$ and *probability bound* p. Both $\mathcal{A}$ and $\mathcal{G}$ represent linear-time properties, and are given as DTMAs. Semantically, since each automaton represents a set of traces, a probabilistic contract represents the set of all I/O behaviors such that, whenever the assumption is satisfied, the probability of the guarantee respects the probability bound. These probabilistic contracts can be composed by negation ($\neg$), parallel composition ($\|$), and conjunction ($\wedge$).

We begin by formalizing the way in which automata represent sets of traces, using the notion of *accepted trace*. First, a trace θ for X is said to be *consistent with* a timed predicate sequence τ if for each $t \in \mathbb{R}_{\geq 0}$ and $q \in \boldsymbol{\mathcal{Q}}$, the predicate application $q(\theta|_{\texttt{var}(q)}(t))$ evaluates to true if and only if $q \in \tau^*(t)$.

Definition 22 (Accepted Trace). *A TMA $\mathcal{M}$* accepts *a trace θ for X if there exists a timed predicate sequence τ such that $\mathcal{M}$ accepts τ and θ is consistent with τ.*

Given a TMA $\mathcal{M}$, let $L^*(\mathcal{M})$ denote the set of traces accepted by $\mathcal{M}$. Furthermore, if E is a set of variables such that $\mathtt{var}(\mathcal{M}) \subseteq E \subseteq X$, then let $L^E(\mathcal{M})$ denote the set $\{\theta|_E \mid \theta \in L^*(\mathcal{M})\}$ of accepted traces restricted to the variables E.

Definition 23 (Probabilistic Contract Formula). *A* probabilistic contract formula (PCF) *ϕ is defined inductively using the grammar*

$$\phi ::= P_{\sim p}(\mathcal{A}, \mathcal{G}) \mid \neg\phi \mid \phi \parallel \phi \mid \phi \wedge \phi$$

where $\sim \in \{<, \leq, \geq, >\}$ is a comparison operator, $p \in [0, 1]$ is the probability bound*, and $\mathcal{A}$ and $\mathcal{G}$, the* assumption *and* guarantee*, respectively, are complete DTMAs such that $\mathtt{var}(\mathcal{G}) \setminus \mathtt{var}(\mathcal{A}) \neq \emptyset$.*

Remark 1. Timed predicate sequences, which are an intermediate step between automata and their accepted traces, intrinsically obey the so-called *finite variability* property [3]. This property states that each bounded time interval contains at most finitely many state changes, which according to [3] is an adequate assumption for modelling discrete-state systems. However, the discreteness of timed predicate sequences is merely an abstraction put over traces, which by no means are restricted to obey finite variability. For instance, a trace over real-valued variables can generally change values uncountably many times even within bounded time intervals.

The semantics for a PCF ϕ is defined in terms of its *interpretation* $[\![\phi]\!]$, evaluating to a specification, i.e. a set of I/O behaviors. The interpretation is defined inductively following the structure of the formal grammar. Since parallel composition is only defined for compatible specifications, we require in the following definition that for each PCF of the form $\phi_1 \parallel \phi_2$, the specifications $[\![\phi_1]\!]$ and $[\![\phi_2]\!]$ are compatible. Similarly, for each PCF $\phi_1 \wedge \phi_2$, we require that ϕ_1 and ϕ_2 have the same input and output variables.

Definition 24 (PCF Interpretation). *Given a probabilistic contract $P_{\sim p}(\mathcal{A}, \mathcal{G})$, as shorthand, let I denote $\mathtt{var}(\mathcal{A})$, O denote $\mathtt{var}(\mathcal{G}) \setminus \mathtt{var}(\mathcal{A})$, and $\Theta_{\mathcal{G}}$ denote $\{\theta_O \in \mathtt{tr}(O) \mid \theta_I \parallel \theta_O \in L^{I \cup O}(\mathcal{G})\}$. The interpretation $[\![\phi]\!]$ of a PCF ϕ is defined inductively by:*

$$\begin{aligned}
[\![P_{\sim p}(\mathcal{A}, \mathcal{G})]\!] &= \begin{cases} \{\beta \in \mathtt{beh}(I, O) \mid \beta()(L^O(\mathcal{G})) \sim p\}, & I = \emptyset \\ \{\beta \in \mathtt{beh}(I, O) \mid \forall \theta_I \in L^I(\mathcal{A}) \;.\; \beta(\theta_I)(\Theta_{\mathcal{G}}) \sim p\}, & I \neq \emptyset \end{cases} \\
[\![\neg\phi]\!] &= \{\beta \in \mathtt{beh}(\mathtt{in}([\![\phi]\!]), \mathtt{out}([\![\phi]\!])) \mid \beta \notin [\![\phi]\!]\} \\
[\![\phi_1 \parallel \phi_2]\!] &= [\![\phi_1]\!] \parallel [\![\phi_2]\!] \\
[\![\phi_1 \wedge \phi_2]\!] &= [\![\phi_1]\!] \cap [\![\phi_2]\!]
\end{aligned}$$

We extend the notion of *input variables* and *output variables* also to PCFs ϕ such that $\mathtt{in}(\phi) = \mathtt{in}([\![\phi]\!])$ and $\mathtt{out}(\phi) = \mathtt{out}([\![\phi]\!])$. See Section 6 for examples of PCFs.

The refinement-verification algorithm in the next section works analogously to the language inclusion algorithm for automata, in which the top-level specification is negated and intersected with the component specifications and checked for language emptiness. For this to work, the negated specification must have no input variables and be expressed as a single probabilistic contract, rather than a general PCF. The following proposition states that, under these restrictions, negation works in the way we require.

Proposition 4. *Suppose that $\phi = P_{\sim p}(\mathcal{A}, \mathcal{G})$ is a probabilistic contract with no input variables. Then $[\![\neg\phi]\!] = [\![P_{\sim^c p}(\mathcal{A}, \mathcal{G})]\!]$, where $\sim^c$ is the complement of the operator $\sim$, so that $\leq^c$ is equivalent to $>$, $>^c$ is equivalent to $\leq$, $<^c$ is equivalent to $\geq$, and $\geq^c$ is equivalent to $<$.*

Proof. Let I denote $\texttt{in}(\phi)$ and O denote $\texttt{out}(\phi)$. Since $\phi = P_{\sim p}(\mathcal{A}, \mathcal{G})$ is a probabilistic contract without input, Definition 24 gives $[\![\phi]\!] = [\![P_{\sim p}(\mathcal{A}, \mathcal{G})]\!] = \{\beta \in \texttt{beh}(I, O) \mid \beta()(L^O(\mathcal{G})) \sim p\}$. Using this, Definition 24 furthermore gives

$$\begin{aligned}
[\![\neg\phi]\!] &= \{\beta \in \texttt{beh}(I, O) \mid \beta \notin [\![\phi]\!]\} \\
&= \{\beta \in \texttt{beh}(I, O) \mid \beta \notin [\![P_{\sim p}(\mathcal{A}, \mathcal{G})]\!]\} \\
&= \{\beta \in \texttt{beh}(I, O) \mid \beta()(L^O(\mathcal{G})) \nsim p\} \\
&= \{\beta \in \texttt{beh}(I, O) \mid \beta()(L^O(\mathcal{G})) \sim^c p\} \\
&= [\![P_{\sim^c p}(\mathcal{A}, \mathcal{G})]\!] \ .
\end{aligned}$$

. □

5 Algorithm for Verifying Refinement

In this section, we present the main contribution of the paper, namely an algorithm for verifying refinement between PCFs. To guarantee termination, we make a reduction to the well-studied language emptiness problem for timed automata. The algorithm takes two inputs ϕ and ϕ_0, each being a PCF, and outputs `true` only if ϕ refines ϕ_0. In this sense, the algorithm is sound. However, it is not complete, as will be discussed in more detail below.

To understand the algorithm, note that ϕ refines ϕ_0 if and only if there exists no behavior implementing ϕ but not ϕ_0. The key insight is that this is equivalent to the statement that no behavior implements both ϕ and $\neg\phi_0$. That is, no probability measure over the traces will respect the bounds imposed by both ϕ and $\neg\phi_0$ simultaneously. To prove that no such measure exists, we first partition the possible traces into subsets Θ_j according to the assumptions and guarantees. More precisely, for any automaton $\mathcal{M}$ being either an assumption or a guarantee of some probabilistic contract appearing in ϕ or ϕ_0, either all or no traces in Θ_j satisfy $\mathcal{M}$. We only consider the possible probability measures over the sets Θ_j. This is because, given any probabilistic contract $\mathcal{P}_{\sim p}(\mathcal{A}, \mathcal{G})$ appearing in ϕ or ϕ_0, the sets Θ_j are fine-grained enough for expressing the conditional probability of the guarantee $\mathcal{G}$ given the assumption $\mathcal{A}$. These conditional probabilities can be encoded as a system of linear inequalities. If the system of inequalities lacks solution, it means that no probability measure can possibly respect the bounds imposed by both ϕ and $\neg\phi_0$, and we conclude that ϕ refines ϕ_0. Of importance is the fact that, although the set of all probability spaces over the possible traces $\texttt{tr}(X)$ is, in the general case, uncountably infinite, the linear inequalities over the finitely many Θ_j can be solved using e.g. the simplex method, see [8, 23].

The algorithm is presented in pseudocode below, where we assume that ϕ is a composition of the form $\phi_1 \parallel \phi_2 \parallel \cdots \parallel \phi_m$, each being a probabilistic contract, and that also ϕ_0 is a probabilistic contract. Furthermore, due to the difficulties in solving linear inequalities with non-strict inequalities, we assume that the probability bound of each ϕ_i, $i \geq 1$, is non-strict and the probability bound of ϕ_0 (which will be complemented) is strict. On line 5, the notation $\mathcal{M} \in \Omega_j$, where $\mathcal{M}$ is a DTMA, means that $\mathcal{M}$ appears as part of the intersection Ω_j. The details of the algorithm are demonstrated in Section 6 on an example.

Algorithm 1 Verify that a PCF $\phi_1 \parallel \cdots \parallel \phi_m$ refines a PCF ϕ_0.

```
1:  $\Omega \leftarrow \begin{pmatrix} \mathcal{A}_0 \sqcap \mathcal{G}_0 \sqcap \mathcal{A}_1 \sqcap \mathcal{G}_1 \sqcap \cdots \sqcap \mathcal{A}_m \sqcap \mathcal{G}_m, \\ \mathcal{A}_0 \sqcap \mathcal{G}_0 \sqcap \mathcal{A}_1 \sqcap \mathcal{G}_1 \sqcap \cdots \sqcap \mathcal{A}_m \sqcap \mathcal{G}_m^c, \\ \mathcal{A}_0 \sqcap \mathcal{G}_0 \sqcap \mathcal{A}_1 \sqcap \mathcal{G}_1 \sqcap \cdots \sqcap \mathcal{A}_m^c \sqcap \mathcal{G}_m, \\ \vdots \\ \mathcal{A}_0^c \sqcap \mathcal{G}_0^c \sqcap \mathcal{A}_1^c \sqcap \mathcal{G}_1^c \sqcap \cdots \sqcap \mathcal{A}_m^c \sqcap \mathcal{G}_m^c \end{pmatrix}$
2:  Let $z_0, z_1, \ldots, z_{k-1}$ be fresh variables, where $k = |\Omega|$
3:  ineqs $\leftarrow \{z_0 + z_1 + \cdots + z_{k-1} = 1\}$
4:  for $\phi_i \in \{\phi_1, \ldots, \phi_m, \neg\phi_0\}$ do
5:      ineqs $\leftarrow$ ineqs $\cup \left\{ \frac{\sum_{j:\mathcal{A}_i \sqcap \mathcal{G}_i \in \Omega_j} z_j}{\sum_{j:\mathcal{A}_i \in \Omega_j} z_j} \sim_i p_i \right\}$
6:  for $j \in \{0, \ldots, k-1\}$ do
7:      if $L(\Omega_j)$ is empty then
8:          ineqs $\leftarrow$ ineqs $\cup \{z_j = 0\}$
9:      else
10:         ineqs $\leftarrow$ ineqs $\cup \{z_j \geq 0\}$
11: return true if ineqs has no real solutions; unknown otherwise
```

Note that, whenever a solution to the system of linear inequalities is found, the algorithm outputs `unknown` rather than `false`. To see why, consider one of the probabilistic contracts ϕ_i of the composition, and suppose that it has both input and output variables. Then the behaviors in $[\![\phi]\!]$ respect the probability bound of ϕ on *each* input trace satisfying the assumption. Intuitively, this means that the granularity is fine. On the other hand, since the algorithm considers conditional probabilities rather than individual input traces, any proposed solution only needs to respect this bound *on average*, taken over all input traces satisfying the assumption. Intuitively, the granularity is coarser. This is a weaker condition, potentially resulting in spurious solutions. Since a fine-grained solution always implies a course-grained solution, but not vice versa, the algorithm is sound but not complete.

Theorem 1 (Soundness). *Let ϕ_0 be a probabilistic contract and ϕ be a PCF of the form $\phi_1 \parallel \phi_2 \parallel \cdots \parallel \phi_m$, each ϕ_i being a probabilistic contract, such that ϕ_0*

and ϕ have the same (non-empty) set of output variables and no input variables. If the procedure $\texttt{Refines}(\phi, \phi_0)$ *given by Algorithm 1 returns* `true` *then ϕ refines ϕ_0.*

Proof. By contraposition: assume that ϕ does not refine ϕ_0 and show that it is possible to assign probabilities satisfying the system of linear inequalities contained in the set `ineqs`. The assumption is equivalent to the existence of a behavior β such that $\beta \in [\![\phi]\!]$ and $\beta \notin [\![\phi_0]\!]$. Take an arbitrary such β and assign to each variable z_j the value $\beta()(L^*(\omega_j))$. Because Ω constitutes a partition of the trace space, i.e. the sets $L^*(\omega_j)$ are disjoint and span $\texttt{tr}(X)$, and $\beta()(\texttt{tr}(X)) = 1$, then due to finite additivity, $\sum_{\omega_j \in \Omega} \beta()(\omega_j) = 1$. This satisfies the equality of line 3. The inequalities of line 10 are satisfied trivially since for any measurable set $A \subseteq \texttt{tr}(X)$, $\beta()(A) \geq 0$. If furthermore A is empty, then $\beta()(A) = 0$, satisfying also the inequalities of line 8.

The only remaining inequalities are those of line 5. Due to the interpretation of $\neg$, we know that $\beta \in [\![\neg\phi_0]\!]$. Since the assumption $\mathcal{A}$ of ϕ_0 is $\top$ which accepts all traces, we have $\sum_{j:\mathcal{A}\in\omega_j} \beta()(L^*(\omega_j)) = 1$. Due to the interpretation of PCF,

$$p \sim^c \beta()(L^O(\mathcal{G})) = \beta()(L^*(\mathcal{G})) = \beta()\left(\bigcup_{j:\mathcal{G}\in\omega_j} L^*(\omega_j)\right)$$

$$= \beta()\left(\bigcup_{j:\mathcal{A}_j\sqcap\mathcal{G}_j\in\omega_j} L^*(\omega_j)\right) = \sum_{j:\mathcal{A}_j\sqcap\mathcal{G}_j\in\omega_j} \beta()(\omega_j)$$

$$= \frac{\sum\limits_{j:\mathcal{A}_j\sqcap\mathcal{G}_j\in\omega_j} \beta()(\omega_j)}{1} = \frac{\sum\limits_{j:\mathcal{A}_j\sqcap\mathcal{G}_j\in\omega_j} \beta()(L^*(\omega_j))}{\sum\limits_{j:\mathcal{A}\in\omega_j} \beta()(L^*(\omega_j))} ,$$

satisfying the conditional inequality of line 5 for the case $i = 0$. Since also $\texttt{in}(\phi_1) = \emptyset$, the same reasoning applies also to the case $i = 1$ but where the inequality $p_1 \sim_1 \beta()(L^*(\mathcal{G}_1))$ comes from the definition of I/O behavior composition. (More precisely, β guarantees the existence, through marginalization, of a behavior $\beta_1 \in \texttt{beh}(\emptyset, \texttt{out}(\phi_1))$ s.t. $p_1 \sim_1 \beta_1(L^{\texttt{out}(\phi_1)}\mathcal{G}_1)$.)

We now consider the rest of the cases $i = 2, 3, \ldots, m$. Let $\beta_1, \beta_2, \ldots, \beta_m$ be behaviors such that $\beta = \beta_1 \parallel \beta_2 \parallel \cdots \parallel \beta_m$ and, for each $i \in \{2, 3, \ldots, m\}$, $\beta_i \in [\![\phi_i]\!]$. Such behaviors always exist because ϕ itself is defined as the composition $\phi_1 \parallel \phi_2 \parallel \cdots \parallel \phi_m$ for which the interpretation $[\![\phi_1 \parallel \phi_2 \parallel \cdots \parallel \phi_m]\!] = [\![\phi_1]\!] \parallel [\![\phi_2]\!] \parallel \cdots \parallel [\![\phi_m]\!]$ only contains behaviors being a composition of individual behaviors β_i of each ϕ_i, respectively. Definition 24 gives, for each $i \in \{2, 3, \ldots, m\}$,

$$\forall \theta_{I_i} \in L^{I_i}(\mathcal{A}_i) \ . \ \beta_i(\theta_{I_i})(\{\theta_{O_i} \mid \theta_{I_i} \parallel \theta_{O_i} \in L^{I\cup O}(\mathcal{G}_i)\}) \sim_i p_i \ .$$

This implies that for each set $\Theta_{\mathcal{A}_i}$ satisfying

$$\Theta_{\mathcal{A}_i} \subseteq L^*(\mathcal{A}_i) \ , \tag{1}$$

we have

$$\beta()(L^*(\mathcal{G}_i) \mid \Theta_{\mathcal{A}_i}) \sim_i p_i \ . \tag{2}$$

For each i, let

$$A_i = \bigcup_{j:\mathcal{A}_i \in \omega_j} L^*(\omega_j)$$

and

$$B_i = \bigcup_{j:\mathcal{A}_i \sqcap \mathcal{G}_i \in \omega_j} L^*(\omega_j) \ .$$

Since $A_i = L^*(\mathcal{A}_i)$ satisfies (1), and $B_i = A_i \cap L^*(\mathcal{G}_i)$, it follows from (2) that

$$\begin{aligned} p_i &\sim_i \beta()(B_i \mid A_i) \\ &= \frac{\beta()(B_i \cap A_i)}{\beta()(A_i)} \\ &= \frac{\beta()(B_i)}{\beta()(A_i)} \\ &= \frac{\beta()(\bigcup_{j:\mathcal{A}_i \sqcap \mathcal{G}_i \in \omega_j} L^*(\omega_j))}{\beta()(\bigcup_{j:\mathcal{A}_i \in \omega_j} L^*(\omega_j))} \\ &= \frac{\sum_{j:\mathcal{A}_i \sqcap \mathcal{G}_i \in \omega_j} \beta()(L^*(\omega_j))}{\sum_{j:\mathcal{A}_i \in \omega_j} \beta()(L^*(\omega_j))} \ , \end{aligned}$$

satisfying the rest of the inequalities of line 5. □

6 Illustrative Example

Following the setting of [15], consider a hypothetical power generating system composed of the two components g_1 and g_2 representing the *main* and *backup* generator, respectively. In order for the backup to work properly, it must first have been charged by the main, demonstrating a sort of dependence between the two. Assume that we are given, in natural language, a top-level specification S_0 and two component specifications S_1 and S_2, one for each generator. Our objective is to answer whether or not the decomposition is correct. That is, does the parallel composition $S_1 \parallel S_2$ of the component specifications refine the top-level specification S_0? Let the three specifications be given as follows:

S_0: "total power output shall be at least 1 kW throughout the first 7 hours with over 45% probability",
S_1: "main power output shall be at least 1 kW throughout the first 6 hours with at least 70% probability",
S_2: "assuming main power output is at least 1 kW throughout the first 3 hours, then the first time t it declines below 1 kW, with at least 80% probability, backup power output shall be at least 1 kW throughout the interval $[t, t+2h]$".

To formally reason about this decomposition, let us express the above specifications using the framework of the present paper. For each generator g_i, let the real-valued variable x_i represent its current power output. Each possible *trace* for the set of variables $\{x_1, x_2\}$ therefore gives, for each point in time, the power output of the main and backup generator, separately. For $i \in \{1, 2\}$, let q_i be the *predicate* $x_i \geq 1$ kW, asserting that generator g_i outputs at least 1 kW of power. We now express each specification S_j above as a corresponding PCF ϕ_j. Starting with the top-level specification S_0, let

$$\phi_0 = \mathcal{P}_{>0.45}\Big(\top,\quad \begin{tikzcd} {} \arrow[r] & \substack{q_1 \vee q_2 \\ c_0 \leq 7} \arrow[rr] & & c_0 > 7 \end{tikzcd}\Big).$$

Denoting the double-ringed location as l, let the acceptance family of the guarantee be the set $\{\{l\}\}$. We will continue to use this convention whenever there is only one double-ringed location. This PCF puts no assumptions on the environment and contains the guarantee that at least 1 kW is output through the first 7 h by either the main or backup generator, or both. This time limit of 7 h is enforced by the clock c_0 and clock constraint $c_0 > 7$, since the accepting location can only be entered if the constraint $q_1 \vee q_2$ has remained true at all times before then. Otherwise, the run moves to an implicit trap location, see Section 3.3. The probability bound states that all traces satisfying this guarantee must have a total probability mass strictly greater than 0.45. For the main generator specification S_1, let

$$\phi_1 = \mathcal{P}_{\geq 0.7}\Big(\top,\quad \begin{tikzcd} {} \arrow[r] & \substack{q_1 \\ c_1 \leq 6} \arrow[rr] & & c_1 > 6 \end{tikzcd}\Big).$$

This PCF follows the same pattern as ϕ_0 but, since the predicate q_2 is not used, only constrains the main generator. Note that ϕ_1 by itself neither refines nor is refined by ϕ_0 due to the shorter time limit of 6 h, which is why the backup generator is needed for the decomposition to work.

Since the natural-language specification S_2 for the backup g_2 contains an assumption about the power output of g_1, then so must the corresponding PCF:

$$\phi_2 = \mathcal{P}_{\geq 0.8}\Big(\begin{tikzcd} {} \arrow[d] & \\ \substack{q_1 \\ c_{2,1} \leq 3} \arrow[r] & c_{2,1} > 3 \end{tikzcd},\quad \begin{tikzcd} {} \arrow[r] & q_1 \arrow[rr, "c_{2,2} := 0"] & & \substack{\neg q_1 \wedge q_2 \\ c_{2,2} = 0} \arrow[dl] \\ & c_{2,2} > 2 & \substack{q_2 \\ c_{2,2} \leq 2} \arrow[l] & \end{tikzcd}\Big).$$

The assumption, which is put on the main generator, encodes the charging requirement of the backup using the same pattern as before. In the guarantee, let l_1 and l_2 denote the two double-ringed locations, from top to bottom respectively. The acceptance family is the set $\{\{l_1\}, \{l_2\}\}$, which captures two scenarios. The first, captured by l_1, is where q_1 never turns false, in which case the backup is not required to do anything. The second, captured by l_2, is where q_1 turns false and the backup compensates, turning q_2 true. The time at which this happens is recorded by the clock reset $c_{2,2} := 0$, and q_2 must then remain true for the next 2 h for the trace to be accepted.

Verifying that this decomposition is correct amounts to evaluating whether $\phi_1 \parallel \phi_2$ refines ϕ_0. This is done following Algorithm 1, by checking emptiness of each possible combination of the four automata or their complements, and reducing to determining a solution for a system of linear inequalities. For example, it is clear that the language $L(\Omega_0) = L(\mathcal{G}_0 \sqcap \mathcal{G}_1 \sqcap \mathcal{A}_2 \sqcap \mathcal{G}_2) = L(\mathcal{G}_0) \cap L(\mathcal{G}_1) \cap L(\mathcal{A}_2) \cap L(\mathcal{G}_2)$ is non-empty because e.g. the constant trace $\theta(t) = \{x_1 \mapsto 1000, x_2 \mapsto 1000\}, t \in \mathbb{R}_{\geq 0}$, in which both the main and backup generator outputs 1 kW constantly, is accepted by each automaton. In contrast, the language $L(\Omega_2) = L(\mathcal{G}_0 \sqcap \mathcal{G}_1 \sqcap \mathcal{A}_2^c \sqcap \mathcal{G}_2) = L(\mathcal{G}_0) \cap L(\mathcal{G}_1) \cap L(\mathcal{A}_2^c) \cap L(\mathcal{G}_2)$ is empty. To see this, note that $\mathcal{G}_1$ specifies 1 kW for at least 6 h while $\mathcal{A}_2$ specifies the same output for the shorter duration of at least 3 h. It is obviously not possible for a trace to uphold 1 kW for 6 h but not for 3 h, implying $L(\mathcal{G}_1) \cap L(\mathcal{A}_2^c) = \emptyset$. Checking emptiness for each combination gives

$$\left\{\begin{matrix} L(\Omega_0), & L(\Omega_1), & L(\Omega_2), & L(\Omega_3), \\ L(\Omega_4), & L(\Omega_5), & L(\Omega_6), & L(\Omega_7), \\ L(\Omega_8), & L(\Omega_9), & L(\Omega_{10}), & L(\Omega_{11}), \\ L(\Omega_{12}), & L(\Omega_{13}), & L(\Omega_{14}), & L(\Omega_{15}), \end{matrix}\right\} = \left\{\begin{matrix} \cdot, & \cdot, & \emptyset, & \emptyset, \\ \cdot, & \cdot, & \cdot, & \emptyset, \\ \emptyset, & \cdot, & \emptyset, & \emptyset, \\ \cdot, & \cdot, & \cdot, & \cdot, \end{matrix}\right\} .$$

Let us now construct the system of linear inequalities expressing the probability bounds of $[\![\neg\phi_0 \wedge (\phi_1 \parallel \phi_2)]\!]$. Note that, in accordance with Algorithm 1, the top-level specification has been complemented. For each composition Ω_k, we define a corresponding variable z_k to represent the probability mass of all traces accepted by Ω_k. That is, for a behavior $\beta \in [\![\neg\phi_0 \wedge (\phi_1 \parallel \phi_2)]\!]$, z_k represents the probability $\beta()(L^*(\Omega_k))$. The resulting inequalities can be constructed as follows:

$$\begin{cases} \sum_{j=0}^{15} z_j = 1 & (3) \\ (z_2, z_3, z_7, z_8, z_{10}, z_{11}) = (0,0,0,0,0,0) & (4) \\ (z_0, z_1, z_4, z_5, z_6, z_9, z_{12}, z_{13}, z_{14}, z_{15}) \geq (0,0,0,0,0,0,0,0,0,0) & (5) \\ z_0 + z_1 + z_2 + z_3 + z_4 + z_5 + z_6 + z_7 \leq 0.45 & (6) \\ z_0 + z_1 + z_2 + z_3 + z_8 + z_9 + z_{10} + z_{11} \geq 0.7 & (7) \\ \left(\dfrac{z_0 + z_4 + z_8 + z_{12}}{z_0 + z_1 + z_4 + z_5 + z_8 + z_9 + z_{12} + z_{13}}\right) \geq 0.8 \ . & (8) \end{cases}$$

The equality (3) represents the fact that all variables must sum to 1 to constitute a probability measure. Next, (4) constrains the variables corresponding to empty languages to have probability 0, and (5) constrains the rest to be at least non-negative. Lastly, reconstructing the probability bounds given by the complement of ϕ_0 and by ϕ_1 and ϕ_2 gives the remaining inequalities (6), (7) and (8). Using e.g. the simplex method, we see that this system of linear inequalities has no solution. According to Theorem 1, we conclude that $\phi_1 \parallel \phi_2$ refines ϕ_0.

7 Related Work

A related field is the area of *model checking* [20, 27]. However, this is different from the current paper, since the goal of model checking is to verify implementation rather than refinement.

Specification theories for *Probabilistic Automata* have been proposed [17, 29, 31], in which the semantics is defined as sets of so-called *trace distributions*, resembling the behaviors of the present paper. A similar notion of *bundles*, which are generated by *probabilistic modules* instead of probabilistic automata, is studied in [9]. Although both of these theories facilitate refinement verification, they assume discrete time and a specific underlying probability structure, while the present paper gives explicit support for continuous time and general probability measures. Also in the continuous setting, *probabilistic I/O automata* [33] and *interactive Markov chains* [14] combine exponential distributions with non-deterministic choice, and *constraint Markov chains* [6] can be used as a finite representation of infinite sets of continuous-time Markov chains. While these theories assume the memoryless property, the present paper allows general probability spaces. Also supporting general probability spaces, a stochastic process algebra called *SPADES* [11] has been introduced. However, the focus there is on discrete-event simulation while the present paper studies refinement of specifications.

In the probabilistic contract setting, the papers [10] and [24] present two different compositional contract-based specification theories for stochastic systems. Both theories include refinement verification, although under the assumption of discrete time in contrast to the continuous-time support of the present paper.

8 Conclusions

For cyber-physical systems, the ability to *specify* probabilistic properties, e.g. related to safety, is fundamental. Also fundamental is the ability to handle systems with continuous state spaces. The present paper has presented a theory for such specifications based upon *probabilistic contracts*. In particular, the main contribution is an algorithm for verification of *refinement* between such specifications. The algorithm utilizes a reduction to the language emptiness problem, making the algorithm terminate after a finite number of computations.

References

1. Alur, R., Dill, D.: Automata for modeling real-time systems. In: Automata, Languages and Programming: 17th International Colloquium Warwick University, England, July 16–20, 1990 Proceedings 17. pp. 322–335. Springer (1990)
2. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science **126**(2), 183–235 (1994). https://doi.org/https://doi.org/10.1016/0304-3975(94)90010-8
3. Alur, R., Feder, T., Henzinger, T.A.: The benefits of relaxing punctuality. Journal of the ACM (JACM) **43**(1), 116–146 (1996)
4. Alur, R., Henzinger, T.A.: Logics and models of real time: A survey. In: Real time: Theory in practice. vol. 600, pp. 74–75 (1992)
5. Benveniste, A., Caillaud, B., Ferrari, A., Mangeruca, L., Passerone, R., Sofronis, C.: Multiple viewpoint contract-based specification and design. In: Formal Methods for Components and Object, pp. 200–225. Springer-Verlag, Berlin, Heidelberg (2008)
6. Caillaud, B., Delahaye, B., Larsen, K.G., Legay, A., Pedersen, M.L., Wasowski, A.: Compositional design methodology with constraint markov chains. In: 2010 Seventh International Conference on the Quantitative Evaluation of Systems. pp. 123–132. IEEE (2010)

7. Cuoq, P., Kirchner, F., Kosmatov, N., Prevosto, V., Signoles, J., Yakobowski, B.: Frama-c. In: International conference on software engineering and formal methods. pp. 233–247. Springer (2012)
8. Dantzig, G.B.: Origins of the simplex method. In: A history of scientific computing, pp. 141–151 (1990)
9. De Alfaro, L., Henzinger, T.A., Jhala, R.: Compositional methods for probabilistic systems. In: CONCUR 2001—Concurrency Theory: 12th International Conference Aalborg, Denmark, August 20–25, 2001 Proceedings. pp. 351–365. Springer (2001)
10. Delahaye, B., Caillaud, B., Legay, A.: Probabilistic contracts: a compositional reasoning methodology for the design of systems with stochastic and/or non-deterministic aspects. Formal Methods in System Design **38**(1), 1–32 (2011)
11. D'Argenio, P.R., Katoen, J.P., Brinksma, E.: An algebraic approach to the specification of stochastic systems. In: Programming Concepts and Methods PROCOMET'98: IFIP TC2/WG2. 2, 2.3 International Conference on Programming Concepts and Methods (PROCOMET'98) 8–12 June 1998, Shelter Island, New York, USA. pp. 126–147. Springer (1998)
12. D'Souza, D., Prabhakar, P.: On the expressiveness of mtl in the pointwise and continuous semantics. International Journal on Software Tools for Technology Transfer **9**, 1–4 (2007)
13. Fahrenberg, U., Legay, A., Traonouez, L.M.: Specification theories for probabilistic and real-time systems. In: From Programs to Systems. The Systems perspective in Computing: ETAPS Workshop, FPS 2014, in Honor of Joseph Sifakis, Grenoble, France, April 6, 2014. Proceedings. pp. 98–117. Springer (2014)
14. Gössler, G., Xu, D.N., Girault, A.: Probabilistic contracts for component-based design. Formal Methods in System Design **41**(2), 211–231 (2012)
15. Hampus, A., Nyberg, M.: Formally verifying decompositions of stochastic specifications. In: International Conference on Formal Methods for Industrial Critical Systems. pp. 193–210. Springer (2022)
16. Konur, S.: Real-time and probabilistic temporal logics: An overview. arXiv preprint arXiv:1005.3200 (2010)
17. Kwiatkowska, M., Norman, G., Parker, D., Qu, H.: Assume-guarantee verification for probabilistic systems. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 23–37. Springer (2010)
18. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, pp. 152–166. Springer (2004)
19. Maler, O., Nickovic, D., Pnueli, A.: From mitl to timed automata. In: International conference on formal modeling and analysis of timed systems. pp. 274–289. Springer (2006)
20. Mereacre, A., Katoen, J.P., Han, T., Chen, T.: Model checking of continuous-time markov chains against timed automata specifications. Logical Methods in Computer Science **7** (2011)
21. Meyer, B.: Applying'design by contract'. Computer **25**(10), 40–51 (1992)
22. Moura, L.d., Bjørner, N.: Z3: An efficient smt solver. In: International conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 337–340. Springer (2008)
23. Nash, J.C.: The (dantzig) simplex method for linear programming. Computing in Science & Engineering **2**(1), 29–31 (2000)
24. Nuzzo, P., Li, J., Sangiovanni-Vincentelli, A.L., Xi, Y., Li, D.: Stochastic assume-guarantee contracts for cyber-physical system design. ACM Transactions on Embedded Computing Systems (TECS) **18**(1), 1–26 (2019)

25. Nyberg, M., Westman, J., Gurov, D.: Formally proving compositionality in industrial systems with informal specifications. In: International Symposium on Leveraging Applications of Formal Methods. pp. 348–365. Springer (2020)
26. Ouaknine, J., Worrell, J.: Some recent results in metric temporal logic. In: Formal Modeling and Analysis of Timed Systems: 6th International Conference, FORMATS 2008, Saint Malo, France, September 15-17, 2008. Proceedings 6. pp. 1–13. Springer (2008)
27. Paolieri, M., Horváth, A., Vicario, E.: Probabilistic model checking of regenerative concurrent systems. IEEE Transactions on Software Engineering **42**(2), 153–169 (2015)
28. Roever, W.P.d.: The need for compositional proof systems: A survey. In: International Symposium on Compositionality. pp. 1–22. Springer (1997)
29. Segala, R.: A compositional trace-based semantics for probabilistic automata. In: CONCUR'95: Concurrency Theory: 6th International Conference Philadelphia, PA, USA, August 21–24, 1995 Proceedings 6. pp. 234–248. Springer (1995)
30. Slind, K., Norrish, M.: A brief overview of hol4. In: International Conference on Theorem Proving in Higher Order Logics. pp. 28–32. Springer (2008)
31. Stoelinga, M.: An introduction to probabilistic automata. Bulletin of the EATCS **78**(176-198), 2 (2002)
32. Westman, J., Nyberg, M.: Conditions of contracts for separating responsibilities in heterogeneous systems. Formal Methods in System Design (Sep 2017). https://doi.org/10.1007/s10703-017-0294-7
33. Wu, S.H., Smolka, S.A., Stark, E.W.: Composition and behaviors of probabilistic i/o automata. Theoretical Computer Science **176**(1-2), 1–38 (1997)