Formally Verifying Decompositions of Stochastic Specifications (With Proofs)*

Anton Hampus[1][0000−0002−3939−3919](✉) and Mattias Nyberg[1,2](✉)

[1] KTH Royal Institute of Technology, Stockholm, Sweden
ahampus@kth.se
[2] Scania, Södertälje, Sweden
mattias.nyberg@scania.com

This version of the paper contains the proof of Thm. 1 and lemmas needed for the proof, together with some minor editorial changes.

Abstract. According to the principles of compositional verification, verifying that lower-level components satisfy their specification will ensure 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 for safety-critical systems, specifications are often of stochastic nature, for example giving a bound on the probability that 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 behaviors. 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 decomposition 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

1 Introduction

The principle of *compositional verification* [32] has been proposed as a solution to verify large complex systems built up by 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 will then ensure that the whole system implements its top-level

* Supported by Vinnova FFI through the SafeDim project.

specification. The key difficulty is (2), which can also be expressed as to ensure 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 [19] and ISO 21434 [18]. In these standards, specifications in the form of safety and cyber-security requirements are decomposed into lower-level specifications. The standards also require these decompositions to be correct and complete.

In the present paper, we consider general cyber-physical systems, and have therefore chosen a representation based on continuous time. Based upon logic and various extensions to include time, a number of frameworks are available to express specifications and to verify refinement between specifications, e.g. [10, 26, 29, 34]. A limitation with 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 to occur 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 an algorithm for the analysis of whether a composition of stochastic specifications refines another stochastic specification.

The approach taken in the paper is that *behaviors* of components and systems are characterized by traces and probability measures over sets of traces. Rather than being expressed explicitly, 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 [4, 5, 17] but views specifications generally as a probabilistic extension to *assume-guarantee contracts* [7, 25, 36]. 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, given that the environment satisfies the assumption, the probability that the guarantee is satisfied shall be less than p.

The literature contains some other proposed frameworks for defining stochastic specifications and verifying properties such as refinement, e.g. [8, 13, 14, 16, 20, 21, 23, 28, 33]. However, in contrast to all of these previous works, the present paper uses continuous time and considers component behaviors purely in terms of traces—no particular modeling formalism for generating the traces is assumed.

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

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 this, 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

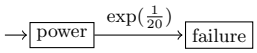

Fig. 1: Possible main power component

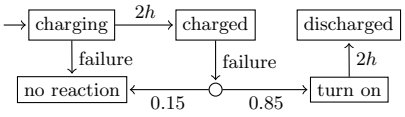

Fig. 2: Possible backup component

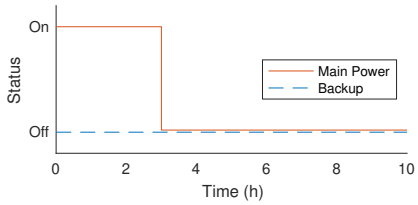

Fig. 3: Failed backup activation

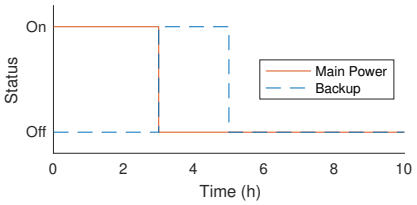

Fig. 4: Successful backup activation

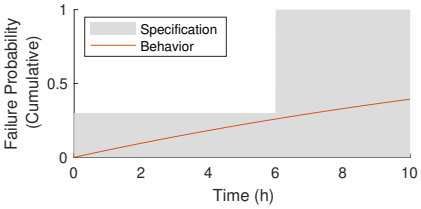

Fig. 5: Main power specification

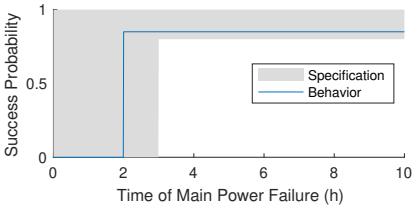

Fig. 6: Backup specification

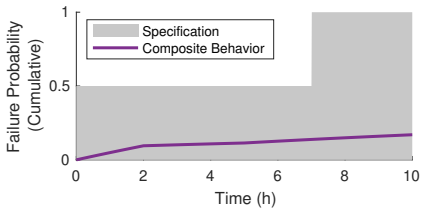

Fig. 7: Top-level specification

of such a system is depicted 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 main power failure occurs, 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 turned on, the backup will output power also for 2 hours, until entering a discharged state.

Assume the top-level specification to be: "the system shall output power continuously during the first 7 hours with over 50% probability". Instead of merely verifying that the system composed of the components 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 sure to implement the top-level specification. As our attempt for doing so, let the main power source specification be: "main power failure shall occur before 6 hours with at most 30% probability". Meanwhile, the backup specification will be 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 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 can 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 corresponds to specifying two sets of behaviors; thus, we must translate the natural language specifications to sets of "allowed" probability distributions.

Fig. 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 $\exp(\frac{1}{20})$.

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 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 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 this: does the composition of the two component specifications refine the top-level specification? The purpose of the rest of the paper is to formalise these notions of *traces*, *behaviors*, and *specifications*, and to provide an algorithm for verifying refinement.

3 A Theory for Specifying Stochastic Behavior

3.1 Traces and Behaviors

A *behavior* is meant to represent the possible executions, or *traces*, of a component, as well as how likely they are. In short, we represent a trace as an assignment of values to variables at each point in time, and a behavior as a distribution over traces. We will also extend behaviors to incorporate input as well as output, calling them input/output behaviors.

We consider a universal set of variables $X = \{x_1, x_2, \ldots, x_n\}, n \geq 1$, each $x_i \in X$ ranging over a non-empty countable set V_{x_i} of values. Given a non-empty set of variables $E \subseteq X$, a *valuation for* E is a function $\nu : E \to \bigcup_{x_i \in X} V_{x_i}$ 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 $\mathtt{val}(E)$.

Definition 1 (Trace). *Given a non-empty set of variables $E \subseteq X$, a* trace over *E is a right-continuous function $\theta : \mathbb{R}_{\geq 0} \to \mathtt{val}(E)$ defined on the timeline.* □

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 $\emptyset$. Furthermore, for any trace $\theta \in \mathtt{tr}(E)$ and set 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 2 (Behavior). *Given a non-empty set of variables $E \subseteq X$, a* behavior over *E is a probability measure defined on a sigma algebra on the set $\mathtt{tr}(E)$.* □

Let $\mathtt{beh}(E)$ denote the set of all possible behaviors over a non-empty $E \subseteq X$.

We will now extend behaviors into *input/output behaviors*, which intuitively have control over output variables while being dependent on input variables.

Definition 3 (Input/Output Behavior). *Given two disjoint sets of variables $I \subseteq X$ and $O \subseteq X$, where O is non-empty, an* input/output behavior from *I* to *O is a function $\beta : \mathtt{tr}(I) \to \mathtt{beh}(O)$ such that for any pair of traces $\theta_1, \theta_2 \in \mathtt{tr}(I)$, the behaviors $\beta(\theta_1)$ and $\beta(\theta_2)$ share the same sigma algebra denoted σ_β.* □

Given a possibly empty $I \subseteq X$ and a non-empty $O \subseteq X$, let $\texttt{beh}(I, O)$ denote the set of all possible input/output behaviors from I to O. Furthermore, for an input/output behavior β from I to O, let $\texttt{in}(\beta)$ and $\texttt{out}(\beta)$ denote the sets I and O of input and output variables, respectively. From now on, "input/output" will often be abbreviated as I/O.

Example 1. Consider $I = \emptyset$ and $O = \{x\}$. Then an I/O behavior from I to O is a function $\beta : \emptyset \to \texttt{beh}(\{x\})$. Thus, the I/O behavior from I to O reduces to a behavior over $\{x\}$, i.e. $\beta \in \texttt{beh}(\{x\})$.

Composition of Behaviors When composing two behaviors β_1 and β_2, for the sake of simplicity, we restrict ourselves to the case where β_1 has no input, and its output is exactly the input of β_2, i.e. $\texttt{in}(\beta_1) = \emptyset$ and $\texttt{out}(\beta_1) = \texttt{in}(\beta_2)$. The implication of this is that composing β_1 with β_2 results in yet another behavior without input. The composition of β_1 and β_2, denoted $\beta_1 \| \beta_2$, is the I/O behavior from $\texttt{in}(\beta_1) = \emptyset$ to $\texttt{out}(\beta_1) \cup \texttt{out}(\beta_2)$ formed as follows.

We assume that $\beta_2(\cdot)(\Theta_2)$, for any fixed Θ_2, is a measurable function from the measurable space $(\texttt{out}(\beta_1), \sigma_{\beta_1})$ to the measurable space $([0,1], \mathcal{B}([0,1]))$. Then according to [31] (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 of $\Theta_1 \times \Theta_2 \in \sigma_{\beta_1} \times \sigma_{\beta_2}$ and its unique extension a probability measure on the product sigma algebra $\sigma_{\beta_1} \times \sigma_{\beta_2}$. This result is the basis for the following definition.

Definition 4 (Composition of I/O Behaviors). *Let β_1 and β_2 be two I/O behaviors such that $\texttt{in}(\beta_1) = \emptyset$, $\texttt{in}(\beta_2) = \texttt{out}(\beta_1)$, and $\beta_2(\cdot)(\Theta_2)$ is a measurable function from $(\texttt{out}(\beta_1), \sigma_{\beta_1})$ 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 $\texttt{out}(\beta_1) \cup \texttt{out}(\beta_2)$, i.e. a probability measure*

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

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, and defined on $\sigma_{\beta_1} \times \sigma_{\beta_2}$. □

Note that according to this definition, we only obtain a measure on the sigma algebra $\sigma_{\beta_1} \times \sigma_{\beta_2}$. As a consequence, we assume that any subset of $\texttt{tr}(\texttt{out}(\beta_1)) \times \texttt{tr}(\texttt{in}(\beta_2))$ that we want to measure the probability of, and that is not an element of the cartesian product $\sigma_{\beta_1} \times \sigma_{\beta_2}$, can be approximated to arbitrary precision by some countable disjoint union of elements in the cartesian product $\sigma_{\beta_1} \times \sigma_{\beta_2}$. Note further that the output of the composition $\beta_1 \| \beta_2$ simply becomes the union of β_1 and β_2 and it is presumed that $\texttt{in}(\beta_2) = \texttt{out}(\beta_1)$ and $\texttt{in}(\beta_1) = \emptyset$. Clearly, a less restrictive definition can be created, but for the sake of simplicity, these generalizations are left out of scope of the current paper.

3.2 Specifications

In short, we view a specification simply as the set of behaviors that implement it. A specification *refines* another if each behavior implementing it also implements the other. This is captured by the following three definitions.

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

Definition 6 (Implements). *An I/O behavior β from I to O* implements *a specification Σ from I to O if $\beta \in \Sigma$.* □

Definition 7 (Refines). *A specification Σ_1 from I to O* refines *a specification Σ_2 from I to O if $\Sigma_1 \subseteq \Sigma_2$.* □

Given a possibly empty set $I \subseteq X$ and non-empty set $O \subseteq X$, let $\mathtt{spec}(I, O)$ denote the set of all possible specifications from I to O. Given a specification Σ, $\mathtt{in}(\Sigma)$ and $\mathtt{out}(\Sigma)$ are defined in a similar manner as with I/O behaviors.

Note that, according to Def. 4, $\beta_1 \| \beta_2$ 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 will be called *compatible*.

In analogy with the notion of compatible behaviors, we say that two specifications Σ_1 and Σ_2 are *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 $\mathtt{in}(\Sigma_2) = \mathtt{out}(\Sigma_1)$.

Definition 8 (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 $\beta_1 \| \beta_2 \in \Sigma_1 \| \Sigma_2$.

3.3 Trace Automata

The specification language presented in this paper, as well as its semantics and the verification method, are based on *timed automata*, as introduced by Alur and Dill [2, 3]. 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 will often use the notation ν_C for a valuation over clocks, as opposed to ν, which is used for a valuation over variables in X. For $t \in \mathbb{R}_{\geq 0}$, let $\nu_C + t$ denote the clock valuation $\{c \mapsto \nu_C(c) + t \mid c \in C\}$. Given a set $C = \{c_1, \ldots, c_m\}$ of clocks, a *clock constraint δ on C* is defined inductively by the grammar

$$\delta ::= c < k \mid c \geq k \mid \delta \wedge \delta,$$

where c ranges over clocks C and k ranges over constant real numbers $\mathbb{R}$. A clock valuation ν_C for C is said to *satisfy* a clock constraint δ on C if $\delta[c_1 \mapsto \nu_C(c_1), \ldots, c_m \mapsto \nu_C(c_m)]$ evaluates to true. Given a set C of clocks, let $\Delta(C)$ denote the set of all possible clock constraints on C.

Definition 9 (Timed Automaton). *A* timed automaton *is a tuple* $\mathcal{A} = \langle V, L, l_0, C, \rightarrow, F\rangle$ *where* V *is a countable alphabet,* L *is a countable set of locations,* $l_0 \in L$ *is a start location,* C *is a countable 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 $\mathcal{A} = \langle V, L, l_0, C, \rightarrow, F\rangle$, we denote by $V_\mathcal{A}$, $L_\mathcal{A}$, $l_{0_\mathcal{A}}$, $C_\mathcal{A}$, $\rightarrow_\mathcal{A}$, and $F_\mathcal{A}$ the elements V, L, l_0, C, $\rightarrow$, and F, respectively. 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 both clock constraints.

In what follows, only a special class of timed automata, called *trace automata*, will be considered. These are characterized by the fact that their alphabets consist of variable valuations, resulting in the ability to read traces as input. This leads us to use the letter ν to denote an input symbol.

Given a timed automaton $\mathcal{A} = \langle V, L, l_0, C, \rightarrow, F\rangle$, locations $l, l' \in L$, clock valuations ν_C, ν'_C for C, and an alphabet symbol $\nu \in V$, we will denote by $(l, \nu_C) \xrightarrow{\nu}_\mathcal{A} (l', \nu'_C)$ the logical statement that $\rightarrow$ contains a transition $\langle l, \nu, r, \delta, l'\rangle$ with $r = \{c_1, \ldots, c_m\}$ such that ν_C satisfies δ and $\nu'_C = \nu_C[c_1 \mapsto 0, \ldots, c_m \mapsto 0]$.

In order to give a concise and well-defined semantics for trace automata, we require that only a finite number of transitions are possible within 0 time. This fact is captured in the following definition.

Definition 10 (Trace Automaton). *Given a non-empty set* $E \subseteq X$ *of variables, a deterministic timed automaton* $\mathcal{A} = \langle V, L, l_0, C, \rightarrow, F\rangle$ *is a* trace automaton for E *if* $V = \mathtt{val}(E)$ *and, for each* $l \in L$ *and* $\nu_C \in \mathtt{val}(C)$*, there exists no infinite sequence* $(l, \nu_C) \xrightarrow{\nu}_\mathcal{A} (l_1, \nu_{C_1}) \xrightarrow{\nu}_\mathcal{A} (l_2, \nu_{C_2}) \xrightarrow{\nu}_\mathcal{A} \ldots$. □

Note from Def. 10 that the condition about infinite transition sequences applies both to loops, including self loops, as well as to infinite location spaces with an infinite number of transitions. For instance, trace automata never allow self loops $\langle l, \nu, r, \delta, l\rangle$ in which the set r of clocks to reset is empty.

The semantics of a trace automaton $\mathcal{A} = \langle V, L, l_0, C, \rightarrow, F\rangle$ for a non-empty E is defined as follows. Consider a trace $\theta \in \mathtt{tr}(E)$ to be given. A configuration is a tuple $\mu_i = (l_i, \nu_{C_i}, t_i) \in L \times \mathtt{val}(C) \times \mathbb{R}_{\geq 0}$, containing a location, clock configuration and time value. Initially, $\mu_0 = (l_0, \bar{0}, 0)$, where $\bar{0}$ denotes the clock valuation $\{c \mapsto 0 \mid c \in C\}$. Inductively, consider a configuration $\mu_i = (l_i, \nu_{C_i}, t_i)$ and the smallest time increment t^+ causing the automaton to transition. That is, $t^+ = \min\{t \in \mathbb{R}_{\geq 0} \mid \exists \langle l_i, \theta(t_i + t), r, \delta, l'_i\rangle \in \rightarrow . \nu_{C_i} + t \text{ satisfies } \delta\}$. The successor of μ_i becomes $\mu_{i+1} = (l_{i+1}, \nu_{C_{i+1}}, t_{i+1})$ such that $t_{i+1} = t_i + t^+$ and there exists a maximal transition sequence $(l_i, \nu_{C_i} + t^+) \xrightarrow{\theta(t_{i+1})} \ldots \xrightarrow{\theta(t_{i+1})} (l_{i+1}, \nu_{C_{i+1}})$. Note that the sequence $\mu_0, \mu_1, \ldots$ generated in this way is unique since $\mathcal{A}$ is deterministic. Thus, we can define *the execution of* $\mathcal{A}$ *on* θ, denoted $\mathcal{A}(\theta)$, as this sequence $\mu_0, \mu_1, \ldots$. Let furthermore $\mathcal{A}(\theta)|_L$ denote the sequence $l_0, l_1, \ldots$ of locations visited along the execution. In the following definition, let * denote Kleene star, i.e. the operator that, when applied to a set L, generates the set L^* of all strings over elements in L.

Definition 11 (Path). *Given a set $E \subseteq X$ of variables, a trace automaton $\mathcal{A} = \langle V, L, l_0, C, \to, F\rangle$ for E, and a location sequence $\pi \in L^*$, the sequence π is a* path of $\mathcal{A}$ *if there exists a trace $\theta \in \mathtt{tr}(E)$ such that $\pi = \mathcal{A}(\theta)|_L$.* □

Given a trace automaton $\mathcal{A}$ for E, the set of all possible paths of $\mathcal{A}$ is denoted $\mathtt{paths}(\mathcal{A})$. Furthermore, given an infinite path $l_0, l_1, \dots$ of $\mathcal{A}$, the limit $\lim_{i\to\infty} l_i$ exists if and only if there exists an index $a \in \mathbb{N}$ such that for each $b \geq a$, $l_b = l_a$. In that case, we define $\lim_{i\to\infty} l_i = l_a$. For finite paths $l_0, l_1, \dots, l_k$, we use the convention that $\lim_{i\to\infty} l_i = l_k$. If for each trace $\theta \in \mathtt{tr}(E)$ the path $\mathcal{A}(\theta)|_L = l_0, l_1, \dots$ has a limit $\lim_{i\to\infty} l_i$, then $\mathcal{A}$ is said to be *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". Furthermore, although some types of liveness properties are not possible to expresss, such as "at all times, each request shall be followed by an answer", 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 $\pi = l_0, l_1, \dots$ of an automaton $\mathcal{A} \in \mathbb{A}_E$, let $\mathtt{last}(\pi)$ denote the last visited location $\lim_{i\to\infty} l_i$. We also extend $\mathtt{last}(\cdot)$ to executions, so that if ω is an execution, then $\mathtt{last}(\omega) = \mathtt{last}(\omega|_L)$. Furthermore, if π is a path of $\mathcal{A} \in \mathbb{A}_E$, then $\Theta_\mathcal{A}(\pi)$ will denote the set of all traces $\theta \in \mathtt{tr}(E)$ corresponding to π, i.e. the set $\{\theta \in \mathtt{tr}(E) \,|\, \mathcal{A}(\theta)|_L = \pi\}$. As an extension, if Π is a set of paths of $\mathcal{A}$, then $\Theta_\mathcal{A}(\Pi) = \{\Theta_\mathcal{A}(\pi) \mid \pi \in \Pi\}$. 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 execution. This is captured in the next definition.

Definition 12 (Composition of Trace Automata). *Let $\mathcal{A}_1 = \langle V_1, L_1, l_{0_1}, C_1, \to_1 \rangle \in \mathbb{A}_{E_1}$ be a trace automaton for E_1 and $\mathcal{A}_2 = \langle V_2, L_2, l_{0_2}, C_2, \to_2\rangle \in \mathbb{A}_{E_2}$ 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 trace automaton $\mathcal{A}_1 \| \mathcal{A}_2 = \langle \mathtt{val}(E_1 \cup E_2), L_1 \times L_2, (l_{0_1}, l_{0_2}), C_1 \cup C_2, \to_1 \| \to_2\rangle \in \mathbb{A}_{E_1 \cup E_2}$ where $\to_1 \| \to_2$ is constructed as follows:*

1. *For each location pair $(l_1, l_2) \in L_1 \times L_2$, each transition $\langle l_1, \nu_1, r_1, \delta_1, l_1'\rangle \in \to_1$ and each valuation $\nu \in \mathtt{val}(E_1 \cup E_2)$ such that $\nu|_{E_1} = \nu_1$, let T_2^ν be the set of transitions of $\mathcal{A}_2$ coherent with ν, i.e.*

$$T_2^\nu = \{\langle l_2, \nu_2, r_2, \delta_2, l_2'\rangle \in \to_2 \;\mid \nu|_{E_2} = \nu_2\}$$

and let $\overline{T_2^\nu}$ be the subset that is mutually executable with $\langle l_1, \nu_1, r_1, \delta_1, l_1'\rangle$, i.e.

$$\overline{T_2^\nu} = \{\langle l_2, \nu_2, r_2, \delta_2, l_2'\rangle \in T_2^\nu \mid \exists \nu_C \in \mathtt{val}(C_1 \cup C_2) \text{ s.t. } \nu_C \text{ satisfies both } \delta_1 \text{ and } \delta_2\}\,.$$

   - *If $\overline{T_2^\nu} = \emptyset$, then $\to_1 \| \to_2$ contains the transition $\langle (l_1, l_2), \nu, r_1, \delta_1, (l_1', l_2)\rangle$.*

- *Otherwise, for each* $\langle l_2, \nu_2, r_2, \delta_2, l_2' \rangle \in \overline{T_2^\nu}$, $\rightarrow_1 \| \rightarrow_2$ *contains the transitions* $\langle (l_1, l_2), \nu, r_1 \cup r_2, \delta_1 \wedge \delta_2, (l_1', l_2') \rangle$ *and* $\langle (l_1, l_2), \nu, r_1, \delta_1 \wedge \neg\delta_2, (l_1', l_2) \rangle$.

2. *Repeat step 1 but with the indices* 1 *and* 2 *interchanged, i.e. iterating over* $\rightarrow_2$ *instead of* $\rightarrow_1$. □

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

3.4 Probabilistic Automaton Contracts

For specifying I/O behaviors in practice, we will use a contract-based approach. A contract consists of an *assumption* and a *guarantee* together with a probability bound. 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 guarantee are specified using terminating trace automata. 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 $\mathcal{A} \| \top = \top \| \mathcal{A} = \mathcal{A}$.

The choice of using automata for specifying system properties is motivated by their flexibility—while temporal logics offer their own advantages, it may be difficult, or even impossible, to specify some complex systems using them [9]. In general, it is always possible to construct some automaton corresponding to a given temporal logic formula.

Definition 13 (Accepts). *Given a non-empty set* $E \subseteq X$ *of variables, an automaton* $\mathcal{A} \in \mathbb{A}_E$*, and a trace* $\theta \in \mathtt{tr}(E)$, $\mathcal{A}$ accepts θ *if* $\mathtt{last}(\mathcal{A}(\theta)) \in F$. □

We also extend the notion of acceptance to the non-assumption $\top$, so that $\top$ is considered to accept each possible trace $\theta \in \bigcup_{E \subseteq X} \mathtt{tr}(E)$. For an automaton $\mathcal{A} \in \mathbb{A}_E \cup \{\top\}$, let $\mathtt{acc}(\mathcal{A})$ denote the set of all traces that $\mathcal{A}$ accepts.

Definition 14 (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, $\mathtt{in}(\phi)$ and $\mathtt{out}(\phi)$ are defined for PACs ϕ in a similar manner as for I/O behaviors and specifications. For a PAC $\phi = \mathcal{P}_{\bowtie p}(\mathcal{A}, \mathcal{G})$, we will denote its assumption $\mathcal{A}$, guarantee $\mathcal{G}$, probability value p and comparison operator $\bowtie$ by $\mathcal{A}_\phi$, $\mathcal{G}_\phi$, p_ϕ, and $\bowtie_\phi$, respectively.

To understand *trace composition* in the following definition of *PAC interpretation*, 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 $x \in E_2$.

In the next definition of *PAC interpretation*, given a set $O \subseteq X$ of variables, we will make use of a particular σ-algebra σ_O that, for each automaton $\mathcal{A} \in \mathbb{A}_O$ and each path $\pi \in \mathtt{paths}(\mathcal{A})$, contains the set $\Theta_{\mathcal{A}}(\pi)$.

Definition 15 (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 specification, i.e. the set of I/O behaviors from I to O, with the σ-algebra σ_O such that for each $\beta \in [\![\phi]\!]$, it holds:*

1. *in the case $I = \emptyset$, $\beta()(\mathtt{acc}(\mathcal{G})) \bowtie p$,*
2. *in the case $I \neq \emptyset$, for each trace $\theta_I \in \mathtt{tr}(I)$, if $\theta_I \in \mathtt{acc}(\mathcal{A})$, then $\beta(\theta_I)(\{\theta_O \in \mathtt{tr}(O) \mid \theta_I \| \theta_O \in \mathtt{acc}(\mathcal{G})\}) \bowtie p$.* □

Note that the choice of σ_O guarantees the values $\beta()(\mathtt{acc}(\mathcal{G}))$ and $\beta(\theta_I)(\{\theta_O \in \mathtt{tr}(O) \mid \theta_I \,\|\, \theta_O \in \mathtt{acc}(\mathcal{G})\})$ in the above definition to be defined. For a PAC $\phi = \mathcal{P}_{\bowtie p}(\mathcal{A}, \mathcal{G})$, we denote by ϕ^c the PAC $\mathcal{P}_{\bowtie^c p}(\mathcal{A}, \mathcal{G})$ in which the comparison operator has been complemented. For instance, the complement of $<$ is $\geq$. 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, combining PACs using negation, conjunction and disjunction as well as defining an *until* operator and nesting PACs within PACs. Although this possibility is interesting, it lies outside the scope of the present paper. Instead, as will be introduced in the next definition, we will work with *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 16 (Composite Probabilistic Automaton Contract). *Given two PACs ϕ_1 and ϕ_2 with compatible interpretations, the term $\phi_1 \| \phi_2$ is a* composite probabilistic automaton contract (cPAC) *with interpretation $[\![\phi_1 \| \phi_2]\!] = [\![\phi_1]\!] \| [\![\phi_2]\!]$. Inductively, if ϕ_1 is a PAC or cPAC and ϕ_2 is a PAC or cPAC such that $[\![\phi_1]\!]$ and $[\![\phi_2]\!]$ are compatible, then the term $\phi_1 \| \phi_2$ 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

A common technique for formal verification found throughout literature is to formulate specifications using automata and then solve the language inclusion problem using automata theory [9, 22]. This is also the foundation for the method developed in this paper, except here, languages are sets of I/O behaviors instead of strings. The intuition for verifying that a composition $\phi = \phi_1 \| \ldots \| \phi_k$ of component specifications refines a top-level specification $\widehat{\phi}$ is as follows. We want to verify that the set $[\![\phi]\!]$ is a subset of $[\![\widehat{\phi}]\!]$. The strategy is to check emptiness of the specification $[\![\phi]\!] \cap [\![\widehat{\phi^c}]\!]$. To do so, we will compose all automata found in the two specifications and map each joint path to the assumptions and guarantees that it satisfies. From this, a set of linear inequalities can be generated, each

representing a conditional probability of a guarantee given an assumption. If this system of inequalities has no solutions, it means that no valid probability measure can possibly exist in $[\![\phi]\!] \cap [\![\widehat{\phi}^c]\!]$, thereby proving emptiness. The last step can be calculated using e.g. the simplex method [11, 27].

Although this method is a semi-decision procedure, i.e. is not guaranteed to return `true` if refinement holds and `false` otherwise, it is sound in the sense that refinement does indeed hold whenever the algorithm returns `true`.

To verify refinement, the restrictions imposed on the specifications are as follows. While the component specification ϕ can be either a single PAC or a cPAC consisting of multiple PACs $\phi_1, \ldots, \phi_m$, the top-level specificaion $\widehat{\phi}$ must be a single PAC. Moreover, we require that $\texttt{in}(\phi) = \texttt{in}(\widehat{\phi}) = \emptyset$ and $\texttt{out}(\phi) = \texttt{out}(\widehat{\phi}) \neq \emptyset$. Each $\bowtie_{\phi_i}$ must be one of $\leq$ or $\geq$, and $\bowtie_{\widehat{\phi}}$ must be either $<$ or $>$. Lastly, given that ϕ is the PAC or cPAC $\phi = \phi_1 \| \ldots \| \phi_m$ for some $m \geq 1$ and denoting $\mathcal{A} = \mathcal{A}_{\phi_1} \| \mathcal{G}_{\phi_1} \| \ldots \| \mathcal{A}_{\phi_m} \| \mathcal{G}_{\phi_m} \| \mathcal{A}_{\widehat{\phi}} \| \mathcal{G}_{\widehat{\phi}}$, we assume that $\texttt{paths}(\mathcal{A})$ is finite and can be found in finite time. This should be the case for many industrial applications, for instance when the number of transitions is finite.

The algorithm works as follows. Let $\phi = \phi_1 \| \ldots \| \phi_m$, $m \geq 1$, be a PAC or cPAC and $\widehat{\phi}$ be a PAC. The problem is to decide whether ϕ refines $\widehat{\phi}$. We construct the composition $\mathcal{A} = \mathcal{A}_{\phi_1} \| \mathcal{G}_{\phi_1} \| \ldots \| \mathcal{A}_{\phi_m} \| \mathcal{G}_{\phi_m} \| \mathcal{A}_{\widehat{\phi}} \| \mathcal{G}_{\widehat{\phi}}$ and, from that, a system of linear inequalities as follows. For each $\phi_i \in \{\phi_1, \ldots, \phi_m\}$, let $\{\pi_1, \ldots, \pi_q\}$ be the set of paths of $\mathcal{A}$ accepted by the assumption $\mathcal{A}_{\phi_i}$ and let $\{\pi_{j_1}, \ldots, \pi_{j_s}\} \subseteq \{\pi_1, \ldots, \pi_q\}$ be the set of paths of $\mathcal{A}$ accepted by both the guarantee $\mathcal{G}_{\phi_i}$ and assumption $\mathcal{A}_{\phi_i}$. Then we add an inequality

$$\frac{\pi_{j_1} + \cdots + \pi_{j_s}}{\pi_1 + \cdots + \pi_q} \bowtie_{\phi_i} p_{\phi_i} \ .$$

Do the same for $\widehat{\phi}$ except with $\bowtie_{\widehat{\phi}^c}$ substituted for $\bowtie_{\widehat{\phi}}$. To ensure that the probabilities sum to 1, add the equality $1 = \sum_{\pi \in \texttt{paths}(\mathcal{A})} \pi$. Lastly, if the solution space for this system of inequalities is empty, we conclude that ϕ refines $\widehat{\phi}$.

Pseudocode for this procedure is presented in Algorithm 1. Here, the variable `ineqs` stores the set of linear inequalities, which is incrementally updated to contain the inequality generated from each conditional probability.

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

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

Due to the assumption of finitely many paths, the algorithm will terminate in finite time. However, time complexity depends on operations for enumerating these paths. Therefore, practical implementations call for efficient search algorithms and data structures for this. The following theorem states that Algorithm 1 is sound. To make the proof more readable, the lemmas are given separately below.

Theorem 1. *A PAC or cPAC $\phi_1 \parallel \ldots \parallel \phi_m$ refines a PAC $\widehat{\phi}$ if the procedure* $\texttt{Refines}(\phi_1 \parallel \ldots \parallel \phi_m, \widehat{\phi})$ *given by Algorithm 1 returns* `true`. □

Proof. By contraposition. We want to prove that whenever $\phi_1 \parallel \ldots \parallel \phi_m$ does not refine $\widehat{\phi}$, $\texttt{Refines}(\phi_1 \parallel \ldots \parallel \phi_m, \widehat{\phi})$ does not answer `true`. To do this, assume there is a behavior $\beta \in \texttt{beh}(\texttt{out}(\phi_1 \parallel \ldots \parallel \phi_m))$ such that $\beta \in [\![\phi_1 \parallel \ldots \parallel \phi_m]\!]$ and $\beta \notin [\![\widehat{\phi}]\!]$. First, denote for each $i \in \{1, \ldots, m\}$ $\texttt{out}(\phi_i)$ by O_i, $\mathcal{A}_{\phi_i}$ by $\mathcal{A}_i$, $\mathcal{G}_{\phi_i}$ by $\mathcal{G}_i$, $\bowtie_{\phi_i}$ by $\bowtie_i$, and p_{ϕ_i} by p_i. Furthermore, denote the composition $\mathcal{A}_1 \parallel \mathcal{G}_1 \parallel \ldots \parallel \mathcal{A}_m \parallel \mathcal{G}_m \parallel \mathcal{A}_{\widehat{\phi}} \parallel \mathcal{G}_{\widehat{\phi}}$ by $\mathcal{A}$, and the set $\texttt{out}(\phi_1 \parallel \ldots \parallel \phi_m) = \texttt{out}(\widehat{\phi})$ by O.

Since $\texttt{in}(\phi_1) = \emptyset$ and $\texttt{out}(\phi_1) = \texttt{in}(\phi_2)$, Lemma 1 implies that, for each $\beta_1 \in [\![\phi_1]\!]$, there exists a $\beta_{1..2} \in [\![\phi_1 \parallel \phi_2]\!]$ such that $\beta_1(\texttt{acc}(\mathcal{G}_1)) = \beta_{1..2}(\{\theta \mid \theta|_{O_1} \in \texttt{acc}(\mathcal{G}_1)\})$. Note that $\texttt{acc}(\mathcal{G}_1)$ is measurable w.r.t. the σ-algebra $\sigma_{\texttt{out}(\phi_1)}$ of β_1 from Def. 15. Because also each $\beta_1 \in [\![\phi_1]\!]$ satisfies $\beta_1(\texttt{acc}(\mathcal{G}_1)) \bowtie_1 p_1$ as per Def. 15, it follows that each $\beta_{1..2} \in [\![\phi_1 \parallel \phi_2]\!]$ satisfies $\beta_{1..2}\{\theta \mid \theta|_{O_1} \in \texttt{acc}(\mathcal{G}_1)\} \bowtie_1 p_1$. Furthermore, since $\texttt{in}(\phi_2) = \texttt{out}(\phi_1)$, it follows from Lemma 2 that, for each $\beta_{1..2} \in [\![\phi_1 \parallel \phi_2]\!]$,

$$\frac{\beta_{1..2}\{\theta \mid \theta|_{O_1 \cup O_2} \in \texttt{acc}(\mathcal{G}_2) \text{ and } \theta|_{O_1} \in \texttt{acc}(\mathcal{A}_2)\}}{\beta_{1..2}\{\theta \mid \theta|_{O_1} \in \texttt{acc}(\mathcal{A}_2)\}} \bowtie_2 p_2 \ .$$

Because once again $\texttt{in}(\beta_{1..2}) = \emptyset$, we can repeat this procedure, starting from $\phi_1 \parallel \phi_2$, until covering all of $\phi_1 \parallel \ldots \parallel \phi_m$, preserving, for each $i \in \{1, \ldots, m\}$,

$$\frac{\beta\{\theta \mid \theta|_{O_1 \cup \cdots \cup O_i} \in \texttt{acc}(\mathcal{G}_i) \text{ and } \theta|_{O_1 \cup \cdots \cup O_{i-1}} \in \texttt{acc}(\mathcal{A}_i)\}}{\beta\{\theta \mid \theta|_{O_1 \cup \cdots \cup O_{i-1}} \in \texttt{acc}(\mathcal{A}_i)\}} \bowtie_i p_i \ . \tag{1}$$

Note that in the special case of ϕ_1, we could rewrite

$$\beta\{\theta \mid \theta|_{O_1} \in \texttt{acc}(\mathcal{G}_1)\} = \frac{\beta\{\theta \mid \theta|_{O_1} \in \texttt{acc}(\mathcal{G}_1) \text{ and } \theta|_{\emptyset} \in \texttt{acc}(\mathcal{A}_1)\}}{\beta\{\theta \mid \theta|_{\emptyset} \in \texttt{acc}(\mathcal{A}_1)\}} \bowtie_1 p_1$$

to treat it in the same way as $\phi_2, \ldots, \phi_m$. This comes from the fact that $\texttt{in}(\phi_1) = \emptyset$ and, due to Def. 14, $\mathcal{A}_1$ must therefore equal $\top$. Since $\texttt{acc}(\top)$ is a superset of $\texttt{acc}(\mathcal{G}_1)$, $\texttt{acc}(\mathcal{G}_1) \cap \texttt{acc}(\mathcal{A}_1)$ is simply $\texttt{acc}(\mathcal{G}_1)$. Furthermore, because β is a probability measure over the space $\{\theta \mid \theta|_{\emptyset} \in \texttt{acc}(\mathcal{A}_1)\}$, the denominator $\beta\{\theta \mid \theta|_{\emptyset} \in \texttt{acc}(\mathcal{A}_1)\}$, i.e. the measure of the entire space, is equal to 1.

Using the definition of $\texttt{acc}(\cdot)$ together with first Lemma 3 and then Lemma 4, we see that for $i \in \{1, \ldots, m\}$,

$$\{\theta \mid \theta|_{O_1 \cup \cdots \cup O_{i-1}} \in \texttt{acc}(\mathcal{A}_i)\} = \Theta_{\mathcal{A}}(\Pi_{\mathcal{A}_i}) \ . \tag{2}$$

Using a similar reasoning, we see that also

$$\{\theta \mid \theta|_{O_1 \cup \cdots \cup O_i} \in \texttt{acc}(\mathcal{G}_i) \text{ and } \theta|_{O_1 \cup \cdots \cup O_{i-1}} \in \texttt{acc}(\mathcal{A}_i)\} = \Theta_{\mathcal{A}}(\Pi_{\mathcal{G}_i} \cap \Pi_{\mathcal{A}_i}) \ . \tag{3}$$

Thus, we can rewrite Inequality (1) to get

$$\frac{\beta(\Theta_{\mathcal{A}}(\Pi_{\mathcal{G}_i} \cap \Pi_{\mathcal{A}_i}))}{\beta(\Theta_{\mathcal{A}}(\Pi_{\mathcal{G}_i}))} \bowtie_i p_i \ . \tag{4}$$

In order to construct a similar inequality for $\widehat{\phi}$, first note that because $\beta \notin [\![\widehat{\phi}]\!]$, case 1 of Def. 15 tells us that $\beta(\mathcal{G}_{\widehat{\phi}}) \bowtie_{\widehat{\phi}} p_{\widehat{\phi}}$ does not hold. This means that the complement $\beta(\texttt{acc}(\mathcal{G}_{\widehat{\phi}})) \bowtie^{c}_{\widehat{\phi}} p_{\widehat{\phi}}$ holds. From the definition of complement of PAC, together with case 1 of Def. 15, this can be stated equivalently as $\beta(\texttt{acc}(\mathcal{G}_{\widehat{\phi}^c})) \bowtie_{\widehat{\phi}^c} p_{\widehat{\phi}^c}$.

Once again, using the definition of $\texttt{acc}(\cdot)$ together with Lemma 3 and Lemma 4,

$$\texttt{acc}(\mathcal{G}_{\widehat{\phi}^c}) = \Theta_{\mathcal{A}}(\Pi_{\mathcal{G}_{\widehat{\phi}^c}}) \ ,$$

which implies

$$\beta(\Theta_{\mathcal{A}}(\Pi_{\mathcal{G}_{\widehat{\phi}^c}})) \bowtie_{\widehat{\phi}^c} p_{\widehat{\phi}^c} \ . \tag{5}$$

We now want to show that, given that such a β exists, Algorithm 1 does not return `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 $V : \texttt{paths}(\mathcal{A}) \to \mathbb{R}$ satisfying all generated inequalities. First, given any individual automaton $\mathcal{M}$ being a member of the composition $\mathcal{A}$, let $\Pi_{\mathcal{M}}$ denote the set of paths of $\mathcal{A}$ ending in an accepting location of $\mathcal{M}$, i.e. $\Pi_{\mathcal{M}} = \{\pi \in \texttt{paths}(\mathcal{A}) \mid \texttt{last}(\pi)|_{\mathcal{M}} \in F_{\mathcal{M}}\}$. For the non-assumption $\top$, let $\Pi_{\top}$ denote the set $\texttt{paths}(\mathcal{A})$ of all paths of $\mathcal{A}$. Consider now an arbitrary PAC $\phi_i, i \in \{1, \ldots, m\}$. For this ϕ_i, the algorithm adds a single inequality

$$\frac{\sum\limits_{\pi \in \Pi_{\mathcal{G}_i} \cap \Pi_{\mathcal{A}_i}} \pi}{\sum\limits_{\pi \in \Pi_{\mathcal{A}_i}} \pi} \bowtie_i p_i \ .$$

Note that the paths π used in this generated inequality are purely syntactical. They are used merely to represent variables, and the inequality is satisfied by a solution V if

$$\frac{\sum\limits_{\pi \in \Pi_{\mathcal{G}_i} \cap \Pi_{\mathcal{A}_i}} V(\pi)}{\sum\limits_{\pi \in \Pi_{\mathcal{A}_i}} V(\pi)} \bowtie_i p_i \ . \tag{6}$$

Using the fact that the traces of different paths are disjoint, together with the fact that probability measures are σ-additive, we can choose V such that, for each $i \in \{1, \ldots, m\}$,

$$\sum_{\pi \in \Pi_{\mathcal{A}_i}} V(\pi) = \beta(\Theta_{\mathcal{A}}(\Pi_{\mathcal{A}_i})) \ , \tag{7}$$

$$\sum_{\pi \in \Pi_{\mathcal{G}_i} \cap \Pi_{\mathcal{A}_i}} V(\pi) = \beta(\Theta_{\mathcal{A}}(\Pi_{\mathcal{G}_i} \cap \Pi_{\mathcal{A}_i})) \ , \tag{8}$$

and

$$\sum_{\pi \in \Pi_{\mathcal{G}_{\widehat{\phi}^c}}} V(\pi) = \beta(\Theta_{\mathcal{A}}(\Pi_{\mathcal{G}_{\widehat{\phi}^c}})) \ . \tag{9}$$

To see that such a choice is possible, let $\mathbf{\Pi}$ be the collection of all sets of paths used in (7), (8) and (9), i.e. $\mathbf{\Pi} = \{\Pi_{\mathcal{A}_i} \mid i \in \{1, \ldots, m\}\} \cup \{\Pi_{\mathcal{G}_i} \cap \Pi_{\mathcal{A}_i} \mid i \in \{1, \ldots, m\}\} \cup \Pi_{\mathcal{G}_{\widehat{\phi}^c}}$. Partition the set $\texttt{paths}(\mathcal{A})$ into the set of subsets

$$\mathbf{P} = \left\{ \left(\bigcap_{m \in C} m^c \right) \cap \left(\bigcap_{m \in \mathbf{\Pi} \setminus C} m \right) \;\middle|\; C \subseteq \mathbf{\Pi} \right\} \ ,$$

where m^c denotes the complement $\texttt{paths}(\mathcal{A}) \setminus m$. The elements of $\mathbf{P}$ are pairwise disjoint sets of paths, and, because σ-algebrae are closed under intersection and complement, correspond to measurable sets of traces. To choose V as above, all we need to do is to ensure that, for each element $\Pi \in \mathbf{P}$ of the partition, $\sum_{\pi \in \Pi} V(\pi) = \Theta_{\mathcal{A}}(\Pi)$. Because the elements $\Pi \in \mathbf{P}$ are pairwise disjoint sets, we can distribute the value $\Theta_{\mathcal{A}}(\Pi)$ however we want among the paths $\pi \in \Pi$ without interfering with any other element $\Pi' \neq \Pi$ in the partition $\mathbf{P}$.

Choosing V according to (7), (8) and (9) gives

$$\frac{\sum\limits_{\pi \in \Pi_{\mathcal{G}_i} \cap \Pi_{\mathcal{A}_i}} V(\pi)}{\sum\limits_{\pi \in \Pi_{\mathcal{A}_i}} V(\pi)} = \frac{\beta(\Theta_{\mathcal{A}}(\Pi_{\mathcal{G}_i} \cap \Pi_{\mathcal{A}_i}))}{\beta(\Theta_{\mathcal{A}}(\Pi_{\mathcal{G}_i}))} \ ,$$

whereby Inequality (6) follows immediately from Inequality (4).

Because $\texttt{paths}(\mathcal{A})$ constitues a partition of the entire trace space $\texttt{tr}(O)$ due to Lemma 5, and β is a probability measure such that $\beta(\texttt{tr}(O)) = 1$, it follows that any such assignment V also satisfies $\sum_{\pi \in \texttt{paths}(\mathcal{A})} V(\pi) = 1$, fulfilling the equality that the algorithm adds to ensure that the probabilities sum to 1. Note

that here, the value $\sum_{\pi \in \texttt{paths}(\mathcal{A})} V(\pi)$ is uniquely determined by Equation (7) for the case $i = 1$, since $\mathcal{A}_1 = \top$ and thus $\Pi_{\mathcal{A}_1} = \Pi_\top = \texttt{paths}(\mathcal{A})$.

Lastly, when it comes to $\widehat{\phi}^c$, the single inequality

$$\frac{\sum\limits_{\pi \in \Pi_{\mathcal{G}_{\widehat{\phi}^c}} \cap \Pi_{\mathcal{A}_{\widehat{\phi}^c}}} V(\pi)}{\sum\limits_{\pi \in \Pi_{\mathcal{A}_{\widehat{\phi}^c}}} V(\pi)} \bowtie_{\widehat{\phi}^c} p_{\widehat{\phi}^c}$$

is satisfied since

$$\frac{\sum\limits_{\pi \in \Pi_{\mathcal{G}_{\widehat{\phi}^c}} \cap \Pi_{\mathcal{A}_{\widehat{\phi}^c}}} V(\pi)}{\sum\limits_{\pi \in \Pi_{\mathcal{A}_{\widehat{\phi}^c}}} V(\pi)} = \frac{\sum\limits_{\pi \in \Pi_{\mathcal{G}_{\widehat{\phi}^c}}} V(\pi)}{\sum\limits_{\pi \in \texttt{paths}(\mathcal{A})} V(\pi)} = \frac{\sum\limits_{\pi \in \Pi_{\mathcal{G}_{\widehat{\phi}^c}}} V(\pi)}{1} = \beta(\Theta_{\mathcal{A}}(\Pi_{\mathcal{G}_{\widehat{\phi}^c}})) \bowtie_{\widehat{\phi}^c} p_{\widehat{\phi}^c} \ ,$$

where the last inequality is Inequality (5) and the first equality comes from the fact that $\mathcal{A}_{\widehat{\phi}^c} = \top$ which implies that $\Pi_{\mathcal{A}_{\widehat{\phi}^c}} = \texttt{paths}(\mathcal{A})$ and, in turn, that $\Pi_{\mathcal{G}_{\widehat{\phi}^c}} \cap \Pi_{\mathcal{A}_{\widehat{\phi}^c}} = \Pi_{\mathcal{G}_{\widehat{\phi}^c}}$.

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

Lemma 1. *Consider compatible specifications Σ_1 and Σ_2 and denote $\texttt{out}(\Sigma_1) = O_1$ and $\texttt{out}(\Sigma_2) = O_2$. Then, for each $\beta_1 \in \Sigma_1$, there exists a $\beta \in \Sigma_1 \parallel \Sigma_2$ such that for any set of traces $\Theta_1 \in \sigma_{\beta_1}$, it holds that $\beta_1(\Theta_1) = \beta(\{\theta \in \texttt{tr}(O_1 \cup O_2) \mid \theta|_{O_1} \in \Theta_1\})$.* □

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 $\beta_1(\Theta_1) = \beta_1 \parallel \beta_2(\{\theta \in \texttt{tr}(O_1 \cup O_2) \mid \theta|_{O_1} \in \Theta_1\})$, or equivalently $\beta_1(\Theta_1) = \beta_1 \| \beta_2(\Theta_1 \times \Omega_2)$, where $\Omega_2 = \texttt{tr}(O_2)$. Note that according to Def. 4, it holds that

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

Thus,

$$\beta_1 \| \beta_2(\Theta_1 \times \Omega_2) = \int_{\Theta_1} \beta_2(\theta_1)(\Omega_2) \beta_1(d\theta_1) = \int_{\Theta_1} 1 \beta_1(d\theta_1) = \beta_1(\Theta_1) \ . \tag{10}$$

□

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 $\mathtt{out}(\phi_1) = O_1$ and $\mathtt{out}(\phi_2) = O_2$. Then, for each $\beta \in [\![\phi_1 \,\|\, \phi_2]\!]$, it holds that*

$$\frac{\beta(\{\theta \in \mathtt{tr}(O_1 \cup O_2) \mid \theta \in \mathtt{acc}(\mathcal{G}_{\phi_2}) \text{ and } \theta|_{O_1} \in \mathtt{acc}(\mathcal{A}_{\phi_2})\})}{\beta(\{\theta \in \mathtt{tr}(O_1 \cup O_2) \mid \theta|_{O_1} \in \mathtt{acc}(\mathcal{A}_{\phi_2})\})} \bowtie_{\phi_2} p_{\phi_2} \ .$$

□

Proof. Pick an arbitrary $\beta \in [\![\phi_1 \,\|\, \phi_2]\!]$. From Def. 16, it holds that $\beta \in [\![\phi_1 \,\|\, \phi_2]\!] = [\![\phi_1]\!] \| [\![\phi_2]\!]$. Then, from Def. 8, 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 $\beta_2 \in [\![\mathcal{P}_{\bowtie_{\phi_2} p_{\phi_2}}(\mathcal{A}_{\phi_2}, \mathcal{G}_{\phi_2})]\!]$.

According to the assumption stated below Def. 4, the set

$$\{\theta \in \mathtt{tr}(O_1 \cup O_2) \mid \theta \in \mathtt{acc}(\mathcal{G}_{\phi_2}) \text{ and } \theta|_{O_1} \in \mathtt{acc}(\mathcal{A}_{\phi_2})\} \tag{11}$$

can be approximated to arbitrary precision by a countable disjoint union of sets in $\sigma_{\beta_1} \times \sigma_{\beta_2}$. Consider such approximations, and in particular over-estimations. Such an approximation means that

$$\{\theta \in \mathtt{tr}(O_1 \cup O_2) \mid \theta \in \mathtt{acc}(\mathcal{G}_{\phi_2}) \text{ and } \theta|_{O_1} \in \mathtt{acc}(\mathcal{A}_{\phi_2})\} \cup \Psi = \bigcup_j \Theta_1^j \times \Theta_2^j$$

for some set $\Psi \subseteq \mathtt{tr}(O_1 \cup O_2)$ of traces, where all $\Theta_1^j \subseteq \mathtt{tr}(O_1)$ are disjoint, all $\Theta_2^j \subseteq \mathtt{tr}(O_2)$ are disjoint, and Ψ is disjoint with the set (11). Because these sets are disjoint, and because β is σ-additive, we have

$$\beta\big(\{\theta \in \mathtt{tr}(O_1 \cup O_2) \mid \theta \in \mathtt{acc}(\mathcal{G}_{\phi_2}) \text{ and } \theta|_{O_1} \in \mathtt{acc}(\mathcal{A}_{\phi_2})\}\big) + \beta(\Psi) = \beta\big(\bigcup_j \Theta_1^j \times \Theta_2^j\big) \ .$$

Due to the assumption of approximation to arbitrary precision, we know that for any arbitrary small ϵ, we can find a partitioning $\bigcup_j \Theta_1^j \times \Theta_2^j$ such that

$$0 \leq \beta\big(\bigcup_j \Theta_1^j \times \Theta_2^j\big) - \beta\big(\{\theta \in \mathtt{tr}(O_1 \cup O_2) \mid \theta \in \mathtt{acc}(\mathcal{G}_{\phi_2}) \text{ and } \theta|_{O_1} \in \mathtt{acc}(\mathcal{A}_{\phi_2})\}\big) < \epsilon \ . \tag{12}$$

We will now consider a more specific partitioning $\bigcup_j \Theta_1^j \times \Theta_2^j$ constructed as follows. Let $\Theta_{A_2} = \mathtt{acc}(\mathcal{A}_{\phi_2})$. We then partition Θ_{A_2} into n sets $\Theta_{A_2}^i$, $i \in \{1, \ldots, n\}$. Let $\hat{\Theta}_{G_2}(\Theta_{A_2}^i) = \min\{\Theta_2 \subseteq \mathtt{tr}(O_2) \mid \mathtt{acc}(\mathcal{G}_{\phi_2}) \cap (\Theta_{A_2}^i \times \mathtt{tr}(O_2)) \subseteq \Theta_{A_2}^i \times \Theta_2\}$. This means that the sets $\{\Theta_{A_2}^i \times \hat{\Theta}_{G_2}(\Theta_{A_2}^i)\}_i$ are disjoint and thereby constitute a partition. Furthermore, its union is an over-estimation of the set (11).

Now, note that any given considered partition $\{\Theta_1^j \times \Theta_2^j\}_j$ can be replaced by a partition of the kind $\{\Theta_{A_2}^i \times \hat{\Theta}_{G_2}(\Theta_{A_2}^i)\}_i$, where $\bigcup_i \big(\Theta_{A_2}^i \times \hat{\Theta}_{G_2}(\Theta_{A_2}^i)\big)$ will be an over-estimation of (11), and at least as tight as $\bigcup_j (\Theta_1^j \times \Theta_2^j)$. Furthermore, the result (12) will still hold. Another way of writing this is by using the notation of limit, i.e.

$$\beta(\{\theta \in \mathtt{tr}(O_1 \cup O_2) \mid \theta \in \mathtt{acc}(\mathcal{G}_{\phi_2}) \text{ and } \theta|_{O_1} \in \mathtt{acc}(\mathcal{A}_{\phi_2})\}) = \lim_{n \to \infty} R_n \ ,$$

where

$$R_n = \sum_{i=1}^{n} \int_{\Theta^i_{A_2}} \beta_2(\theta_1)(\hat{\Theta}_{G_2}(\Theta^i_{A_2}))\beta_1(d\theta_1) \ , \tag{13}$$

for some sequence $\left(\{\Theta^i_{A_2}\}_{i=1}^n\right)_{n\geq 1}$ of partitions.

Now, first consider the case where $\bowtie_{\phi_2}$ is the operator $\geq$. For each $\theta_1 \in \mathtt{acc}(\mathcal{A}_{\phi_2})$, let $\Theta_{G_2}(\theta_1)$ denote the set $\Theta_{G_2}(\theta_1) = \{\theta_2 \in \mathtt{tr}(O_2) | \, \theta_1 \| \theta_2 \in \mathtt{acc}(\mathcal{G}_{\phi_2})\}$. Note that for each $\theta_1 \in \Theta^i_{A_2}$, $i \in \{1, \ldots, n\}$, it holds that $\hat{\Theta}_{G_2}(\Theta^i_{A_2}) \supseteq \Theta_{G_2}(\theta_1)$, and consequently, $\beta_2(\theta_1)(\hat{\Theta}_{G_2}(\Theta^i_{A_2})) \geq \beta_2(\theta_1)(\Theta_{G_2}(\theta_1))$. From Def. 15, and since $\beta_2 \in [\![\mathcal{P}_{\bowtie_{\phi_2} p_{\phi_2}}(\mathcal{A}_{\phi_2}, \mathcal{G}_{\phi_2})]\!]$ and $\theta_1 \in \mathtt{acc}(\mathcal{A}_{\phi_2})$, it follows that $\beta_2(\theta_1)(\Theta_{G_2}(\theta_1)) \geq p_{\phi_2}$.

Therefore, for any partition $\Theta^i_{A_2}$, $i \in \{1, \ldots, n\}$,

$$\sum_{i=1}^{n} \int_{\Theta^i_{A_2}} \beta_2(\theta_1)(\hat{\Theta}_{G_2}(\Theta^i_{A_2}))\beta_1(d\theta_1) \geq \sum_{i=1}^{n} \int_{\Theta^i_{A_2}} \beta_2(\theta_1)(\Theta_{G_2}(\theta_1))\beta_1(d\theta_1) \geq$$
$$\geq \sum_{i=1}^{n} \int_{\Theta^i_{A_2}} p_{\phi_2}\beta_1(d\theta_1) = p_{\phi_2} \sum_{i=1}^{n} \int_{\Theta^i_{A_2}} \beta_1(d\theta_1) \ . \tag{14}$$

Since $\int_{\Theta^i_{A_2}} \beta_1(d\theta_1) = \beta_1(\Theta^i_{A_2})$, it holds that

$$\sum_{i=1}^{n} \int_{\Theta^i_{A_2}} \beta_1(d\theta_1) = \sum_{i=1}^{n} \beta_1(\Theta^i_{A_2}) = \beta_1(\Theta_{A_2}) \ .$$

By combining these results, we obtain $R_n \geq p_{\phi_2}\beta_1(\Theta_{A_2})$, and further on,

$$\beta(\{\theta \in \mathtt{tr}(O_1 \cup O_2) \mid \theta \in \mathtt{acc}(\mathcal{G}_{\phi_2}) \text{ and } \theta|_{O_1} \in \mathtt{acc}(\mathcal{A}_{\phi_2})\}) =$$
$$= \lim_{n\to\infty} R_n \geq p_{\phi_2}\beta_1(\Theta_{A_2}) \ . \tag{15}$$

Next, note that $\{\theta \in \mathtt{tr}(O_1 \cup O_2) \mid \theta|_{O_1} \in \mathtt{acc}(\mathcal{A}_{\phi_2})\} = \Theta_{A_2} \times \Omega_2$, where $\Omega_2 = \mathtt{tr}(O_2)$. Thus, $\beta(\{\theta \in \mathtt{tr}(O_1 \cup O_2) \mid \theta|_{O_1} \in \mathtt{acc}(\mathcal{A}_{\phi_2})\}) = \beta(\Theta_{A_2} \times \Omega_2)$. 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, for the case where $\bowtie_{\phi_2}$ is the operator $\geq$, then follows from

$$\frac{\beta(\{\theta \in \mathtt{tr}(O_1 \cup O_2) \mid \theta \in \mathtt{acc}(\mathcal{G}_{\phi_2}) \text{ and } \theta|_{O_1} \in \mathtt{acc}(\mathcal{A}_{\phi_2})\})}{\beta(\{\theta \in \mathtt{tr}(O_1 \cup O_2) \mid \theta|_{O_1} \in \mathtt{acc}(\mathcal{A}_{\phi_2})\})} \geq \frac{p_{\phi_2}\beta_1(\Theta_{A_2})}{\beta_1(\Theta_{A_2})} = p_{\phi_2} \ .$$

The case where $\bowtie_{\phi_2}$ is the operator $\leq$ is proven similarly but by using $\check{\Theta}_{G_2}(\Theta^i_{A_2}) = \max\{\Theta_2 \subseteq \mathtt{tr}(O_2) \,|\, \mathtt{acc}(\mathcal{G}_{\phi_2}) \supseteq \Theta^i_{A_2} \times \Theta_2\}$ instead of $\hat{\Theta}_{G_2}(\Theta^i_{A_2})$ in (13) and (14). □

The following lemma states that automaton composition preserves the last location of execution.

Lemma 3. *Consider a set of terminating trace automata* $\{\mathcal{A}_i\}_{i=1}^k$*, each automaton defined for a corresponding variable set* E_i*, and denote by* $\mathcal{A}$ *the composition automaton* $\mathcal{A}_1 \parallel \ldots \parallel \mathcal{A}_k$ *being defined for* $E = E_1 \cup \cdots \cup E_k$*. Then, for each index* $i \in \{1, \ldots, k\}$ *and trace* $\theta \in \mathtt{tr}(E)$*,* $\mathtt{last}(\mathcal{A}_i(\theta|_{E_i})) = \mathtt{last}(\mathcal{A}(\theta))|_{\mathcal{A}_i}$*.* □

Proof. Because parallel composition of trace automata preserves determinism, and the executions of the composed automata are independent given an input trace, it follows that each transition of the composition is equivalent to each individual automaton taking any possible transitions, respectively. Thus, individual executions are preserved in executions of the composition. □

Lemma 4. *Consider a set of terminating trace automata* $\{\mathcal{A}_i\}_{i=1}^k$*, each automaton defined for a corresponding variable set* E_i*, and denote by* $\mathcal{A}$ *the composition automaton* $\mathcal{A}_1 \parallel \ldots \parallel \mathcal{A}_k$ *defined for* $E = E_1 \cup \cdots \cup E_k$*. Then, for each individual automaton* $\mathcal{A}_i, i \in \{1, \ldots, k\}$*, and each of its locations* $l \in L_{\mathcal{A}_i}$*,* $\{\theta \in \mathtt{tr}(\mathcal{A}) \mid \mathtt{last}(\mathcal{A}(\theta))|_{\mathcal{A}_i} = l\} = \Theta_{\mathcal{A}}(\{\pi \in \mathtt{paths}(\mathcal{A}) \mid \mathtt{last}(\pi)|_{\mathcal{A}_i} = l\})$*.* □

Proof. Follows from the fact that each trace $\theta \in \mathtt{tr}(\mathcal{A})$ has a corresponding execution $\mathcal{A}(\theta)$ and, from Def. 11, a corresponding path $\pi = \mathcal{A}(\theta)|_L$ with $\mathtt{last}(\mathcal{A}(\theta)) = \mathtt{last}(\pi)$ and, therefore, $\mathtt{last}(\mathcal{A}(\theta))|_{\mathcal{A}_i} = \mathtt{last}(\pi)|_{\mathcal{A}_i}$. In the reverse direction, each path π has a corresponding set Θ of traces such that $\forall \theta \in \Theta \,.\, \mathcal{A}(\theta)|_L = \pi$ and, once again, $\mathtt{last}(\mathcal{A}(\theta))|_{\mathcal{A}_i} = \mathtt{last}(\pi)|_{\mathcal{A}_i}$. □

The following lemma states that the possible paths of an automaton is a partition of the trace space.

Lemma 5. *Given a set of variables* $E \subseteq X$ *and a deterministic 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. Consider an arbitrary trace $\theta \in \mathtt{tr}(E)$. Using the fact that $\mathcal{A}$ is deterministic, there exists exactly one execution $\mathcal{A}(\theta)$ of $\mathcal{A}$ on θ. According to Def. 11, it follows that $\mathcal{A}(\theta)|_{L_{\mathcal{A}}}$ is a path of $\mathcal{A}$. Thus, $\theta \in \cup_{\pi \in \mathtt{paths}(\mathcal{A})} \Theta_{\mathcal{A}}(\pi)$, and, because θ was arbitrarily chosen 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 $\theta_\pi \in \Theta_{\mathcal{A}}(\pi)$ for $\pi \in \mathtt{paths}(\mathcal{A})$ is a trace, $\cup_{\pi \in \mathtt{paths}(\mathcal{A})} \Theta_{\mathcal{A}}(\pi) \subseteq \mathtt{tr}(E)$, implying (1). Furthermore, the location sequence $\mathcal{A}(\theta)|_{L_{\mathcal{A}}}$ is a unique path, i.e. each trace corresponds to no more than one path, implying (2). □

5 Case Study

Recall the two-component system from Sec. 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 Sec. 4.

Once again, the natural language top-level specification is: "the system shall output power continuously during the first 7 hours with over 50% probability". To represent this specification, we can define the PAC

$$\widehat{\phi} = \mathcal{P}_{>0.5}\Big(\top, \quad \rightarrow \underset{\text{ok}}{\textcircled{\textcircled{}}} \xrightarrow[c_M < 7]{\{p_M \mapsto 0, p_B \mapsto 0\}} \underset{\text{pre}}{\textcircled{}}\Big).$$

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 before 7 hours with neither main nor backup power. The probability bound put on the guarantee is > 0.5.

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

$$\phi_M = \mathcal{P}_{\geq 0.7}\Big(\top, \quad \rightarrow \underset{\text{ok}}{\textcircled{\textcircled{}}} \xrightarrow[c_M < 6]{\{p_M \mapsto 0\}} \underset{\text{pre}}{\textcircled{}}\Big),$$

$$\phi_B = \mathcal{P}_{\geq 0.8}\Big(\underset{\text{U}}{\textcircled{}} \xrightarrow[c_M \geq 3]{\{p_M \mapsto 0\}} \underset{\text{T}}{\textcircled{\textcircled{}}}, \quad \underset{\text{wait}}{\textcircled{}} \xrightarrow[c_B := 0]{\{p_M \mapsto 0, p_B \mapsto 1\}} \underset{\text{ok}}{\textcircled{\textcircled{}}} \Big),$$

$$\underset{\text{U}}{\textcircled{}} \xrightarrow[c_M < 3]{\{p_M \mapsto 0\}} \underset{\text{F}}{\textcircled{}}, \qquad \underset{\text{wait}}{\textcircled{}} \xrightarrow{\{p_M \mapsto 0, p_B \mapsto 0\}} \underset{\text{fail}}{\textcircled{}} \xrightarrow[c_B < 2]{\{p_B \mapsto 0\}} \underset{\text{ok}}{\textcircled{\textcircled{}}}$$

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.

Following Algorithm 1, we first construct the composition $\mathcal{A} = \mathcal{A}_{\widehat{\phi}} \| \mathcal{G}_{\widehat{\phi}} \| \mathcal{A}_{\phi_M} \| \mathcal{G}_{\phi_M} \| \mathcal{A}_{\phi_B} \| \mathcal{G}_{\phi_B}$. The resulting automaton is shown in Fig. 8, where only the reachable part is included. Next to each location, there is a tuple giving the initials of the corresponding component automaton locations, where e.g. (p,p,f,f) refers to locations *pre*, *pre*, *F*, and *fail* of $\mathcal{G}_{\widehat{\phi}}$, $\mathcal{G}_{\phi_M}$, $\mathcal{A}_{\phi_B}$, 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 $\{p_M \mapsto 1, p_B \mapsto 0\}$. Lastly, note 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 $c_M < 6 \wedge \neg(c_M < 3)$.

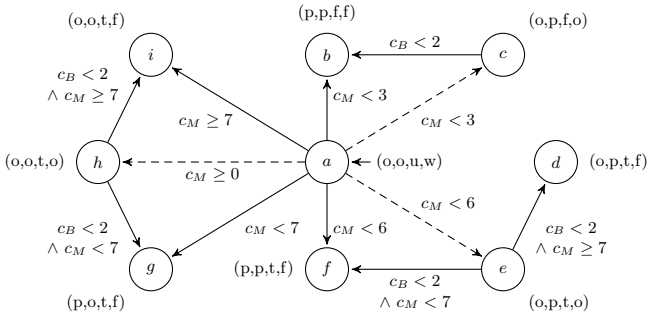

Fig. 8: The composition $\mathcal{A} = \mathcal{A}_{\widehat{\phi}} \| \mathcal{G}_{\widehat{\phi}} \| \mathcal{A}_{\phi_M} \| \mathcal{G}_{\phi_M} \| \mathcal{A}_{\phi_B} \| \mathcal{G}_{\phi_B}$

Identifying the paths ending in accepting locations of each automaton results in the sets $\Pi_{\mathcal{A}_{\widehat{\phi}}} = \Pi_{\mathcal{A}_{\phi_M}} = \texttt{paths}(\mathcal{A}) = \{a, ab, ac, acb, ae, aed, aef, af, ag, ah, ahg, ahi, ai\}$, $\Pi_{\mathcal{G}_{\widehat{\phi}} \wedge \mathcal{A}_{\widehat{\phi}}} = \{a, ac, ae, aed, ah, ahi\}$, $\Pi_{\mathcal{G}_{\phi_M} \wedge \mathcal{A}_{\phi_M}} = \{a, ag, ah, ahg, ahi, ai\}$, $\Pi_{\mathcal{A}_{\phi_B}} = \{ae, aed, aef, ag, ah, ahg, ahi, ai\}$, and $\Pi_{\mathcal{G}_{\phi_B} \wedge \mathcal{A}_{\phi_B}} = \{ae, ah\}$. This results in the following system of linear inequalities:

$$\begin{cases} a + ag + ah + ahg + ahi + ai \geq 0.7 \\ \dfrac{ae + ah}{ae + aed + aef + ag + ah + ahg + ahi + ai} \geq 0.8 \\ a + ac + ae + aed + ah + ahi \leq 0.5 \\ a + ab + ac + acb + ae + aed + aef + af + ag + ah + ahg + ahi + ai = 1 \ . \end{cases}$$

Running a linear optimization solver, e.g. [1], on this instance shows that the solution space is empty. Thus, we have verified that the composition of ϕ_M and ϕ_B refines $\widehat{\phi}$. 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.

6 Related Work

A related field is the area of model checking. 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. [24, 30].

The literature contains various proposed specification theories for stochastic systems, supporting for instance constraint Markov chains [8], abstract probabilistic automata [14], interactive Markov chains [16], and a variety of probabilistic transition systems [20, 21, 23, 33]. In the contract context, Nuzzo et al.

[28] 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) [6]. 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 [13] is also trace-based, but in discrete time.

Both automata and temporal logics can be used for specifying properties of systems. For specifying stochastic systems in continuous time, Continuous Stochastic Logic (CSL) is commonly used [17]. The extension CSL^{TA} allows specifying properties through single-clock automata and has been used for model checking CTMCs [15]. A specification theory allowing compositional reasoning has been developed for timed I/O automata [12]; however, this framework gives no explicit support for probabilities. In a discrete-time setting, temporal operators defined by finite automata are included in a temporal logic presented by [35], and in an extension to computation tree logic, called ECTL [9].

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 system failure will occur before a given time. A decomposition of such a specification requires techniques beyond traditional theorem proving.

As presented in Sec. 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 of *refinement* between specifications in the form of assume-guarantee contracts. This is needed to support compositional verification, which in turn is a key solution to specify and verify large-scale complex systems. A main 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 Sec. 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 implementations of the algorithm, e.g. by replacing explicit path enumeration, and experimental evaluation using larger and more realistic case studies motivated by industry.

References

1. Linear optimization. https://online-optimizer.appspot.com, (Accessed on 05/27/2022)

2. Alur, R., Dill, D.: Automata for modeling real-time systems. In: International colloquium on automata, languages, and programming. pp. 322–335. Springer (1990)
3. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical computer science **126**(2), 183–235 (1994)
4. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.: Verifying continuous time markov chains. In: International Conference on Computer Aided Verification. pp. 269–276. Springer (1996)
5. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.: Model-checking continuous-time markov chains. ACM Transactions on Computational Logic (TOCL) **1**(1), 162–170 (2000)
6. Baier, C., Katoen, J.P., Hermanns, H., Wolf, V.: Comparative branching-time semantics for markov chains. Information and computation **200**(2), 149–214 (2005)
7. Benveniste, A., Caillaud, B., Ferrari, A., Mangeruca, L., Passerone, R., Sofronis, C.: Multiple viewpoint contract-based specification and design. In: Formal Methods for Components and Object, pp. 200–225. Springer-Verlag, Berlin, Heidelberg (2008)
8. 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)
9. 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 (1989)
10. Cuoq, P., Kirchner, F., Kosmatov, N., Prevosto, V., Signoles, J., Yakobowski, B.: Frama-c. In: International conference on software engineering and formal methods. pp. 233–247. Springer (2012)
11. Dantzig, G.B.: Origins of the simplex method. In: A history of scientific computing, pp. 141–151 (1990)
12. 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)
13. Delahaye, B., Caillaud, B., Legay, A.: Probabilistic contracts: a compositional reasoning methodology for the design of systems with stochastic and/or nondeterministic aspects. Formal Methods in System Design **38**(1), 1–32 (2011)
14. Delahaye, B., Katoen, J.P., Larsen, K.G., Legay, A., Pedersen, M.L., Sher, F., Wasowski, A.: Abstract probabilistic automata. In: International Workshop on Verification, Model Checking, and Abstract Interpretation. pp. 324–339. Springer (2011)
15. Donatelli, S., Haddad, S., Sproston, J.: Model checking timed and stochastic properties with csl^{TA}. IEEE Transactions on Software Engineering **35**(2), 224–240 (2008)
16. Gössler, G., Xu, D.N., Girault, A.: Probabilistic contracts for component-based design. Formal Methods in System Design **41**(2), 211–231 (2012)
17. Grunske, L.: Specification patterns for probabilistic quality properties. In: 2008 ACM/IEEE 30th International Conference on Software Engineering. pp. 31–40. IEEE (2008)
18. ISO 21434: "Road vehicles – Cybersecurity engineering" (2021)
19. ISO 26262: "Road vehicles - Functional safety" (2018)
20. 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 Computer Society (1991)

21. Jonsson, B., Yi, W.: Testing preorders for probabilistic processes can be characterized by simulations. Theoretical Computer Science **282**(1), 33–51 (2002)
22. Kern, C., Greenstreet, M.R.: Formal verification in hardware design: a survey. ACM Transactions on Design Automation of Electronic Systems (TODAES) **4**(2), 123–193 (1999)
23. Lanotte, R., Maggiolo-Schettini, A., Troina, A.: Parametric probabilistic transition systems for system design and analysis. Formal Aspects of Computing **19**(1), 93–109 (2007)
24. Mereacre, A., Katoen, J.P., Han, T., Chen, T.: Model checking of continuous-time markov chains against timed automata specifications. Logical Methods in Computer Science **7** (2011)
25. Meyer, B.: Applying'design by contract'. Computer **25**(10), 40–51 (1992)
26. Moura, L.d., Bjørner, N.: Z3: An efficient smt solver. In: International conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 337–340. Springer (2008)
27. Nash, J.C.: The (dantzig) simplex method for linear programming. Computing in Science & Engineering **2**(1), 29–31 (2000)
28. Nuzzo, P., Li, J., Sangiovanni-Vincentelli, A.L., Xi, Y., Li, D.: Stochastic assume-guarantee contracts for cyber-physical system design. ACM Transactions on Embedded Computing Systems (TECS) **18**(1), 1–26 (2019)
29. Nyberg, M., Westman, J., Gurov, D.: Formally proving compositionality in industrial systems with informal specifications. In: International Symposium on Leveraging Applications of Formal Methods. pp. 348–365. Springer (2020)
30. Paolieri, M., Horváth, A., Vicario, E.: Probabilistic model checking of regenerative concurrent systems. IEEE Transactions on Software Engineering **42**(2), 153–169 (2015)
31. Resnick, S.: A Probability Path. Birkhäuser Boston (2019)
32. Roever, W.P.d.: The need for compositional proof systems: A survey. In: International Symposium on Compositionality. pp. 1–22. Springer (1997)
33. Segala, R., Lynch, N.: Probabilistic simulations for probabilistic processes. In: International Conference on Concurrency Theory. pp. 481–496. Springer (1994)
34. Slind, K., Norrish, M.: A brief overview of hol4. In: International Conference on Theorem Proving in Higher Order Logics. pp. 28–32. Springer (2008)
35. Vardi, M.Y., Wolper, P.: Reasoning about infinite computations. Information and computation **115**(1), 1–37 (1994)
36. Westman, J., Nyberg, M.: Conditions of contracts for separating responsibilities in heterogeneous systems. Formal Methods in System Design (Sep 2017). https://doi.org/10.1007/s10703-017-0294-7