International Journal on Software Tools for Technology Transfer (2024) 26:207–228
https://doi.org/10.1007/s10009-024-00742-5

GENERAL

Special Issue: FMICS 2022

Formally verifying decompositions of stochastic specifications

Anton Hampus[1] · Mattias Nyberg[1,2]

Accepted: 6 February 2024 / Published online: 21 February 2024

Abstract

According to the principles of compositional verification, verifying that lower-level components satisfy their specification ensures that the whole system satisfies its top-level specification. The key step is to ensure that the lower-level specifications constitute a correct decomposition of the top-level specification. In a non-stochastic context, such decomposition can be analyzed using techniques of theorem proving. In industrial applications, especially in safety-critical systems, specifications are often of stochastic nature, for example, giving a bound on the probability that a system failure will occur before a given time. A decomposition of such a specification requires techniques beyond traditional theorem proving. The first contribution of the paper is a theoretical framework that allows the representation of, and reasoning about, stochastic and timed behavior of systems as well as specifications for such behavior. The framework is based on traces that describe the continuous-time evolution of a system, and specifications are formulated using timed automata combined with probabilistic acceptance conditions. The second contribution is a novel approach to verifying decompositions of such specifications by reducing the problem to checking emptiness of the solution space for a system of linear inequalities.

Keywords Specification theory · Refinement · Contracts · Automata

1 Introduction

The principle of *compositional verification* [1] has been proposed as a solution to verify large complex systems built up from smaller components. The key idea is to verify that: (1) each component implements its specification, and (2) the composition of these component specifications refines the top-level system specification. This ensures that the whole system implements its top-level specification. The key difficulty is (2), which can also be expressed as ensuring that the component specifications constitute a correct decomposition of the top-level specification.

Although decomposition of specifications is in general difficult, its importance is stressed by its role in recent industrial standards such as ISO 26262 [2] and ISO 21434 [3]. In these standards, specifications in the form of safety and cyber-security requirements are decomposed into lower-level specifications. The standards also require that these decompositions are complete, in the sense that if the lower-level requirements are satisfied, then the upper-level requirements are also satisfied.

In the present paper, we consider general cyber-physical systems and have therefore chosen a representation compliant with continuous time. Based upon logic and various extensions to include time, a number of frameworks are available to express specifications and to verify refinements between specifications, e.g., [4–7]. A limitation of these frameworks is that they do not consider probabilistic or stochastic behaviors. On the other hand, from an industrial standpoint, the ability to include stochastics is fundamentally important since the exact purpose of many specifications, especially within safety, is to set limits on the probability of undesired events occurring within certain time intervals.

In order to allow the study of stochastic specifications, the present paper proposes, as its first contribution, a novel framework covering: syntax and semantics of stochastic specifications, and composition and refinement of such specifications. To support the industrial applicability of the framework, as the second contribution, the paper proposes also

✉ A. Hampus
ahampus@kth.se

M. Nyberg
mattias.nyberg@scania.com

[1] KTH Royal Institute of Sweden, Stockholm, Sweden

[2] Scania, Södertälje, Sweden

an algorithm for the analysis of whether a composition of stochastic specifications refines another stochastic specification.

The approach taken in this paper is that behavior of components and systems is characterized by traces and probability measures over sets of traces. Such behaviors are used as an abstract tool for defining the semantics of specifications as sets of behaviors.

The syntax of specifications bears a resemblance to CSL [8–10], but views specifications generally as a stochastic extension to assume-guarantee contracts [11–13]. In such a specification, denoted $\mathcal{P}_{<p}(\mathcal{A},\mathcal{G})$, both the assumption $\mathcal{A}$ and the guarantee $\mathcal{G}$ of the contract is represented by a deterministic timed automaton responding to traces. The specification states that if the assumption is satisfied, the probability that the guarantee is satisfied shall be less than p. Using automata to formulate assumptions and guarantees stem from their flexibility, expressiveness, and presence in literature. They are in many ways comparable to temporal logic, and provide a basis for the proposed refinement verification algorithm.

To perform such verification, the main assumptions are that dependence between specifications is non-cyclical and that traces contain a bounded number of events. On the other hand, the framework aims to be as general as possible, supporting a dense time domain and arbitrary probability measures for the underlying systems. For instance, we allow both discrete and continuous distributions, including, but not limited to, normal and exponential distributions.

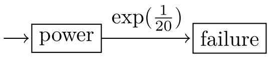
The literature contains some other proposed frameworks for defining stochasticspecifications and verifying properties such as refinement, e.g., [14–22]. However, in contrast to all of these previous works, the present paper uses continuous time and considers component behavior purely in terms of traces—no particular modeling formalism for generating the traces is assumed.

The present paper extends the conference paper [23]. Besides being more detailed, it provides the following:

- a proof of the main theorem,
- an updated trace semantics relying on interval sequences, avoiding Zeno behavior and allowing more general clock constraints,
- a formal semantics of automata composition and a proof that it preserves determinism and termination,
- a concrete σ-algebra for the set of traces, removing assumptions about trace sets being measurable.

The paper is organized as follows. Section 2 uses an example to illustrate the problem and sketch the proposed solution. Sections 3 and 4 describe the proposed framework and algorithm. Section 5 applies the framework and the algorithm to an extended version of the example studied in Section 2. Finally, Sections 6 and 7 present related work and conclusions.

Fig. 1 Example of a main power behavior.



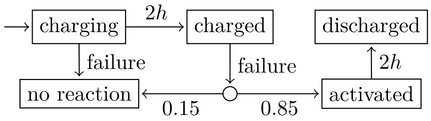


Fig. 2 Example of a backup behavior.

2 Problem illustration

Consider a two-component system consisting of a main and backup power source. The idea is that whenever there is a main power failure, the backup is activated. The purpose of the backup is to prolong the duration of power output by the system. However, in order for the backup to correctly do so, it needs to first be charged by the main power source for a certain amount of time. Furthermore, even if charged, there is a probability that it will fail prematurely.

An example of a behavior of such a system is modeled in Fig. 1 and 2. In these diagrams, main power failure occurs exponentially with rate $\frac{1}{20}$ (per hour), while the backup component responds to this failure probabilistically. More precisely, when the main power fails, the backup is activated with 85% probability if it has finished charging and 0% probability otherwise. This fact is represented in Fig. 2 by the edges labeled *failure*. The required charging time for this specific backup is 2 hours. Once activated, the backup will output power also for 2 hours, until entering a discharged state. Note that these are just examples of possible behavior of a main and backup power source, not the actual specifications.

Assume that the top-level specification is: “The system shall output power continuously during the first 7 hours with over 45% probability”. Instead of merely verifying that the system composed of components with behaviors as in Fig. 1 and 2 implements the top-level specification, we want to formulate two component specifications and verify that any system composed of a main and backup implementing its component specification is certain to implement the top-level specification. As our attempt to do this, let the main power source specification be: “Main power failure shall occur before 6 hours with at most 30% probability”. Meanwhile, the backup specification is an assume-guarantee contract: “Assuming main power failure occurs after at least 3 hours, then with at least 80% probability, the backup shall output power continuously for at least 2 hours starting at this time”. Note that, since the main power specification only concerns the first 6 hours, it does not refine the top-level specification by itself and needs to be supplemented by the backup specification to extend this time interval.

As a sketch of what refinement means, we first observe that the outcomes, i.e. the traces, of the components are generated

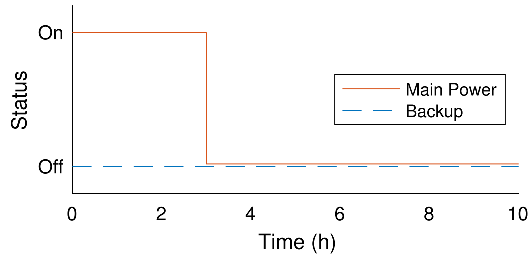


Fig. 3 Failed backup activation.

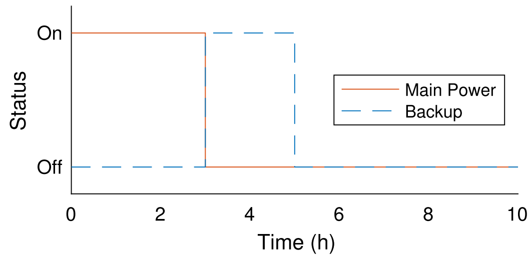


Fig. 4 Successful backup activation.

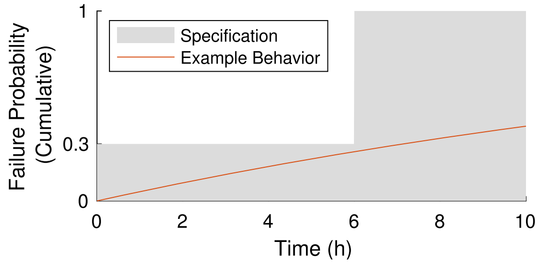


Fig. 5 Main power specification.

stochastically. Fig. 3 and 4 show two possible traces of a main and backup power source. In both traces, main power failure occurs at exactly 3 hours. However, backup power activation fails in Fig. 3, while succeeding in Fig. 4. Once activated, it manages to prolong power output by 2 hours, resulting in the system continuously outputting power for 5 hours instead of 3, as would be the case without the backup.

We view these traces as samples drawn from some underlying probability distribution. For example, the main power trace might be drawn from the process of Fig. 1 and the backup trace from Fig. 2. Such an underlying probability distribution is referred to as a *behavior*. As a result, specifying the two components amounts to specifying two sets of behaviors; thus, we must translate the natural language specifications to sets of “allowed” probability distributions.

Figure 5 depicts the specification for the main power source in terms of the behaviors it contains, represented by the gray region. The convention used here is that a behavior, represented by the cumulative distribution function (CDF) of the time to failure, implements the specification if it lies completely within the gray region. Note that the region extends to positive infinity along the horizontal axis. To better understand this graphical representation of the specification, an example behavior, drawn as a CDF, is included inside the region. Note that this CDF in fact represents the behavior generated by the process of Fig. 1, following the exponential distribution exp($\frac{1}{20}$).

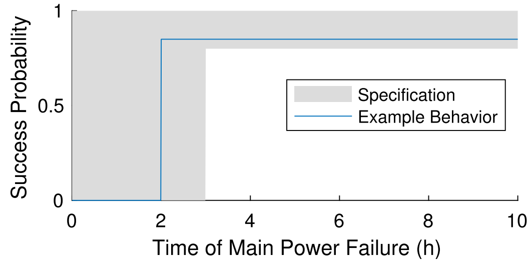


Fig. 6 Backup specification.

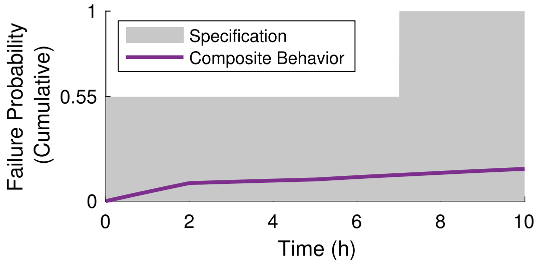


Fig. 7 Top-level specification.

The backup specification is depicted in Fig. 6 using a similar approach. However, this region does not represent a set of failure CDFs, but instead a set of success probabilities, given as functions of the time when the main power failure occurs. Here, success means that the backup is able to output power for at least 2 hours. The example behavior shown within the region corresponds to a backup power source that needs 2 hours to charge, and, once charged, has a success rate of 85% regardless of when the main power fails. Note that whenever the assumption is unfulfilled, nothing is required of the backup. That is, within the first 3 hours, all success rates from 0% to 100% are allowed.

Lastly, the top-level specification is depicted in Fig. 7, showing a region of allowed failure CDFs of total power output, ignoring whether the main or backup is responsible for outputting it. The question now is: Does the composition of the two component specifications refine the top-level specification? The purpose of the rest of the paper is to formalize these notions of traces, behaviors, and specifications, and to introduce an algorithm for verifying refinement between specifications.

3 A theory for specifying stochastic behavior

Looking back at the core problem described in Sections 1 and 2, our goal in this section is to develop a theory that enables us to: (1) represent stochastic behaviors over input and output, (2) represent and compose specifications over such behaviors, (3) reason about implementation and refinement, and (4) express specifications using a contract-based approach.

In short, a behavior is meant to represent the possible executions, or traces, of a component, as well as how likely they are. We represent a trace as an assignment of values to variables at each point in time and a behavior as a probability measure over traces. The motivation for choosing a trace-based approach is its generality—no particular modeling formalism for generating the traces is assumed. We will also extend behaviors to incorporate input as well as output, calling them input/output behaviors. In accordance with [24], a specification is simply the set of all input/output behaviors that implement it, and a specification refines another if each behavior implementing it also implements the other. Thus, the framework presented in this paper constitutes an instantiation of the main part of [24]. Lastly, we present probabilistic automaton contracts (PACs) as a syntax for expressing specifications.

3.1 Traces

We consider a universal set of variables $X = \{x_1, x_2, \dots, x_n\}$, $n \geq 1$, each $x_i \in X$ ranging over a non-empty set V_{x_i} of values with cardinality not greater than the cardinality of $\mathbb{R}$. Given a non-empty set of variables $E \subseteq X$, a valuation for E is a function $\nu : E \rightarrow \bigcup_{x_i \in X} V_{x_i}$ associating each $x_i \in E$ with a value in its range V_{x_i}. The set of all possible valuations for a non-empty set $E \subseteq X$ is denoted $\texttt{val}(E)$. Throughout the paper, we often use notation of the form $\{x_1 \mapsto v_1, x_2 \mapsto v_2, \dots\}$ to denote the function associating x_1 with v_1, x_2 with v_2, and so on.

To define traces, we use a continuous-time semantics based on interval sequences [25]. We only consider traces that do not exhibit so-called Zeno behaviors, in which a system changes state an infinite number of times within a finite time interval [25, 26]. In [25], this avoidance of Zeno behaviors is referred to as the *finite-variability condition* and is stated to be an adequate assumption for modeling discrete-state systems.

A time interval $I \subseteq \mathbb{R}_{\geq 0}$ is a bounded subset of the non-negative real numbers, taking the form $[a, b)$ where $a, b \in \mathbb{R}_{\geq 0}$ and $a < b$. 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 $r(I_1) = l(I_2)$. Let $\mathbb{N}$ denote the non-negative integers $\{0, 1, 2, \dots\}$.

Definition 1 (Interval Sequence [25])
An *interval sequence* is an infinite sequence of time intervals $I_0 I_1 I_2 \dots$ that partition the non-negative real line such that, for each $i \in \mathbb{N}$, the intervals I_i and I_{i+1} are adjacent.

Given a non-empty set $E \subseteq X$ of variables, a *timed valuation sequence* over E is a pair $\gamma = (\bar{\nu}, \bar{I})$ where $\bar{\nu} = \nu_0 \nu_1 \nu_2 \dots$ is an infinite sequence of valuations $\nu_i \in \texttt{val}(E)$ and $\bar{I} = I_0 I_1 I_2 \dots$ is an interval sequence. To later define traces, we first introduce the notation γ^* to allow us to represent a timed valuation sequence γ as a functions of time rather than a pair of sequences. Simply put, γ^* maps each point in time to the corresponding valuation in γ. More formally, given a timed valuation sequence γ over E, let $\gamma^* : \mathbb{R}_{\geq 0} \rightarrow \texttt{val}(E)$ be the function such that, for each $i \in \mathbb{N}$ and time $t \in I_i$, $\gamma^*(t) = \nu_i$. For instance, if γ is the timed valuation sequence $((\{x \mapsto 1\}, \{x \mapsto 0\}, \{x \mapsto 1\}, \{x \mapsto 0\}, \dots), ([0, 1), [1, 2), [2, 3), [3, 4), \dots))$, then γ^* is the function

$$\gamma^*(t) = \begin{cases} \{x \mapsto 1\} & \text{if } t \in [2i, 2i+1) \\ \{x \mapsto 0\} & \text{if } t \in [2i+1, 2i+2)\,, \end{cases}$$

where $i \in \{0, 1, 2, \dots\}$.

Definition 2 (Trace)
Given a non-empty set of variables $E \subseteq X$, a *trace over* E is a function $\theta : \mathbb{R}_{\geq 0} \rightarrow \texttt{val}(E)$ such that there exists a timed valuation sequence γ over E satisfying $\gamma^*(t) = \theta(t)$ for each $t \in \mathbb{R}_{\geq 0}$.

Example 1 (Trace)
Consider again the trace depicted in Fig. 3. Let x_m and x_b be two variables ranging over {on, off}, representing main and backup status, respectively. In accordance with Definition 2, we represent the trace as the function $\theta : \mathbb{R}_{\geq 0} \mapsto \texttt{val}(\{x_m, x_b\})$ such that

$$\theta(t) = \begin{cases} \{x_m \mapsto \text{on}, x_b \mapsto \text{off}\}, & t \in [0, 3) \\ \{x_m \mapsto \text{off}, x_b \mapsto \text{off}\}, & t \in [3, \infty)\,. \end{cases}$$

Note that, since time intervals take the form $[a, b)$, traces are intrinsically right-continuous. This restriction is not necessary for defining the framework and algorithm, but it makes the timed semantics of trace automata in Section 3.5 more intuitive and the later-presented σ-algebra simpler to define.

Due to their avoidance of Zeno behavior, traces are essentially multivalued piecewise constant functions. As such, they cannot explicitly express properties about continuous change of values. However, note that time nonetheless moves over the dense domain $\mathbb{R}_{\geq 0}$. It is also possible to view each attained value in a given trace as a general logic proposition that holds true at the corresponding time. Using an appropriate interpretation, such propositions can represent the grouping of possibly uncountable sets of “underlying” values. Traces over such propositions thus encapsulate truly continuous underlying traces, but lose the exact continuous information as a result of discretization. In other words, piecewise constant functions do not impose a restriction on the underlying systems, but rather on the timed properties that can be expressed for them. This is a common restriction within formal verification, where, for instance, timed automata and temporal logics often use logical propositions to obtain similar discretizations.

Let $\mathtt{tr}(E)$ denote the set of all possible traces over E. By convention, let $\mathtt{tr}(\emptyset) = \emptyset$, i.e., the set of all possible traces over the empty set of variables is itself empty. Furthermore, for any trace $\theta \in \mathtt{tr}(E)$ and non-empty set $E' \subseteq E$ of variables, let $\theta|_{E'}$ denote the projection $\theta' : \mathbb{R}_{\geq 0} \to \mathtt{val}(E')$ such that $\forall t \in \mathbb{R}_{\geq 0} \,.\, \forall x \in E' \,.\, \theta'(t)(x) = \theta(t)(x)$.

Definition 3 (Valuation Change)
Given a set $E \subseteq X$ of variables and a trace $\theta \in \mathtt{tr}(E)$ over E, a *valuation change in* θ is a time-point $t \in \mathbb{R}_{>0}$ such that there exists a real positive number ϵ satisfying $\forall \epsilon' > 0 \,.\, \epsilon' < \epsilon \implies \theta(t - \epsilon') \neq \theta(t)$.

Example 2 (Valuation Change)
Consider the trace depicted in Fig. 4 and let the variables x_m and x_b denote main power status and backup power status, respectively. The time 3 h is a valuation change, at which the valuation changes from $\{x_m \mapsto \text{On}, x_b \mapsto \text{Off}\}$ to $\{x_m \mapsto \text{Off}, x_b \mapsto \text{On}\}$. Time 5 h is also a valuation change in which the valuation changes from $\{x_m \mapsto \text{Off}, x_b \mapsto \text{On}\}$ to $\{x_m \mapsto \text{Off}, x_b \mapsto \text{Off}\}$. Other valuation changes are not depicted in the figure.

3.2 Behaviors

To formalize a notion of probability over traces, we will use probability measures, which are a generalization of probability distributions. The motivation behind using these is that they enable a completely general framework, including continuous, discrete, and mixtures of continuous and discrete distributions. They also facilitate a proof of correctness for the later presented refinement verification algorithm.

We first need to establish a σ-algebra on traces. That is, we need to specify the sets of traces that are measurable. Assume that each trace contains at most finitely many valuation changes, bounded by a number M. Note, however, that the bound M need not be calculated or chosen explicitly, but is rather assumed for theoretical reasons to assure a finite-dimension sample space for reasoning about probabilities. Note also that, since traces are non-Zeno, they automatically contain at most finitely many valuation changes if we are only interested in finite time, for instance the lifetime of the considered system. Furthermore, due to software running on fixed clock periods, together with an inherent inertia within physical systems, bounded system lifetime often implies a bounded number of valuation changes for cyber-physical systems in practice. Even systems with unbounded lifetimes may be guaranteed to enter a final state, e.g., success or failure, after which the remaining valuation changes can be ignored.

To define a σ-algebra on traces, we first show that each trace can be represented as a single point in a finite-dimension real coordinate space. Using such a representation, we can simply adopt the corresponding Borel σ-algebra. Let $E \subseteq X$ be a non-empty set of variables and θ an arbitrary trace over E. Then θ can be uniquely represented as a collection of the time-points for each valuation change together with the sequence of valuations themselves. Each trace is then uniquely determined using $M + 1$ valuations and M time-points. Since each range V_{x_i} for $x_i \in E$ has cardinality not greater than that of $\mathbb{R}$, we can represent each valuation as a real number. The set $\mathtt{tr}(E)$ of all traces over E can therefore be represented using a subset of $\mathbb{R}^{2M+1}$. Thus, to specify the measurable sets of traces, we may use the Borel σ-algebra $\mathcal{B}(\mathbb{R}^{2M+1})$. This connection between a set E of variables and the corresponding σ-algebra on traces over E is captured in the following definition.

Definition 4 (The σ-algebra $\mathcal{B}(E)$)
Given a non-empty set $E \subseteq X$ of variables, let $\mathcal{B}(E)$ denote the Borel σ-algebra $\mathcal{B}(\mathbb{R}^{2M+1})$, where M is a global upper bound for the number of valuation changes in traces.

See Appendix B for details about the trace sets contained within the σ-algebra $\mathcal{B}(E)$ for $E \subseteq X$. This is the σ-algebra we use for the following definition of a behavior. With slight abuse of notation, we make no distinction between traces θ and their corresponding representatives in $\mathbb{R}^{2M+1}$. Thus, we consider the pair $(\mathtt{tr}(E), \mathcal{B}(E))$ as a measurable space for any non-empty set $E \subseteq X$.

Definition 5 (Behavior)
Given a non-empty set of variables $E \subseteq X$, a *behavior over* E is a probability measure β defined on the σ-algebra $\mathcal{B}(E)$ satisfying $\beta(\mathtt{tr}(E)) = 1$.

Note that the preceding definition includes the condition that $\beta(\mathtt{tr}(E)) = 1$. This is to ensure that all probability mass is distributed between valid traces rather than arbitrary points in $\mathbb{R}^{2M+1}$ not corresponding to any trace.

Example 3 (Behavior)
Consider again the main power behavior of Fig. 1 and let x be a variable ranging over $\{0, 1\}$ with 1 representing power and 0 representing failure. Let us figure out what such a behavior β would look like within the context of Definition 5.

Each trace of the component can be represented as a vector $(1,t,0)$, representing that x changes value from 1 to 0 at time t. That is, since the component moves from a state of power generation to failure, and never back in the other direction, the probability mass of e.g. traces taking the form $(1,t_1,0,t_2,1)$ is 0. On the other hand, for any set Θ of traces of the form $(1,t,0)$, it suffices to view each trace through the time t of failure, which is exponentially distributed according to Fig. 1. More precisely, the probability $\beta(\Theta)$ is uniquely determined by an exponential distribution with parameter $1/20$.

Let $\mathtt{beh}(E)$ denote the set of all possible behaviors over a non-empty set $E \subseteq X$ of variables. We now extend behaviors into *input/output behaviors*, which intuitively have control over output variables while being dependent on input variables.

Definition 6 (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 $\beta : \mathtt{tr}(I) \to \mathtt{beh}(O)$.

Given a possibly empty set $I \subseteq X$ and a non-empty set $O \subseteq X$ of variables, let $\mathtt{beh}(I,O)$ denote the set of all possible input/output behaviors from I to O. Furthermore, for any input/output behavior β from I to O, let $\mathtt{in}(\beta)$ and $\mathtt{out}(\beta)$ denote the sets I and O of input and output variables, respectively, and, for ease of notation, let σ_β denote the σ-algebra $\mathcal{B}(O)$ of $\beta(\cdot)$. From now on, "input/output" is often abbreviated as I/O.

Consider the special case of I/O behavior where $I = \emptyset$. Then an I/O behavior from I to O is a function $\beta : \emptyset \to \mathtt{beh}(O)$. That is, β is a nullary function, or, in other words, a constant, taking a value from $\mathtt{beh}(O)$. Thus, the I/O behavior from I to O reduces to a behavior over O, which means that I/O behaviors are a generalization of behaviors.

3.3 Composition of behaviors

When composing two behaviors β_1 and β_2, we restrict ourselves to the case where β_1 has no input, and its output is exactly the input of β_2, i.e., $\mathtt{in}(\beta_1) = \emptyset$ and $\mathtt{out}(\beta_1) = \mathtt{in}(\beta_2)$. Since these restrictions concern only the variables being used, checking that they are satisfied is trivial. The restrictions result in the convenience that composing β_1 with β_2 results in yet another behavior without input. However, this also implies that "open" systems with unresolved input cannot be constructed using such a composition. It is also not possible to directly compose behaviors β, β', β'' if either (a) β has input from both β' and β'' or (b) both β' and β'' have input from β. Clearly, a less restrictive definition of behavior composition can be created. For instance, whenever β' and β'' have no shared input or output variables, we could define composition for cases (a) and (b) by first constructing the product measure of β' and β'' and then composing it with β. Furthermore, to enable the construction of open systems, we could remove the requirement that $\mathtt{in}(\beta_1) = \emptyset$ and, loosely speaking, apply composition of β_1 and β_2 for each input trace of β_1. However, these generalizations are left out of scope of the current paper.

Another implication of the two restrictions is that it is not possible to compose behaviors whose inputs and outputs form a cycle. However, this is also the case with Bayesian networks, which are defined using directed acyclic graphs [27, 28] and for which the theory illustrates underlying difficulties in handling cycles in probabilistic models [28]. These are closely related in the sense that they, too, describe the joint probabilities of components that depend on each other. Despite not supporting cycles, their usefulness has nonetheless been proven in numerous applications in all kinds of domains [28]. Thus, lifting these restrictions lies outside the scope of the present paper, which comprises a first attempt at devising a framework and algorithm for verifying refinement between specifications.

The composition of β_1 and β_2, denoted $\beta_1 \| \beta_2$, is the I/O behavior from $\mathtt{in}(\beta_1) = \emptyset$ to $\mathtt{out}(\beta_1) \cup \mathtt{out}(\beta_2)$ formed as follows. We first assume that $\beta_2(\cdot)(\Theta_2)$, for any fixed Θ_2, is a measurable function from the measurable space $(\mathtt{tr}(\mathtt{out}(\beta_1)), \sigma_{\beta_1})$ to the measurable space $([0,1], \mathcal{B}([0,1]))$. Since both β_1 and β_2 are non-negative and finite, this is equivalent to assuming that $\beta_2(\cdot)(\Theta_2)$ is Lebesgue integrable. Note that I/O behaviors are not meant to be constructed explicitly, but rather seen as theoretical support for the later defined concepts of specifications, refinement, and probabilistic automaton contracts. Thus, checking fulfilment of this assumption is not necessary in practice, but may nonetheless be done using standard results from measure theory.

In the definition of behavior composition, we use the notation $\sigma_{\beta_1} \otimes \sigma_{\beta_2}$ to denote the σ-algebra generated by the Cartesian product $\sigma_{\beta_1} \times \sigma_{\beta_2}$, i.e. $\sigma_{\beta_1} \otimes \sigma_{\beta_2} = \sigma(\sigma_{\beta_1} \times \sigma_{\beta_2})$. According to [29] (Thm. 5.8.1 and Thm. 2.4.3), $\beta_1 \| \beta_2(\cdot)$ defined as $\beta_1 \| \beta_2(\Theta_1 \times \Theta_2) = \int_{\Theta_1} \beta_2(\theta_1)(\Theta_2)\beta_1(d\theta_1)$ is a probability measure of sets $\Theta_1 \times \Theta_2 \in \sigma_{\beta_1} \times \sigma_{\beta_2}$, and its unique extension is a probability measure on the product σ-algebra $\sigma_{\beta_1} \otimes \sigma_{\beta_2}$. This result is the basis for the following definition.

Definition 7 (Composition of I/O Behaviors)
Let β_1 and β_2 be two I/O behaviors such that $\mathtt{in}(\beta_1) = \emptyset$, $\mathtt{in}(\beta_2) = \mathtt{out}(\beta_1)$, and, for any $\Theta_2 \in \sigma_{\beta_2}$, the function $\beta_2(\cdot)(\Theta_2)$ is a measurable function from $(\mathtt{tr}(\mathtt{out}(\beta_1)), \sigma_{\beta_1})$ to $([0,1], \mathcal{B}([0,1]))$. The *composition of β_1 and β_2*, denoted $\beta_1 \| \beta_2$, is an I/O behavior from $\emptyset$ to $\mathtt{out}(\beta_1) \cup \mathtt{out}(\beta_2)$, i.e. a probability measure

$$\beta_1 \| \beta_2 \in \mathtt{beh}(\mathtt{out}(\beta_1) \cup \mathtt{out}(\beta_2)) \,,$$

defined by

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

and its unique extension defined on $\sigma_{\beta_1} \otimes \sigma_{\beta_2} = \mathcal{B}(\mathtt{out}(\beta_1) \cup \mathtt{out}(\beta_2))$.

3.4 Specifications

Definition 8 (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 behavior $\mathtt{beh}(I,O)$, i.e., $\Sigma \subseteq \mathtt{beh}(I,O)$.

Example 4 (Specification)
Consider again the main power specification depicted in Fig. 5 and let x_m be a variable that ranges over $\{\text{on}, \text{off}\}$ and represents the main power status. Formally, the specification is the set of all I/O behaviors β such that $\beta()(\Theta) \leq 0.3$ whenever Θ consists of traces θ for which $\theta(t) = \{x_m \mapsto \text{off}\}$ for some time $t \in [0,6)$.

Definition 9 (Implements)
An I/O behavior β from I to O *implements* a specification Σ from I to O if $\beta \in \Sigma$.

Definition 10 (Refines)
A specification Σ_1 from I to O *refines* a specification Σ_2 from I to O if $\Sigma_1 \subseteq \Sigma_2$.

Example 5 (Refines)
Let Σ denote the specification in Example 4, which represents the main power specification of Section 2. Recall the main power behavior depicted in Fig. 1, containing a single exponential distribution. Let us consider the possible parameters to the exponential distribution causing the behavior of the component to implement Σ. More precisely, let Σ_{exp} be the set of all behaviors β such that $\beta()$ is an exponential distribution and $\beta()(\Theta) \leq 0.3$ whenever Θ consists of traces θ for which $\theta(t) = \{x_m \mapsto \text{off}\}$ for some time $t \in [0,6)$. Then Σ_{exp} refines Σ, but Σ does not refine Σ_{exp}.

The preceding example highlights that exponential distributions are a special case of behaviors. In fact, since behaviors are defined as general probability measures, any concrete class of probability distributions over traces is a special case of behaviors. The example also shows that specifications can be constructed to consist solely of a particular class of distributions, in this case the class of exponential distributions.

Given a possibly empty set $I \subseteq X$ and a non-empty set $O \subseteq X$ of variables such that I and O are disjoint, let $\mathtt{spec}(I,O)$ denote the set of all possible specifications from I to O. We extend the notation $\mathtt{in}(\Sigma)$ and $\mathtt{out}(\Sigma)$ from I/O behaviors to specifications in the obvious way.

Note that, according to Definition 7, $\beta_1 \| \beta_2$ is only defined for cases where $\mathtt{in}(\beta_1) = \emptyset$, $\mathtt{in}(\beta_2) = \mathtt{out}(\beta_1)$, and $\beta_2(\cdot)(\Theta_2)$ is a measurable function from $(\mathtt{out}(\beta_1), \sigma_{\beta_1})$ to $([0,1], \mathcal{B}([0,1]))$. Behaviors fulfilling these conditions are said to be compatible. Likewise, two specifications Σ_1 and Σ_2 are said to be compatible if each $\beta_1 \in \Sigma_1$ is compatible with each $\beta_2 \in \Sigma_2$. Note that a prerequisite for this is that $\mathtt{in}(\Sigma_1) = \emptyset$ and $\mathtt{out}(\Sigma_1) = \mathtt{in}(\Sigma_2)$. Again, the following definition is in accordance with [24].

Definition 11 (Parallel Composition of Specifications)
Given two compatible specifications Σ_1 and Σ_2, the *parallel composition of* Σ_1 *and* Σ_2, denoted $\Sigma_1 \| \Sigma_2$, is the specification $\Sigma_1 \| \Sigma_2 = \{\beta_1 \| \beta_2 \mid \beta_1 \in \Sigma_1, \beta_2 \in \Sigma_2\}$.

The essence of this definition is that we can take any pair $\beta_1 \in \Sigma_1$ and $\beta_2 \in \Sigma_2$, and be sure that $\beta_1 \| \beta_2 \in \Sigma_1 \| \Sigma_2$.

3.5 Trace automata

The specification language presented in this paper, as well as its semantics and the refinement verification method, are based on timed automata, as introduced by Alur and Dill [30, 31]. The following definitions follow closely this literature, except that traces are assumed as input, rather than timed words, to fit the current setting.

Let a clock be a variable ranging over the entire timeline $\mathbb{R}_{\geq 0}$. We use the notation v_C for a valuation for clocks from a set C, as opposed to v, which is used for a valuation for variables from X. For $t \in \mathbb{R}_{\geq 0}$, let $v_C + t$ denote the clock valuation $\{c \mapsto v_C(c) + t \mid c \in C\}$. Given a set $C = \{c_1, \ldots, c_l\}$ 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 $\mathbb{Q}$. A clock valuation v_C for C is said to *satisfy* a clock constraint δ on C if $\delta[c_1 \mapsto v_C(c_1), \ldots, c_l \mapsto v_C(c_l)]$ evaluates to true in the usual logic sense. Note that it is possible to formulate clock constraints *true* and *false*, being satisfied by every clock valuation and being satisfied by no clock valuation, respectively. Given a set C of clocks, let $\Delta(C)$ denote the set of all possible clock constraints on C.

Definition 12 (Timed Automaton)
A *timed automaton* is a tuple $\mathcal{A} = \langle V, L, l_0, C, \rightarrow, F \rangle$ where V is a non-empty alphabet, L is a non-empty finite set of locations, $l_0 \in L$ is a start location, C is a non-empty finite set of clocks, $\rightarrow \subseteq L \times V \times 2^C \times \Delta(C) \times L$ is a transition relation, and $F \subseteq L$ is a set of accepting locations.

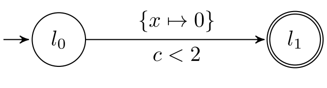
For a timed automaton $\mathcal{A} = \langle V, L, l_0, C, \rightarrow, F\rangle$, let $V_\mathcal{A}$, $L_\mathcal{A}$, $l_{0_\mathcal{A}}$, $C_\mathcal{A}$, $\rightarrow_\mathcal{A}$ and $F_\mathcal{A}$ denote the elements V, L, l_0, C, $\rightarrow$, and F, respectively. Note that in accordance with e.g. [31, 32] we enforce finite sets of clocks and locations and in clock constraints compare clocks only to rational numbers $k \in \mathbb{Q}$. These conditions ensure finite search spaces for the path explorations of timed automata, see more details in Section 4. Note also that we do not allow invariants in the form of clock constraints within locations that must hold at all times. This restriction is made for simplicity, and adding support for such invariants would not restrain the method for verifying refinement presented in Section 4. Furthermore, introducing invariants would only require minor additions to the semantics of timed automata, or, alternatively, be treated by using a translation into additional (perhaps implicit) locations and transitions. Using such a translation, the violation of an invariant may be represented by a transition into a designated "trap" location.

A timed automaton is said to be deterministic if, for each pair of distinct transitions originating from the same location and sharing the same alphabet symbol, there exists no clock valuation satisfying the clock constraints of both transitions. As will become evident in the definition of a run, even deterministic timed automata allow a sort of timed nondeterminism, in which the clock constraint of a transition is satisfied, yet time may pass without the transition being used. However, to ensure that each trace gives rise to exactly one sequence of locations, we require that no transitions are "missed". That is, whenever a transition can be made, then it eventually will be, before any others are made.

In what follows, only a special class of timed automata, called trace automata, is considered. These are characterized by the fact that their alphabets consist of variable valuations, resulting in the ability to read traces as input.

Definition 13 (Trace Automaton)
Given a non-empty set $E \subseteq X$ of variables, a *trace automaton for E* is a deterministic timed automaton $\mathcal{A} = \langle V, L, l_0, C, \rightarrow, F\rangle$ such that $V = \mathtt{val}(E)$.

Example 6 (Trace automaton)
Let x be a variable that ranges over $\mathbb{R}$. Consider the following automaton:



It contains a single clock c and a single transition from a location l_0 to an accepting location l_1 on the condition that x takes the value 0 within 2 time units. The exact semantics of such an automaton, as well as its interpretation as a set of traces, is explained in the remainder of this section.

We define the semantics of trace automata in the following definition of a *run*. Simply put, a run of an automaton $\mathcal{A}$ on a trace θ is the sequence of steps that the automaton takes while reading θ. In each step i, both the location l_i and the time interval I_i spent there are recorded together with the next clock valuation $\nu_{C_{i+1}}$ and the clocks R_{i+1} to be reset upon the next step. Given a run, the time intervals I_i may be split up into smaller pieces to form yet another run with more steps, meaning that runs are not unique. At each new step $i+1$, the automaton takes any outgoing transition $\langle l_i, \nu_{i+1}, R_{i+1}, \delta, l_{i+1}\rangle$ if, at the start of the new time interval I_{i+1}, δ is satisfied and θ outputs ν_{i+1}. This increases the clock valuation by the time spent between the two steps and then resets the clocks in R_{i+1}. If no such transition exists, the clock valuation is simply increased by the time spent between the two steps.

Definition 14 (Run)
Given a set $E \subseteq X$ of variables, a trace automaton $\mathcal{A} = \langle V, L, l_0, C, \rightarrow, F\rangle$ for E, and a trace θ over E, a *run* ρ of $\mathcal{A}$ on θ is a (finite or infinite) sequence

$$\rho = \xrightarrow[\nu_{C_0}]{} (l_0, I_0) \xrightarrow[\nu_{C_1}]{R_1} (l_1, I_1) \xrightarrow[\nu_{C_2}]{R_2} (l_2, I_2) \xrightarrow[\nu_{C_3}]{R_3} \cdots,$$

where $\nu_{C_i} \in \mathtt{val}(C)$ are clock valuations, $l_i \in V$ are locations, $I_0 I_1 I_2 \ldots$ is an interval sequence, and $R_i \subseteq C$ are clock sets, such that the following holds:

- For each clock $c \in C$, the initial valuation ν_{C_0} satisfies $\nu_{C_0}(c) = 0$.
- For each $i \in \mathbb{N}$, the trace θ remains constant throughout the interval I_i. That is, for each $t, t' \in I_i$, the equality $\theta(t) = \theta(t')$ holds.
- For each $i \in \mathbb{N}$, if there exists a transition $\langle l_i, \nu, R, \delta, l\rangle$ in $\rightarrow$ such that
  - $\nu = \theta(r(I_i))$,
  - the clock valuation $\nu_{C_i} + r(I_i) - l(I_i)$ satisfies δ,

  then it holds that $R_{i+1} = R$, $\nu_{C_{i+1}} = \left(\nu_{C_i} + r(I_i) - l(I_i)\right) \times [R_{i+1} \mapsto 0]$, and $l_{i+1} = l$. That is, the transition is made. If no such transition exists, then $R_{i+1} = \emptyset$, $\nu_{C_{i+1}} = (\nu_{C_i} + (r(I_i) - l(I_i)))$, and $l_{i+1} = l_i$. That is, no transition is made.
- For each $i \in \mathbb{N}$, if there exists a time $l(I_i) < t < r(I_i)$ and a transition $\langle l_i, \nu, R, \delta, l\rangle$ in $\rightarrow$ such that
  - $\nu = \theta(t)$,
  - the clock valuation $\nu_{C_i} + t - l(I_i)$ satisfies δ,

  then also
  - $\nu = \theta(r(I_i))$,
  - the clock valuation $\nu_{C_i} + r(I_i) - l(I_i)$ satisfies δ.

  That is, the transition is not missed.

Example 7 (Run)
Consider the trace automaton from Example 6 and let θ be the trace

$$\theta(t) = \begin{cases} \{x \mapsto 5\}, & t \in [0,1) \\ \{x \mapsto 0\}, & t \in [1, \inf) \end{cases}.$$

One possible run of the automaton on the trace θ is the sequence

$$\rho_1 = \xrightarrow[\{c=0\}]{} (l_0,[0,1)) \xrightarrow[\{c=1\}]{\emptyset} (l_1,[1,\infty)) \ .$$

Since the automaton *must* transition at time $t = 1$ when the trace changes values, all other possible runs can be constructed by splitting the intervals into smaller pieces. For instance,

$$\rho_2 = \xrightarrow[\{c=0\}]{} (l_0,[0,0.6)) \xrightarrow[\{c=0.6\}]{\emptyset} (l_0,[0.6,1)) \xrightarrow[\{c=1\}]{\emptyset} (l_1,[1,\infty))$$

is another possible run.

Given a run ρ, let ρ^* denote the function associating each time-point with the current location in ρ. That is, given a run $\rho = \xrightarrow[\nu_{C_0}]{} (l_0,I_0) \xrightarrow[\nu_{C_1}]{R_1} (l_1,I_1) \xrightarrow[\nu_{C_2}]{R_2} (l_2,I_2) \xrightarrow[\nu_{C_3}]{R_3} \cdots$ of a trace automaton $\mathcal{A} = \langle V, L, l_0, C, \rightarrow, F\rangle$, the function $\rho^* : \mathbb{R}_{\geq 0} \rightarrow L$ satisfies $\rho^*(t) = l_i$ for each $i \in \mathbb{N}$ and $t \in I_i$.

Given a trace automaton $\mathcal{A}$, a trace θ, and a run ρ of $\mathcal{A}$, let $\mathcal{A}(\theta)$ denote the sequence of locations visited along ρ^*. Note that, since any trace automaton $\mathcal{A}$ is deterministic, all runs of $\mathcal{A}$ on any given trace θ share the same sequence of locations $\mathcal{A}(\theta)$.

Definition 15 (Path)
Given a set $E \subseteq X$ of variables and a trace automaton $\mathcal{A} = \langle V, L, l_0, C, \rightarrow, F\rangle$ for E, a (finite or infinite) sequence π of locations from L is a *path of* $\mathcal{A}$ if there exists a trace $\theta \in \mathtt{tr}(E)$ such that $\pi = \mathcal{A}(\theta)$.

Given a trace automaton $\mathcal{A}$ for E, let $\mathtt{paths}(\mathcal{A})$ denote the set of all possible paths of $\mathcal{A}$.

Definition 16 (Terminating)
Given a set $E \subseteq X$ of variables and a trace automaton $\mathcal{A}$ for E, if each path $\pi \in \mathtt{paths}(\mathcal{A})$ is finite, then $\mathcal{A}$ is *terminating*.

Henceforth, we will only consider terminating trace automata. This is done both for the sake of simplicity and to provide a refinement verification algorithm that is guaranteed to terminate. Note that terminating automata still allow us to express safety properties over infinite traces, such as "The system shall never crash". It can also be noted that some types of liveness properties are not possible to express using terminating automata, such as "At all times, each request shall be followed by an answer". However, we can still express liveness properties such as "The system eventually finishes", or liveness within bounded time, such as "During the system lifetime of 10,000 hours, each request shall be followed by an answer".

Let $\mathbb{A}_E$ denote the set of all terminating trace automata for any non-empty set of variables $E \subseteq X$, and let $\mathbb{A}_\emptyset = \emptyset$ by convention. For a path $\pi = l_0 l_1 \ldots l_k$ of an automaton $\mathcal{A} \in \mathbb{A}_E$, let $\mathtt{last}(\pi)$ denote the last visited location l_k. Furthermore, if π is a path of $\mathcal{A}$, then let $\Theta_{\mathcal{A}}(\pi)$ denote the set of all traces $\theta \in \mathtt{tr}(E)$ corresponding to π. That is, $\Theta_{\mathcal{A}}(\pi) = \{\theta \in \mathtt{tr}(E) \mid \mathcal{A}(\theta) = \pi\}$. As an extension, if Π is a set of paths of $\mathcal{A}$, then $\Theta_{\mathcal{A}}(\Pi) = \{\Theta_{\mathcal{A}}(\pi) \mid \pi \in \Pi\}$.

Definition 17 (Accepts)
Given a non-empty set $E \subseteq X$ of variables, a terminating trace automaton $\mathcal{A} \in \mathbb{A}_E$, and a trace $\theta \in \mathtt{tr}(E)$, $\mathcal{A}$ *accepts* θ if $\mathtt{last}(\mathcal{A}(\theta)) \in F$.

For any terminating trace automaton $\mathcal{A} \in \mathbb{A}_E$, let $[\![\mathcal{A}]\!]$ denote the set of all traces that $\mathcal{A}$ accepts. Note that, according to Proposition 5 in Appendix B, such trace sets are σ_β-measurable for any behavior β over E.

Example 8 (Accepts)
Consider the trace θ from Example 7 and the automaton $\mathcal{A}$ from Examples 6 and 7. From the runs of Example 7, we see that $\mathcal{A}(\theta) = l_0 l_1$ and $\mathtt{last}(\mathcal{A}(\theta)) = l_1$. Since $l_1 \in F_{\mathcal{A}}$ is an accepting location, the automaton $\mathcal{A}$ accepts θ. Due to the clock constraint $c < 2$, the set $[\![\mathcal{A}]\!]$ is the set of all traces in which x takes the value 0 within 2 time units. That is, $[\![\mathcal{A}]\!] = \{\theta \in \mathtt{tr}(\{x\}) \mid \exists t \in [0,2) \ . \ \theta(t)(x) = 0\}$.

Given trace automata $\mathcal{A}_1 \in \mathbb{A}_{E_1}$ and $\mathcal{A}_2 \in \mathbb{A}_{E_2}$, the composition of $\mathcal{A}_1$ and $\mathcal{A}_2$, denoted $\mathcal{A}_1 \| \mathcal{A}_2$, is the trace automaton giving their joint run. In simple terms, whenever two individual transitions of $\mathcal{A}_1$ and $\mathcal{A}_2$ agree, they give rise to a joint transition of $\mathcal{A}_1 \| \mathcal{A}_2$ that encapsulates the effect of them both. To cover the remaining possibilities, each individual transition of $\mathcal{A}_i$, $i = 1,2$, also gives rise to a joint transition that encapsulates the effects on $\mathcal{A}_i$, while the other automaton remains stationary. This is captured in the following definition.

Definition 18 (Composition of Trace Automata)
Let $\mathcal{A}_1 = \langle V_1, L_1, l_{0_1}, C_1, \rightarrow_1, F_1\rangle \in \mathbb{A}_{E_1}$ be a trace automaton for E_1 and $\mathcal{A}_2 = \langle V_2, L_2, l_{0_2}, C_2, \rightarrow_2, F_2\rangle \in \mathbb{A}_{E_2}$ be a trace automaton for E_2. Then the *composition of* $\mathcal{A}_1$ *and* $\mathcal{A}_2$, *denoted* $\mathcal{A}_1 \| \mathcal{A}_2$, is the timed automaton $\mathcal{A}_1 \| \mathcal{A}_2 = \langle \mathtt{val}(E_1 \cup E_2), L_1 \times L_2, (l_{0_1}, l_{0_2}), C_1 \cup C_2, \rightarrow_1 \| \rightarrow_2, F_1 \times F_2\rangle$ where $\rightarrow_1 \| \rightarrow_2$ is the smallest set such that, for each pair of locations $(l_1, l_2) \in L_1 \times L_2$ and each valuation $\nu \in \mathtt{val}(E_1 \cup E_2)$,

(i) it contains a transition

$$\langle (l_1,l_2), \nu, r_1 \cup r_2, \delta_1 \wedge \delta_2, (l_1', l_2')\rangle$$

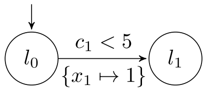


Fig. 8 A trace automaton $\mathcal{A}_1$

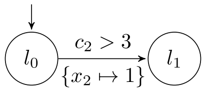


Fig. 9 A trace automaton $\mathcal{A}_2$

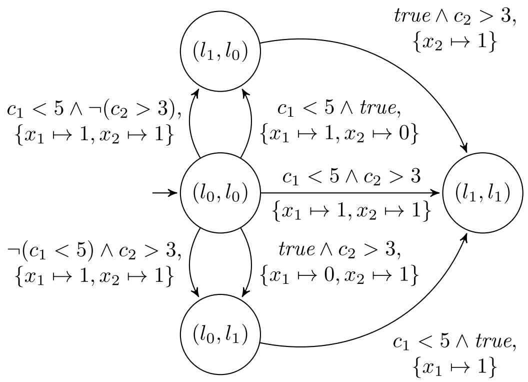

(l_1, l_0)
$true \wedge c_2 > 3,$
$\{x_2 \mapsto 1\}$
$c_1 < 5 \wedge \neg(c_2 > 3),$
$\{x_1 \mapsto 1, x_2 \mapsto 1\}$
$c_1 < 5 \wedge true,$
$\{x_1 \mapsto 1, x_2 \mapsto 0\}$
(l_0, l_0)
$c_1 < 5 \wedge c_2 > 3$
$\{x_1 \mapsto 1, x_2 \mapsto 1\}$
(l_1, l_1)
$\neg(c_1 < 5) \wedge c_2 > 3,$
$\{x_1 \mapsto 1, x_2 \mapsto 1\}$
$true \wedge c_2 > 3,$
$\{x_1 \mapsto 0, x_2 \mapsto 1\}$
(l_0, l_1)
$c_1 < 5 \wedge true,$
$\{x_1 \mapsto 1\}$

Fig. 10 The composition $\mathcal{A}_1 \| \mathcal{A}_2$

for every pair of individual transitions $\langle l_1, v_1, r_1, \delta_1, l_1' \rangle \in \rightarrow_1$ and $\langle l_2, v_2, r_2, \delta_2, l_2' \rangle \in \rightarrow_2$ satisfying $v_1 = v|_{E_1}$ and $v_2 = v|_{E_2}$.

(ii) it contains a transition

$$\langle (l_1, l_2), v, r_1, \delta_1 \wedge \neg\delta, (l_1', l_2) \rangle$$

or a transition

$$\langle (l_1, l_2), v, r_2, \neg\delta \wedge \delta_2, (l_1, l_2') \rangle$$

for every individual transition $\langle l_i, v_i, r_i, \delta_i, l_i' \rangle \in \rightarrow_i$, $i = 1,2$, satisfying $v_i = v|_{E_i}$ and such that the following holds. Let $i' = 3 - i$ denote the other of the two composition components, and let $\delta_{i_1'}, \delta_{i_2'}, \ldots$ be all clock constraints corresponding to transitions $\langle l_{i'}, v_{i'}, r_{i'}, \delta_{i'_j}, l_{i'}' \rangle \in \rightarrow_{i'}$ satisfying $v_{i'} = v|_{E_{i'}}$, that is, being mutually executable. Then δ is the clock constraint $\delta_{i_1'} \vee \delta_{i_2'} \vee \ldots$. If no such clock constraints $\delta_{i_1'}, \delta_{i_2'}, \ldots$ exist, then δ is the clock constraint *false*, implying that $\neg\delta$ is the clock constraint *true*.

Given a joint location $l = (l_1, l_2) \in L_{\mathcal{A}_1 \| \mathcal{A}_2}$, let $l|_{\mathcal{A}_1}$ and $l|_{\mathcal{A}_2}$ denote the individual locations l_1 and l_2, respectively.

Example 9 (Automata Composition)
To illustrate Definition 18, Fig. 8-10 depict two simple trace automata $\mathcal{A}_1 \in E_1$ and $\mathcal{A}_2 \in E_2$, and their composition $\mathcal{A}_1 \| \mathcal{A}_2$. Assume that, for $i \in \{1,2\}$, $E_i = \{x_i\}$ where x_i ranges over the set $\{0,1\}$. Thus, for each E_i, there are only two possible valuations $\{x_i \mapsto 0\}$ and $\{x_i \mapsto 1\}$. For joint transitions of $\mathcal{A}_1 \| \mathcal{A}_2$, we will use shorthand notation such that e.g. a transition with the constraint $\{x_1 \mapsto 1\}$ represents two distinct transitions, where one has the constraint $\{x_1 \mapsto 1, x_2 \mapsto 0\}$ and the other has the constraint $\{x_1 \mapsto 1, x_2 \mapsto 1\}$. Consider the cases (i) and (ii) from Definition 18. The composition $\mathcal{A}_1 \| \mathcal{A}_2$ contains only one transition from the case (i), namely from (l_0, l_0) to (l_1, l_1). This transition corresponds to both $\mathcal{A}_1$ and $\mathcal{A}_2$ transitioning at the same time. The remaining six transitions are from the case (ii), the four rightmost being cases where a tautology *true* is added. To see why such tautologies are introduced, consider the individual transition of $\mathcal{A}_1$ as the joint automaton is residing in the location (l_0, l_1). As $\mathcal{A}_2$ has no outgoing transitions from l_1, it should not matter what the value of the clock c_2 is. And in fact, a tautology on c_2 accurately represents this non-condition, while still allowing the transition to neatly fall into case (ii) instead of needing a separate case.

Proposition 1
Given terminating trace automata $\mathcal{A}_1$ and $\mathcal{A}_2$, the composition $\mathcal{A}_1 \| \mathcal{A}_2$ is also a terminating trace automaton.

Proof
We need to prove that $\mathcal{A}_1 \| \mathcal{A}_2$ is both deterministic and terminating. We begin by proving termination. Assume the contrary, namely that there exists a trace θ causing the path $(\mathcal{A}_1 \| \mathcal{A}_2)(\theta)$ to be an infinite sequence. Let E_i, for $i = 1,2$, denote the set of variables such that $\mathcal{A}_i$ is a trace automaton for E_i. Since each transition of $\mathcal{A}_1 \| \mathcal{A}_2$ corresponds to at least one transition of $\mathcal{A}_1$ or $\mathcal{A}_2$, it must be the case that either $\mathcal{A}_1(\theta|_{E_1})$ or $\mathcal{A}_2(\theta|_{E_2})$ is an infinite sequence. However, this contradicts the premise that both $\mathcal{A}_1$ and $\mathcal{A}_2$ are terminating.

To prove determinism, consider a location $(l_1, l_2) \in L_1 \times L_2$, a valuation $v \in \mathtt{val}(E_1 \cup E_2)$ and two distinct transitions $\langle (l_1, l_2), v, R, \delta_1 \wedge \delta_2, (l_1', l_2') \rangle$ and $\langle (l_1, l_2), v, R', \delta_1' \wedge \delta_2', (l_1'', l_2'') \rangle$ in $\rightarrow_1 \| \rightarrow_2$. We want to show that their clock constraints mutually exclusive in the usual logic sense. We have three cases: either both transitions are of the form (i) of Definition 18, or both transitions are of the form (ii), or one is of the form (i) and the other is of the form (ii).

If both transitions are of the form (i), then, for either $i = 1$ or $i = 2$, δ_i and δ_i' are clock constraints of two distinct transitions of $\mathcal{A}_i$. Because $\mathcal{A}_i$ itself is deterministic, there exists no clock valuation v_C satisfying both δ_i and δ_i' and thus no v_C satisfying both $\delta_1 \wedge \delta_2$ and $\delta_1' \wedge \delta_2'$.

If both transitions are of the form (ii), either (a) they are the result of two distinct transitions from the same automaton $\mathcal{A}_i, i \in \{1,2\}$, or (b) one is from $\mathcal{A}_1$ and the other is from $\mathcal{A}_2$. For case (a), the same reasoning as above applies. That is, the two individual clock constraints are mutually exclusive, implying that so is the case for the two joint transitions. For case (b), without loss of generality, let δ_1 be the clock constraint of the transition taken from $\mathcal{A}_1$ and let $\delta_\vee$ be the negated disjunction $\neg(\delta_{1_1} \vee \delta_{1_2} \vee \dots)$ of the other joint transition, in accordance with Definition 18. Note that each δ_{1_j} contains only clocks from $\mathcal{A}_1$. By Definition 18, the clock constraint $\delta_{\vee}$ is constructed as to be mutually exclusive with δ_1. Thus, there exists no clock valuation satisfying both clock constraints of the two joint transitions.

Lastly, if one transition is of the form (i) and the other transition is of the form (ii), then the reasoning from case (b) applies again. □

3.6 Probabilistic automaton contracts

For specifying I/O behaviors in practice, we will use a contract-based approach. A probabilistic automaton contract consists of an assumption and a guarantee together with a probability bound. The interpretation of such a contract is a specification, i.e. a set of behaviors. Intuitively, an I/O behavior implements a contract if, for each input trace satisfying the assumption, the probability over all output traces satisfying the guarantee respects the probability bound. Both the assumption and the guarantee are specified using terminating trace automata. An advantage of using automata rather than temporal logic for specifying system properties is their flexibility—while temporal logics offer their own advantages, it may be difficult, or even impossible, to specify some complex systems using them [33]. In general, it is always possible to construct some automaton corresponding to a given temporal logic formula.

For convenience, we will also allow a special non-assumption $\top$ that carries the meaning of always being satisfied. We use the convention that composing any automaton $\mathcal{A}$ with $\top$ results in $\mathcal{A}$ itself, so that $\mathcal{A}\|\top = \top\|\mathcal{A} = \mathcal{A}$. We also extend the notion of acceptance and the notation $[\![\cdot]\!]$ to the non-assumption $\top$, so that $\top$ is considered to accept each possible trace.

Definition 19 (Probabilistic Automaton Contract)
Given a set of variables $I \subseteq X$, a non-empty set of variables $O \subseteq X$ disjoint from I, an assumption $\mathcal{A} \in \mathbb{A}_I \cup \{\top\}$, a guarantee $\mathcal{G} \in \mathbb{A}_{I \cup O}$, a probability value $p \in [0,1]$, and a comparison operator $\bowtie \in \{<, \leq, \geq, >\}$, a formula $\phi = \mathcal{P}_{\bowtie p}(\mathcal{A}, \mathcal{G})$ *is a probabilistic automaton contract (PAC) from I to O.*

Once again, we extend the notation $\mathtt{in}(\phi)$ and $\mathtt{out}(\phi)$ from I/O behaviors and specifications in the obvious way. For a PAC $\phi = \mathcal{P}_{\bowtie p}(\mathcal{A}, \mathcal{G})$, let $\mathcal{A}_\phi$, $\mathcal{G}_\phi$, p_ϕ, and $\bowtie_\phi$ denote its assumption $\mathcal{A}$, guarantee $\mathcal{G}$, probability value p, and comparison operator $\bowtie$, respectively.

To understand trace composition in the following definition of the interpretation of a PAC, consider two traces θ_1 and θ_2 over disjoint sets of variables E_1 and E_2, respectively. The composition of θ_1 and θ_2 is the trace $\theta_1\|\theta_2 : \mathbb{R}_{\geq 0} \to \mathtt{val}(E_1 \cup E_2)$ such that $(\theta_1\|\theta_2)(t)(x)$ equals $\theta_1(t)(x)$ if $x \in E_1$ and $\theta_2(t)(x)$ if $x \in E_2$.

Definition 20 (PAC Interpretation)
Given a set of variables $I \subseteq X$, a non-empty set of variables $O \subseteq X$, and a PAC $\phi = \mathcal{P}_{\bowtie p}(\mathcal{A}, \mathcal{G})$ from I to O, the *interpretation of ϕ*, written $[\![\phi]\!]$, is the largest specification, i.e. the largest set of I/O behaviors from I to O, such that, for each $\beta \in [\![\phi]\!]$,

1. if $I = \emptyset$ then $\beta([\![\mathcal{G}]\!]) \bowtie p$,
2. if $I \neq \emptyset$ then for each trace $\theta_I \in [\![\mathcal{A}]\!]$, it holds that $\beta(\theta_I)(\{\theta_O \in \mathtt{tr}(O) \mid \theta_I\|\theta_O \in [\![\mathcal{G}]\!]\}) \bowtie p$.

For a PAC $\phi = \mathcal{P}_{\bowtie p}(\mathcal{A}, \mathcal{G})$, let ϕ^c denote the PAC $\mathcal{P}_{\bowtie^c p}(\mathcal{A}, \mathcal{G})$, in which the comparison operator has been complemented, where the complement of $<$ is $\geq$ and vice versa, and the complement of $\leq$ is $>$ and vice versa. The PAC ϕ^c is called the complement of ϕ.

Of course, one could imagine the possibility of creating more complex, even nested, contract-based formulae following a recursively defined grammar. For instance, this might include combining PACs using negation, conjunction and disjunction as well as defining an until operator and nesting PACs within PACs. Although this possibility might be of interest, it lies out of scope of the present paper. Instead, as introduced in the next definition, we will consider so-called composite PACs, which consist of multiple PACs and represent their parallel composition. As with composition of behaviors and specifications, in the next definition, we consider only the case of $\mathtt{in}(\phi_1) = \emptyset$ and $\mathtt{in}(\phi_2) = \mathtt{out}(\phi_1)$.

Definition 21 (Composite Probabilistic Automaton Contract)
Given two PACs ϕ_1 and ϕ_2 such that $[\![\phi_1]\!]$ and $[\![\phi_2]\!]$ are compatible, the formula $\phi_1\|\phi_2$ is a *composite probabilistic automaton contract (cPAC)* with interpretation $[\![\phi_1\|\phi_2]\!] = [\![\phi_1]\!]\|[\![\phi_2]\!]$. Inductively, let ϕ_1 be a PAC or cPAC and ϕ_2 be a PAC or cPAC such that $[\![\phi_1]\!]$ and $[\![\phi_2]\!]$ are compatible. Then the formula $\phi_1\|\phi_2$ is a cPAC.

The notation $\mathtt{in}(\phi)$ and $\mathtt{out}(\phi)$ is extended also to cPACs ϕ, and the notions of *implement* and *refine* are extended to PACs and cPACs by defining that β *implements* ϕ if $\beta \in [\![\phi]\!]$, and ϕ_1 *refines* ϕ_2 if $[\![\phi_1]\!] \subseteq [\![\phi_2]\!]$.

4 Verification of refinement

In this section, we present an algorithm for verifying refinement between specifications.

A common technique for formal verification in general, found throughout literature, is to formulate specifications using automata and then solve the corresponding language inclusion problem using automata theory [33, 34]. However, standard approaches assume a non-probabilistic setting. In these approaches, checking the inclusion $[\![\mathcal{A}]\!] \subseteq [\![\mathcal{A}_0]\!]$ is typically done by determining language emptiness of the intersection $[\![\mathcal{A}]\!] \cap [\![\mathcal{A}_0]\!]^c$, in which the language of $\mathcal{A}_0$ has been complemented. This is also the foundation for the method developed in this paper, except here, languages are sets of I/O behaviors instead of sets of strings. In our context, we check language emptiness of $[\![\phi]\!] \cap [\![\phi_0]\!]^c$, which represents a lack of counterexamples to the refinement statement $\phi \sqsubseteq \phi_0$. To account for probability, emptiness of $[\![\phi]\!] \cap [\![\phi_0]\!]^c$ is checked by determining whether a system of linear inequalities constraining the probabilities lacks solution.

The algorithm presented here takes two inputs ϕ and ϕ_0 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 later. We assume that ϕ is a PAC or cPAC $\phi = \phi_1 \| \phi_2 \| .. \| \phi_m$, where each ϕ_i, $i = 1, \ldots, m$, is a PAC representing a component specification. Furthermore, we assume that ϕ_0 is a PAC representing the top-level specification. The intuition for verifying that ϕ refines ϕ_0 is as follows. We want to verify that the set $[\![\phi]\!]$ is a subset of $[\![\phi_0]\!]$, or, equivalently, that the specification $[\![\phi]\!] \cap [\![\phi_0]\!]^c$ is the empty set. This is done by proving that it is impossible to construct a probability measure that satisfies each bound $\mathcal{P}_{\bowtie p}(\mathcal{A}, \mathcal{G})$ imposed by either ϕ or ϕ_0^c. To do so, we express each such bound as a linear inequality, resulting in a system of linear inequalities that can be solved using, e.g., the simplex method [35, 36]. Each variable in this system of inequalities represents the probability value of a joint path through the automata in ϕ and ϕ_0. Thus, to express a bound $\mathcal{P}_{\bowtie p}(\mathcal{A}, \mathcal{G})$ as an inequality, we simply use the conditional probability of all accepted paths of the guarantee $\mathcal{G}$, given that they are accepted by the assumption $\mathcal{A}$, bounded by p according to $\bowtie$.

Due to the difficulties in solving linear inequalities with non-strict inequalities, we restrict the PACs such that each $\bowtie_{\phi_i}$, $i \in \{1, \ldots, m\}$, must be one of $\leq$ or $\geq$, and $\bowtie_{\phi_0}$ must be either $<$ or $>$. In words, 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. This restriction is often insignificant in practice, since any PAC with non-strict bound can be approximated to arbitrary precision by one with a strict bound, and vice versa.

Pseudocode for the algorithm is presented in Algorithm 1. Here, the variable `ineqs` stores the set of linear inequalities. Initially, on line 3, it stores only the equality representing the total probability of the paths being equal to 1. Note that, in the algorithm, each syntactic path π simply represents a mathematical variable, which in turn represents to the probability mass of $\Theta_{\mathcal{A}}(\pi)$. Thus, each inequality added to `ineqs` constrains the probabilities of trace sets that correspond to paths of $\mathcal{A}$. On line 8, `ineqs` is incrementally updated to store the inequality generated from the conditional probability bound given by each ϕ_i, $i \geq 1$, and by ϕ_0^c. The algorithm relies on finding all possible paths of a composite automaton constructed on line 2. Due to Proposition 1, the composite automaton is terminating, meaning that each such path is finite. Therefore, there are only finitely many possible paths, and they can all be found in finite time.

Note that, whenever a solution to the system of linear inequalities is found, the algorithm outputs `unknown` rather than `false`. This is because the algorithm is not complete. More precisely, an identified solution to the inequalities might be spurious in the sense that it represents a behavior not actually implementing all specifications. To see why such spurious solutions may exist, consider one of the probabilistic contracts ϕ_i of the composition, and suppose that it has both input and output variables. Due to Definitions 20 and 21, the composition $[\![\phi]\!]$ respects the probability bound of ϕ_i on every input trace satisfying the assumption. However, the algorithm considers only conditional probabilities rather than individual traces, and, in so doing, lumps traces together and forgets some of the “fine resolution”. Thus, any proposed solution to the system of linear inequalities must only respect the probability bound of ϕ_i on average, over all traces satisfying the assumption. This is a weaker condition than what follows from Definitions 20 and 21, which may result in spurious solutions. These spurious solutions are exactly what causes the algorithm to output `unknown` even though refinement actually holds. In other words, the algorithm is not complete. On the other hand, whenever the algorithm outputs `true`, it is because no solutions are found whatsoever. In this case, we can be certain that refinement does in fact hold, meaning that the algorithm is sound. This is captured in the following theorem. The lemmas are given separately in Appendix A.

Theorem 1
A PAC or cPAC $\phi_1 \| \ldots \| \phi_m$ refines a PAC ϕ_0 if the procedure `Refines`$(\phi_1 \| \ldots \| \phi_m, \phi_0)$ *given by Algorithm 1 returns* `true`.

Proof
By contraposition. We want to prove that whenever $\phi_1 \| \ldots \| \phi_m$ does not refine ϕ_0, `Refines`$(\phi_1 \| \ldots \| \phi_m, \phi_0)$ does not return `true`. To do this, assume there is a behavior $\beta \in$ `beh`(`out`$(\phi_1 \| \ldots \| \phi_m))$ such that $\beta \in [\![\phi_1 \| \ldots \| \phi_m]\!]$ and $\beta \notin [\![\phi_0]\!]$. First, for each $i \in \{0, \ldots, m\}$, let O_i denote `out`(ϕ_i), $\mathcal{A}_i$ denote $\mathcal{A}_{\phi_i}$, $\mathcal{G}_i$ denote $\mathcal{G}_{\phi_i}$, $\bowtie_i$ denote $\bowtie_{\phi_i}$, and p_i denote p_{ϕ_i}. Furthermore, let $\mathcal{A}$ denote the composition $\mathcal{A}_1 \| \mathcal{G}_1 \| \ldots \| \mathcal{A}_m \| \mathcal{G}_m \| \mathcal{A}_{\phi_0} \| \mathcal{G}_{\phi_0}$ and O denote the set `out`$(\phi_1 \| \ldots \| \phi_m) =$ `out`(ϕ_0) of output variables.

Since `in`$(\phi_1) = \emptyset$ and `out`$(\phi_1) =$ `in`(ϕ_2), Lemma 1 implies that, for each $\beta_1 \in [\![\phi_1]\!]$ and $\beta_2 \in [\![\phi_2]\!]$, it holds that $\beta_1([\![\mathcal{G}_1]\!]) = \beta_1 \| \beta_2(\{\theta \mid \theta|_{O_1} \in [\![\mathcal{G}_1]\!]\})$. Note that a condition for applying Lemma 1 is that $[\![\mathcal{G}_1]\!]$ is measurable, which follows from Proposition 5. Because also each $\beta_1 \in [\![\phi_1]\!]$ satisfies $\beta_1([\![\mathcal{G}_1]\!]) \bowtie_1 p_1$ as per Definition 20, it follows,

Algorithm 1 Verify that a PAC or cPAC refines a PAC.

```
function Refines($\phi_1\|\ldots\|\phi_m, \phi_0$)
    $\mathcal{A} = \mathcal{A}_{\phi_1}\|\mathcal{G}_{\phi_1}\|\ldots\|\mathcal{A}_{\phi_m}\|\mathcal{G}_{\phi_m}\|\mathcal{A}_{\phi_0}\|\mathcal{G}_{\phi_0}$
    ineqs $\leftarrow \left\{1 = \sum_{\pi\in\mathtt{paths}(\mathcal{A})}\pi\right\}$
    for $\phi_i \in \{\phi_1,\ldots,\phi_m,\phi_0^c\}$ do
        $\Pi_{\mathcal{A}} \leftarrow \{\pi \in \mathtt{paths}(\mathcal{A}) \mid \mathtt{last}(\pi)|_{\mathcal{A}_{\phi_i}} \in F_{\mathcal{A}_{\phi_i}}\}$
        $\Pi_{\mathcal{G}} \leftarrow \{\pi \in \mathtt{paths}(\mathcal{A}) \mid \mathtt{last}(\pi)|_{\mathcal{G}_{\phi_i}} \in F_{\mathcal{G}_{\phi_i}}\}$
        $\Pi_{\mathcal{G}\wedge\mathcal{A}} \leftarrow \Pi_G \cap \Pi_A$
        ineqs $\leftarrow$ ineqs $\cup \left\{\left(\sum_{\pi\in\Pi_{\mathcal{G}\wedge\mathcal{A}}}\pi\right) / \left(\sum_{\pi\in\Pi_{\mathcal{A}}}\pi\right) \bowtie_{\phi_i} p_{\phi_i}\right\}$
    end for
    return true if the solution space for ineqs is empty; unknown otherwise
end function
```

that for each $\beta_2 \in [\![\phi_2]\!]$, the composition $\beta_1\|\beta_2$ satisfies $\beta_1\|\beta_2(\{\theta \mid \theta|_{O_1} \in [\![\mathcal{G}_1]\!]\}) = \beta_1([\![\mathcal{G}_1]\!]) \bowtie_1 p_1$. Furthermore, since $\mathtt{in}(\phi_2) = \mathtt{out}(\phi_1)$, it follows from Lemma 2 that

$$\frac{\beta_1\|\beta_2(\{\theta \mid \theta|_{O_1\cup O_2} \in [\![\mathcal{G}_2]\!],\ \theta|_{O_1} \in [\![\mathcal{A}_2]\!]\})}{\beta_1\|\beta_2(\{\theta \mid \theta|_{O_1} \in [\![\mathcal{A}_2]\!]\})} \bowtie_2 p_2 .$$

Since once again $\mathtt{in}(\beta_1\|\beta_2) = \emptyset$, we can repeat this procedure, starting from each $\beta_1\|\beta_2 \in [\![\phi_1\|\phi_2]\!]$, until covering all $\beta_1\|..\|\beta_m \in \phi_1\|\ldots\|\phi_m$, preserving

$$\beta(\{\theta \mid \theta|_{O_1} \in \mathcal{G}_1\}) \bowtie_1 p_1 \qquad (1)$$

and, for each $i \in \{2,\ldots,m\}$,

$$\frac{\beta(\{\theta \mid \theta|_{O_1..O_i} \in [\![\mathcal{G}_i]\!],\ \theta|_{O_1..O_{i-1}} \in [\![\mathcal{A}_i]\!]\})}{\beta(\{\theta \mid \theta|_{O_1..O_{i-1}} \in [\![\mathcal{A}_i]\!]\})} \bowtie_i p_i\ , \qquad (2)$$

where $O_1..O_i$ denotes the union $O_1 \cup O_2 \cup \cdots \cup O_i$.

Consider any component automaton $\mathcal{M}$ of the composition $\mathcal{A}$, i.e., $\mathcal{M} = \mathcal{A}_i$ or $\mathcal{M} = \mathcal{G}_i$ for some $i \in \{0,\ldots,m\}$. Then let $\Pi_{\mathcal{M}}$ denote the set of paths of $\mathcal{A}$ ending in an accepting location of $\mathcal{M}$. That is, $\Pi_{\mathcal{M}} = \{\pi \in \mathtt{paths}(\mathcal{A}) \mid \mathtt{last}(\pi)|_{\mathcal{M}} \in F_{\mathcal{M}}\}$. For the non-assumption $\top$, let $\Pi_\top$ denote the set $\mathtt{paths}(\mathcal{A})$ of all paths of $\mathcal{A}$. For $i \in \{1,\ldots,m\}$, we then have

$$\begin{aligned}
&\{\theta \mid \theta|_{O_1..O_i} \in [\![\mathcal{G}_i]\!]\} = \\
&\qquad = \{\theta \mid \mathtt{last}(\mathcal{G}_i(\theta|_{O_1..O_i})) \in \mathcal{F}_{\mathcal{G}_i}\} = \\
&\qquad = \{\theta \mid \mathtt{last}(\mathcal{A}(\theta))|_{\mathcal{G}_i} \in \mathcal{F}_{\mathcal{G}_i}\} = \\
&\qquad = \Theta_{\mathcal{A}}(\pi \in \mathtt{paths}(\mathcal{A}) \mid \mathtt{last}(\pi)|_{\mathcal{G}_i} \in \mathcal{F}_{\mathcal{G}_i}) = \\
&\qquad = \Theta_{\mathcal{A}}(\Pi_{\mathcal{G}_i})\ , \qquad (3)
\end{aligned}$$

where the first equality follows from Definition 17, the second follows from Lemma 4, the third follows from Lemma 5, and the fourth is just a notational substitution. Similarly, for $i \in \{2,\ldots,m\}$,

$$\{\theta \mid \theta|_{O_1..O_{i-1}} \in [\![\mathcal{A}_i]\!]\} = \Theta_{\mathcal{A}}(\Pi_{\mathcal{A}_i})\ . \qquad (4)$$

Thus,

$$\{\theta \mid \theta|_{O_1..O_i} \in [\![\mathcal{G}_i]\!],\ \theta|_{O_1..O_{i-1}} \in [\![\mathcal{A}_i]\!]\} = \Theta_{\mathcal{A}}(\Pi_{\mathcal{G}_i} \cap \Pi_{\mathcal{A}_i})\ . \qquad (5)$$

Using (3), we can rewrite (1) to get

$$\beta(\Theta_{\mathcal{A}}(\Pi_{\mathcal{G}_1})) \bowtie_1 p_1, \qquad (6)$$

Similarly, using (4) and (5) in the denominator and numerator, respectively, we can rewrite (2) to get

$$\frac{\beta(\Theta_{\mathcal{A}}(\Pi_{\mathcal{G}_i} \cap \Pi_{\mathcal{A}_i}))}{\beta(\Theta_{\mathcal{A}}(\Pi_{\mathcal{G}_i}))} \bowtie_i p_i \qquad (7)$$

for $i \in \{2,\ldots,m\}$.

In order to construct a similar inequality for ϕ_0, first note that because $\beta \notin [\![\phi_0]\!]$, case 1 of Definition 20 tells us that $\beta(\mathcal{G}_{\phi_0}) \bowtie_{\phi_0} p_{\phi_0}$ does not hold. This means that the complement $\beta([\![\mathcal{G}_{\phi_0}]\!]) \bowtie^c_{\phi_0} p_{\phi_0}$ holds. From the definition of the complement of a PAC, together with case 1 of Definition 20, this can be stated equivalently as $\beta([\![\mathcal{G}_{\phi_0^c}]\!]) \bowtie_{\phi_0^c} p_{\phi_0^c}$.

Once again, using Definition 17 together with Lemma 4 and Lemma 5,

$$[\![\mathcal{G}_{\phi_0^c}]\!] = \Theta_{\mathcal{A}}(\Pi_{\mathcal{G}_{\phi_0^c}})\ ,$$

which implies

$$\beta(\Theta_{\mathcal{A}}(\Pi_{\mathcal{G}_{\phi_0^c}})) \bowtie_{\phi_0^c} p_{\phi_0^c}\ . \qquad (8)$$

We now want to show that Algorithm 1 does not return $\mathtt{true}$, or in other words, that the solution space for the system of linear inequalities generated by the algorithm is non-empty. This is the case if and only if there exists an assignment $V : \mathtt{paths}(\mathcal{A}) \to \mathbb{R}$ satisfying all generated inequalities. We prove the existence of such an assignment V by choosing $V(\pi) = \beta(\Theta_{\mathcal{A}}(\pi))$ for each $\pi \in \mathtt{paths}(\mathcal{A})$.

We begin by proving that the equality generated on line 3 is satisfied, i.e.,

$$\sum_{\pi \in \mathtt{paths}(\mathcal{A})} V(\pi) = 1 \,.$$

Because $\mathtt{paths}(\mathcal{A})$ constitutes a partition of the entire trace space $\mathtt{tr}(O)$, and furthermore $\beta(\mathtt{tr}(O)) = 1$, countable additivity of β gives

$$\begin{aligned}
\sum_{\pi \in \mathtt{paths}(\mathcal{A})} V(\pi) &= \sum_{\pi \in \mathtt{paths}(\mathcal{A})} \beta(\Theta_{\mathcal{A}}(\pi)) \\
&= \beta\left(\bigcup_{\pi \in \mathtt{paths}} \Theta_{\mathcal{A}}(\pi) \right) \\
&= \beta(\Theta_{\mathcal{A}}(\mathtt{paths}(\mathcal{A}))) \\
&= \beta(\mathtt{tr}(O)) \\
&= 1 \,.
\end{aligned}$$

Next, we prove that the inequalities from line 8 are satisfied. That is, inequalities of the form

$$\frac{\sum_{\pi \in \Pi_{\mathcal{G}_i} \cap \Pi_{\mathcal{A}_i}} \pi}{\sum_{\pi \in \Pi_{\mathcal{A}_i}} \pi} \bowtie_i p_i \,,$$

for $i \in \{1, \ldots, m\}$, as well as for the complemented specification ϕ_0^c. Note that the paths π used in this generated inequality are purely syntactical, with each path simply being a variable. This inequality is satisfied by V if

$$\frac{\sum_{\pi \in \Pi_{\mathcal{G}_i} \cap \Pi_{\mathcal{A}_i}} V(\pi)}{\sum_{\pi \in \Pi_{\mathcal{A}_i}} V(\pi)} \bowtie_i p_i \,. \tag{9}$$

According to Proposition 4, any set $\Theta_{\mathcal{A}}(\pi)$ is measurable whenever $\pi \in \mathtt{paths}(\mathcal{A})$. Thus, using our choice $V(\pi) = \beta(\Theta_{\mathcal{A}}(\pi))$ together with countable additivity of β, we can prove that (9) is satisfied be rewriting

$$\begin{aligned}
\frac{\sum_{\pi \in \Pi_{\mathcal{G}_i} \cap \Pi_{\mathcal{A}_i}} V(\pi)}{\sum_{\pi \in \Pi_{\mathcal{A}_i}} V(\pi)} &= \frac{\sum_{\pi \in \Pi_{\mathcal{G}_i} \cap \Pi_{\mathcal{A}_i}} \beta(\Theta_{\mathcal{A}}(\pi))}{\sum_{\pi \in \Pi_{\mathcal{A}_i}} \beta(\Theta_{\mathcal{A}}(\pi)} \\
&= \frac{\beta(\Theta_{\mathcal{A}}(\Pi_{\mathcal{G}_i} \cap \Pi_{\mathcal{A}_i}))}{\beta(\Theta_{\mathcal{A}}(\Pi_{\mathcal{G}_i}))} \,,
\end{aligned}$$

whereby (7) implies that (9) is satisfied for the cases $i \in \{2, \ldots, m\}$. Moreover, since $\mathtt{in}(\phi_1) = \emptyset$, it must be the case that $\mathcal{A}_1 = \top$. Thus, $\Pi_{\mathcal{A}_1} = \mathtt{paths}(\mathcal{A})$ and $\Pi_{\mathcal{G}_1} \cap \Pi_{\mathcal{A}_1} = \Pi_{\mathcal{G}_1}$, implying that (9) is also satisfied for the case $i = 1$, since

$$\begin{aligned}
\frac{\sum_{\pi \in \Pi_{\mathcal{G}_1} \cap \Pi_{\mathcal{A}_1}} V(\pi)}{\sum_{\pi \in \Pi_{\mathcal{A}_1}} V(\pi)} &= \frac{\sum_{\pi \in \Pi_{\mathcal{G}_1}} V(\pi)}{\sum_{\pi \in \mathtt{paths}(\mathcal{A})} V(\pi)} \\
&= \frac{\sum_{\pi \in \Pi_{\mathcal{G}_1}} \beta(\Theta_{\mathcal{A}}(\pi))}{\sum_{\pi \in \mathtt{paths}(\mathcal{A})} \beta(\Theta_{\mathcal{A}}(\pi))} \\
&= \frac{\beta(\Theta_{\mathcal{A}}(\Pi_{\mathcal{G}_1})}{1} \\
&\bowtie_1 p_1 \,,
\end{aligned}$$

where the last inequality follows from (6).

Lastly, when it comes to ϕ_0^c, the single generated inequality

$$\frac{\sum_{\pi \in \Pi_{\mathcal{G}_{\phi_0^c}} \cap \Pi_{\mathcal{A}_{\phi_0^c}}} V(\pi)}{\sum_{\pi \in \Pi_{\mathcal{A}_{\phi_0^c}}} V(\pi)} \bowtie_{\phi_0^c} p_{\phi_0^c}$$

is satisfied since

$$\begin{aligned}
\frac{\sum_{\pi \in \Pi_{\mathcal{G}_{\phi_0^c}} \cap \Pi_{\mathcal{A}_{\phi_0^c}}} V(\pi)}{\sum_{\pi \in \Pi_{\mathcal{A}_{\phi_0^c}}} V(\pi)} &= \frac{\sum_{\pi \in \Pi_{\mathcal{G}_{\phi_0^c}}} V(\pi)}{\sum_{\pi \in \mathtt{paths}(\mathcal{A})} V(\pi)} \\
&= \frac{\beta(\Theta_{\mathcal{A}}(\Pi_{\mathcal{G}_{\phi_0^c}}))}{1} \\
&\bowtie_{\phi_0^c} p_{\phi_0^c} \,,
\end{aligned}$$

where the last inequality is the same as (8) and the first equality comes from the fact that $\mathcal{A}_{\phi_0^c} = \top$, with similar reasoning as in the case $i = 1$.

Thus, there exists an assignment V satisfying all inequalities generated by Algorithm 1, implying that $\mathtt{Refines}(\phi_1 \parallel \ldots \parallel \phi_m, \phi_0)$ does not return $\mathtt{true}$. This concludes the proof. □

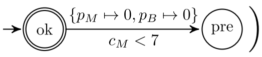
For illustrational purposes, Algorithm 1 is manually applied to the case from Section 2 in the next section. Implementing the algorithm itself would be straight-forward, with the two main tasks being path exploration and solving the system of linear inequalities. Path exploration must take into account the locations of the automata along with the set of possible clock valuations at each step of the path. Usually, these sets of possible clock valuations are partitioned into regions [31] or zones [32, 37], which can be represented by difference bound matrices (DBMs) that express constraints between pairs of clocks [37]. Using DBMs, the successors of a given zone can be efficiently computed, resulting in a way to explore the possible paths. The second task, solving the system of linear inequalities, can be implemented using standard linear programming approaches, for example using the simplex method as mentioned earlier.

The main difficulty when developing a useful tool for checking refinement comes from the fact that the total number of possible paths, and their lengths, can be very large, depending on the automata used in the specifications. Therefore, practical implementations call for efficient search algorithms and data structures for managing the paths and the corresponding systems of inequalities. For instance, if a subset of a system of inequalities lacks solution, then so does the full system. This suggests the possibility to avoid the need to find all possible paths and construct all possible inequalities, and that heuristics can be used to search for paths in a meaningful order.

5 Illustrative case study

Recall the two-component system from Section 2 consisting of a main and backup power source. The purpose of this section is to solve the refinement verification problem for the specifications presented there, using the algorithm from Section 4.

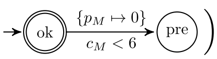
Once again, the natural language top-level specification is: "The system shall output power continuously during the first 7 hours with over 45% probability". To represent this specification, we can define the PAC

$$\phi_0 = \mathcal{P}_{>0.45}\Big(\top,$$ 

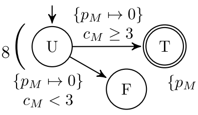
$$\Big).$$

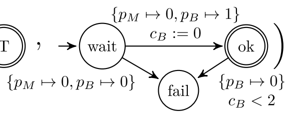
Here, the non-assumption is used together with a guarantee automaton over the considered variables p_M and p_B, denoting main power and backup power, respectively. Each variable is Boolean, where 1 corresponds to power output and 0 corresponds to no power output. The guarantee accepts all traces for which the location *ok* is never left. Looking at the only outgoing transition, this captures the traces such that there exists no time-point before 7 hours with neither main nor backup power. The probability bound put on the guarantee is > 0.45.

Likewise, the natural language specification for the main power source is stated as: "Main power failure shall occur before 6 hours with at most 30% probability", and for the backup power source as: "Assuming main power failure occurs after at least 3 hours, then with at least 80% probability, the backup shall output power continuously for at least 2 hours starting at this time". These natural language specifications can be represented by the two PACs

$$\phi_M = \mathcal{P}_{\geq 0.7}\Big(\top,$$ 

$$\Big),$$

$$\phi_B = \mathcal{P}_{\geq 0.8}\Big($$ 

$$,$$ 
wait
$\{p_M \mapsto 0, p_B \mapsto 1\}$
$c_B := 0$
ok
$\{p_M \mapsto 0, p_B \mapsto 0\}$
fail
$\{p_B \mapsto 0\}$
$c_B < 2$
$$\Big),$$

respectively. Because an assumption is present in the natural language backup specification, the PAC ϕ_B must include a corresponding assumption automaton. Here, the assumption location U denotes *undecided*, T denotes *true* and F denotes *false*. The assumption automaton accepts traces in which main power failure occurs at some time after 3 hours. Meanwhile, the guarantee waits for this occurrence, after which failure to turn on the backup results in entering the *fail* location; otherwise the *ok* location is entered. Now, in order for the guarantee to accept the trace, backup power must be held for at least 2 hours. After that, the accepting location *ok* can never be left.

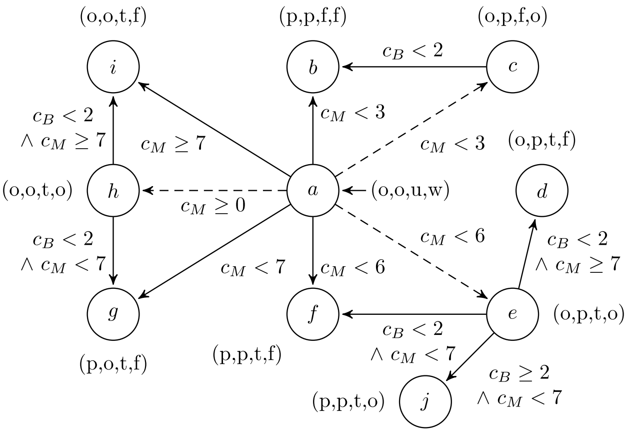


Fig. 11 The composition $\mathcal{A} = \mathcal{A}_{\phi_0} \| \mathcal{G}_{\phi_0} \| \mathcal{A}_{\phi_M} \| \mathcal{G}_{\phi_M} \| \mathcal{A}_{\phi_B} \| \mathcal{G}_{\phi_B}$.

We now verify that the cPAC $\phi_M \| \phi_B$ of component specifications refines the top-level PAC ϕ_0. Following Algorithm 1 on the call `Refines`$(\phi_M \| \phi_B, \phi_0)$, on line 2, we first construct the composition $\mathcal{A} = \mathcal{A}_{\phi_0} \| \mathcal{G}_{\phi_0} \| \mathcal{A}_{\phi_M} \| \mathcal{G}_{\phi_M} \| \mathcal{A}_{\phi_B} \| \mathcal{G}_{\phi_B}$. The resulting automaton is shown in Fig. 11, where only the reachable part is included, since unreachable locations do not contribute to possible paths. Such a reachable part can be computed using standard approaches, see more details regarding so-called regions and zones at the end of Section 4. Note that $\mathcal{A}$ is terminating and, as a result, contains only finitely many possible paths, each of which is itself finite. Next to each location, there is a tuple giving the initials of the corresponding individual automaton locations, e.g., (p,p,f,f) refers to locations *pre*, *pre*, *F*, and *fail* of $\mathcal{G}_{\phi_0}$, $\mathcal{G}_{\phi_M}$, $\mathcal{A}_{\phi_B}$, and $\mathcal{G}_{\phi_B}$, respectively. Dashed arrows denote transitions on the valuation $\{p_M \mapsto 0, p_B \mapsto 1\}$, in which the backup correctly responds to main power failure. Solid lines originating from location a denote transitions on the valuation $\{p_M \mapsto 0, p_B \mapsto 0\}$ in which none of the power sources output power, and solid lines originating from any other location denote transitions on valuations in which the backup does not output power, i.e., both $\{p_M \mapsto 0, p_B \mapsto 0\}$ and $\{p_M \mapsto 1, p_B \mapsto 0\}$. We use the convention that the clock constraints of transitions sharing the same source location and valuation are disjoint, so that e.g. $c_M < 6$ is shorthand for $c_M < 6 \wedge \neg(c_M < 3)$ as a result of the transition from a to b. Lastly, accepting locations of $\mathcal{A}$ are not explicitly marked

Fig. 12 System of linear inequalities generated by Algorithm 1 for the inputs $\phi_M \| \phi_B$ and ϕ_0 from Section 5.

$$\begin{cases} a + ab + ac + acb + ae + aed + aef + aej + af + ag + ah + ahg + ahi + ai = 1 \\ a + ag + ah + ahg + ahi + ai \geq 0.7 \\ \dfrac{ae + aej + ah}{ae + aed + aef + aej + ag + ah + ahg + ahi + ai} \geq 0.8 \\ a + ac + ae + aed + ah + ahi \leq 0.45\ . \end{cases}$$

since these are irrelevant. Rather, we use the accepting locations of each individual automaton, which can be identified through the corresponding position in the location tuples.

The set $\mathtt{paths}(\mathcal{A})$ of all possible paths of $\mathcal{A}$ is $\{a, ab, ac, acb, ae, aed, aef, af, ag, ah, ahg, ahi, ai\}$. Using this, line 3 of Algorithm 1 adds the first inequality of the system of linear inequalities shown in Fig. 12. In accordance with lines 4-7, for $\phi_i = \phi_M, \phi_B, \phi_0^c$, we construct the sets $\Pi_{\mathcal{A}_{\phi_i}}$ and $\Pi_{\mathcal{G}_{\phi_i} \wedge \mathcal{A}_{\phi_i}}$. The resulting sets are $\Pi_{\mathcal{A}_{\phi_M}} = \Pi_{\mathcal{A}_{\phi_0^c}} = \mathtt{paths}(\mathcal{A}) = \{a, ab, ac, acb, ae, aed, aef, af, ag, ah, ahg, ahi, ai\}$, $\Pi_{\mathcal{A}_{\phi_B}} = \{ae, aed, aef, aej, ag, ah, ahg, ahi, ai\}$, $\Pi_{\mathcal{G}_{\phi_0^c} \wedge \mathcal{A}_{\phi_0^c}} = \{a, ac, ae, aed, ah, ahi\}$, $\Pi_{\mathcal{G}_{\phi_M} \wedge \mathcal{A}_{\phi_M}} = \{a, ag, ah, ahg, ahi, ai\}$, and $\Pi_{\mathcal{G}_{\phi_B} \wedge \mathcal{A}_{\phi_B}} = \{ae, aej, ah\}$. Line 8 then adds the last three inequalities of Fig. 12. Running a linear optimization solver, e.g. [38], on this instance shows that the solution space is empty. As a result, line 10 returns `true`. Thus, we have verified that the composition of ϕ_M and ϕ_B refines ϕ_0. Or, in other words, combining any main power source and any backup power source implementing its corresponding specification will surely implement the top-level specification ϕ_0.

6 Related work

A related field is the area of probabilistic or stochastic model checking. However, in contrast to the present paper, which treats refinement of specifications, the goal of model checking is to verify that a given model implements a given specification, see, e.g., [39, 40].

The literature contains various proposed specification theories for stochastic systems, supporting for instance constraint Markov chains [14], abstract probabilistic automata [16], interactive Markov chains [17], and a variety of probabilistic transition systems [18–20, 22]. In the contract context, Nuzzo et al. [21] present a specification theory for probabilistic assume-guarantee contracts. While these previous theories are based on discrete time, the present paper gives explicit support for continuous time. Also in the continuous setting, simulation and bisimulation have been studied for continuous-time Markov chains (CTMCs) [41]. However, this theory assumes that systems follow a particular stochastic process. Similarly, the rest of the papers above assume a particular formalism or system structure, in contrast to the purely trace-based approach of the present paper. The contract theory of [15] is also trace-based, although in discrete time.

Both automata and temporal logic can be used for specifying properties of systems. To specify stochastic systems in continuous time, continuous stochastic logic (CSL) is commonly used [10]. The extension CSL$^{\mathrm{TA}}$ allows specifying properties through single-clock automata and has been used for model checking CTMCs [42]. However, to the best of our knowledge, none of these theories provide a framework for analyzing refinement between specifications.

A specification theory allowing compositional reasoning has been developed for timed I/O automata [43]. In a discrete-time setting, temporal operators defined by finite automata are included in a temporal logic presented by [44] as well as in an extension to computation tree logic called ECTL [33]. However, these frameworks give no explicit support for probabilities.

The systems of linear inequalities generated using Algorithm 1 are essentially instances of generalized probabilistic satisfiability (GenPSAT), which is an NP-complete problem [45, 46].

7 Conclusions

In industrial applications, especially for safety-critical systems, specifications are often of stochastic nature, for example giving a bound on the probability that a system failure will occur before a given time. A decomposition of such a specification requires techniques beyond traditional theorem proving.

As presented in Section 3, the first contribution of the paper is a theoretical framework that allows the representation of, and reasoning about, stochastic and continuous-time behaviors of systems, as well as specifications for such behaviors. The main goal has been to provide a framework that can handle reasoning about refinement between specifications in the form of assume-guarantee contracts. This is needed to support compositional verification, which in turn plays a key role in specifying and verifying large-scale complex systems. A goal has also been to approach the problem from a general perspective, leading to our choice of representing behaviors of components as probability measures on sets of traces. The second contribution, presented in Section 4, is an algorithm for the verification of stochastic specification refinement by reducing the problem to checking emptiness of the solution space for a system of linear inequalities. Future work includes investigating more efficient versions of the algorithm, implementations, and experimental evaluation using larger and more realistic case studies motivated by industry.

Appendix A: Lemmas

This Appendix includes necessary lemmas for proving Theorem 1.

Lemma 1
Consider compatible specifications Σ_1 and Σ_2, and let O_1 denote $\mathtt{out}(\Sigma_1)$ and O_2 denote $\mathtt{out}(\Sigma_2)$. Then, for any behaviors $\beta_1 \in \Sigma_1$ and $\beta_2 \in \Sigma_2$, and any trace set $\Theta_1 \in \sigma_{\beta_1}$, it holds that $\beta_1(\Theta_1) = \beta_1 \| \beta_2(\{\theta \in \mathtt{tr}(O_1 \cup O_2) \mid \theta|_{O_1} \in \Theta_1\})$.

Proof
Pick arbitrary behaviors $\beta_1 \in \Sigma_1$ and $\beta_2 \in \Sigma_2$. Since Σ_1 and Σ_2 are compatible, also β_1 and β_2 are compatible. We will prove that $\beta_1(\Theta_1) = (\beta_1 \| \beta_2)(\{\theta \in \mathtt{tr}(O_1 \cup O_2) \mid \theta|_{O_1} \in \Theta_1\})$, or, equivalently, $\beta_1(\Theta_1) = \beta_1 \| \beta_2(\Theta_1 \times \Omega_2)$, where $\Omega_2 = \mathtt{tr}(O_2)$. This equality follows from Definition 7 as follows:

$$\begin{aligned}\beta_1 \| \beta_2(\Theta_1 \times \Omega_2) &= \int_{\Theta_1} \beta_2(\theta_1)(\Omega_2)\beta_1(d\theta_1)\\ &= \int_{\Theta_1} 1\beta_1(d\theta_1)\\ &= \beta_1(\Theta_1)\ . \qquad (10)\end{aligned}$$

□

Lemma 2
Consider a cPAC or PAC ϕ_1 and a PAC ϕ_2 such that $\mathtt{in}(\phi_1) = \emptyset$ and $\mathtt{out}(\phi_1) = \mathtt{in}(\phi_2)$, and denote $\mathtt{out}(\phi_1) = O_1$, $\mathtt{out}(\phi_2) = O_2$, and $O_1 \cup O_2 = O$. Then, for each $\beta \in [\![\phi_1 \| \phi_2]\!]$, it holds that

$$\frac{\beta(\{\theta \in \mathtt{tr}(O) \mid \theta \in [\![\mathcal{G}_{\phi_2}]\!], \theta|_{O_1} \in [\![\mathcal{A}_{\phi_2}]\!]\})}{\beta(\{\theta \in \mathtt{tr}(O) \mid \theta|_{O_1} \in [\![\mathcal{A}_{\phi_2}]\!]\})} \bowtie_{\phi_2} p_{\phi_2}.$$

Proof
Pick an arbitrary $\beta \in [\![\phi_1 \| \phi_2]\!]$. From Definition 21, it holds that $\beta \in [\![\phi_1 \| \phi_2]\!] = [\![\phi_1]\!] \| [\![\phi_2]\!]$. Then, from Definition 11, it holds that there are behaviors $\beta_1 \in [\![\phi_1]\!]$ and $\beta_2 \in [\![\phi_2]\!]$ such that $\beta = \beta_1 \| \beta_2$. For any such β_2, it holds that $\beta_2 \in [\![\mathcal{P}_{\bowtie_{\phi_2} p_{\phi_2}}(\mathcal{A}_{\phi_2}, \mathcal{G}_{\phi_2})]\!]$.

Let $\Theta_{G_2}(\theta_1) = \{\theta_2 \in \mathtt{tr}(O) \mid \theta_1 \| \theta_2 \in [\![\mathcal{G}_{\phi_2}]\!]\}$. Then, from Definition 20, and since $\beta_2 \in [\![\mathcal{P}_{\bowtie_{\phi_2} p_{\phi_2}}(\mathcal{A}_{\phi_2}, \mathcal{G}_{\phi_2})]\!]$ and $\{\theta_1\} \times \Theta_{G_2}(\theta_1) \subseteq [\![\mathcal{G}_{\phi_2}]\!]$, it follows that for any $\theta_1 \in [\![\mathcal{A}_{\phi_2}]\!]$, it holds that $\beta_2(\theta_1)(\Theta_{G_2}(\theta_1)) \bowtie_{\phi_2} p_{\phi_2}$.

Let $\Theta_{A_2} = \{\theta_1 \in \mathtt{tr}(I) \mid \exists \theta_2 \in \mathtt{tr}(O). \theta_1 \| \theta_2 \in [\![\mathcal{A}_{\phi_2}]\!]\}$. Then, from Lemma 3, we have

$$\begin{aligned}&\beta(\{\theta \in \mathtt{tr}(O) \mid \theta \in [\![\mathcal{G}_{\phi_2}]\!], \theta|_{O_1} \in [\![\mathcal{A}_{\phi_2}]\!]\}) =\\ &\qquad \int_{\Theta_{A_2}} \beta_2(\theta_1)(\Theta_{G_2}(\theta_1))d\beta_1(\theta_1) \qquad (11)\end{aligned}$$

From above, we have

$$\begin{aligned}&\int_{\Theta_{A_2}} \beta_2(\theta_1)(\Theta_{G_2}(\theta_1))d\beta_1(\theta_1) \bowtie_{\phi_2}\\ &\qquad \bowtie_{\phi_2} \int_{\Theta_{A_2}} p_{\phi_2} d\beta_1(\theta_1) =\\ &\qquad = p_{\phi_2} \int_{\Theta_{A_2}} d\beta_1(\theta_1) = p_{\phi_2}\beta_1(\Theta_{A_2})\end{aligned}$$

Next, note that $\{\theta \in \mathtt{tr}(O_1 \cup O_2) \mid \theta|_{O_1} \in [\![\mathcal{A}_{\phi_2}]\!]\} = \Theta_{A_2} \times \Omega_2$, where $\Omega_2 = \mathtt{tr}(O_2)$. Thus, $\beta(\{\theta \in \mathtt{tr}(O_1 \cup O_2) \mid \theta|_{O_1} \in [\![\mathcal{A}_{\phi_2}]\!]\}) = \beta(\Theta_{A_2} \times \Omega_2)$. Note that $\beta(\Theta_{A_2} \times \Omega_2) = \beta_1 \| \beta_2(\Theta_{A_2} \times \Omega_2) = \beta_1(\Theta_{A_2})$ due to (10).

The lemma then follows from

$$\begin{aligned}&\frac{\beta(\{\theta \in \mathtt{tr}(O) \mid \theta \in [\![\mathcal{G}_{\phi_2}]\!],\ \theta|_{O_1} \in [\![\mathcal{A}_{\phi_2}]\!]\})}{\beta(\{\theta \in \mathtt{tr}(O) \mid \theta|_{O_1} \in [\![\mathcal{A}_{\phi_2}]\!]\})} \bowtie_{\phi_2}\\ &\qquad \bowtie_{\phi_2} \frac{p_{\phi_2}\beta_1(\Theta_{A_2})}{\beta_1(\Theta_{A_2})} = p_{\phi_2}\ .\end{aligned}$$

□

Lemma 3
Given any set $\Theta \in \mathcal{B}(\mathtt{out}(\beta_1) \cup \mathtt{out}(\beta_2))$, the extension of $\beta_1 \| \beta_2$ is

$$\beta(\Theta) = \int_{\Theta_1} \beta_2(\theta_1)(\Theta_2(\theta_1))d\beta_1(\theta_1)\,, \qquad (12)$$

where $\Theta_1 = \{\theta_1 \in \mathtt{tr}(I) \mid \exists \theta_2 \in \mathtt{tr}(O).\ \theta_1 \| \theta_2 \in \Theta\}$ and $\Theta_2(\theta_1) = \{\theta_2 \in \mathtt{tr}(O) \mid \theta_1 \| \theta_2 \in \Theta\}$.

Proof
We already know from Section 3.3 that $\beta_1 \| \beta_2$ defined on all sets $\Theta_1 \times \Theta_2$ has a unique extension to $\mathcal{B}(\mathtt{out}(\beta_1) \cup \mathtt{out}(\beta_2))$. To conclude that β in (12) is this extension, we just need to prove that β is a measure defined on $\mathcal{B}(\mathtt{out}(\beta_1) \cup \mathtt{out}(\beta_2))$ and that for any $\Theta_1 \times \Theta_2 \in \mathcal{B}(\mathtt{out}(\beta_1) \cup \mathtt{out}(\beta_2))$, it holds that $\beta(\Theta_1 \times \Theta_2) = \beta_1 \| \beta_2(\Theta_1 \times \Theta_2)$.

To prove that β is a measure, we need to show that: (a) $\beta(\emptyset) = 0$, and (b) $\beta(\cup_j^\infty \Theta^j) = \sum_j^\infty \beta(\Theta^j)$ for any sequence $\{\Theta^j\}_{j=1}^\infty$ of pairwise disjoint sets in $\mathcal{B}(\mathtt{out}(\beta_1) \cup \mathtt{out}(\beta_2))$.

First, (a) is proven by $\beta(\emptyset) = \int_\emptyset \beta_2(\theta_1)(\emptyset)d\beta_1(\theta_1) = 0$. Next, in order to prove (b), note that for any set $\Theta = \cup_{j=1}^\infty \Theta^j$, it holds that $\Theta_1 = \cup_{j=1}^\infty \Theta_1^j$ and $\Theta_2(\theta_1) = \cup_{j=1}^\infty \Theta_2^j(\theta_1)$. Then, due to σ-additivity of the measure $\beta_2(\theta_1)(\cdot)$, we have

$$\begin{aligned}\beta\Big(\bigcup_{j=1}^{\infty} \Theta^j\Big) &= \int_{\cup_{j=1}^\infty \Theta_1^j} \beta_2(\theta_1)(\cup_{j=1}^\infty \Theta_2^j(\theta_1))d\beta_1(\theta_1) =\\ &= \int_{\cup_{j=1}^\infty \Theta_1^j} \sum_{i=1}^{\infty} \beta_2(\theta_1)(\Theta_2^i(\theta_1))d\beta_1(\theta_1) =\end{aligned}$$

$$= \sum_{i=1}^{\infty} \int_{\cup_{j=1}^{\infty} \Theta_1^j} \beta_2(\theta_1)(\Theta_2^i(\theta_1)) d\beta_1(\theta_1) \ .$$

Next, note that for any $\theta_1 \notin \Theta_1^i$, it holds $\Theta_2^i(\theta_1) = \emptyset$ and consequently $\beta_2(\theta_1)(\Theta_2^i(\theta_1)) = 0$. This means

$$\int_{\cup_{j=1}^{\infty} \Theta_1^j} \beta_2(\theta_1)(\Theta_2^i(\theta_1)) d\beta_1(\theta_1) =$$

$$= \int_{\Theta_1^i} \beta_2(\theta_1)(\Theta_2^i(\theta_1)) d\beta_1(\theta_1) = \beta(\Theta^i) \ .$$

Combining these facts directly yields the property (b). Thus, we have proven that β is a measure defined on $\mathcal{B}(\mathtt{out}(\beta_1) \cup \mathtt{out}(\beta_2))$.

Next, we will prove that for any $\Theta_1 \times \Theta_2 \in \mathcal{B}(\mathtt{out}(\beta_1) \cup \mathtt{out}(\beta_2))$, it holds that $\beta(\Theta_1 \times \Theta_2) = \beta_1 \| \beta_2(\Theta_1 \times \Theta_2)$. Since $\Theta = \Theta_1 \times \Theta_2$, it holds $\Theta_2(\theta_1) = \Theta_2$. This implies

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

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

Lemma 4

Let $\mathcal{A}_1 \in \mathbb{A}_{E_1}$ and $\mathcal{A}_2 \in \mathbb{A}_{E_2}$ be terminating trace automata and let $\mathcal{A}$ denote the composition $\mathcal{A}_1 \| \mathcal{A}_2$. Then, for each trace $\theta \in \mathtt{tr}(E_1 \cup E_2)$ and index $i \in \{1, 2\}$,

$$\mathtt{last}(\mathcal{A}_i(\theta|_{E_i})) = \mathtt{last}(\mathcal{A}(\theta))|_{\mathcal{A}_i} \ .$$

Proof

Let $\theta \in \mathtt{tr}(E_1 \cup E_2)$ be an arbitrary trace. It follows from Proposition 1 that $\mathcal{A}$ is both deterministic and terminating, resulting in a unique finite path $\pi = \mathcal{A}(\theta)$ with a unique last location $\mathtt{last}(\pi)$. The idea is to show that, given an arbitrary trace θ, whenever $\mathcal{A}_1$ or $\mathcal{A}_2$ uses an individual transition, then $\mathcal{A}$ mimics that action in terms of the next location and clock resets, and vice versa. This ensures that throughout the run of $\mathcal{A}$, the current location is always the same as for the individual runs of $\mathcal{A}_1$ and $\mathcal{A}_2$, implying that also the last location is the same.

We start by showing that each individual transition is mimicked by a joint one with corresponding effect. Without loss of generality, due to symmetry, consider an individual transition $\langle l_1, \nu_1, R_1, \delta_1, l_1' \rangle$ of $\mathcal{A}_1$ and a clock valuation ν_C satisfying δ_1. There are now two possibilities: either (a) there exists a transition $\langle l_2, \nu_2, R_2, \delta_2, l_2' \rangle$ of $\mathcal{A}_2$ with $\nu_2|_{E_1 \cap E_2} = \nu_1|_{E_1 \cap E_2}$ such that ν_C satisfies δ_2, or (b) no such transition exists. Due to Definition 18, if (a) is true, then $\mathcal{A}$ contains a transition $\langle (l_1, l_2), \nu, R, \delta_1 \wedge \delta_2, (l_1', l_2') \rangle$ of the form (i) with $\nu|_{E_1} = \nu_1$ and $R \cap E_1 = R_1$ such that ν_C satisfies $\delta_1 \wedge \delta_2$. On the other hand, if (b) is true, then due to the negated disjunction of (ii) covering all remaining clock valuations, $\mathcal{A}$ contains a transition $\langle (l_1, l_2), \nu, R_1, \delta_1 \wedge \delta_2, (l_1', l_2) \rangle$ with $\nu|_{E_1} = \nu_1$ such that ν_C satisfies $\delta_1 \wedge \delta_2$. Thus, each transition of $\mathcal{A}_1$ is contained within a joint transition of $\mathcal{A}$. Due to symmetry, each transition of $\mathcal{A}_2$ is contained within a joint transition of $\mathcal{A}$.

We now show that each joint transition is mimicked by one or two individual ones with corresponding effect. It is easy to see that each joint transition of $\mathcal{A}$ has been added as the result of either (i) or (ii) of Definition 18. In the case (i), the transition contains a constituent transition of each of $\mathcal{A}_1$ and $\mathcal{A}_2$, preserving their respective effects. That is, the destination location of the joint transition consists of the destination location of both individual transitions, and the clocks to reset in the joint transition is the union of clocks to reset in the individual transitions. In the case (ii), $\mathcal{A}$ contains a constituent transition of either $\mathcal{A}_1$ or $\mathcal{A}_2$, preserving its effects while not affecting the other. Since, in this case, the other automaton would not have transitioned at all, the joint transition reflects precisely the joint transition of $\mathcal{A}_1$ and $\mathcal{A}_2$, and in so doing, invariantly preserves the joint path of $\mathcal{A}_1$ and $\mathcal{A}_2$. □

Lemma 5

Let $\mathcal{A}_1 \in \mathbb{A}_{E_1}$ and $\mathcal{A}_2 \in \mathbb{A}_{E_2}$ be terminating trace automata and let $\mathcal{A} = \mathcal{A}_1 \| \mathcal{A}_2$. Then, for each index $i \in \{1, 2\}$ and each of location $l_i \in L_{\mathcal{A}_i}$,

$$\{\theta \in \mathtt{tr}(E_1 \cup E_2) \mid \mathtt{last}(\mathcal{A}(\theta))|_{\mathcal{A}_i} = l_i\} =$$

$$= \Theta_{\mathcal{A}}(\{\pi \in \mathtt{paths}(\mathcal{A}) \mid \mathtt{last}(\pi)|_{\mathcal{A}_i} = l_i\}) \ .$$

Proof

To prove the equality, we will show that the left-hand side of the equation is a subset of the right-hand side, and vice versa. Due to Proposition 1, $\mathcal{A}$ is deterministic and terminating. As a result, each trace $\theta \in \mathtt{tr}(E_1 \cup E_2)$ corresponds to a unique finite path $\pi = \mathcal{A}(\theta)$. Trivially, for $i \in \{1, 2\}$, $\mathtt{last}(\pi)|_{\mathcal{A}_i} = \mathtt{last}(\mathcal{A}(\theta))|_{\mathcal{A}_i}$, implying that the left-hand side is a subset of the right-hand side. For the opposite direction, each path π has a corresponding set $\Theta_{\mathcal{A}}(\pi)$ of traces for which $\mathcal{A}(\theta) = \pi$ holds for each $\theta \in \Theta_{\mathcal{A}}(\pi)$. Thus, once again, $\mathtt{last}(\mathcal{A}(\theta))|_{\mathcal{A}_i} = \mathtt{last}(\pi)|_{\mathcal{A}_i}$, and the right-hand side is a subset of the left-hand side. □

Lemma 6

Given a set of variables $E \subseteq X$ and a terminating trace automaton $\mathcal{A} \in \mathbb{A}_E$, the collection $\{\Theta_{\mathcal{A}}(\pi) \mid \pi \in \mathtt{paths}(\mathcal{A})\}$ consisting of sets of traces corresponding to each path is a partition of $\mathtt{tr}(E)$.

Proof

We need to prove that: (1) the set of traces $\cup_{\pi \in \mathtt{paths}(\mathcal{A})} \Theta_{\mathcal{A}}(\pi)$ equals $\mathtt{tr}(E)$ and (2) the sets $\Theta_{\mathcal{A}}(\pi)$ s.t. $\pi \in \mathtt{paths}(\mathcal{A})$ are disjoint. To prove (1), we will show that $\mathtt{tr}(E)$ is a subset of $\cup_{\pi \in \mathtt{paths}(\mathcal{A})} \Theta_{\mathcal{A}}(\pi)$ and vice versa. Consider an arbitrary trace $\theta \in \mathtt{tr}(E)$. Using the fact that $\mathcal{A}$ is deterministic, there exists a unique sequence $\mathcal{A}(\theta)$. According to Definition 15, it follows that $\mathcal{A}(\theta)$ is a path of $\mathcal{A}$, i.e. $\mathcal{A}(\theta) \in \mathtt{paths}(\mathcal{A})$, and $\theta \in \Theta_{\mathcal{A}}(\mathcal{A}(\theta))$. Thus, also $\theta \in \cup_{\pi \in \mathtt{paths}(\mathcal{A})} \Theta_{\mathcal{A}}(\pi)$. Because θ was chosen arbitrarily from $\mathtt{tr}(E)$, it follows that $\mathtt{tr}(E) \subseteq \cup_{\pi \in \mathtt{paths}(\mathcal{A})} \Theta_{\mathcal{A}}(\pi)$. Because also each element $\theta_\pi \in \Theta_{\mathcal{A}}(\pi)$, for any path $\pi \in \mathtt{paths}(\mathcal{A})$, is a trace, we have $\cup_{\pi \in \mathtt{paths}(\mathcal{A})} \Theta_{\mathcal{A}}(\pi) \subseteq \mathtt{tr}(E)$, implying (1). Furthermore, since $\mathcal{A}$ is deterministic, the location sequence $\mathcal{A}(\theta)$, for any $\theta \in \mathtt{tr}(E)$, is a unique path. That is, each trace corresponds to no more than one path, implying (2). □

Appendix B: Triangle sets

In this appendix, we introduce triangle sets as a way to represent the trace sets considered throughout the paper. The purpose is to attain a more explicit representation of such trace sets and thereby be able to prove that they are measurable in the required context. More precisely, the appendix includes proof that each trace set used in the paper as an argument to any behavior β over any variable set E is measurable with regard to its corresponding Borel σ-algebra $\mathcal{B}(E)$.

Definition 22 (Triangle)
Let $E \subseteq X$ be a set of variables, $\bar{v}_0\bar{v}_1 \dots \bar{v}_m$be a sequence of valuations for E in vector form, and $I_1 I_2 \dots I_m$ be a sequence of functions such that, for each $i \in \{1,2,\dots,m\}$, $I_i : \mathbb{R}^{i-1} \rightarrow 2^{\mathbb{R}}$ maps each sequence of time-points $h_i = t_1 t_2 \dots t_{i-1}$ to an interval

$$I_i(h_i) = \Big[a_i(h_i), b_i(h_i)\Big)$$

where $a_i(h_i)$ and $b_i(h_i)$ are functions of the form

$$a_i(h_i) = \max_{c \in N_{a,i}} \left(d_c + \sum_{j=1}^{i-1} k_c t_j \right)$$

and

$$b_i(h_i) = \min_{c \in N_{b,i}} \left(d_c + \sum_{j=1}^{i-1} k_c t_j \right),$$

where $N_{a,i}$ and $N_{b,i}$ are any countable index sets, each constant d_c is a real number and each constant k_c is either 0 or -1. Then the sequence $\bar{v}_0 I_1 \bar{v}_1 \dots I_m \bar{v}_m$ is a *triangle on E*.

Intuitively, the intervals I_i represent the set of all possible time delays between valuations $\bar{v}_{i-1}$ and $\bar{v}_i$. Note, however, that any two adjacent valuations $\bar{v}_i$, $\bar{v}_{i+1}$ are allowed to be identical. This is what accommodates less than M valuation changes, say M', by letting the remaining $M - M'$ valuations be duplicates. This intuition is captured in the following definition, which establishes a connection between triangles and the trace sets that they represent.

Definition 23 (Interpretation of Triangle)
Given a set $E \subseteq X$ of variables and a triangle $\alpha = \bar{v}_0 I_1 \bar{v}_1 \dots I_m \bar{v}_m$ on E, the *interpretation of* α is the largest trace set, denoted $D(\bar{v}_0 I_1 \bar{v}_1 \dots I_m \bar{v}_m) \subseteq \mathtt{tr}(E)$, such that each trace $\theta \in D(\bar{v}_0 I_1 \bar{v}_1 \dots I_m \bar{v}_m)$ satisfies the following:

(i) $\theta(0) = \bar{v}_0$.
(ii) Let $t_0 = 0$. Inductively, for $i = 1 \dots m$, there exists a time-point $t_i \in I_i(t_1, t_2, \dots, t_{i-1})$ such that $\theta(t_i) = \bar{v}_i$ and $\forall t \in [t_{i-1}, t_i)\ .\ \theta(t) = \bar{v}_{i-1}$.

Any trace set T that can be expressed as $D(\bar{v}_0 I_1 \bar{v}_1 \dots I_m \bar{v}_m)$ for some triangle $\bar{v}_0 I_1 \bar{v}_1 \dots I_m \bar{v}_m$ on E is referred to as a *triangle set on E*, or, if the set E is unimportant, simply as a *triangle set*.

Proposition 2
Given a set $E \subseteq X$ of variables, each triangle set on E is an element of the Borel σ-algebra $\mathcal{B}(E)$.

Proof
Volumes between linear functions of the form $d_c + \sum_{j=1}^{i-1} k_c t_j$ are easily seen to be Borel sets, because they can be expressed using a countable number of unions and intersections of open sets. Furthermore, allowing boundaries defined as the maximum or minimum of a countable number of such functions does not change Borel-measurability. Lastly, each valuation is a point value and therefore constitutes a Borel set. □

We now need to prove that trace sets $\Theta(\pi)$ of paths π can be expressed as a countable disjoint union of triangle sets. To make the proof simpler, we first introduce change-enabled trace automata. Their characteristic property is that, whenever they encounter a valuation change in the trace that they are reading, they must transition to another location. For the following definition, given a trace automaton $\mathcal{M} = \langle V, L, l_0, C, \rightarrow, F \rangle$, let $l \xrightarrow{v} l'$ denote the logical statement that a transition from l to l' under v is possible. That is, there exists a transition $(l, v, r, \delta, l') \in \rightarrow$, clock valuation $v_C \in \mathtt{val}(C)$, clock set $r \subseteq C$, and clock constraint $\delta \in \Delta(C)$ such that v_C satisfies δ. Furthermore, let $l \rightharpoonup v$ denote the statement that, for some clock valuation, the automaton will stay in l when reading v. That is, there exists a clock valuation $v_C \in \mathtt{val}(C)$ for which there exists no transition $(l, v, r, \delta, r, l') \in \rightarrow$, clock set $r \subseteq C$, and clock constraint $\delta \in \Delta(C)$ such that v_C satisfies δ.

Definition 24 (Change-Enabled Automaton)
A trace automaton $\mathcal{M} = \langle V, L, l_0, C, \rightarrow, F \rangle$ is *change-enabled*

if, for any locations $l, l' \in L$ and valuations $\nu, \nu' \in V$ such that $\nu \neq \nu'$, the following holds:

(i) If $l_0 \rightharpoonup \nu$ then $l_0 \not\rightharpoonup \nu'$.
(ii) If $l \xrightarrow{\nu} l'$ then $l' \not\rightharpoonup \nu'$.

It is easy to realize that, if traces are restricted to at most M valuation changes, then any trace automaton $\mathcal{M}$ can be converted to a change-enabled trace automaton $\mathcal{M}'$ such that $[\![\mathcal{M}]\!] = [\![\mathcal{M}']\!]$. This is done by simply expanding each location l into one location l_ν per possible new valuation, and duplicating all incoming and outgoing transitions. Of course, transitions $(l, \nu, \emptyset, true, l_\nu)$ are also added. As a result, we may assume without loss of generality that any given automaton is change-enabled.

Proposition 3
Given a terminating trace automaton $\mathcal{M}$ and a path π of $\mathcal{M}$, the set $\Theta(\pi)$ can be expressed as a countable union of triangle sets.

Proof
Let $\mathcal{M} = \langle V, L, l_0, C, \rightarrow, F \rangle$ be an arbitrary terminating trace automaton and $\pi = l_0 l_1 \ldots l_m$ be a path of $\mathcal{M}$. Assume without loss of generality that $\mathcal{M}$ is change-enabled. The proof strategy is to construct one triangle $\Delta_{\boldsymbol{\tau}}^{\bar{v}_0} = \bar{v}_0 I_1 \bar{v}_1 \ldots I_m \bar{v}_m$ per possible initial valuation $\bar{v}_0$ and sequence $\boldsymbol{\tau} = \tau_1 \tau_2 \ldots \tau_m$ of transitions through π such that $D(\bar{v}_0 I_1 \bar{v}_1 \ldots I_m \bar{v}_m)$ is the set of all traces giving rise to $\tau_1 \tau_2 \ldots \tau_m$. We then show that the union of these triangle sets $D(\Delta_{\boldsymbol{\tau}}^{\bar{v}_0})$ over all possible initial valuations $\bar{v}_0$ and transition sequences $\boldsymbol{\tau}$ results in the set $\Theta(\pi)$.

Formally, let $\boldsymbol{\tau} = \tau_1 \tau_2 \ldots \tau_m$ be an arbitrary sequence of transitions traversing π. Consider now, for any given initial valuation $\bar{v}_0$, the valuation-interval sequence $\Delta_{\boldsymbol{\tau}}^{\bar{v}_0} = \bar{v}_0 I_1 \bar{v}_1 \ldots I_m \bar{v}_m$ constructed as follows. First, $\bar{v}_1 \bar{v}_2 \ldots \bar{v}_m$ is the unique sequence of valuations corresponding to $\tau_1 \tau_2 \ldots \tau_m$, respectively. Here, uniqueness follows from $\mathcal{M}$ being change-enabled. Note, however, that there may exist $\bar{v}_i$, $\bar{v}_{i+1}$ with $\bar{v}_i = \bar{v}_{i+1}$ because of transitions taken due to the satisfaction of some clock constraint rather than a valuation change. We now construct each interval I_j for $j = 1, \ldots, m$ to represent the set of possible time delays t_j between transitions τ_{j-1} and τ_j. Let δ_j denote the clock constraint of τ_j and let C_j be the set of all clocks appearing in δ_j. The set of all clock valuations satisfying δ_j and possibly resulting in τ_j being used can be expressed as an interval $[a_{cj}, b_{cj})$ per clock $c \in C_j$. Note that a_{cj} and b_{cj} represent bounds for the absolute values for the clock, rather than the time delay between τ_{j-1} and τ_j. The concrete absolute clock value $\nu_C(c)$ of c depends on the time delay since the last reset of c, which can be expressed as the sum of individual delays between the transitions since that reset. That is, if e_{cj} is the number of transitions since the last clock reset of c up to (but not including) τ_j, then $\nu_C(c) = \sum_{u=1}^{e_{cj}} t_{j-u}$ where each t_{j-u} is the delay between transitions τ_{j-u-1} and τ_{j-u}. Thus, to convert all absolute value intervals $[a_{cj}, b_{cj})$ for clocks $c \in C_j$ to a single interval I_j for the delay between τ_{j-1} and τ_j, we must make sure that each $c \in C_j$ satisfies $\nu_C(c) \in \left[a_{cj},\, b_{cj}\right)$. This is equivalent to the delay t_j between τ_{j-1} and τ_j, satisfying $t_j \in \left[a_{cj} - \sum_{u=1}^{e_{cj}} t_{j-u},\, b_{cj} - \sum_{u=1}^{e_{cj}} t_{j-u}\right)$ for each $c \in C_j$. Therefore, the interval I_j is the intersection of such intervals over all $c \in C_j$, which can be expressed as

$$I_j = \left[\max_{c \in C_j}\left(a_{cj} - \sum_{u=1}^{e_{cj}} t_{j-u}\right),\ \min_{c \in C_j}\left(b_{cj} - \sum_{u=1}^{e_{cj}} t_{j-u}\right)\right).$$

Due to the above reasoning, given an initial valuation $\bar{v}_0$, the set $D(\bar{v}_0 I_1 \bar{v}_1 \ldots I_m \bar{v}_m)$ consists of precisely the traces giving rise to $\tau_1 \tau_2 \ldots \tau_m$. Thus, the union

$$\bigcup_{\boldsymbol{\tau}} D\left(\bar{v}_0 I_1 \bar{v}_1 \ldots I_m \bar{v}_m\right) .$$

over all transition sequences $\boldsymbol{\tau}$ traversing π is the set of all traces $\theta \in \Theta(\pi)$ having initial valuation $\theta(0) = \bar{v}_0$. Considering each possible initial valuation $\bar{v}_0 \in \mathtt{tr}(E)$ gives

$$\Theta(\pi) = \bigcup_{\bar{v}_0, \boldsymbol{\tau}} D\left(\bar{v}_0 I_1 \bar{v}_1 \ldots I_m \bar{v}_m\right), \qquad (13)$$

as desired. Since there is only a countable number of transitions, the set of all possible initial valuations can be partitioned into a countable number of equivalence classes, such that each pair of valuations $\bar{v}_0$, $\bar{v}_0'$ taken from the same class satisfies

$$D\left(\bar{v}_0 I_1 \bar{v}_1 \ldots I_m \bar{v}_m\right) = D\left(\bar{v}_0' I_1 \bar{v}_1 \ldots I_m \bar{v}_m\right) .$$

Thus, (13) can be expressed using a countable union. □

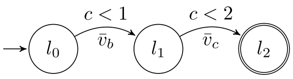


Fig. 13 A trace automaton with two successive clock constraints on the same clock and no reset in-between.

Example 10
Consider the automaton in Fig. 13 and the path $\pi = l_0 l_1 l_2$. The subset of traces corresponding to π, assuming $\bar{v}_a$ is the initial valuation, can be expressed as the triangle set $D(\bar{v}_a, [0,1), \bar{v}_b, [0, 2 - t_1), \bar{v}_c)$. To represent the entire set $\Theta(\pi)$, we use the union $\bigcup_{\bar{v}_0} D(\bar{v}_0, [0,1), \bar{v}_b, [0, 2 - t_1), \bar{v}_c)$ over all possible starting valuations $\bar{v}_0$. Note that I_1 is a constant

interval $[0, 1)$ that does not depend on t_2, while I_2 is a diagonal interval $I_2 = [0, 2 - t_1)$ that depends on t_1.

Proposition 4
Given a set $E \subseteq X$ of variables, a trace automaton $\mathcal{M}$ over E and a path π of $\mathcal{M}$, the set $\Theta(\pi)$ is an element of the Borel σ-algebra $\mathcal{B}(E)$.

Proof
Follows from Propositions 2 and 3, because $\Theta(\pi)$ can be expressed as a countable union of triangle sets, each being an element of the Borel σ-algebra $\mathcal{B}(E)$. □

Proposition 5
Given a set $E \subseteq X$ of variables and a trace automaton $\mathcal{M}$ over E, the set $[\![\mathcal{M}]\!]$ is an element of the Borel σ-algebra $\mathcal{B}(E)$.

Proof
Follows from Proposition 4 because each set of accepting traces $[\![\mathcal{M}]\!]$ can be expressed as a countable union $\bigcup_{i \in \mathbb{N}} \Theta(\pi_i)$, where each π_i is a path of $\mathcal{M}$. □

Acknowledgements Supported by Vinnova FFI, Sweden, through the SafeDim project.

Funding Open access funding provided by Royal Institute of Technology.

References

1. de Roever, W.-P.: The need for compositional proof systems: a survey. In: International Symposium on Compositionality, pp. 1–22. Springer, Berlin (1997)
2. ISO 26262: "Road vehicles - Functional safety", Geneva, Switzerland (2018)
3. ISO 21434: "Road vehicles – Cybersecurity engineering", Geneva, Switzerland (2021)
4. 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, Berlin (2012)
5. 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, Berlin (2008)
6. 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, Berlin (2020)
7. Slind, K., Norrish, M.: A brief overview of HOL4. In: International Conference on Theorem Proving in Higher Order Logics, pp. 28–32. Springer, Berlin (2008)
8. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.: Verifying continuous time Markov chains. In: International Conference on Computer Aided Verification, pp. 269–276. Springer, Berlin (1996)
9. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.: Model-checking continuous-time Markov chains. ACM Trans. Comput. Log. **1**(1), 162–170 (2000)
10. Grunske, L.: Specification patterns for probabilistic quality properties. In: 2008 ACM/IEEE 30th International Conference on Software Engineering, pp. 31–40. IEEE (2008)
11. 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, Berlin (2008)
12. Meyer, B.: Applying 'design by contract'. Computer **25**(10), 40–51 (1992)
13. Westman, J., Nyberg, M.: Conditions of contracts for separating responsibilities in heterogeneous systems. Form. Methods Syst. Des. (2017). https://doi.org/10.1007/s10703-017-0294-7
14. 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)
15. Delahaye, B., Caillaud, B., Legay, A.: Probabilistic contracts: a compositional reasoning methodology for the design of systems with stochastic and/or non-deterministic aspects. Form. Methods Syst. Des. **38**(1), 1–32 (2011)
16. Delahaye, B., Katoen, J.-P., Larsen, K.G., Legay, A., Pedersen, M.L., Sher, F., Wąsowski, A.: Abstract probabilistic automata. In: International Workshop on Verification, Model Checking, and Abstract Interpretation, pp. 324–339. Springer, Berlin (2011)
17. Gössler, G., Xu, D.N., Girault, A.: Probabilistic contracts for component-based design. Form. Methods Syst. Des. **41**(2), 211–231 (2012)
18. Jonsson, B., Larsen, K.G.: Specification and refinement of probabilistic processes. In: Proceedings 1991 Sixth Annual IEEE Symposium on Logic in Computer Science, pp. 266–267. IEEE Comput. Soc., Los Alamitos (1991)
19. Jonsson, B., Yi, W.: Testing preorders for probabilistic processes can be characterized by simulations. Theor. Comput. Sci. **282**(1), 33–51 (2002)
20. Lanotte, R., Maggiolo-Schettini, A., Troina, A.: Parametric probabilistic transition systems for system design and analysis. Form. Asp. Comput. **19**(1), 93–109 (2007)
21. Nuzzo, P., Li, J., Sangiovanni-Vincentelli, A.L., Xi, Y., Li, D.: Stochastic assume-guarantee contracts for cyber-physical system design. ACM Trans. Embed. Comput. Syst. **18**(1), 1–26 (2019)
22. Segala, R., Lynch, N.: Probabilistic simulations for probabilistic processes. In: International Conference on Concurrency Theory, pp. 481–496. Springer, Berlin (1994)
23. Hampus, A., Nyberg, M.: Formally verifying decompositions of stochastic specifications. In: Formal Methods for Industrial Critical Systems: 27th International Conference, FMICS 2022, Proceedings, Warsaw, Poland, September 14–15, 2022, pp. 193–210. Springer, Berlin (2022)
24. 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, Berlin (2020)

25. Alur, R., Feder, T., Henzinger, T.A.: The benefits of relaxing punctuality. J. ACM **43**(1), 116–146 (1996)
26. Heymann, M., Lin, F., Meyer, G., Resmerita, S.: Analysis of Zeno behaviors in a class of hybrid systems. IEEE Trans. Autom. Control **50**(3), 376–383 (2005)
27. Ben-Gal, I.: Bayesian networks. Encyclopedia of statistics in quality and **reliability** (2008)
28. Koller, D., Friedman, N.: Probabilistic Graphical Models: Principles and Techniques. MIT Press, Cambridge (2009)
29. Resnick, S.: A Probability Path. Birkhäuser, Boston (2019)
30. Alur, R., Dill, D.: Automata for modeling real-time systems. In: International Colloquium on Automata, Languages, and Programming, pp. 322–335. Springer, Berlin (1990)
31. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2), 183–235 (1994)
32. Alur, R.: Timed automata. In: Computer Aided Verification: 11th International Conference, CAV'99, Proceedings 11, Trento, Italy, July 6–10, 1999, pp. 8–22. Springer, Berlin (1999)
33. Clarke, E.M., Grumberg, O., Kurshan, R.P.: A synthesis of two approaches for verifying finite state concurrent systems. In: International Symposium on Logical Foundations of Computer Science, pp. 81–90. Springer, Berlin (1989)
34. Kern, C., Greenstreet, M.R.: Formal verification in hardware design: a survey. ACM Trans. Des. Autom. Electron. Syst. **4**(2), 123–193 (1999)
35. Dantzig, G.B.: Origins of the simplex method. In: A History of Scientific Computing, pp. 141–151 (1990)
36. Nash, J.C.: The (Dantzig) simplex method for linear programming. Comput. Sci. Eng. **2**(1), 29–31 (2000)
37. Bengtsson, J., Yi, W.: Timed automata: semantics, algorithms and tools. In: Advanced Course on Petri Nets, pp. 87–124. Springer, Berlin (2003)
38. Linear Optimization. https://online-optimizer.appspot.com. (Accessed on 05/27/2022)
39. Mereacre, A., Katoen, J.-P., Han, T., Chen, T.: Model checking of continuous-time Markov chains against timed automata specifications. Log. Methods Comput. Sci. **7** (2011)
40. Paolieri, M., Horváth, A., Vicario, E.: Probabilistic model checking of regenerative concurrent systems. IEEE Trans. Softw. Eng. **42**(2), 153–169 (2015)
41. Baier, C., Katoen, J.-P., Hermanns, H., Wolf, V.: Comparative branching-time semantics for Markov chains. Inf. Comput. **200**(2), 149–214 (2005)
42. Donatelli, S., Haddad, S., Sproston, J.: Model checking timed and stochastic properties with CSL^{TA}. IEEE Trans. Softw. Eng. **35**(2), 224–240 (2008)
43. David, A., Larsen, K.G., Legay, A., Nyman, U., Wasowski, A.: Timed I/o automata: a complete specification theory for real-time systems. In: Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control, pp. 91–100 (2010)
44. Vardi, M.Y., Wolper, P.: Reasoning about infinite computations. Inf. Comput. **115**(1), 1–37 (1994)
45. Caleiro, C., Casal, F., Mordido, A.: Generalized probabilistic satisfiability. Electron. Notes Theor. Comput. Sci. **332**, 39–56 (2017)
46. Hansen, P., Jaumard, B.: Probabilistic satisfiability. In: Handbook of Defeasible Reasoning and Uncertainty Management Systems: Algorithms for Uncertainty and Defeasible Reasoning, pp. 321–367 (2000)