Alcom-FT Technical Report Series
ALCOMFT-TR-02-137

Formal Modeling and Analysis of Power-Aware Real-Time Systems*

Insup Lee[1], Anna Philippou[2], and Oleg Sokolsky[3]

[1]University of Pennsylvania, USA. lee@central.cis.upenn.edu
[2]University of Cyprus, Cyprus. annap@ucy.ac.cy
[1]University of Pennsylvania, USA. sokolsky@saul.cis.upenn.edu

Abstract — **The paper describes a unified formal framework for designing and reasoning about power-constrained, timed systems. The framework is based on *process algebra*, a formalism which has been developed to describe and analyze communicating, concurrent systems. The proposed extension allows the modeling of probabilistic resource failures, priorities of resource usages, and power consumption by resources within the same formalism. Thus, it is possible to study several alternative power-consumption behaviors and tradeoffs in their timing and other characteristics. This paper describes the modeling and analysis techniques, and illustrates them with examples, including a dynamic voltage-scaling algorithm.**

1 Introduction

In recent years, there have been great technological advances in wireless communication and mobile computing. These advances have given rise to sophisticated embedded devices (e.g., PDA, cell phones, smart sensors) and wireless network infrastructures that are becoming widespreadly available. In addition, new applications with powerful functionalities are being developed to meet the ever-increasing demand by the users. A serious limitation of the mobile embedded devices is the battery life available to them. Although a great deal of power-intensive computation has to be performed to carry out application-specific functionalities such as video streaming, this has to be done on a limited amount of power. To cope with this fact, a number of power-aware algorithms and protocols have been proposed aiming to make energy savings by dynamically altering the power consumed by a processor while still achieving the required behavior. However, in time-constrained applications often found in embedded systems, applying power-saving techniques can lead to serious problems. This is because changing the power available to tasks can affect their execution time which may lead to violation of their timing constraints and other desirable properties. A challenge presented by such systems is the development of robust algorithms that incorporate power-saving techniques and task management without sacrificing timing and performance guarantees. An example of such a proposal can be found in [14].

*This research was supported in part by NSF CCR-9988409, NSF CCR-0086147, NSF CISE-9703220, ARO DAAD19-01-1-0473, and by the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

The main purpose of this paper is to provide a unified formal framework for designing and reasoning about power-constrained, timed systems. The framework we propose is based on *process algebra* a formalism which has been developed to describe and analyze communicating, concurrently-executing systems. The most salient aspect of process algebras is that they support the *modular* specification and verification of systems and they enable the verification of a whole system by reasoning about its parts. Process algebras are being used widely in specifying and verifying concurrent systems and they have been extended to take account of phenomena such as time and probabilistic behavior.

In this paper we present the process algebra $\mathrm{P^2ACSR}$ which extends our previous work on formal methods for real-time [11] and probabilistic systems [13] by incorporating the ability of reasoning about power consumption. The Algebra of Communicating Shared Resources (ACSR) [11] is a timed process algebra which represents a real-time system as a collection of concurrent processes. Each process can engage in two kinds of activities: communication with other processes by means of instantaneous *events* and computation by means of timed *actions*. Executing an action requires access to a set of resources and takes a non-zero amount of time measured by an implicit global clock. Resources are serially reusable. The notion of a resource, which is already important in the specification of real-time systems, additionally provides a convenient abstraction mechanism for capturing a variety of aspects of systems behavior. One such aspect is the failure of physical devices: in a probabilistic extension of ACSR, PACSR [13], resources are extended with the ability to fail, and are associated with a probability of failure. In $\mathrm{P^2ACSR}$, the resource model of PACSR is further extended to reason about power consumption. Resources in $\mathrm{P^2ACSR}$ specifications are accompanied with information about the power consumption of each resource use. Thus, for each execution step requiring access to a set of power-consuming resources, we can compute the power consumed by the step and similarly for a sequence of steps.

We are interested in being able to specify and verify high-level requirements of $\mathrm{P^2ACSR}$ specifications, and, in particular, to study their power-consumption behavior and tradeoffs in their timing and other characteristics. To do this we provide an operational semantics of $\mathrm{P^2ACSR}$ via *labeled concurrent Markov chains* [17], which are transition systems containing both probabilistic and nondeterministic behavior. Probabilistic behavior is present in the model due to resource failure and nondeterministic behavior due to the fact that $\mathrm{P^2ACSR}$ specifications typically consist of several parallel processes producing events concurrently.

We describe analysis methods that can be carried out on this model by extending model-checking techniques to allow reasoning about power consumption properties. Temporal logics are commonly used to expressed high-level requirements of systems including probabilistic criteria that apply to fragments of systems executions. We present a probabilistic, power-aware, temporal logic for expressing properties of $\mathrm{P^2ACSR}$ expressions. In particular, we associate power-consumption constraints with temporal operators and thus express power-consumption constraints to complex execution fragments. For example, given a communication protocol in which a sender inquires about the readiness of a receiver and, after obtaining an acknowledgement, sends data, a useful property to check would be that this exchange happens correctly without consuming more than an acceptable level of power of the sender's processing unit. To further enable power-consumption calculations, we also propose analysis of systems via computing bounds on a system's power consumption. We present model-checking algorithms both for the logic and for bound computations. We illustrate the usefulness of the proposed formalism for two examples including a dynamic voltage-scaling algorithm for real-time, power-aware systems [14]. In the latter example, we use resources to model the power-consuming processing unit which can be used at different power levels on different occasions. Furthermore, we account for the probabilistic nature of task duration in the system by employing probabilistic resources.

The rest of the paper is organized as follows: the next two sections present the P^{2}ACSR syntax and semantics. Section 4 describes analysis techniques for P^{2}ACSR processes, and Section 5 presents the case study in which a power-aware real-time scheduling algorithm is modeled and analyzed. We conclude with some final remarks and discussion of further work.

2 The general framework

We assume that a system contains a finite set of serially reusable resources drawn from a countably infinite set of resources $\mathcal{R}$. Resources can correspond to physical entities, such as processor units and communication channels, or to abstract notions such as message arrival. They can have a set of numerical attributes that let us capture quantitative aspects of resource-constrained computations. In general, resource attributes can be associated to the resource itself, or separately to each resource use. In P^{2}ACSR we have three resource attributes: As in PACSR, one attribute captures the probability of resource failure, which remains constant throughout a system specification, and another attribute represents action priority, which may be different in different uses of the resource. In addition, we employ a third attribute corresponding to power consumption, which, like the priority attribute, may be different in each resource use.

Probabilistic resource failures. As in PACSR, we associate with each resource a probability. This probability captures the rate at which the resource may fail. A failure may correspond to either a physical failure, such as a message loss in a communication channel, or a failure of some abstract condition, for example no message arrival when one was expected. We assume that in each execution step, resources fail independently. To capture the notion of a failed resource we also consider the set $\overline{\mathcal{R}}$ that contains for each $r \in \mathcal{R}$, its dual element $\overline{r}$, representing the *failed* resource r and write R for $\overline{\mathcal{R}} \cup \mathcal{R}$. Thus, for all $r \in \mathcal{R}$ we denote by $\pi(r) \in [0, 1]$ the probability of resource r being up in a given step, while $\pi(\overline{r}) = 1 - \pi(r)$ denotes the probability of r failing in a given step. The use of failed resources is useful when we need to specify recovery from failures.

Resources and power consumption. In order to reason about power consumption in distributed settings, the set of resources $\mathcal{R}$ is partitioned into a finite set of disjoint classes $\mathcal{R}_i$, for some index set I. Intuitively, each $\mathcal{R}_i$ corresponds to a distinct power source which can provide a limited amount of power at any given time. This limit is denoted by c_i. Each resource $r \in \mathcal{R}_i$ consumes a certain amount of power from the source $\mathcal{R}_i$. As we will see below, the rate of power consumption is specified in *timed actions*.

2.1 The Syntax

As PACSR, P^{2}ACSR has three types of actions: instantaneous events, timed actions, and probabilistic actions. We discuss these three concepts below.

Instantaneous Events. Instantaneous actions are called *events*. Events provide the basic synchronization primitives in the process algebra. An event is denoted as a pair (a, p) where a is the label of the event and p, a natural number, is the priority of the event. Labels are drawn from the set $\mathsf{L} = \mathcal{L} \cup \overline{\mathcal{L}} \cup \{\tau\}$, where if $a \in \mathcal{L}$, $\overline{a} \in \overline{\mathcal{L}}$ is its *inverse* label. The special label τ, also called the silent action, arises when two events with inverse labels are executed concurrently. We let a, b range over labels. Further, we use $\mathcal{D}_E$ to denote the domain of events.

Timed actions A timed action consists of several resources, each resource being used at some priority and at some level of power consumption, and consumes one unit of time. Formally, an action is a finite set of triples of the form (r, p, c), where r is a resource, p is the priority of the resource usage and c is the rate of power consumption, with the restriction that each resource is represented at most once.

An example of an action is given by $\{(cpu, 2, 3),(msg, 1, 0)\}$. This action, takes one unit of time, and uses resource cpu representing a processor unit, at priority level two, consuming three units of power. The processor can fail with probability $\pi(\overline{cpu})$. This action also assumes that the processor receives a message, represented by resource msg. The fact that the message may or may not arrive is modeled as a failure of resource msg. This is not a physical failure, but rather a failed assumption. The action takes place assuming that none of the resources cpu and msg fail. On the other hand, action $\{(cpu, 2, 3), (\overline{msg}, 1, 0)\}$ takes place given that resource msg fails and resource cpu does not. The action $\emptyset$ represents idling for one unit of time, since no resource is consumed.

We let A, B range over timed actions and $\mathcal{D}_R$ to denote the domain of actions. We denote the set of resources used in an action A as $\rho(A)$. We also denote the priority level and the power consumption level of resource r in action A as $\mathsf{pr}_r(A)$ and $\mathsf{pc}_r(A)$, respectively.

Probabilistic transitions. As already mentioned resources are associated with a probability of failure. Thus, the behavior of a resource-consuming system has certain probabilistic aspects to it which are reflected in the operational semantics of the algebra. For example consider action $\{(cpu, 2, 3), (msg, 1, 0)\}$, where resources cpu and msg have probabilities of failure 0 and 1/3, respectively, that is $\pi(cpu) = 1$ and $\pi(msg) = 2/3$. Then the action takes place with probability $\pi(cpu) \cdot \pi(msg) = 2/3$ and fails with probability 1/3.

Processes We let P, Q range over processes and we assume a set of process constants each with an associated definition of the kind $X \stackrel{\text{def}}{=} P$. The following grammar describes the syntax of P²ACSR processes.

$$P \quad ::= \quad \mathrm{NIL} \mid (a, n).\, P \mid A{:}P \mid b \rightarrow P \mid P + P \mid P \| P \mid P \backslash F \mid [P]_I \mid rec\ X.P \mid X$$

Process NIL represents the inactive process. There are two prefix operators, corresponding to the two types of actions. The first, $(a, n).\, P$, executes the instantaneous event (a, n) and proceeds to P. When it is not relevant for the discussion we omit the priority of an event in a process and simply write $a.\, P$. The second, $A : P$, executes a resource-consuming action A during the first time unit and proceeds to P. As we will specify precisely when we give the semantics of the language, an action can take place if none of the resources used by it fail and assuming that it does not violate the system's power constraints. Otherwise, $A : P$ cannot execute the action and behaves as NIL. Process $b \rightarrow P$

behaves as P if condition b is true, otherwise it behaves as NIL. Process $P+Q$ represents a nondeterministic choice between the two summands. Process $P\|Q$ describes the concurrent composition of P and Q: the component processes may proceed independently or interact with one another while executing events, and they synchronize on timed actions. In $P\backslash F$, where $F \subseteq L$, the scope of labels in F is restricted to process P: components of P may use these labels to interact with one another but not with P's environment. The construct $[P]_I$, $I \subseteq \mathsf{R}$, produces a process that reserves the use of resources in I for itself, extending every action A in P with resources in $I - \rho(A)$ at priority 0 and power consumption 0. Finally, the process $rec\ X.P$ denotes standard recursion. We write Proc for the set of P^{2}ACSR processes.

As an example of a process, consider the process

$$P \stackrel{\text{def}}{=} \{(cpu, 2, 3), (msg, 1, 0)\} : P_1 + \{(cpu, 2, 2), (\overline{msg}, 1, 0)\} : P_2 \ .$$

Process P represents a processor that can accept messages from a channel. We assume that reading the message from the channel requires additional power. Depending on whether the message arrives or not, P has two alternative behaviors. If the message arrives, that is, resource msg is up, the processor receives the message, consuming 3 units of power, and proceeds to process it as P_1. Otherwise, if the message does not arrive, msg is down and that action cannot proceed. This is specified as $\overline{msg}$ is up, and the processor consumes only 2 units of power and continues as P_2.

As a syntactic convenience, we allow P^{2}ACSR processes to be parametrized by a set of index variables. Each index variable is given a fixed range of values. This restricted notion of parameterization allows us to represent collections of similar processes concisely. For example, the parameterized process

$$P_t \stackrel{\text{def}}{=} t < 2 \rightarrow (a_t, p_t).P_{t+1}, \quad t \in \{0..2\}$$

is equivalent to the following three processes:

$$P_0 \stackrel{\text{def}}{=} (a_0, p_0).P_1, \quad P_1 \stackrel{\text{def}}{=} (a_1, p_1).P_2, \quad P_2 \stackrel{\text{def}}{=} \text{NIL}$$

The informal account of behavior of the algebra's constructs, just given is made precise via a family of rules that define the labeled transition relations $\longrightarrow_n$ and $\longrightarrow_p$ on processes. This is presented in the next section. First we have some useful definitions. The function $\mathsf{imr}(P)$, defined inductively below, associates each P^{2}ACSR process with the set of resources on which its behavior immediately depends:

$\mathsf{imr}(\text{NIL}) = \emptyset$
$\mathsf{imr}((a, n).\, P) = \emptyset$
$\mathsf{imr}(A : P) = \rho(A)$
$\mathsf{imr}(b \rightarrow P) = \mathsf{imr}(P)$
$\mathsf{imr}(P_1 + P_2) = \mathsf{imr}(P_1) \cup \mathsf{imr}(P_2)$
$\mathsf{imr}(P_1 \| P_2) = \mathsf{imr}(P_1) \cup \mathsf{imr}(P_2)$
$\mathsf{imr}([P]_I) = \mathsf{imr}(P) \cup I$
$\mathsf{imr}(P\backslash F) = \mathsf{imr}(P)$
$\mathsf{imr}(rec\ X.P) = \mathsf{imr}(P)$

3 Operational Semantics

The semantics of the process algebra is given in two steps. At the first level, a transition system is given that captures the nondeterministic and probabilistic behavior of processes, ignoring the presence of priorities. Subsequently this is refined via a second transition system which takes action priorities into account.

We begin with the unprioritized semantics. As for PACSR, it is based on the notion of a *world*, which keeps information about the state of the resources of a process. A *world* is any subset of set R with the restriction that it contains information about each resource at most once. Related to the concept of worlds we have the following definition:

Definition 3.1 Let $Z = \{c_1, \ldots, c_n\} \subseteq \mathsf{R}$. We write

- $\mathcal{W}(Z) = \{Z' \subseteq Z \cup \overline{Z} \mid x \in Z' \text{ iff } \overline{x} \notin Z'\}$, for the set of all worlds involving resources Z,
- $\pi(Z) = \Pi_{1 \le i \le n} \pi(c_i)$, for the probability of worlds Z,
- $res(Z) = \{r \in \mathcal{R} \mid r \in Z \text{ or } \overline{r} \in Z\}$, for the resources of set Z. □

Behavior of a given process P can be given only with respect to the world P is in. A *configuration* is a pair of the form $(P, W) \in \mathsf{Proc} \times 2^{\mathsf{R}}$, representing a P^{2}ACSR process P in world W. We write S for the set of all configurations. The semantics is given in terms of a labeled transition system whose states are configurations and whose transitions are either probabilistic or nondeterministic.

The intuition for the semantics is as follows: for a process P, we begin with the configuration $(P, \emptyset)$. As computation proceeds, probabilistic transitions are performed from probabilistic configurations to determine the status of resources immediately relevant for execution (as specified by $\mathsf{imr}(P)$) but for which there is no knowledge in the configuration's world. Once the status of a resource is determined by some probabilistic transition, it cannot change until the next timed action occurs. Once a timed action occurs, the state of resources has to be determined anew, since in each time unit resources can fail independently from any previous failures. Nondeterministic transitions (which can involve events or actions) are performed from nondeterministic configurations.

With this view of computation in mind, we partition S as follows:

$S_n = \{(P, W) \in S \mid res(\mathsf{imr}(P)) - res(W) = \emptyset\}$, the set of nondeterministic configurations, and
$S_p = \{(P, W) \in S \mid res(\mathsf{imr}(P)) - res(W) \neq \emptyset\}$, the set of probabilistic configurations.

Let $\longrightarrow_p \subset S_p \times [0,1] \times S_n$ be the probabilistic transition relation. A triple in $\longrightarrow_p$, written $(P, W) \xrightarrow{p}_p (P', W')$, denotes that process P in world W may become P' and enter world W' with probability p. Furthermore, let $\longrightarrow \subset S_n \times Act \times S$ be the nondeterministic transition relation where Act, the set of actions, is given by $\mathcal{D}_E \cup \mathcal{D}_R$. A triple in $\longrightarrow$ is written $(P, W) \xrightarrow{\alpha} (P', W')$, capturing that process P in world W may nondeterministically perform α and become (P', W').

The probabilistic transition relation is given by the following rule:

$$(\mathsf{PROB}) \quad \frac{(P, W) \in S_p, Z_1 = res(\mathsf{imr}(P)) - res(W), Z_2 \in \mathcal{W}(Z_1)}{(P, W) \xrightarrow{\pi(Z_2)}_p (P, W \cup Z_2)}$$

Thus, given a probabilistic configuration (P, W), with Z_1 the immediate resources of P for which the state is not yet determined in W, and $Z_2 \in \mathcal{W}(Z_1)$, P enters the world extended by Z_2 with probability $\pi(Z_2)$. Note that configuration (P, W) evolves into $(P, W \cup Z_2)$ which is, by definition, a nondeterministic configuration.

To illustrate the probabilistic transition relation, consider process

$$P \stackrel{\text{def}}{=} \{(r_1, 2, 1), (r_2, 2, 2)\} : P_1 + (e, 1).P_2$$

in the initial configuration $(P, \emptyset)$. The immediate resources of P are $\{r_1, r_2\}$. Since there is no knowledge in the configuration's world regarding these resources, the configuration belongs to the set of probabilistic configurations S_p, from where we have four probabilistic transitions that determine the states of r_1 and r_2:

$$(P, \emptyset) \xrightarrow{\pi(r_1)\cdot\pi(r_2)}_p (P, \{r_1, r_2\}), \qquad (P, \emptyset) \xrightarrow{\pi(r_1)\cdot\pi(\overline{r_2})}_p (P, \{r_1, \overline{r_2}\}),$$

$$(P, \emptyset) \xrightarrow{\pi(\overline{r_1})\cdot\pi(r_2)}_p (P, \{\overline{r_1}, r_2\}), \qquad \text{and} \qquad (P, \emptyset) \xrightarrow{\pi(\overline{r_1})\cdot\pi(\overline{r_2})}_p (P, \{\overline{r_1}, \overline{r_2}\}).$$

All of the resulting configurations are nondeterministic since they contain full information about P's immediate resources.

We have the following lemma.

Lemma 3.2 For all $s \in S_p$, $\Sigma\{\!|p \mid (s, p, s') \in \longrightarrow_p |\!\} = 1$, where $\{\!|$ and $|\!\}$ are multiset brackets and the summation over the empty multiset is 1. $\square$

The nondeterministic transition relation for configurations $(P, W) \in S_n$ is presented in Table 1. In this table and hereafter, we use α, β to range over $\mathcal{D}_E \cup \mathcal{D}_R$. Further, we use the predicate $\mathsf{valid}(A)$ to distinguish actions that do not violate their power consumption requirements. Specifically, $\mathsf{valid}(A) = \bigwedge_{i\in I}(\sum_{r\in\mathcal{R}_i} \mathsf{pc}_r(A) \leq c_i)$. Note that the symmetric versions of rules (Sum) and (Par1) have been omitted. The rules are the same as for PACSR except from the action prefix operator and the parallel composition operator. In both (Act2) and (Par3), for an action to take place, in addition to all other constraints, it must be valid according to the power requirements.

The first two rules define the semantics of the prefix operators. Note that for the timed action A to be performed by configuration $(A : P, B)$, it must be that all resources in A are available in the configuration's world and that it has valid power requirements. The next three rules are straightforward: rule (Sum) captures the behavior of the summation operator, and rules (Par1) and (Par2) the behavior of the parallel composition operator with respect to instantaneous actions: component processes may proceed independently or synchronize with one another. Rule (Par3) describes the behavior of the parallel operator with respect to timed actions: note that all processes in a parallel composition need to synchronize on a timed action, making timed transitions truly synchronous, in that a process only advances if both of its subprocesses take a step. The condition states that only one process may use a resource during any time step and that the resulting action is valid with respect to its power consumption. Rule (Res) establishes that the set of labels F is restricted from the interface with the environment. Rules (Cl1) and (Cl2) describe the behavior of the close operator. When a process is embedded in a closed context, such as $[P]_I$, we ensure that there is no further sharing of the resources $r \in I - \rho(A)$ by employing all of these resources at power consumption level 0 and priority level 0. Instantaneous events are not affected by the close operator. Rule (Rec) is the standard rule for recursion.

(Act1)	$((a,n).P,B) \xrightarrow{(a,n)} (P,B)$	(Act2)	$(A:P,B) \xrightarrow{A} (P,\emptyset)$, if $\rho(A) \subseteq B$, valid(A)
(Sum)	$\dfrac{(P_1,B) \xrightarrow{\alpha} (P,B')}{(P_1+P_2,B) \xrightarrow{\alpha} (P,B')}$	(Cond)	$\dfrac{(P,B) \xrightarrow{\alpha} (P',B')}{(true \rightarrow P,B) \xrightarrow{\alpha} (P',B')}$
(Par1)	$\dfrac{(P_1,B) \xrightarrow{(a,n)} (P_1',B)}{(P_1\|P_2,B) \xrightarrow{(a,n)} (P_1'\|P_2,B)}$	(Par2)	$\dfrac{(P_1,B) \xrightarrow{(a,n)} (P_2',B),\ \ (P_2,B) \xrightarrow{(\overline{a},n)} (P_2',B)}{(P_1\|P_2,B) \xrightarrow{(\tau,n+m)} (P_1'\|P_2',B)}$
(Par3)	$\dfrac{(P_1,B) \xrightarrow{A_1} (P_1',B'),\ (P_2,B) \xrightarrow{A_2} (P_2',B')}{(P_1\|P_2,B) \xrightarrow{A_1 \cup A_2} (P_1'\|P_2',B')}$,		$\rho(A_1) \cap \rho(A_2) = \emptyset$ and valid$(A_1 \cup A_2)$
(Cl1)	$\dfrac{(P,B) \xrightarrow{(a,n)} (P',B)}{([P]_I,B) \xrightarrow{(a,n)} ([P']_I,B)}$	(Cl2)	$\dfrac{(P,B) \xrightarrow{A_1} (P',B'), A_2 = \{(r,0,0) \mid r \in B \cap (I \cup \overline{I})\}}{([P]_I,B) \xrightarrow{A_1 \cup A_2} ([P']_I,B')}$
(Res)	$\dfrac{(P,B) \xrightarrow{(a,n)} (P',B),\ \ a \notin F}{(P \backslash F,B) \xrightarrow{(a,n)} (P' \backslash F,B)}$	(Rec)	$\dfrac{(P[rec\ X.P/X],B) \xrightarrow{\alpha} (P',B')}{(rec\ X.P,B) \xrightarrow{\alpha} (P',B')}$

Table 1: The nondeterministic relation

Returning to the previous example, the nondeterministic configuration $(P, \{r_1, r_2\})$, where $P \stackrel{\text{def}}{=} \{(r_1,2,1),(r_2,2,2)\} : P_1 + (e,1).P_2$ has two nondeterministic transitions:

$$(P, \{r_1, r_2\}) \xrightarrow{\{(r_1,2,1),(r_2,2,2)\}} (P_1, \emptyset) \qquad \text{and} \qquad (P, \{r_1, r_2\}) \xrightarrow{e} (P_2, \{r_1, r_2\}).$$

The other configurations $(P, \{r_1, \overline{r_2}\})$, $(P, \{\overline{r_1}, r_2\})$, and $(P, \{\overline{r_1}, \overline{r_2}\})$, allow only the e-labeled transition since either r_1 or r_2 is failed.

We now proceed to the prioritized transition system for P^{2}ACSR. The prioritized transition system is based on the notion of *preemption* and refines the nondeterministic transition relation $\longrightarrow$ by taking priorities into account. It is given by the pair of transition relations $\longrightarrow_p$ and $\longrightarrow_n$, the latter of which is defined below. The preemption relation $\prec$ on Act is defined as for ACSR, specifying when two actions are comparable with respect to priorities. For example, $\emptyset \prec A$ for all $A \in \mathcal{D}_R$, that is, the idle action $\emptyset$ is preemptable by all other timed actions, and $(a,p) \prec (a,p')$, whenever $p < p'$. For the precise definition of $\prec$ we refer to [11]. The basic idea behind $\longrightarrow_n$ is that a nondeterministic transition of the form $(P,W) \xrightarrow{\alpha}_n (P',W')$ is permitted if and only if there are no higher-priority transitions enabled in (P,W), that is, for all β enabled in (P,W) it is not the case that $\alpha \prec \beta$. Thus we have that the prioritized nondeterministic transition system is obtained from the unprioritized one by pruning away preemptable transitions.

Definition 3.3 The prioritized labeled transition system $\longrightarrow_n$ is defined as follows: $(P,W) \xrightarrow{\alpha}_n (P',W')$ if and only if (1) $(P,W) \xrightarrow{\alpha} (P',W')$ is an unprioritized nondeterministic transition, and (2) there is no unprioritized transition $(P,W) \xrightarrow{\beta} (P'',W'')$ such that $\alpha \prec \beta$. $\square$

As an example, consider process

$$Q \stackrel{\text{def}}{=} \{(r,2,3)\} : \text{NIL} + (e,1).\text{NIL} \ \ \| \ \ (e,2).\text{NIL} \ \ \| \ \ (e,3).\text{NIL}$$

in the initial configuration $(Q, \emptyset)$. The immediate resources of Q are $\{r\}$. Since there is no knowledge in the configuration's world regarding this resource, $(Q, \emptyset)$ is a probabilistic configuration from which we have two probabilistic transitions.

$$(Q, \emptyset) \xrightarrow{\pi(r)}_p (Q, \{r\}), \qquad\qquad (Q, \emptyset) \xrightarrow{\pi(\overline{r})}_p (Q, \{\overline{r}\}),$$

The resulting configurations are nondeterministic. Consider $(Q, \{r\})$. Although in the non-prioritized semantics

$$\begin{array}{lcl}
(Q,\ \{r\}) & \xrightarrow{\{(r,2,3)\}} & ((e,2).\mathrm{NIL}\ \|\ (e,3).\mathrm{NIL},\ \ \emptyset), \\
(Q,\ \{r\}) & \xrightarrow{(e,1)} & ((e,2).\mathrm{NIL}\ \|\ (e,3).\mathrm{NIL},\ \ \{r\}), \\
(Q,\ \{r\}) & \xrightarrow{(e,2)} & (\{(r,2,3)\} : \mathrm{NIL} + (e,1).\mathrm{NIL}\ \|\ (e,3).\mathrm{NIL},\ \ \{r\}), \\
(Q,\ \{r\}) & \xrightarrow{(e,3)} & (\{(r,2,3)\} : \mathrm{NIL} + (e,1).\mathrm{NIL}\ \|\ (e,2).\mathrm{NIL},\ \ \{r\}),
\end{array}$$

the prioritized transition relation allows only the first and last transitions, that is:

$$\begin{array}{lcl}
(Q,\ \{r\}) & \xrightarrow{\{(r,2,3)\}}_n & ((e,2).\mathrm{NIL}\ \|\ (e,3).\mathrm{NIL},\ \ \emptyset), \\
(Q,\ \{r\}) & \xrightarrow{(e,3)}_n & (\{(r,2,3)\} : \mathrm{NIL} + (e,1).\mathrm{NIL}\ \ \|\ \ (e,2).\mathrm{NIL},\ \ \{r\})
\end{array}$$

are the only allowable initial transitions from configuration $(Q, \{r\})$.

4 Analysis

In this section we discuss possible analysis that can be performed on P^2ACSR specifications.

We begin by presenting the formal model underlying P^2ACSR processes which is that of *labeled concurrent Markov chains* [17].

Definition 4.1 A *labeled concurrent Markov chain* (LCMC) is a tuple $\langle S_n, S_p, Act, \longrightarrow_n, \longrightarrow_p, s_0\rangle$, where S_n is the set of nondeterministic states, S_p is the set of probabilistic states, Act is the set of labels, $\longrightarrow_n \subset S_n \times Act \times (S_n \cup S_p)$ is the nondeterministic transition relation, $\longrightarrow_p \subset S_p \times (0,1] \times S_n$ is the probabilistic transition relation, satisfying $\Sigma_{(s,\pi,t)\in\longrightarrow_p} \pi = 1$ for all $s \in S_p$, and $s_0 \in S_n \cup S_p$ is the initial state. □

We may see that the operational semantics of P^2ACSR yields transition systems that are LCMCs where $Act = \mathcal{D}_E \cup \mathcal{D}_R$ and the sets S_n, S_p are the sets of nondeterministic and probabilistic configurations, respectively. In what follows, we let ℓ range over $Act \cup [0,1]$.

Computations of LCMCs arise by resolving the nondeterministic and probabilistic choices: a *computation* in $T = \langle S_n, S_p, Act, \longrightarrow_n, \longrightarrow_p, s_0\rangle$ is either a finite sequence $c = s_0\,\ell_1\,s_1 \ldots \ell_k\,s_k$, where s_k has no transitions, or an infinite sequence $c = s_0\,\ell_1\,s_1 \ldots \ell_k\,s_k \ldots$, such that $s_i \in S_n \cup S_p$, $\ell_{i+1} \in Act \cup [0,1]$ and $(s_i, \ell_{i+1}, s_{i+1}) \in \longrightarrow_p \cup \longrightarrow_n$, for all $0 \le i$. We denote by $\mathsf{comp}(T)$ the set of all computations of T and by $\mathsf{Pcomp}(T)$ the set of all partial computations of T, i.e. $\mathsf{Pcomp}(T) = \{s_0\ell_1 \ldots \ell_k s_k \mid \exists c \in \mathsf{comp}(T).\ c = s_0\ell_1 \ldots \ell_k s_k \ldots \text{ and } s_k \in S_n\}$. Given a computation $c = s_0\ell_1 \ldots \ell_k s_k$, we define

$$\mathsf{trace}\,(c) \quad = \quad \ell_1 \ldots \ell_k\!\restriction Act - \{\tau\},$$

$$
\begin{aligned}
\mathsf{states}\,(c) &= \{s_0, s_1, \ldots s_k\}\\
\mathsf{time}\,(c) &= \#(\ell_1 \ldots \ell_k \lceil \mathcal{D}_R)\\
\mathsf{init}\; c &= s_0 \ldots s_{k-1} \text{ and } \mathsf{last}\; c \;=\; s_k\\
\mathsf{pow}\,(\ell) &= \sum_{r \in \rho(\ell)} \mathsf{pc}_r(\ell)\\
\mathsf{pow}\,(\ell, R) &= \sum_{r \in \rho(\ell) \cap R} \mathsf{pc}_r(\ell)\\
\mathsf{power}\,(c) &= \sum_{1 \le i \le k, \ell_i \in \mathcal{D}_R} \mathsf{pow}\,(\ell_i),\\
\mathsf{power}\,(c, R) &= \sum_{1 \le i \le k, \ell_i \in \mathcal{D}_R} \mathsf{pow}\,(\ell_i, R)
\end{aligned}
$$

To define probability measures on computations of an LCMC the nondeterminism present must be resolved. To achieve this, the notion of a scheduler has been employed [17, 9, 15]. A scheduler is an entity that, given a partial computation ending in a nondeterministic state, chooses the next transition to be executed.

Definition 4.2 A *scheduler* of an LCMC T is a partial function $\sigma : \mathsf{Pcomp}(T) \mapsto \longrightarrow_n$, such that if $pc \in \mathsf{Pcomp}(T)$ and $\sigma(pc) = (s, \alpha, s')$, then $s = \mathsf{last}\, pc$. We use $\mathsf{Sched}(T)$ to denote the set of all schedulers of T. □

$\mathsf{Sched}(T)$ is potentially an infinite set. We let σ range over schedulers. For an LCMC T and a scheduler $\sigma \in \mathsf{Sched}(T)$ we define the set of *scheduled computations* $\mathsf{Scomp}(T, \sigma) \subseteq \mathsf{comp}(T)$ to be the computations $c = s_0\, \ell_1 \ldots \ell_k\, s_k \ldots$ such that for all $s_i \in S_n$, $\sigma(s_0 \ell_1 \ldots \ell_i s_i) = (s_i, \ell_{i+1}, s_{i+1})$.

Each scheduler induces a probability space [8] on the set of computations allowed by the scheduler, $\mathsf{Scomp}(T, \sigma)$, and allows us to compute the probability of computations of interest. In particular, we define $\mathsf{Scomp}_{fin}(T, \sigma)$ to be the set of all partial computations that are a prefix of some $c \in \mathsf{Scomp}(T, \sigma)$, and $\mathcal{A}^\sigma(T)$ to be the sigma-algebra generated by the basic cylinders $C(\omega) = \{c \in \mathsf{Scomp}(T, \sigma) \mid \omega \text{ is a prefix of } c\}$, where $\omega \in \mathsf{Scomp}_{fin}(T, \sigma)$. Then the probability measure $\mathcal{P}$ on $\mathcal{A}^\sigma(T)$ is the unique measure such that if $\omega = s_0 \ell_1 s_1 \ldots \ell_k s_k$ then $\mathcal{P}(C(\omega)) = \Pi\{\!|\ell_i \in [0, 1] \mid 1 \le i \le k|\!\}$, that is, it is the product of the probabilities arising along the path.

4.1 Model Checking for P²ACSR

The first technique we propose for analysing P²ACSR specifications is that of model-checking. Model checking is a verification technique aimed at determining whether a system specification satisfies a property typically expressed as a temporal logic formula. To allow model checking on P²ACSR specifications, in this section we introduce a probabilistic temporal logic that allows one to associate power consumption constraints with fragments of behaviors. Behavioral fragments of interest are expressed in terms of regular expressions over *Act*, the set of observable actions. The associated model-checking algorithm, also presented in this section, is used to check whether these constraints are satisfied and thus whether formulae of the logic are satisfied by system specifications.

We now introduce our logic for P²ACSR. This is an extension of our logic of [13], which, in turn, is based on the Hennessy-Milner Logic (HML) with *until* [7]. The extension established allows for quantitative analysis of power consumption properties of a system by associating a condition with the *until* operator. The condition takes the form of $\leq pc$ or $\geq pc$ for a constant pc. In this way we can express a property that an execution, timed or untimed, satisfies a power consumption constraint, with a certain probability (which may be equal to one). We also include a second construct that allows a similar type of reasoning but specifying the power sources for which analysis is to be performed.

Definition 4.3 *(Power-aware PHML with until)* The syntax of $\mathcal{L}^{pc}_{PHMLu}$ is defined by the following grammar, where f, f', range over $\mathcal{L}^{pc}_{PHMLu}$-formulae, Φ is a regular expression over Act, R is a subset of the set of resources $\mathcal{R}$, p a number in $[0,1]$ representing a probability, t a number representing a time limit, pc a number representing a power consumption, and $\bowtie \in \{\leq, <, \geq, >\}$:

$$f \;::=\; tt \;\;|\;\; \neg f \;\;|\;\; f \wedge f' \;\;|\;\; f\langle\Phi\rangle^{\bowtie' pc}_{\bowtie p} f' \;\;|\;\; f\langle\Phi\rangle^{\bowtie' pc}_{\bowtie p,t} f' \;\;|\;\; f\langle\Phi\rangle^{\bowtie' pc,R}_{\bowtie p} f' \;\;|\;\; f\langle\Phi\rangle^{\bowtie' pc,R}_{\bowtie p,t} f'$$

□

$\mathcal{L}^{pc}_{PHMLu}$-formulae are interpreted over states of LCMCs. Informally, formulae of the form $f\langle\Phi\rangle f'$ state that there is some some execution and some integer l such that f holds for the first $l-1$ steps and f' becomes true in the l-th step and the observable behavior of the l-step execution involves some behavior from Φ. The subscript $\bowtie p$ denotes that the probability of paths fulfilling the formula is $\bowtie p$ and the use of subscript t denotes that the paths of interest are only those that achieve the goal in at most t time units. Finally, the superscript $\bowtie' pc$ requires paths to use $\bowtie' pc$ units of power, and the use of R, restricts power consumption calculations to the set of resources R. For instance, formula $tt\langle Act^*\rangle^{\leq pc}_{\geq 1} f$ expresses that there is some execution of the system for which eventually f becomes true, with probability 1, without consuming more than pc units of power. Similarly, $\neg(tt\langle Act^*\rangle^{\geq pc}_{>0} tt)$, specifies that the power consumption never exceeds the threshold of pc units, whereas $\neg(tt\langle Act^*\rangle^{\geq pc,\{cpu\}}_{>0} tt)$, specifies that the power consumption of resource cpu never exceeds the threshold of pc units.

In order to present the semantics of the logic, and in particular of the four until operators, we need to compute the probabilities that certain behaviors are satisfied. Then, for example, if we compute that under a certain scheduler the probability measure of all scheduled computations of a system for which f is satisfied until f' becomes true within t time units and consuming less than pc units of power, via computations that have observable behavior from the set Φ, to be p, then the system satisfies formulae $f\langle\Phi\rangle^{<pc}_{\geq p',t} f'$, for all $p' \leq p$.

To compute such probabilities we need some machinery. We present the machinery necessary for formula $f\langle\Phi\rangle^{\leq pc,R}_{\bowtie p,t} f'$.

Let $\Phi \subseteq Act^*$, $\mathcal{M}, A \subseteq S$, $R \subseteq \mathcal{R}$, t an integer, pc a real number, and $\sigma \in \mathsf{Sched}(T)$. We define

$$\begin{aligned}
\mathsf{FPaths}_A(T,\Phi,\mathcal{M},\leq pc,R,t) &= \{c \in \mathsf{Pcomp}(T) \mid \mathsf{last}\, c \in \mathcal{M}, \mathsf{trace}\,(c) \in \Phi, \mathsf{time}\,(c) \leq t,\\
&\qquad \mathsf{states}\,(\mathsf{init}\,(c)) \subseteq A,\ \mathsf{power}\,(c,R) \leq pc\}\\
\mathsf{SPaths}_A(T,\Phi,\mathcal{M},\leq pc,R,t,\sigma) &= \{c \in \mathsf{Scomp}(T,\sigma) \mid c = c_1c_2, c_1 \in \mathsf{FPaths}_A(T,\Phi,\mathcal{M},\leq pc,R,t)\}.
\end{aligned}$$

Thus, $\mathsf{FPaths}_A(T,\Phi,\mathcal{M},\leq pc,R,t)$ denotes the set of partial computations of T that lead to a state in $\mathcal{M}$ via a sequence of actions in Φ with intermediate states in A, in less than t time units and consuming no more than pc units of power of resources in R. On the other hand, $\mathsf{SPaths}_A(T,\Phi,\mathcal{M},\leq pc,R,t\sigma)$, denotes the set of (complete) computations in

$\mathsf{Scomp}(T,\sigma)$ that are extensions of partial computations of the previous set. It is easy to see that this last set is measurable in $\mathcal{A}^\sigma(T)$ as $\mathsf{SPaths}_A(T,\Phi,\mathcal{M},\leq pc,R,t,\sigma)=\bigcup_\omega C(\omega)$, where $\omega\in\mathsf{FPaths}_A(T,\Phi,\mathcal{M},\leq pc,R,t)\cap\mathsf{Scomp}_{fin}(T,\sigma)$.

The probability $\Pr_A(T,\Phi,\mathcal{M},\leq pc,R,t,\sigma,s_0)=\mathcal{P}(\mathsf{SPaths}_A(T,\Phi,\mathcal{M},\leq pc,R,t,\sigma))$ is given as the smallest solution to the following set of equations:

$$\Pr_A(s,\Phi,\mathcal{M},\leq pc,R,t,\sigma,c) \;=\; \begin{cases} 1 & \text{if } \varepsilon\in\Phi, s\in\mathcal{M}, pc\geq 0\; t\geq 0\\ \Sigma_{s',s\xrightarrow{\pi}_p s'}\ \pi\cdot\Pr_A(s',\Phi,\mathcal{M},\leq pc,R,t,\sigma,c\,\pi s') & \\ \qquad\qquad\qquad\text{if } s\in S_p\cap A & \\ \Pr_A(s',\Phi-\alpha,\mathcal{M},\leq pc-\mathsf{pow}\,(\alpha,R),R,t-\mathsf{time}\,(\alpha),\sigma,c\,\alpha\,s') & \\ \qquad\qquad\qquad\text{if } s\in S_n\cap A, \sigma(c)=(s,\alpha,s') & \\ 0 & \text{otherwise} \end{cases}$$

where $\Phi-\alpha$ is $\{\phi\mid\alpha\phi\in\Phi\}$ if $\alpha\neq\tau$ and Φ, otherwise. Thus $\Pr_A(T,\Phi,\mathcal{M},\leq pc,R,t,\sigma,s_0)$ denotes the probability of performing from s_0, under scheduler σ, a sequence of actions in Φ to reach a state in $\mathcal{M}$ while passing only via states in A in less than t times units and consuming less than pc units of power in resources R.

Similarly, we can compute probabilities $\Pr_A(T,\Phi,\mathcal{M},\bowtie pc,R,t,\sigma,s_0)$, where $\bowtie\in\{\geq,<,>\}$, $\Pr_A(T,\Phi,\mathcal{M},\bowtie pc,t,\sigma,s_0)$, $\Pr_A(T,\Phi,\mathcal{M},\bowtie pc,R,\sigma,s_0)$, and $\Pr_A(T,\Phi,\mathcal{M},\bowtie pc,\sigma,s_0)$. To do this we define:

$$\begin{array}{lcl}
\mathsf{FPaths}_A(T,\Phi,\mathcal{M},\bowtie pc) & = & \{c\in\mathsf{Pcomp}(T)\mid \mathsf{last}\,c\in\mathcal{M}, \mathsf{trace}\,(c)\in\Phi,\\
 & & \qquad\mathsf{states}\,(\mathsf{init}\,(c))\subseteq A,\ \mathsf{power}\,(c)\bowtie pc\},\\
\mathsf{FPaths}_A(T,\Phi,\mathcal{M},\bowtie pc,R) & = & \{c\in\mathsf{Pcomp}(T)\mid \mathsf{last}\,c\in\mathcal{M}, \mathsf{trace}\,(c)\in\Phi,\\
 & & \qquad\mathsf{states}\,(\mathsf{init}\,(c))\subseteq A,\ \mathsf{power}\,(c,R)\bowtie pc\},\\
\mathsf{FPaths}_A(T,\Phi,\mathcal{M},\bowtie pc,t) & = & \{c\in\mathsf{FPaths}_A(T,\Phi,\mathcal{M},\bowtie pc)\mid \mathsf{time}\,(c)\leq t\},\\
\mathsf{FPaths}_A(T,\Phi,\mathcal{M},\bowtie pc,R,t) & = & \{c\in\mathsf{FPaths}_A(T,\Phi,\mathcal{M},\bowtie pc,R)\mid \mathsf{time}\,(c)\leq t\},\\
\mathsf{SPaths}_A(T,\Phi,\mathcal{M},\bowtie pc,\sigma) & = & \{c\in\mathsf{Scomp}(T,\sigma)\mid c=c_1c_2, c_1\in\mathsf{FPaths}_A(T,\Phi,\mathcal{M},\bowtie pc)\},\\
\mathsf{SPaths}_A(T,\Phi,\mathcal{M},\bowtie pc,R,\sigma) & = & \{c\in\mathsf{Scomp}(T,\sigma)\mid c=c_1c_2, c_1\in\mathsf{FPaths}_A(T,\Phi,\mathcal{M},\bowtie pc,R)\}\\
\mathsf{SPaths}_A(T,\Phi,\mathcal{M},\bowtie pc,t,\sigma) & = & \{c\in\mathsf{Scomp}(T,\sigma)\mid c=c_1c_2, c_1\in\mathsf{FPaths}_A(T,\Phi,\mathcal{M},\bowtie pc,t)\}\\
\mathsf{SPaths}_A(T,\Phi,\mathcal{M},\bowtie pc,R,t,\sigma) & = & \{c\in\mathsf{Scomp}(T,\sigma)\mid c=c_1c_2, c_1\in\mathsf{FPaths}_A(T,\Phi,\mathcal{M},\bowtie pc,R,t)\}
\end{array}$$

and proceed to define the probability measures of the last four sets in the natural way.

Finally, the satisfaction relation $\models\ \subseteq(S_n\cup S_p)\times\mathcal{L}^{pc}_{PHMLu}$, stating when an LCMC state satisfies a given formula, is defined inductively as follows. Let $T=(S_n,S_p,Act,\longrightarrow_n,\longrightarrow_p,s_0)$, be an LCMC. Then:

$$\begin{array}{lll}
s\models tt & \text{always} & \\
s\models\neg f & \textit{iff} & s\not\models f\\
s\models f\wedge f' & \textit{iff} & s\models f \text{ and } s\models f'\\
s\models f\langle\Phi\rangle^{\bowtie' pc}_{\bowtie p}f' & \textit{iff} & \text{there is } \sigma\in\mathsf{Sched}(s) \text{ such that } \Pr_A(s,\Phi,B,\bowtie' pc,\sigma,s)\bowtie p,\\
 & & \text{where } A=\{s'\mid s'\models f\} \text{ and } B=\{s'\mid s'\models f'\}\\
s\models f\langle\Phi\rangle^{\bowtie' pc,R}_{\bowtie p}f' & \textit{iff} & \text{there is } \sigma\in\mathsf{Sched}(s) \text{ such that } \Pr_A(s,\Phi,B,\bowtie' pc,R,\sigma,s)\bowtie p,\\
 & & \text{where } A=\{s'\mid s'\models f\} \text{ and } B=\{s'\mid s'\models f'\}\\
s\models f\,\langle\Phi\rangle^{\bowtie' pc}_{\bowtie p,t}f' & \textit{iff} & \text{there is } \sigma\in\mathsf{Sched}(s) \text{ such that } \Pr_A(s,\Phi,B,\bowtie' pc,t,\sigma,s)\bowtie p,\\
 & & \text{where } A=\{s'\mid s'\models f\},\ B=\{s'\mid s'\models f'\}\\
s\models f\,\langle\Phi\rangle^{\bowtie' pc,R}_{\bowtie p,t}f' & \textit{iff} & \text{there is } \sigma\in\mathsf{Sched}(s) \text{ such that } \Pr_A(s,\Phi,B,\bowtie' pc,R,t,\sigma,s)\bowtie p,\\
 & & \text{where } A=\{s'\mid s'\models f\},\ B=\{s'\mid s'\models f'\}
\end{array}$$

4.1.1 The Model-Checking Algorithm

Let *closure*(f) denote the set of formulae $\{\ f',\neg f'\mid f' \text{ is a subformula of } f\ \}$. Our model-checking algorithm is similar to the CTL model-checking algorithm of [5]. In order to check that LCMC T satisfies some formula $f \in \mathcal{L}^{pc}_{PHMLu}$, the algorithm labels each state s of T with a set $F \subseteq \mathit{closure}(f)$, such that for every $f' \in F$, $s \models f'$. T satisfies f if and only if s_0, the initial state of T, is labeled with f. The algorithm starts with the atomic subformulae of f and proceeds to more complex subformulae. The labeling rules are straightforward from the semantics of the operators, with the exception of the *until* operator.

In order to decide whether a state s satisfies one of $f\langle\Phi\rangle^{\bowtie' pc}_{\bowtie p} f'$, $f\langle\Phi\rangle^{\bowtie' pc}_{\bowtie p,t} f'$, $f\langle\Phi\rangle^{\bowtie' pc,R}_{\bowtie p} f'$, $f\langle\Phi\rangle^{\bowtie' pc,R}_{\bowtie p,t} f'$, we compute the maximum (minimum) probability of the specified behavior. The maximum value of $\Pr_A(s, \Phi, B, \leq pc, \sigma, s)$ over all σ is computed as the value of the variable $X^s_{f\langle\Phi\rangle^{\leq pc} f'}$ in the smallest solution of the following set of equations:

$$X^s_{f\langle\Phi\rangle^{\leq pc} f'} = \begin{cases} \sum_{s \xrightarrow{\pi}_p s'} \pi \cdot X^{s'}_{f\langle\Phi\rangle^{\leq pc} f'} & \text{if } s \in S_p \\ \max(\{X^{s'}_{f\langle\Phi-\alpha\rangle^{\leq pc - \mathrm{pow}(\alpha)} f'} \mid s \xrightarrow{\alpha}_n s'\}) & \text{if } s \in S_n, s \models f \\ 1 & \text{if } s \in S_n, s \models f', \varepsilon \in \Phi, pc \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

A solution for this set of equations can be computed by solving a linear programming problem, in a manner similar to [2]. More precisely, for all equations of the form $X = \max\{X_1, \ldots X_n\}$, we introduce, the set of inequalities $X \geq X_i$ aiming to minimize the function $\sum_i X_i$. Using algorithms based on the ellipsoid method, this problem can be solved in time polynomial in the number of variables (see, e.g. [10]).

The efficiency of the algorithm can be improved in many obvious ways. In particular, a symbolic version of the algorithm along the lines of [1] is possible.

Example

Consider two systems requiring the use of a resource. Suppose the first system employs a highly reliable resource r that never fails, $\pi(r) = 1$, but consumes a large amount of power during each of its uses. On the other hand the second system opts on using a less reliable resource r' with probability of failure $1/2$ but consumes less power. The description of these systems is as follows:

$$P \stackrel{\text{def}}{=} rec\ X.\{(r,1,2)\} : \overline{succ}.X$$
$$Q \stackrel{\text{def}}{=} rec\ X.(\{(r',1,1)\} : \overline{succ}.X + \{(\overline{r'},1,0)\} : X)$$

We observe that Q attempts to use resource r' and if it is up then it consumes r', performs the event $\overline{succ}$ and returns to its initial state, otherwise, if r' is down, it retries to use r' until it succeeds. The LCMCs corresponding to processes P, Q are given in Figure 1. We may see that although Q risks a delay in successfully using resource r', on average, it consumes less power than P per successful resource use. Specifically, it is easy to show that letting Φ be the set all all sequences of actions of the form $\ell_1, \ldots, \ell_n$, where $\ell_i \in \{\{(r,1,2)\}\{(r',1,1)\}, \{(\overline{r'},1,0)\}\}$ for all $1 \leq i \leq n-1$ and $\ell_n = \overline{succ}$, $(Q, \emptyset) \models tt\langle\Phi\rangle^{\leq 1}_{\geq 1} tt$, since, with probability 1, configuration $(Q, \emptyset)$ can eventually successfully use resource r' using 1 unit of power. On the other hand, $(P, \emptyset) \not\models tt\langle\Phi\rangle^{\leq 1}_{\geq 1} tt$, since a successful usage of resource r consumes two units of power. Introducing a time limit to the property to be checked we can see the tradeoff with respect to the time delay between using resource r' and using resource r: although $(P, \emptyset) \models tt\langle\{\Phi\rangle^{\leq 2}_{\geq 1,1} tt$, $(Q, \emptyset) \not\models tt\langle\Phi\rangle^{\leq 2}_{\geq 1,1} tt$, instead $(Q, \emptyset) \models tt\langle\Phi\rangle^{\leq 1}_{\geq 0.5,1} tt$, and $(Q, \emptyset) \models tt\langle\Phi\rangle^{\leq 2}_{\geq 0.75,2} tt$.

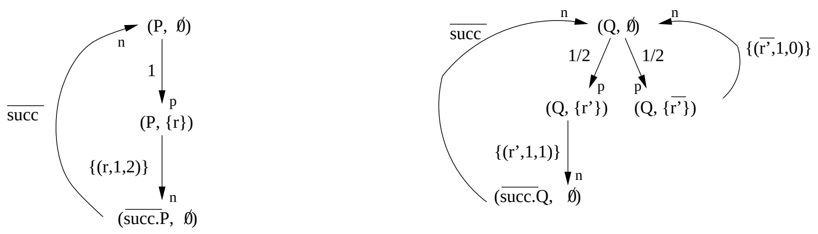

Figure 1: The LCMCs of processes P and Q

4.2 Probabilistic bounds on power consumption

Model checking P^{2}ACSR processes with respect to logical formulae allows us to verify important properties of a process. A disadvantage of this approach, however, is that to reason about power consumption we need to guess and specify a bound on power consumption in the formula. Although, often these bounds come from the requirements for the process, sometimes we do not have them *a priori*. On the contrary, we need to compute bounds of power consumption of processes over a fixed interval of time. In this section we show how such bounds can be calculated.

Let T be an LCMC and σ a finite scheduler in $\mathsf{Sched}(T)$, where by σ being finite we mean that it can schedule a finite number of transitions. We consider the set of computations $\mathsf{Scomp}(T, \sigma)$, and would like to compute their expected power consumption, which we denote by $Pow(T, \sigma)$. This expected value is $Pow(s_0, \sigma, \epsilon)$, where s_0 is the initial state of T, given by the least solution to the following set of equations:

$$Pow(s, \sigma, c) \;=\; \begin{cases} 0 & \sigma(c) = \perp \\ \mathsf{pow}\,(\alpha) + Pow(s', \sigma, c\alpha s') & s \in S_n, \sigma(c) = (s, \alpha, s') \\ \Sigma_{s'} \pi(s, s') \cdot Pow(s', \sigma, c\pi(s, s')s') & s \in S_p \end{cases}$$

Here we assume that $\mathsf{pow}\,(\alpha) = 0$ whenever α is an event.

Given a finite P^{2}ACSR process P, we want to compute the minimum and maximum power consumption over the set of all schedulers. This can be achieved by the following algorithm for the initial state s_0:

compute_bounds(s)
    if $s \in S_n$
        if s has no transitions, $p_{min}(s) = 0, p_{max}(s) = 0$
        else for each s', $s \xrightarrow{\alpha}_n s'$
            compute_bounds(s')
            $p_{min}(s) = \min_{s \xrightarrow{\alpha}_n s'} \mathsf{pow}\,(\alpha) + p_{min}(s')$
            $p_{max}(s) = \max_{s \xrightarrow{\alpha}_n s'} \mathsf{pow}\,(\alpha) + p_{max}(s')$
    if $s \in S_p$
        for each s', $s \xrightarrow{\alpha}_n s'$
            compute_bounds(s')
            $p_{min}(s) = \Sigma_{s \xrightarrow{p}_p s'} p \cdot p_{min}(s')$
            $p_{max}(s) = \Sigma_{s \xrightarrow{p}_p s'} p \cdot p_{max}(s')$

Task	Execution time	Period
1	3	8
2	3	10
3	1	14

Table 2: Example task set

By replacing $\mathsf{pow}(\alpha)$ by $\mathsf{pow}(\alpha, R)$, we can also compute the expected power-consumption bounds regarding the set of resources R.

5 Case Study: Power-Aware Real-Time Scheduling

In this section, we describe the case study of a power-aware application. The case study is based on the work of [14] and concerns the use of dynamic voltage scaling [3] in an embedded real-time system. Dynamic voltage scaling allows us to make a tradeoff between performance and power consumption. A CMOS-based processor operating on a lower frequency can use a lower supply voltage and thus consume less power. At the same time, a lower-frequency execution means that tasks take longer to compute. If the system has real-time requirements associated with it, these requirements may become violated at lower frequency. A power-aware real-time operating system has to decide when it is possible to operate at a lower frequency while at the same time maintaining the timing properties of the system.

In [14], Pillai and Shin propose extensions to real-time scheduling algorithms to make use of dynamic voltage scaling. We concentrate on the extension of the Earliest Deadline First (EDF) scheduling algorithm [12] that utilizes cycles unused by the tasks to lower the operating frequency for other tasks. The algorithm assumes a set of independent periodic tasks $T_1, \ldots, T_n$ that have to be executed on the same processor. Each task T_i has a *period* p_i, a *worst-case execution time* c_i, and a deadline d_i by which the execution must be completed. The ratios of execution time to period in each task define the nominal utilization of the processor by the task set that determines whether the tasks can be scheduled. In reality, tasks often take much less than the worst case to execute. Thus effective utilization of the task set may be much lower than the nominal one.

When the processor operates at a lower frequency, execution times of tasks grow accordingly, increasing nominal utilization so that the task set may become unschedulable. However, the effective utilization may be small enough even for a lower frequency. The power-aware scheduling algorithm of [14] computes effective utilization during execution and switches frequencies to use the lowest frequency for which the task set remains effectively schedulable.

We begin by describing the EDF scheduling algorithm and its encoding as an ACSR process. Then, we turn this description into a PACSR process by extending the tasks with probabilistic early completion. Finally, entering the realm of P^{2}ACSR, we extend the process with power attributes and add dynamic voltage scaling to the model. We demonstrate that the timing constraints of the tasks are maintained even at lower frequencies and show the savings in power consumption offered by the power-aware scheduling.

For the case study, we use the example from [14]. The task set contains three tasks with the parameters shown in Table 2. In each case, the deadline of the task is the same as

$$
\begin{array}{ll}
T_i & = (Job_i \| Actuator_i) \backslash \{start_i\} \\
Actuator_i & = (\overline{start_i}, i).\emptyset^{p_i} : Actuator_i \\
Job_i & = \emptyset : Job_i + (start_i, 0).Exec_{i,0,0} \\
Exec_{i,e,t} & = e < c_i \rightarrow \{(cpu, p_{max} - (p_i - t))\} : Exec_{i,e+1,t+1} \\
 & \qquad\qquad + \emptyset : Exec_{i,e,t+1} \\
 & \quad + e = c_i \rightarrow Job_i \qquad\qquad e \in \{0..c_i\}, t \in \{0..p_i\}
\end{array}
$$

Figure 2: ACSR specification of a task with EDF priorities

its period. Execution times are shown for the maximum processor frequency. We assume that the processor has two possible operating frequencies. For simplicity, we assume that at the reduced frequency tasks take twice as long to execute and consume half of the power.

5.1 EDF Scheduling Algorithm

The ACSR representation of the EDF scheduling algorithm has been presented in [4]. An instance of the scheduling problem is modeled as a collection of processes $T_1, \ldots, T_n$. The process T_i is shown in Figure 2. A task is represented as a parallel composition of two processes: Job_i and $Activator_i$. We mention that an ACSR-specification differs from a P^2ACSR specification in that actions are sets of pairs (r, p) where r is a resource and p its priority, that is, no power consumption rates are present, and, further, resources cannot fail.

The role of the activator is to keep track of the timing constraint of the task. At the beginning of every period, $Activator_i$ sends the signal $start_i$ to Job_i, releasing the task for execution, and then idles until the end of the period. If, by the end of the period, the task has not finished its execution, it will not be able accept the next $start_i$ signal, resulting in a deadlock that will signify the scheduling failure. For, recall from the semantics of P^2ACSR and in particular rule (Par3), that in a parallel composition of processes a timed action can only occur if all processes can engage in a timed action: in the above-mentioned case $Actuator_i$ can only engage in the instantaneous event $(\overline{start_i}, i)$, whereas $Exec_{i,e,t}$ can only engage in timed actions, thus brining the process to a deadlock signifying the scheduling failure.

The other process, Job_i, upon receiving the $start_i$ signal, Job_i begins its execution. At each time step, the task has a priority that is increased as the task approaches its deadline. The task that has been released t time units ago, has $p_i - t$ time units remaining until the deadline and has priority $p_{max} - (p_i - t)$, where $p_{max} = max(p_1, \ldots, p_n) + 1$. When the task receives the processor resource, it executes for one time unit and its accumulated execution time e, is increased together with the elapsed time t. At any time step, the task can be interrupted by another task that has a closer deadline. In this case, the task makes an idling step and its accumulated execution time stays the same while the elapsed time is increased.

Theoretical results from [12] show that a set of tasks is schedulable if the utilization of the task set, $U_i = \Sigma_{i \in \{1..n\}} c_i / p_i$ does not exceed 1. The task set from our example satisfies this criterion, and, by checking the resulting process for the absence of deadlocks, we can indeed verify that all deadlines are met.

$$
\begin{aligned}
Job_i \quad &= \emptyset : Job_i + (start_i, 0).Exec_{i,0,0} \\
Exec_{i,e,t} \quad &= e < c_i \rightarrow \{(cpu, p_{max} - (p_i - t)), (cont, 1)\} : Exec_{i,e+1,t+1} \\
&\qquad\qquad + \{(cpu, p_{max} - (p_i - t)), (\overline{cont}, 1)\} : Job_i \\
&\qquad\qquad + \emptyset : Exec_{i,e,t+1} \\
&\quad + e = c_i \rightarrow Job_i \qquad\qquad\qquad e \in \{0..c_i\}, t \in \{0..p_i\}
\end{aligned}
$$

Figure 3: PACSR specification of a task with probabilistic completion time

5.2 Probabilistic Execution of Tasks

The EDF scheduling algorithm assumes that every task completes its execution in every period according to its worst-case execution parameter. However, in practice, tasks usually take much less to complete than in the worst case. The power-aware scheduling algorithm takes advantage of this fact. In this section, we extend the model of a task to include the potential for early termination. With each task, we can associate a probability distribution on the time it takes to complete the task. For simplicity, we assume that the execution time of a task conforms to the geometric distribution. That is, after every execution step, the task may terminate with probability π and continue its execution with probability $1 - \pi$. Thus the probability that the task takes i time units to execute is $(1-\pi)^{i-1} \cdot \pi$. We assume that this distribution is the same for all tasks. We capture the probabilistic behavior in the model in the PACSR version of the process Job_i shown in Figure 3. We introduce an additional resource *cont* that represents continuation of the task execution. When the resource fails, the task terminates its execution, otherwise the execution continues, up to the worst-case execution time.

5.3 Real-time Dynamic Voltage Scaling

The algorithm of [14] recomputes effective utilization every time a task is released for execution or ends its execution in the current period. Then, it selects the least operating frequency for the processor that would still guarantee schedulability of the task set. The algorithm is shown in Figure 4. Effective utilization is recomputed every time a task is released for execution or completes its execution in the current period. Procedure select_frequency() finds the lowest operational frequency that is consistent with the current effective utilization.

We now show how to represent this algorithm as a P²ACSR process. We first extend the model of a task with the ability to execute faster of slower depending on the state of the system. The new task model is shown in Figure 5. The task uses signals *fast* and *slow* to determine whether the processor is in the fast or slow mode. Then, if the processor is in the slow mode, the next computation step takes two time units. The task uses resource *cont* to determine whether to continue execution during the second time unit. The task T_i uses two additional events, $release_i$ and $end_{i,j}$. These events are used to drive the voltage scaling algorithm and correspond to the release of task T_i and the completion of T_i after j time units, respectively.

Resources used in the model of the task do not consume power since both represent abstract notions: scheduling priorities and probabilistic completion. Power consumed by the processor is captured by a separate resource *power* that is used by the process DVS described below. The process DVS, shown in Figure 6 consists of two parallel

select_frequency():
  use lowest frequency $f_i \in \{f_1, \ldots, f_m \mid f_1 < \ldots < f_m\}$
  such that $U_1 + \ldots + U_n \leq f_i / f_m$

initialize():
  set $U_i = c_i / p_i$;
  select_frequency();

upon release of task T_i:
  set $U_i = c_i / p_i$;
  select_frequency();

upon completion of task T_i:
  set $U_i = c_i^{act} / p_i$;
  where c_i^{act} is the actual time used in the current period by T_i
  select_frequency();

Figure 4: Dynamic voltage scaling for EDF scheduling

$$
\begin{array}{lll}
Job_i & = & \emptyset : Job_i + (start_i, 0).(\overline{release_i}, i).Exec_{i,0,0} \\
Exec_{i,e,t} & = & e < c_i \rightarrow ((fast, i).\ (\ \{(cpu, p_{max} - (p_i - t), 0), (cont, 1, 0)\} : Exec_{i,e+1,t+1} \\
 & & \qquad + \{(cpu, p_{max} - (p_i - t), 0), (\overline{cont}, 1, 0)\} : (\overline{end_{i,e+1}}, i).Job_i \\
 & & \qquad + \emptyset : Exec_{i,e,t+1}) \\
 & & \quad + (slow, i).(\ \{(cpu, p_{max} - (p_i - t), 0)\} : \\
 & & \qquad\quad (\ \{(cpu, p_{max} - (p_i - t), 0), (cont, 1, 0)\} : Exec_{i,e+1,t+2} \\
 & & \qquad\quad + \{(cpu, p_{max} - (p_i - t), 0), (\overline{cont}, 1, 0)\} : (\overline{end_{i,e+1}}, i).Job_i) \\
 & & \qquad + \emptyset : Exec_{i,e,t+1}) \\
 & & + \ e = c_i \rightarrow (\overline{end_{i,c_i}}, i).Job_i \qquad\qquad e \in \{0..c_i\}, t \in \{0..p_i\}
\end{array}
$$

Figure 5: A speed-sensitive task

$$
\begin{aligned}
DVS \quad &= (Scale_{c_1,c_2,c_3} \| DVS_{fast}) \backslash \{f_{up}, f_{down}\} \\
\\
Scale_{e_1,e_2,e_3} \quad &= (release_1, 0).SetNew_{c_1,e_2,e_3} \\
&+ (release_2, 0).SetNew_{e_1,c_2,e_3} \\
&+ (release_3, 0).SetNew_{e_1,e_2,c_3} \\
&+ \Sigma_{c \in \{1..c_1\}} (end_{1,c}, 0).SetNew_{c,e_2,e_3} \\
&+ \Sigma_{c \in \{1..c_2\}} (end_{2,c}, 0).SetNew_{e_1,c,e_3} \\
&+ \Sigma_{c \in \{1..c_3\}} (end_{3,c}, 0).SetNew_{e_1,e_2,c} \\
&+ \emptyset : Scale_{e_1,e_2,e_3} \\
\\
SetNew_{e_1,e_2,e_3} &= e_1/p_1 + e_2/p2 + e_3/p_3 < 1/2 \rightarrow (\overline{f_{down}}, 4).Scale_{e_1,e_2,e_3} \\
&+ e_1/p_1 + e_2/p2 + e_3/p_3 >= 1/2 \rightarrow (\overline{f_{up}}, 4).Scale_{e_1,e_2,e_3}) \\
\\
DVS_{fast} \quad &= \{(power, 1, pw_{fast})\} : DVS_{fast} + (\overline{fast}, 1).DVS_{fast} \\
&+ (f_{down}, 0).DVS_{slow} + (f_{up}, 0).DVS_{fast} + \\
DVS_{slow} \quad &= \{(power, 1, pw_{fast})\} : DVS_{slow} + (\overline{slow}, 1).DVS_{slow} \\
&+ (f_{down}, 0).DVS_{slow} + (f_{up}, 0).DVS_{fast}
\end{aligned}
$$

Figure 6: P²ACSR representation of voltage scaling

parts. The first part, represented by the process $Scale_{e_1,e_2,e_3}$, represents the voltage scaling algorithm itself. Triggered by an event $release_i$ or $end_{i,c}$ that correspond to the release or, respectively, completion of the task T_i after executing for c time units, the process $SetNew$ computes the effective utilization and sends signal f_{down} if a lower operating frequency is possible and signal f_{up} otherwise. The other component of the process DVS keeps the information at the current operating frequency. It has two states, DVS_{fast} and DVS_{slow}. In the former state, the process uses the resource $power$ at the power consumption level of pw_{fast} and in the latter state the same resource is used with power consumption of pw_{slow}, where pw_{fast} and pw_{slow} are parameters of the model.

5.4 Analysis

We began the analysis of the case study by checking that the task set remains schedulable by the power-aware scheduling algorithm. The resulting system does not have any deadlocks, which means that all timing constraints are satisfied. Then we used the algorithm described in Section 4.2 to compute the expected power consumption of the task set from Table 2 for the duration of one major frame, that is, the product of periods of all tasks, $p_1 \cdot p_2 \cdot p_3$ (1120 time units). The probability of the task completion after a computation step was taken to be 1/3, and parameters pw_{fast} and pw_{slow} were 2 and 1, respectively.

The expected minimum and maximum power consumption was calculated to be 1906.66 and 1922.65, respectively. Without the dynamic voltage scaling, when each step would take pw_{fast} power units, the power consumption for the same period would be 2240 units. As a result, expected savings from the dynamic voltage scaling are between 14% and 14.8%.

6 Conclusions

We have presented P²ACSR, a process algebra for resource-oriented real-time systems. The formalism allows one to model the power consumption of resources and perform power-oriented analysis of a system's behavior. We have also presented two techniques for analysing P²ACSR specifications. First, we have proposed a probabilistic temporal power-aware logic in which one can express interesting properties regarding the behavior of power-aware, real-time systems and have given an algorithm for checking whether formulas of the logic are satisfied by P²ACSR specifications. Second, we have presented an algorithm for computing probabilistic bounds on power consumption. We illustrated the utility of the proposed approach using a couple of examples.

We are currently extending the PARAGON toolset [16], which allows the specification and analysis of ACSR and PACSR processes, to handle the power consumption model of P²ACSR. To the present, we have extended PARAGON to accept P²ACSR specifications and construct the LCMCs emanating from them, and we have implemented the algorithm for obtaining probabilistic bounds on power consumption. As future work we will also implement the model-checking algorithm for $\mathcal{L}^{pc}_{PHMLu}$, thus allowing the automatic model-checking of P²ACSR processes against $\mathcal{L}^{pc}_{PHMLu}$-formulas.

Another useful measure to be computed on P²ACSR specifications is that of long-run average performance. Average behavior is particularly interesting for power consumption studies. Average power consumption can be computed per unit of time, or if desired, per periods of interest. It has already been shown in the literature how to evaluate the long-run average behavior of LCMC's [6]. This is achieved by distinguishing the experiment of which the average behavior is to be determined and then computing this long-run average by performing appropriate searches on the LCMC. As future work, we intend to implement the above-mentioned algorithm in the PARAGON toolset.

Finally, we intend to define ordering relations by which to relate protocols that although behaviorally similar, differ in their power consumption rates.

References

[1] Christel Baier, Edmund Clarke, Vassilis Hartonas-Garmhausen, Marta Kwiatkowska, and Mark Ryan. Symbolic model checking for probabilistic processes. In *Proceedings of ICALP '97*, volume 1256 of *Lecture Notes in Computer Science*, pages 430–440. Springer-Verlag, July 1997.

[2] A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In *Proceedings Foundations of Software Technology and Theoretical Computer Science*, volume 1026 of *Lecture Notes in Computer Science*, pages 499–513. Springer-Verlag, 1995.

[3] T. D. Burd and R. W. Brodersen. Energy efficient CMOS microprocessor design. In *Proceedings of the 28th Annual Hawaii International Conference on System Sciences. Volume 1: Architecture*, pages 288–297. IEEE Computer Society Press, 1995.

[4] J-Y. Choi, I. Lee, and H-L Xie. The specification and schedulability analysis of real-time systems using ACSR. In *Real-Time Systems Symposium*. IEEE Computer Society Press, December 1995.

[5] E. Clarke, E. Emerson, and A. Prasad Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2), 1986.

[6] L. de Alfaro. How to specify and verify the long-run average behavior of probabilistic systems. In *Proceedings 13th Annual IEEE Symposium on Logic in Computer Science*, pages 454–465. IEEE, 1998.

[7] Rocco De Nicola and Frits W. Vaandrager. Three logics for branching bisimulation. In *Proceedings of LICS '90*. IEEE Computer Society Press, 1990.

[8] P. Halmos. *Measure Theory*. Springer Verlag, 1950.

[9] H. Hansson. *Time and Probability in Formal Design of Distributed Systems*. PhD thesis, Department of Computer Systems, Uppsala University, 1991. DoCS 91/27.

[10] H. Karloff. *Linear Programming*. Progress in Theoretical Computer Science. Birkhauser, 1991.

[11] I. Lee, P. Brémond-Grégoire, and R. Gerber. A process algebraic approach to the specification and analysis of resource-bound real-time systems. *Proceedings of the IEEE*, pages 158–171, Jan 1994.

[12] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM 20*, 1:46–61, 1973.

[13] A. Philippou, O. Sokolsky, R. Cleaveland, I. Lee, and S. Smolka. Probabilistic resource failure in real-time process algebra. In *Proceedings CONCUR 98*, pages 389–404. Springer-Verlag, 1998.

[14] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proceedings of the* 18th *Annual ACM Symposium on Operating Systems Principles*, 2001.

[15] R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. In B. Jonsson and J. Parrow, editors, *Proceedings CONCUR 94,* Uppsala, Sweden, volume 836 of *Lecture Notes in Computer Science*, pages 481–496. Springer-Verlag, 1994.

[16] O. Sokolsky, I. Lee, and H. Ben-Abdallah. Specification and analysis of real-time systems with PARAGON. *Annals of Software Engineering*, 7:211–234, 1999.

[17] M. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proceedings 26th Annual Symposium on Foundations of Computer Science*, pages 327–338. IEEE, 1985.