Model Checking Probabilistic Distributed Systems

Benedikt Bollig[1] and Martin Leucker[2*]

[1] Lehrstuhl für Informatik II, RWTH Aachen, Germany
bollig@informatik.rwth-aachen.de
[2] IT department, Uppsala University, Sweden
Martin.Leucker@it.uu.se

Abstract. Protocols for distributed systems make often use of random transitions to achieve a common goal. A popular example are randomized leader election protocols. We introduce *probabilistic product automata* (PPA) as a natural model for this kind of systems. To reason about these systems, we propose to use a product version of linear temporal logic ($\mathrm{LTL}^{\otimes}$). The main result of the paper is a model-checking procedure for PPA and $\mathrm{LTL}^{\otimes}$. With its help, it is possible to check qualitative properties of distributed systems automatically.

1 Introduction

Randomization techniques have been employed to solve numerous problems of computing both sequentially and in parallel. Examples of probabilistic algorithms that are asymptotically better than their deterministic counterparts in solving various fundamental problems abound. It has also been shown that they allow solutions of problems which cannot be solved deterministically [LR81]. They have the advantages of simplicity and better performance both in theory and often in practice. Prominent examples for distributed randomized algorithms are randomized leader election protocols [BGJ99]. An overview of the domain of randomized algorithms is given in [PRRR01].

As for any kind of hardware or software system, it is important to develop formal methods and tools for verifying their correctness, thus also for probabilistic programs. *Model checking*, introduced independently in [CES83] and [QS82], turned out to be one fruitful approach to automatically verify systems (see [CW96] for an overview of model checking in practice). In the model-checking approach, usually a finite system $\mathcal{M}$, often an abstraction of a real system, and a property, usually expressed as a temporal-logic formula φ or as an automaton describing the computations that adhere to the property, are given. The *model-checking procedure* decides whether the set or tree formed of all *computations* of $\mathcal{M}$ *satisfies* φ, or, in other words, whether the given system satisfies the required property. Temporal logics used for expressing requirements are usually linear temporal logics (LTL) (as initially proposed by Pnueli in [Pnu77]) or branching-time logics like computation-tree logics CTL and CTL*.

* This author is supported by the European Research Training Network "Games".

In the probabilistic setting, one is no longer interested in *all* but only *almost all* computations. Thus, (sets or paths of trees of) computations with zero probability are ignored when deciding whether a given property is satisfied.

Temporal logics for probabilistic programs were initially studied in [LS82] and [HS84]. [Var85] introduced the notion of *Concurrent Probabilistic Programs* (CPPs) and provided a procedure for checking their LTL properties.

In this paper, we introduce *Probabilistic Product Automata* (PPA) as a model for *distributed* probabilistic systems. Roughly, a PPA is a parallel product of CPPs. They are enriched, however, with *actions* to synchronize the concurrent execution. In the non-randomized setting, product automata have proven to be a reasonable model for distributed systems. To support their analysis, a product version of linear temporal logic ($\mathrm{LTL}^{\otimes}$) has been defined. It allows the definition of properties for each process, i.e., each single automaton or CPP of the overall system. The properties for each process are then combined by means of Boolean connectives to a single formula. Model-checking procedures for $\mathrm{LTL}^{\otimes}$ and product automata have been studied in [Thi95]. We show in this paper, how $\mathrm{LTL}^{\otimes}$ properties of PPA can be checked automatically.

Our method follows the automata-theoretic approach: Given a PPA $\mathcal{A}$, we can construct a single concurrent probabilistic program $\mathcal{M}$ with the same probabilistic behavior. For a property φ expressed as an $\mathrm{LTL}^{\otimes}$ formula, we consider its negation $\neg\varphi$ and construct a (non-randomized) product automaton $\mathcal{A}_{\neg\varphi}$ that captures the behavior violating φ. This automaton is transformed into a deterministic Streett automaton $\mathcal{B}_{\neg\varphi}$. The common behavior of $\mathcal{M}$ and $\mathcal{B}_{\neg\varphi}$ represents the behavior of the underlying system that does not satisfy our property. If the behavior is empty in a probabistic sense, φ is satisfied by $\mathcal{A}$.

We define a kind of intersection of $\mathcal{M}$ and $\mathcal{B}_{\neg\varphi}$ and provide simple graph algorithms that answer the question whether $\mathcal{M}$ and $\mathcal{B}_{\neg\varphi}$ have a common behavior, in the probabilistic sense mentioned above. Our procedure is quadratic in the size of $\mathcal{M}$ (which grows exponentially with the number of parallel CPPs) and double exponential in size of φ.

Often, distributed systems are modelled as a single nondeterministic one. While our model-checking procedure makes use of this idea, our approach starting with a distributed system has several advantages. Firstly, it is more direct to model a distributed protocol as a PPA rather than as a single system. Secondly, it is more direct to formulate properties of the distributed system on a "per process basis" as done in $\mathrm{LTL}^{\otimes}$ rather than for a system capturing the interleaved behavior. Thirdly, performance benefits can be expected in practice, since our procedure works *on-the-fly*: The main question to answer is whether the intersection of $\mathcal{M}$ and $\mathcal{B}_{\neg\varphi}$ has empty behavior. If a single behavior is found, the model checking procedure can stop and provide this counter example. Since the combined system can be checked in a top-down manner, often only a part of this system has to be constructed. This implies that only a part of $\mathcal{M}$ and a part of $\mathcal{B}_{\neg\varphi}$ has to be constructed. Since $\mathcal{M}$ and $\mathcal{B}_{\neg\varphi}$ are (double) exponentially larger than the underlying structures $\mathcal{A}$ and φ, this has a considerable effect in practice.

This paper is organized as follows. In the next section, we introduce the necessary concepts and notation of words, graphs, and automata. Section 3 presents our model for distributed probabilistic systems, probabilistic product automata. LTL$^{\otimes}$ is defined in Section 4. The main contribution of the paper, the model checking algorithm for PPA and LTL$^{\otimes}$ is explained in Section 5. We sum up our approach and mention directions for future work in the last section.

2 Preliminaries

Given an alphabet Σ, Σ^* denotes the set of *finite* and Σ^ω the set of *infinite words* over Σ, respectively. Furthermore, let $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$ be the set of *words*. For a word $\sigma = a_1 a_2 \ldots \in \Sigma^\omega$ and a natural $i \geq 1$, let σ^i denote $a_i a_{i+1} \ldots$, the ith suffix of σ, and let $\sigma(i)$ denote a_i, the ith letter of σ. Furthermore, take $\mathit{inf}(\sigma)$ as the *infinity set* of σ, i.e., the set of letters that occur infinitely often in σ. For a word $\sigma = a_1 \ldots a_n \in \Sigma^*$ and a natural $i \in \{1, \ldots, n+1\}$, we accordingly define σ^i to be the word σ' with $a_1 \ldots a_{i-1}\sigma' = \sigma$ (thus, σ^{n+1} is the empty word ε) and, if $i \leq n$, $\sigma(i)$ to be a_i.

Given a word σ and an alphabet Σ, let $\sigma \downharpoonleft \Sigma$ denote the word we get by erasing from σ the letters that are not contained in Σ.

Given a (directed) graph G with nodes V and edges E, we call a node $v \in V$ (a set $D \subseteq V$ of nodes) *reachable* from $v' \in V$ if there is a path from v' to v (to a node contained in D). A *strongly connected component* (SCC) of G is a maximal set D of nodes such that every two nodes contained in D are reachable from each other. A SCC is called *bottom* if there is no edge leading out of it. We define G to be *strongly connected* if V forms a SCC. Furthermore, G is called *nontrivial* if it contains at least one edge. A set $D \subseteq V$ is said to be *nontrivial* if $G[D]$, the subgraph of G *induced* by D, is nontrivial. The *size* of G, denoted by $|G|$, is defined to be $|V| + |E|$.

Let Σ be an alphabet. An *extended Streett automaton* over Σ is a tuple $\mathcal{A} = (S, S_0, \delta, F, \mathcal{G})$ where S is its nonempty finite set of *states*, $S_0 \subseteq S$ is the set of *initial states*, $\delta : S \times \Sigma \to 2^S$ is its *transition function*, $F \subseteq S$ is its set of *final states*, and $\mathcal{G}$ is a subset of $(2^S)^2$. A *run* of $\mathcal{A}$ on a word $\sigma = a_1 a_2 \ldots \in \Sigma^\omega$ (on a word $\sigma = a_1 \ldots a_n \in \Sigma^*$) is a sequence of states $\rho = s_0 s_1 s_2 \ldots \in S^\omega$ (a sequence of states $\rho = s_0 s_1 \ldots s_n \in S^*$) such that $s_0 \in S_0$ and, for each natural i (for each $i \in \{0, 1, \ldots, |\rho| - 1\}$), $s_{i+1} \in \delta(s_i, a_{i+1})$. We call ρ *accepting* if either σ is finite and $s_{|\sigma|} \in F$ or σ is infinite and, for all pairs $(U, V) \in \mathcal{G}$, $\mathit{inf}(\rho) \cap U \neq \emptyset$ implies $\mathit{inf}(\rho) \cap V \neq \emptyset$. The *language* of $\mathcal{A}$, denoted by $L(\mathcal{A})$, is the set $\{\sigma \in \Sigma^\infty \mid$ there is an accepting run of $\mathcal{A}$ on $\sigma\}$. $\mathcal{A}$ is called *deterministic* if both $|S_0| = 1$ and $|\delta(s, a)| = 1$ for all $s \in S$, $a \in \Sigma$. Furthermore, the *size* $|\mathcal{A}|$ of $\mathcal{A}$ is defined to be the size of its (transition) graph. An extended Büchi automaton over Σ is just an extended Streett automaton, but it employs an acceptance component $\mathcal{F} \subseteq S$ instead of $\mathcal{G}$. A run over an infinite word is henceforth accepting if it visits at least one state from $\mathcal{F}$ infinitely often.

Given an extended Büchi automaton $\mathcal{A}$ over an alphabet Σ, there is a deterministic extended Streett automaton $\mathcal{B}$ over Σ such that $L(\mathcal{B}) = L(\mathcal{A})$ [GTW02].

3 Probabilistic Product Automata

Before we present our model for distributed concurrent probabilistic programs, we define the notion of its building blocks, the concurrent probabilistic programs. Let us describe the systems we want to study intuitively first. Figure 1 shows two concurrent probabilistic programs. The system starts in a *nondeterministic* state, shown as a circle, and selects nondeterministically a randomizing state, represented as a box. The transitions are labelled by actions which will later be used to synchronize the parallel execution of several concurrent probabilistic programs. In a randomizing state, the system chooses a nondeterministic successor state according to the probabilities of the outgoing arcs. Let us be more precise:

Definition 1 (Concurrent Probabilistic Program). *A* concurrent probabilistic program *(CPP) over an alphabet Σ is a tuple $\mathcal{M} = (Q, N, R, \Delta, P, Q^{in})$ where*

- *N and R are disjoint nonempty finite sets of* nondeterministic *and* randomizing states*, respectively, and $Q = N \cup R$ is the set of* states*,*
- *$\Delta \subseteq N \times \Sigma \times R$ is the set of* transitions*,*
- *$P : R \times N \to [0, 1]$ is the* transition probability distribution[1] *such that, for each $q \in R$, $\sum_{q' \in N} P(q, q') = 1$, and*
- *$Q^{in} \subseteq Q$ is the set of* initial states*.*

A nondeterministic state $q \in N$ is called *enabled* in $\mathcal{M}$, if has outgoing transitions, i.e. if there is a $q' \in R$ and an $a \in \Sigma$ such that $(q, a, q') \in \Delta$. The set of enabled states of $\mathcal{M}$ is denoted by $N^{en}_{\mathcal{M}}$.

Sequences of transitions of a CPP involve actions as well as random choices. To be able to handle both kinds of transitions in the same manner, we use the symbol $\mathfrak{p}$ to denote a random move and set $\Sigma_{\mathfrak{p}} := \Sigma \uplus \{\mathfrak{p}\}$. To study the probabilistic behavior of a CPP, we consider its possible random executions, when fixing the nondeterministic choices by means of a scheduler. Given a partial execution of the system, i.e., a sequence of states and actions (including $\mathfrak{p}$), ending in an enabled nondeterministic state, a scheduler tells us which action and successor state to choose: A *scheduler* of a CPP $\mathcal{M} = (Q, N, R, \Delta, P, Q^{in})$ over Σ is a mapping $u : (Q\Sigma_{\mathfrak{p}})^* N^{en}_{\mathcal{M}} \to \Sigma \times R$ such that, for each $x \in (Q\Sigma_{\mathfrak{p}})^*$ and $q \in N^{en}_{\mathcal{M}}$, $u(xq) = (a, q')$ implies $(q, a, q') \in \Delta$.

For the rest of the paper, we fix a natural $K \geq 1$, the set $Proc = \{1, \ldots, K\}$ of *processes*, and a *distributed alphabet* $\tilde{\Sigma} = (\Sigma_1, \ldots, \Sigma_K)$, a tuple of (not necessarily disjoint) alphabets Σ_i. We use $\bigcup_i \Sigma_i$ as a shorthand for $\bigcup_{i \in Proc} \Sigma_i$. For $a \in \bigcup_i \Sigma_i$, let $loc(a) := \{i \in Proc \mid a \in \Sigma_i\}$ be the set of processes that the action a participates in.

We are now ready to define our model for distributed concurrent probabilistic programs:

[1] We usually write $P(q, q')$ instead of $P((q, q'))$.

Definition 2 (Probabilistic Product Automaton). *A* probabilistic product automaton *(PPA) over $\tilde{\Sigma}$ is a structure $\mathcal{A} = ((\mathcal{A}_i)_{i \in Proc}, S^{in})$ such that*

– *for each $i \in Proc$, $\mathcal{A}_i$ is a CPP $(Q_i, N_i, R_i, \Delta_i, P_i)$ (without set of initial states) over Σ_i and*
– *$S^{in} \subseteq \prod_{i \in Proc} Q_i$ is the set of* global initial states.

The two CPPs shown in Figure 1 form together a PPA over $(\{a, b\}, \{b, c\})$ when setting the initial state of the system to (p_0, q_0). Note that the probability distributions P_i are reflected by transition arcs in case of nonzero transition probabilities, respectively.

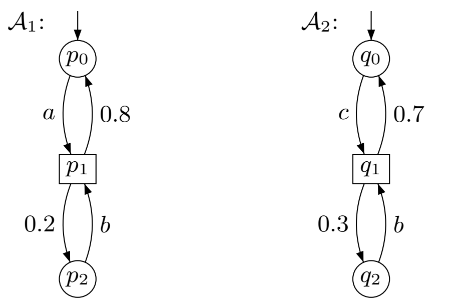

Fig. 1. A probabilistic product automaton

A PPA $\mathcal{A} = ((\mathcal{A}_i)_{i \in Proc}, S^{in})$, $\mathcal{A}_i = (Q_i, N_i, R_i, \Delta_i, P_i)$, determines the CPP $\mathcal{M}_\mathcal{A} = (Q_\mathcal{A}, N_\mathcal{A}, R_\mathcal{A}, \Delta_\mathcal{A}, P_\mathcal{A}, Q^{in}_\mathcal{A})$ over $\bigcup_i \Sigma_i$ where

– $Q_\mathcal{A} = \prod_{i \in Proc} Q_i$
  (in the following, for $\overline{q} = (q_1, \ldots, q_K) \in Q_\mathcal{A}$ and $i \in Proc$, let $\overline{q}[i]$ denote q_i)
– $N_\mathcal{A} = \prod_{i \in Proc} N_i$,
– $R_\mathcal{A} = Q_\mathcal{A} \setminus N_\mathcal{A}$,
– $(\overline{q}, a, \overline{q}') \in \Delta_\mathcal{A}$ if
  • $(\overline{q}[i], a, \overline{q}'[i]) \in \Delta_i$ for all $i \in loc(a)$ and
  • $\overline{q}[i] = \overline{q}'[i]$ for all $i \notin loc(a)$,
– $Q^{in}_\mathcal{A} = S^{in}$, and
– $P(\overline{q}, \overline{q}') = \begin{cases} \prod_{i \in Proc,\ \overline{q}[i] \in R_i} P_i(\overline{q}[i], \overline{q}'[i]) & \text{if, for each } i \in Proc, \\ & \overline{q}[i] \in N_i \text{ implies } \overline{q}[i] = \overline{q}'[i] \\ 0 & \text{otherwise} \end{cases}$

It is easy to verify that $\mathcal{M}_\mathcal{A}$ is indeed a CPP.

The PPA from Figure 1 induces the CPP given by Figure 2.

Let us now recall the probabilistic setting needed to reason about probabilistic programs.

A nonempty set of possible outcomes of an experiment of chance is called *sample space*. Let Ω be a sample space. A set $\mathfrak{B} \subseteq 2^\Omega$ is called *Borel field* over Ω

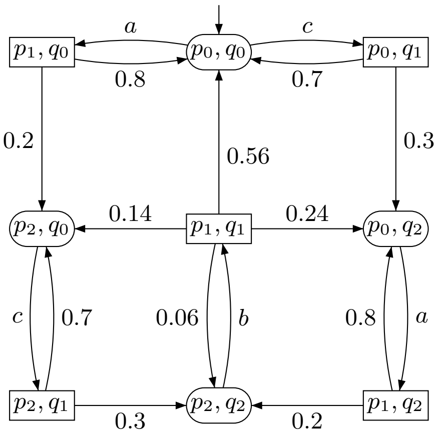

Fig. 2. The CPP generated by a PPA

if it contains Ω, $\Omega \setminus E$ for each $E \in \mathfrak{B}$, and the union of any countable sequence of sets from $\mathfrak{B}$. A Borel field $\mathfrak{B}$ is *generated* by an at most countable set $\mathcal{E}$, denoted by $\mathfrak{B} = \langle \mathcal{E} \rangle$, if $\mathfrak{B}$ is the closure of $\mathcal{E}$'s elements under complement and countable union.

A *probability space* is a triple $\mathcal{PS} = (\Omega, \mathfrak{B}, \mu)$ where Ω is a sample space, $\mathfrak{B}$ is a Borel field over Ω, and μ is a mapping $\mathfrak{B} \to [0,1]$ such that $\mu(\Omega) = 1$ and $\mu(\bigcup_{i=1}^{\infty} E_i) = \sum_{i=1}^{\infty} \mu(E_i)$ for any sequence $E_1, E_2, \ldots$ of pairwise disjoint sets from $\mathfrak{B}$. We call μ a *probability measure*. An event $E \in \mathfrak{B}$ is said to occur *almost surely* if $\mu(E) = 1$.

A scheduler u of a CPP $\mathcal{M} = (Q, N, R, \Delta, P, Q^{in})$ over an alphabet Σ induces a probability space $\mathcal{PS}_{\mathcal{M},u} = (\Omega_{\mathcal{M},u}, \mathfrak{B}_{\mathcal{M},u}, \mu_{\mathcal{M},u})$ as follows:

– the sample space consists all infinite and maximal finite sequences of states and actions (including $\mathfrak{p}$) that respect the transition relation and can be stepwise taken with non-zero probability. Furthermore, the nondeterministic transitions must follow the scheduler. We only consider a finite sequence when it ends in a state with no outgoing transitions: Let $\Omega_{\mathcal{M},u} = \{q_1 a_1 q_2 a_2 \ldots \in (Q\Sigma_{\mathfrak{p}})^{\omega} \mid q_1 \in Q^{in}$ and, for all $i \geq 1$, ($q_i \in R$ and $q_{i+1} \in N$ and $a_i = \mathfrak{p}$ and $P(q_i, q_{i+1}) > 0$) or ($q_i \in N^{en}_{\mathcal{M}}$ and $a_i \in \Sigma$ and $u(q_1 a_1 \ldots q_i) = (a_i, q_{i+1})$)$\} \cup \{q_1 a_1 \ldots q_{n-1} a_{n-1} q_n \in (Q\Sigma_{\mathfrak{p}})^*(N \setminus N^{en}_{\mathcal{M}}) \mid q_1 \in Q^{in}$ and, for all $i \in \{1, \ldots, n-1\}$, ($q_i \in R$ and $q_{i+1} \in N$ and $a_i = \mathfrak{p}$ and $P(q_i, q_{i+1}) > 0$) or ($q_i \in N^{en}_{\mathcal{M}}$ and $u(q_1 a_1 \ldots q_i) = (a_i, q_{i+1})$)$\}$ be the set of *trajectories* of $\mathcal{M}$ wrt. u,
– $\mathfrak{B}_{\mathcal{M},u} = \langle \{\mathcal{C}_{\mathcal{M},u}(x) \mid x \in (Q\Sigma_{\mathfrak{p}})^* N\} \rangle$ where $\mathcal{C}_{\mathcal{M},u}(x) = \{x' \in \Omega_{\mathcal{M},u} \mid x$ is a prefix of $x'\}$ is the *basic cylinder set* of x wrt. $\mathcal{M}$ and u, and
– $\mu_{\mathcal{M},u}$ is uniquely given by $\mu_{\mathcal{M},u}(\Omega_{\mathcal{M},u}) = 1$ and, for $n \geq 1$,

$$\mu_{\mathcal{M},u}(\mathcal{C}_{\mathcal{M},u}(q_1 a_1 \ldots q_{n-1} a_{n-1} q_n)) = P'(q_1, q_2) \cdot \ldots \cdot P'(q_{n-1}, q_n)$$

where

$$P'(q_i, q_{i+1}) = \begin{cases} P(q_i, q_{i+1}) & \text{if } q_i \in R \\ 1 & \text{if } q_i \in N \end{cases}$$

4 Product LTL

In this section, we recall the definition of a product version of linear temporal logic, denoted by LTL$^{\otimes}$, as defined in [MD97]. It is based on Pnueli's LTL. However, since we deal with action-based systems, we provide an action-based version of the logic as well. Furthermore, we extend the definition in a straightforward manner to deal with finite and infinite sequences.

Let Σ be an alphabet. The set LTL(Σ) of Linear Temporal Logic (LTL) formulas over Σ is given by the following grammar:

$$\varphi ::= \mathrm{tt} \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \langle a \rangle \varphi \ (a \in \Sigma) \mid \varphi \mathcal{U} \psi$$

An LTL(Σ) formula is inductively interpreted over $\sigma \in \Sigma^{\infty}$ as follows:

- $\sigma \models_{\Sigma} \mathrm{tt}$
- $\sigma \models_{\Sigma} \neg\varphi$ if $\sigma \not\models_{\Sigma} \varphi$
- $\sigma \models_{\Sigma} \varphi \vee \psi$ if $\sigma \models_{\Sigma} \varphi$ or $\sigma \models_{\Sigma} \psi$
- $\sigma \models_{\Sigma} \langle a \rangle \varphi$ if $\sigma \neq \varepsilon$, $\sigma(1) = a$, and $\sigma^2 \models_{\Sigma} \varphi$
- $\sigma \models_{\Sigma} \varphi \mathcal{U} \psi$ if there is an $i \geq 0$ such that $\sigma^i \models_{\Sigma} \psi$ and, for each $j \in \{1, \ldots, i-1\}$, $\sigma^j \models_{\Sigma} \varphi$

The language of a formula $\varphi \in$ LTL(Σ), denoted by $L(\varphi)$, is defined to be the set $\{\sigma \in \Sigma^{\infty} \mid \sigma \models_{\Sigma} \varphi\}$. Note that one can construct an extended Büchi automaton over Σ such that $L(\mathcal{B}) = L(\varphi)$ [VW86].

Product LTL formulas are Boolean combinations of LTL formulas, each formulated for a single process. More specifically, the set LTL$^{\otimes}(\tilde{\Sigma})$ of Product Linear Temporal Logic (LTL$^{\otimes}$) formulas over $\tilde{\Sigma}$ is given as:

$$\varphi ::= [\psi_i]_i \ (\psi_i \in \mathrm{LTL}(\Sigma_i)) \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2$$

An LTL$^{\otimes}(\tilde{\Sigma})$ formula is inductively interpreted over $\sigma \in (\bigcup_i \Sigma_i)^{\infty}$ as follows:

- $\sigma \models_{\otimes} [\psi_i]_i$ if $\sigma \downharpoonright \Sigma_i \models_{\Sigma_i} \psi_i$
- $\sigma \models_{\otimes} \varphi \vee \psi$ if $\sigma \models_{\otimes} \varphi$ or $\sigma \models_{\otimes} \psi$
- $\sigma \models_{\otimes} \varphi \wedge \psi$ if $\sigma \models_{\otimes} \varphi$ and $\sigma \models_{\otimes} \psi$

Thus, given a system, we restrict the run to the actions "interesting" for the process i and take the usual semantics of the LTL formula.

Note that we did not introduce negation on the outer LTL$^{\otimes}$ level since it can be "pushed inwards" to each single LTL(Σ_i) formula.

One might be tempted to understand an LTL$^{\otimes}$ formula as an LTL formula over the alphabet $\Sigma = \bigcup_i \Sigma_i$ (abstracting away $[]_i$). But it is easy to see that this yields a different semantics.

The language of a formula $\varphi \in$ LTL$^{\otimes}(\tilde{\Sigma})$, denoted by $L(\varphi)$, is defined to be the set $\{\sigma \in (\bigcup_i \Sigma_i)^{\infty} \mid \sigma \models_{\otimes} \varphi\}$. According to the LTL case, we can construct a PPA $\mathcal{A}$ over $\tilde{\Sigma}$ such that $L(\mathcal{A}) = L(\varphi)$ and, from $\mathcal{A}$, build an extended Büchi automaton $\mathcal{B}$ over $\bigcup_i \Sigma_i$ with $L(\mathcal{B}) = L(\mathcal{A}) = L(\varphi)$ [MD97].

5 Probabilistic Model Checking

In this section, we clarify the notion when a PPA satisfies an LTL$^{\otimes}$ formula in a probabilistic sense. Furthermore, we provide an algorithm answering this question.

5.1 Satisfaction

For checking whether a PPA satisfies a formula, we first define the set of sequences following a given scheduler and satisfying a given formula or being accepted by an automaton (let Σ be an alphabet):

1. For a CPP $\mathcal{M}$ over Σ, a scheduler u of $\mathcal{M}$, and a formula $\varphi \in \mathrm{LTL}(\Sigma)$, let $L_{\mathcal{M},u}(\varphi) := \{x \in \Omega_{\mathcal{M},u} \mid x \downharpoonleft \Sigma \models_{\Sigma} \varphi\}$.
2. For a PPA $\mathcal{A}$ over $\tilde{\Sigma}$, a scheduler u of $\mathcal{M}_{\mathcal{A}}$, and a formula $\varphi \in \mathrm{LTL}^{\otimes}(\tilde{\Sigma})$, let $L_{\mathcal{A},u}(\varphi) := \{x \in \Omega_{\mathcal{M}_{\mathcal{A}},u} \mid x \downharpoonleft (\bigcup_i \Sigma_i) \models_{\otimes} \varphi\}$.
3. For a CPP $\mathcal{M}$ over Σ, a scheduler u of $\mathcal{M}$, and an extended Streett (Büchi) automaton $\mathcal{B}$ over Σ, let $L_{\mathcal{M},u}(\mathcal{B}) := \{x \in \Omega_{\mathcal{M},u} \mid x \downharpoonleft \Sigma \in L(\mathcal{B})\}$.

We can show, using a simple induction, that these sets are measurable:

Proposition 1. *Let Σ be an alphabet.*

1. *Given a CPP $\mathcal{M}$ over Σ and a formula $\varphi \in \mathrm{LTL}(\Sigma)$, we have $L_{\mathcal{M},u}(\varphi) \in \mathfrak{B}_{\mathcal{M},u}$ for each scheduler u of $\mathcal{M}$.*
2. *Given a PPA $\mathcal{A}$ over $\tilde{\Sigma}$ and a formula $\varphi \in \mathrm{LTL}^{\otimes}(\tilde{\Sigma})$, it holds $L_{\mathcal{A},u}(\varphi) \in \mathfrak{B}_{\mathcal{M}_{\mathcal{A}},u}$ for each scheduler u of $\mathcal{M}_{\mathcal{A}}$.*
3. *Given a CPP $\mathcal{M}$ over Σ and an extended Streett (Büchi) automaton $\mathcal{B}$ over Σ, we have $L_{\mathcal{M},u}(\mathcal{B}) \in \mathfrak{B}_{\mathcal{M},u}$ for each scheduler u of $\mathcal{M}$.*

We can now define the satisfaction relation for CPPs and LTL, PPA and LTL$^{\otimes}$, as well as CPPs and Streett automata.

Definition 3. *Let Σ be an alphabet.*

1. *A CPP $\mathcal{M}$ over Σ is said to* satisfy *a formula $\varphi \in \mathrm{LTL}(\Sigma)$ if, for all schedulers u of $\mathcal{M}$, $\mu_{\mathcal{M},u}(L_{\mathcal{M},u}(\varphi)) = 1$.*
2. *A PPA $\mathcal{A}$ over $\tilde{\Sigma}$ is said to* satisfy *a formula $\varphi \in \mathrm{LTL}^{\otimes}(\tilde{\Sigma})$ if, for all schedulers u of $\mathcal{M}_{\mathcal{A}}$, $\mu_{\mathcal{M}_{\mathcal{A}},u}(L_{\mathcal{A},u}(\varphi)) = 1$.*
3. *Given a CPP $\mathcal{M}$ over Σ and an extended Streett (Büchi) automaton $\mathcal{B}$ over Σ, $\mathcal{M}$ is said to* satisfy *$\mathcal{B}$ if, for all schedulers u of $\mathcal{M}$, $\mu_{\mathcal{M},u}(L_{\mathcal{M},u}(\mathcal{B})) = 1$.*

For example, the formula $[\Diamond\langle a\rangle\langle b\rangle\mathrm{tt}]_1 \wedge [\Diamond\langle c\rangle\langle b\rangle\mathrm{tt}]_2 \in \mathrm{LTL}^{\otimes}((\{a,b\},\{b,c\}))$ is satisfied by the PPA from Figure 1.

A logic that specifies properties of a product system should not differentiate between different linearizations of its parallel execution, a well-known requirement in the domain of Mazurkiewicz traces [Leu02]. Let us check that this is the case for our notion of satisfaction of PPA and LTL$^{\otimes}$.

For two words $\sigma, \sigma' \in (\bigcup_i \Sigma_i)^*$, we say that they are *equivalent* (with respect to $\tilde{\Sigma}$) and write $\sigma \sim \sigma'$ if $\sigma \downharpoonright \Sigma_i = \sigma' \downharpoonright \Sigma_i$ for all $i \in \mathit{Proc}$. In other words, two words are equivalent if they only differ in the ordering of independent actions. We say that two actions are *independent* if they are not member of a single Σ_i for one $i \in \mathit{Proc}$.

As already a simple product automaton, a PPA is robust with respect to the order in which independent actions are executed. To illustrate this, let us consider the PPA from Figure 1. Both components can independently execute the independent actions a and c whereupon a random move follows, respectively. Such independence is reflected in the global system (cf. Figure 2): Starting from the initial state (p_0, q_0), constituting ac as the order in which a and c are executed spans the same probability space wrt. all the possible nondeterministic successor states (p_0, q_0), (p_0, q_2), (p_2, q_0), and (p_2, q_2) as constituting ca.

5.2 The Algorithm

Model checking PPA against LTL$^{\otimes}$ formulas is the problem whether a given LTL$^{\otimes}$ formula is satisfied by a given PPA. As we will show in this subsection, we can reduce this problem to the question whether the language defined by a scheduler for a Streett automaton has positive measure with respect to a given CPP. Therefore, we study this problem first:

Let Σ be an alphabet, $\mathcal{M} = (Q, N, R, \Delta, P, Q^{in})$ be a CPP over Σ, and $\mathcal{B} = (S, S_0, \delta, F, \mathcal{G})$ be an extended Streett automaton over Σ. The *product* of $\mathcal{M}$ and $\mathcal{B}$, the CPP $\mathcal{M}_{\mathcal{M},\mathcal{B}} = (Q_{\mathcal{M},\mathcal{B}}, N_{\mathcal{M},\mathcal{B}}, R_{\mathcal{M},\mathcal{B}}, \Delta_{\mathcal{M},\mathcal{B}}, P_{\mathcal{M},\mathcal{B}}, Q^{in}_{\mathcal{M},\mathcal{B}}, F_{\mathcal{M},\mathcal{B}}, \mathcal{G}_{\mathcal{M},\mathcal{B}})$ (with acceptance condition) over Σ, is given as follows:

- $Q_{\mathcal{M},\mathcal{B}} = Q \times S$
- $N_{\mathcal{M},\mathcal{B}} = N \times S$
- $R_{\mathcal{M},\mathcal{B}} = R \times S$
- $((q, s), a, (q', s')) \in \Delta_{\mathcal{M},\mathcal{B}}$ if $(q, a, q') \in \Delta$ and $s' \in \delta(s, a)$
- $P_{\mathcal{M},\mathcal{B}}((q, s), (q', s')) = \begin{cases} P(q, q') & \text{if } s = s' \\ 0 & \text{if } s \neq s' \end{cases}$
- $Q^{in}_{\mathcal{M},\mathcal{B}} = Q^{in} \times S_0$
- $F_{\mathcal{M},\mathcal{B}} = (N \setminus N^{en}_{\mathcal{M}}) \times F$
- $\mathcal{G}_{\mathcal{M},\mathcal{B}} = \{((Q \times U), (Q \times V)) \mid (U, V) \in \mathcal{G}\}$

We want to mark some SCCs of $\mathcal{M}_{\mathcal{M},\mathcal{B}}$ to be good in some sense and call a set D of its states *accepting* if, for all pairs $(U, V) \in \mathcal{G}_{\mathcal{M},\mathcal{B}}$, $(q, s) \in D$ with $s \in U$ implies $(q', s') \in D$ for some q' and $s' \in V$. Otherwise, D is called *rejecting.* We say that a state (r, f) of a rejecting set D is *rejecting* if there is a pair $(U, V) \in \mathcal{G}$ such that $f \in U$ and D contains no state (q, s) with $s \in V$.

Theorem 1. *For a CPP $\mathcal{M} = (Q, N, R, \Delta, P, Q^{in})$ over an alphabet Σ and a deterministic extended Streett automaton $\mathcal{B} = (S, \{s_0\}, \delta, F, \mathcal{G})$ over Σ, there is a scheduler u of $\mathcal{M}$ with $\mu_{\mathcal{M},u}(L_{\mathcal{M},u}(\mathcal{B})) > 0$ iff*

– *there is a path in (the graph of) $\mathcal{M}_{\mathcal{M},\mathcal{B}}$ from an initial state to a final state from $F_{\mathcal{M},\mathcal{B}}$ or*
– *there is a set D of states of $\mathcal{M}_{\mathcal{M},\mathcal{B}}$ satisfying the following:*
  (1) $\mathcal{M}_{\mathcal{M},\mathcal{B}}[D]$ is nontrivial and strongly connected,
  (2) D is accepting and reachable from a state of $Q^{in} \times \{s_0\}$, and
  (3) for all transitions $((q,s),(q',s')) \in \Delta_{\mathcal{M},\mathcal{B}}$ with $(q,s) \in D$ and $(q',s') \notin D$, (q,s) is nondeterministic, i.e., it holds $(q,s) \in N_{\mathcal{M},\mathcal{B}}$ (or, equivalently, $q \in N$).

Proof. ($\Leftarrow$) Suppose there is a path $\beta \in (Q_{\mathcal{M},\mathcal{B}})^*$ through $\mathcal{M}_{\mathcal{M},\mathcal{B}}$ from an initial state to a state (q,s) of $F_{\mathcal{M},\mathcal{B}}$. It is easy to see that then a corresponding scheduler (simply following the path) forces $\mathcal{M}_{\mathcal{M},\mathcal{B}}$ to visit (q,s) with nonzero probability. Otherwise, fix a path $\beta \in (Q_{\mathcal{M},\mathcal{B}})^*$ through $\mathcal{M}_{\mathcal{M},\mathcal{B}}$ from an initial state to a state of D. Let β' be the projection of β onto the first component. The scheduler u of $\mathcal{M}'$ satisfying $\mu_{\mathcal{M},u}(L_{\mathcal{M},u}(\mathcal{B})) > 0$ follows β' taking $\mathcal{M}_{\mathcal{M},\mathcal{B}}$ from the initial state to D and, henceforth, forces the trajectory both to stay within D and to almost surely visit each state of D infinitely often. This can be accomplished by, for a given nondeterministic state (q,s), alternately choosing the transitions $(q,s) \xrightarrow{a} (q',s')$ of $\mathcal{M}_{\mathcal{M},\mathcal{B}}$ with $(q',s') \in D$ (recall that the history of a trajectory is at the scheduler's disposal.) Clearly, $\mu_{\mathcal{M},u}(\mathcal{C}_{\mathcal{M},u}(\beta'))$ is nonzero. Given $\mathcal{C}_{\mathcal{M},u}(\beta')$, the conditional probability that $\mathcal{M}$, wrt. u, follows a trajectory that visits each state of D infinitely often is one. As such a trajectory is contained in $L_{\mathcal{M},u}(\mathcal{B})$, we conclude $\mu_{\mathcal{M},u}(L_{\mathcal{M},u}(\mathcal{B})) > 0$.

($\Rightarrow$) Note that a trajectory x of $\mathcal{M}$ wrt. u unambiguously defines a path $\widetilde{x}$ through $\mathcal{M}_{\mathcal{M},\mathcal{B}}$ starting from an initial state. This is due to the fact that $\mathcal{B}$ is deterministic. Let $\mathcal{D}$ contain the subsets D of states of $\mathcal{M}_{\mathcal{M},\mathcal{B}}$ such that $\mathcal{M}_{\mathcal{M},\mathcal{B}}[D]$ is strongly connected. Furthermore, for $D \in \mathcal{D}$, let $E(D) := \{x \in \Omega_{\mathcal{M},u} \mid \mathit{inf}(\widetilde{x}) = D\}$. Now suppose that $\mu_{\mathcal{M},u}(L_{\mathcal{M},u}(\mathcal{B})) > 0$ for a scheduler u of $\mathcal{M}$. If u leads $\mathcal{M}_{\mathcal{M},\mathcal{B}}$ from an initial state into a final state from $F_{\mathcal{M},\mathcal{B}}$, we are done. Otherwise, as

$$L_{\mathcal{M},u}(\mathcal{B}) = \bigcup_{D \in \mathcal{D} \text{ is accepting}} E(D),$$

we can find an accepting set $D \in \mathcal{D}$ that satisfies $\mu_{\mathcal{M},u}(E(D)) > 0$. (Otherwise, the probability of the countable union $L_{\mathcal{M},u}(\mathcal{B})$ of events would be zero.) As D is the infinity set of at least one infinite path through $\mathcal{M}_{\mathcal{M},\mathcal{B}}$ starting from an initial state, it forms a nontrivial (strongly connected) subgraph of $\mathcal{M}_{\mathcal{M},\mathcal{B}}$, satisfying condition (1). Now suppose there is a transition $(q,s) \xrightarrow{\mathfrak{p}} (q',s')$ of $\mathcal{M}_{\mathcal{M},\mathcal{B}}$ with $(q,s) \in D$, $(q',s') \notin D$, and $q \in R$. As, for every trajectory $x \in E(D)$, $\widetilde{x}$ visits (q,s) infinitely often (and each time the probability to exit D is nonzero), it will almost surely leave D infinitely often so that we have $\mu_{\mathcal{M},u}(E(D)) = 0$ contradicting our assumption. It follows that D also satisfies condition (3) from Proposition 1, which concludes our proof. □

Note that, in the above proof, we explicitly make use of the fact that a trajectory of $\mathcal{M}$ determines exactly one corresponding run of $\mathcal{M}_{\mathcal{M},\mathcal{B}}$ starting from an initial state (recall that $\mathcal{B}$ is deterministic).

```
Given a PPA $\mathcal{A}$ over $\tilde{\Sigma}$ and a formula $\varphi \in \mathrm{LTL}^{\otimes}(\tilde{\Sigma})$.
Goal: Decide whether, for all schedulers $u$ of $\mathcal{M}_{\mathcal{A}}$, it holds
$\mu_{\mathcal{M}_{\mathcal{A}},u}(L_{\mathcal{A},u}(\varphi)) = 1$.
Solution:

 1. From $\mathcal{A}$, we construct the CPP $\mathcal{M}_{\mathcal{A}}$, and from $\varphi$, we construct
    the deterministic extended Streett automaton $\mathcal{B}_{\neg\varphi}$ with
    $L(\mathcal{B}_{\neg\varphi}) = L(\neg\varphi)$.
 2. Compute (the graph of) $\mathcal{M}_{\mathcal{M}_{\mathcal{A}},\mathcal{B}_{\neg\varphi}}$, remove those states that are
    not reachable from an initial state of $\mathcal{M}_{\mathcal{M}_{\mathcal{A}},\mathcal{B}_{\neg\varphi}}$, and let $G$ denote
    the resulting graph.
 3. Repeat
    (a) Determine the sets $\mathcal{AC}$ of nontrivial and accepting and $\mathcal{RC}$ of
        nontrivial and rejecting SCCs of $G$, respectively.
    (b) For each $C \in \mathcal{RC}$, remove the transitions going out from
        rejecting states.
    (c) For each $C \in \mathcal{AC}$, do the following:
         i. Find the set $H$ of states $(\overline{q}, s) \in C$ with randomizing $\overline{q}$
            from where there is a transition leaving $C$.
        ii. If $H$ is the empty set, then return "No". Otherwise,
            remove the states of $H$ and corresponding transitions
            from $G$.
    until $\mathcal{AC} \cup \mathcal{RC} = \emptyset$.
 4. Test whether a scheduler can force $\mathcal{M}_{\mathcal{M}_{\mathcal{A}},\mathcal{B}_{\neg\varphi}}$ from an initial
    state into a final state with probability greater than 0, i.e.,
    whether there is a path from an initial state of $\mathcal{M}_{\mathcal{M}_{\mathcal{A}},\mathcal{B}_{\neg\varphi}}$ to a
    state in $F_{\mathcal{M}_{\mathcal{A}},\mathcal{B}_{\neg\varphi}}$. If this is the case, return "No". Otherwise,
    return "Yes".
```

Table 1. Model checking LTL$^{\otimes}$ specifications of PPA

Based on Theorem 1, we now provide an algorithm that solves the model-checking problem for PPA, i.e., it decides for a given PPA $\mathcal{A}$ over $\tilde{\Sigma}$ and a formula $\varphi \in \mathrm{LTL}^{\otimes}(\tilde{\Sigma})$ whether, for all schedulers u of $\mathcal{M}_{\mathcal{A}}$, $\mu_{\mathcal{M}_{\mathcal{A}},u}(L_{\mathcal{A},u}(\varphi)) = 1$ (namely iff there is no scheduler u of $\mathcal{M}_{\mathcal{A}}$ such that $\mu_{\mathcal{M}_{\mathcal{A}},u}(L_{\mathcal{A},u}(\neg\varphi)) > 0$). The algorithm is shown in Table 1. In the first step, the given PPA is transformed into a CPP. The given formula is negated and translated into a product automaton accepting the models of the formula. This step is described in [MD97] and is omitted here. It is straightforward to translate a product automaton into an extended Büchi automaton, which again can be translated into a deterministic extended Streett automaton [GTW02]. In the second step, we combine the obtained CPP and Streett automaton into a single system. The characterization provided in Theorem 1 is used in items 3 and 4 to answer the model checking question. Obviously, the algorithm terminates. Furthermore, it returns the answer "No" iff there is a scheduler u of $\mathcal{M}_{\mathcal{A}}$ such that $\mu_{\mathcal{M}_{\mathcal{A}},u}(L_{\mathcal{M}_{\mathcal{A}},u}(\mathcal{B}_{\neg\varphi})) > 0$.

To simplify our presentation, we described the algorithm in a stepwise manner. It is clear that steps 1 and 2 can be done on demand by steps 3 and 4. Thus, we can get an on-the-fly procedure.

Furthermore, the algorithm can easily be adapted to answer the model-checking problem for LTL or Büchi-automata specifications. Only step 1 has to be adjusted to produce a Streett automaton for a given LTL or Büchi automaton.

Let us discuss the complexity of our algorithm. Starting from a Büchi automaton $\mathcal{B}$ with n states, construct an equivalent deterministic extended Streett automaton $\mathcal{B}'$ with $2^{O(n \log n)}$ states and $O(n)$ pairs in the acceptance component. Say our CPP $\mathcal{M}$ has m states. The number of states of $\mathcal{M}_{\mathcal{M},\mathcal{B}'}$ is not greater than $m \cdot 2^{O(n \log n)}$. Thus, steps (a), (b), and (c) are repeated at most $m \cdot 2^{O(n \log n)}$-times, respectively. Determining the SCCs of G can be done in time linear in the size of $\mathcal{M}_{\mathcal{M},\mathcal{B}'}$. Overall, the algorithm (modified for CPPs and Büchi automata) runs in time $O(m^3 \cdot 2^{O(n \log n)})$, i.e., it is quadratic in $|\mathcal{M}|$ and exponential in $|\mathcal{B}|$.

Proposition 2. *Given a CPP $\mathcal{M}$ and a Büchi automaton $\mathcal{B}$, it can be decided in time $O(|\mathcal{M}|^2 \cdot 2^{O(|\mathcal{B}|)})$ whether $\mu_{\mathcal{M},u}(L_{\mathcal{M},u}(\mathcal{B})) > 0$ for some scheduler u of $\mathcal{M}$.*

Translating an LTL$^{\otimes}$ formula into a product automaton is of exponential complexity with respect to the length of the formula. The product automaton gives rise to a Büchi automaton of same order.[2] Thus, translating an LTL$^{\otimes}$ formula into a deterministic extended Streett automaton is of double exponential complexity. Together with Proposition 2, we get

Theorem 2. *Given a PPA $\mathcal{A}$ and an* LTL$^{\otimes}$ *formula φ, checking whether $\mathcal{A}$ satisfies φ can be done in time polynomial in the size of $\mathcal{A}$ and double exponential in the size of φ.*

6 Conclusion and Future Work

In this paper, we presented probabilistic product automata (PPA) as a model for distributed probabilistic programs. It is based on the well-known model of product automata, but extended by random transitions. Thus, a probabilistic product automaton is a product of probabilistic systems that run in parallel and synchronize by common actions. Every probabilistic system is able to do labelled nondeterministic and as well as randomized transitions.

For the product version of linear temporal logic LTL$^{\otimes}$, originally defined for product automata, we extended the notion of satisfaction to the probabilistic setting. Intuitively, we say that a PPA satisfies an LTL$^{\otimes}$ formula, if for every

[2] Note that a Büchi automaton corresponding to a product automaton grows exponentially in the number of components. Fixing the number of components, however, it grows polynomially with respect to the size of the components.

scheduler (that fixes the nondeterministic choices of the system) almost all runs satisfy the given formula.

The main contribution of the paper is a procedure that automatically answers the question whether a given PPA satisfies a given LTL$^{\otimes}$ formula. This problem is also known as the *model-checking problem*.

Our procedure is automata-based and can be implemented *on-the-fly*, which often provides good run-time behavior in practice, despite high worst-case complexity.

Additionally, we get a procedure for checking automata specifications and LTL specifications of PPA.

It would be interesting to extend our work to the setting of fair executions of the product system and, more specifically, while we check satisfaction for all schedulers, to see whether the restriction to fair schedulers gives different results.

Furthermore, it would be interesting to see whether techniques as used in [CY95] can improve our procedure to get single exponential complexity with respect to the length of the formula.

References

[BGJ99] J. Beauquier, M. Gradinariu, and C. Johnen. Randomized self-stabilizing and space optimal leader election under arbitrary scheduler on rings. Technical Report 99-1225, Université Paris Sud, 1999.

[CES83] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications: A practical approach. In *Conference Record of the Tenth Annual ACM Symposium on Principles of Programming Languages*, pages 117–126, Austin, Texas, January 24–26, 1983. ACM SIGACT-SIGPLAN.

[CW96] Edmund M. Clarke and Jeanette M. Wing. Formal methods: State of the art and future directions. *ACM Computing Surveys*, 28(4):626–643, December 1996.

[CY95] Costas Courcoubetis and Mihalis Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, July 1995.

[GTW02] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics and Infinite Games*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

[HS84] S. Hart and M. Sharir. Probabilistic temporal logics for finite and bounded models. In *ACM Symposium on Theory of Computing (STOC '84)*, pages 1–13, Baltimore, USA, April 1984. ACM Press.

[Leu02] Martin Leucker. *Logics for Mazurkiewicz traces.* PhD thesis, Lehrstuhl für Informatik II, RWTH Aachen, 2002.

[LR81] D. Lehman and M. O. Rabin. On the advantage of free choice: A fully symmetric and fully distributed solution to the dining philosophers problem. In *Proceedings of 10th ACM Symposium of Principles of Programming Languages*, pages 133–138, Williamsburg, 1981.

[LS82] Daniel Lehmann and Saharon Shelah. Reasoning with time and chance. *Information and Control*, 53(3):165–198, June 1982.

[MD97] P. Madhusudan and Deepak D'Souza. On-the-fly verification of Product-LTL. In *Proocedings of the National Seminar on Theoretical Computer Science*, Madras, June 1997.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science (FOCS-77)*, pages 46–57, Providence, Rhode Island, October 31–November 2 1977. IEEE Computer Society Press.

[PRRR01] P. Pardalos, S. Rajasekaran, J. Reif, and J. Rolim, editors. *Handbook on Randomized Computing.* Kluwer Academic Publishers, Dordrecht, The Netherlands, June 2001.

[QS82] J.P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Proceedings of the Fifth International Symposium in Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351, New York, 1982. Springer.

[Thi95] P. S. Thiagarajan. PTL over product state spaces. Technical Report TCS-95-4, School of Mathematics, SPIC Science Foundation, 1995.

[Var85] Moshe Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *26th Annual Symposium on Foundations of Computer Science*, pages 327–338, Portland, Oregon, 21–23 October 1985. IEEE.

[VW86] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Symposium on Logic in Computer Science (LICS'86)*, pages 332–345, Washington, D.C., USA, June 1986. IEEE Computer Society Press.