Probabilistic Resource Failure in Real-Time Process Algebra*

Anna Philippou[1], Rance Cleaveland[2], Insup Lee[1],
Scott Smolka[3], and Oleg Sokolsky[4]

[1] University of Pennsylvania, USA. {annap, lee}@saul.cis.upenn.edu
[2]University of North Carolina, USA. rance@eos.ncsu.edu
[3]SUNY at Stony Brook, USA. sas@cs.sunysb.edu
[4]Computer Command and Control Company, USA. sokolsky@cccc.com

Abstract. PACSR, a probabilistic extension of the real-time process algebra ACSR, is presented. The extension is built upon a novel treatment of the notion of a *resource*. In ACSR, resources are used to model contention in accessing physical devices. Here, resources are invested with the ability to *fail* and are associated with a probability of failure. The resulting formalism allows one to perform probabilistic analysis of real-time system specifications in the presence of resource failures. A probabilistic variant of Hennessy-Milner logic with *until* is presented. The logic features an *until* operator which is parameterized by both a probabilistic constraint and a regular expression over observable actions. This style of parameterization allows the application of probabilistic constraints to complex execution fragments. A model-checking algorithm for the proposed logic is also given. Finally, PACSR and the logic are illustrated with a telecommunications example.

1 Introduction

A common high-level view of a distributed real-time system is that the components of the system compete for access to shared resources, communicating with each other as necessary. To capture this view explicitly in formal specifications, a real-time process algebra ACSR [17] has been developed. ACSR 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, and access to them is governed by priorities. A process that attempts to access a resource currently in use by a higher-priority process is blocked from proceeding.

The notion of a resource, which is already important in the specification of real-time systems, additionally provides a convenient abstraction mechanism for

* This work was supported in part by grants AFOSR F49620-95-1-0508, ARO DAAH04-95-1-0092, NSF CCR-9415346, NSF CCR-9619910, and ONR N00014-97-1-0505 (MURI).

probabilistic aspects of systems behavior. A major source of behavioral variation in a process is failure of physical devices, such as processors, memory units, and communication links. These are exactly the type of objects that are captured as resources in ACSR specifications. Therefore, it is natural to use resources as a means of exploring the impact of failures on a system's performance.

In this paper, we present PACSR, a process algebra that extends the resource model of ACSR with the ability to reason about resource failures. Each resource used by the system is associated with a probability of failure. If a process attempts to access a failed resource, it is blocked. Resource failures are assumed to be independent. Then, for each execution step that requires access to a set of resources, we can compute the probability of being able to take the step. This approach allows us to reason quantitatively about a system's behavior.

Previous work on extending process algebra with probability information (discussed below) typically associates probabilities with process terms. An advantage of associating probabilities with resources, rather than with process terms, is that the specification of a process does not involve probabilities directly. In particular, a specification simply refers to the resources required by a process. Failure probabilities of individual resources are defined separately and are used only during analysis. This makes the specification simpler and ensures a more systematic way of applying probabilistic information. In addition, this approach allows one to explore the impact of changing probabilities of failures on the overall behavior, without changing the specification.

We are also interested in being able to specify and verify high-level requirements for a PACSR specification. Temporal logics are commonly used to express such high-level requirements. In the probabilistic setting, the requirements usually include probabilistic criteria that apply to large fragments of the system's execution. We present a simple temporal logic suitable for expressing properties of PACSR expressions. As is common with probabilistic extensions of temporal logics, we associate probabilistic constraints with temporal operators. The novel feature of the logic is that we allow temporal operators to be parameterized with regular expressions over the set of observable actions. Such parameterization allows us to apply probabilistic constraints to complex execution fragments.

For example, consider a communication protocol in which a sender inquires about the readiness of a receiver, obtains an acknowledgement, and sends data. A reasonable requirement for the system would be that this exchange happens with a certain probability. To express this property, one usually needs two nested temporal *until* operators. Since probabilistic constraints are associated with temporal operators, the single constraint has to be artificially split in two to apply to each of the operators. With the proposed extension, we need only one temporal operator, and the property is expressed naturally. A model-checking algorithm for the logic, suitable for finite-state PACSR specifications, is also given.

In terms of related work, a number of process algebras have been proposed that extend process terms with probability information, including [12, 21, 2, 10, 16, 20]. The approach of [12] is particularly relevant as it also adds probability to a real-time process algebra. It does not, however, consider the notions of

resource and resource probability, nor use priorities to control communication and resource access. In [18], an automata-based formalism that combines the notions of real-time and probabilities is presented. It employs a different notion of time in that transitions can have variable durations. Also, probabilities are associated with instantaneous events.

Since a PACSR specification typically consists of several parallel processes, concurrent events in these processes are the source of non-deterministic behavior, which cannot be resolved through probabilities. To provide for both probabilistic and non-deterministic behavior, the semantics of PACSR processes are given via *labeled concurrent Markov chains* [22]. This model has also been employed in [12], and variations of it appeared in [18, 6].

Regarding previous work on model checking for probabilistic systems, a closely related approach involves associating a probability threshold with the *until* operator of the temporal logic CTL [7]. For example, see [13, 4, 6, 14]. We find that this approach can become problematic when expressing properties that require multiple, nested *until*s. Our proposed extension of the *until* operator, which uses regular expressions and probability, serves to alleviate this deficiency.

The rest of the paper is organized as follows: the next section presents the syntax of PACSR and its semantics is given in Section 3. Section 4 discusses the temporal logic and the model-checking algorithm. In Section 5, we present an application of PACSR for the analysis of a probabilistic telecommunications system. We conclude with some final remarks and discussion of future work.

2 The Syntax of PACSR

2.1 Resource Probabilities and Actions

PACSR (Probabilistic ACSR) extends the process algebra ACSR by associating with each resource a probability. This probability captures the rate at which the resource may fail. PACSR also has two types of actions: instantaneous events and timed actions, the latter of which specifies access to a (possibly empty) set of resources. We discuss these three concepts below.

Instantaneous events. PACSR 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*. Labels are drawn from the set $\mathsf{L} = \mathcal{L} \cup \overline{\mathcal{L}} \cup \{\tau\}$, where if a is a given label, $\overline{a}$ is its *inverse* label. The special label τ 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. We assume that a system contains a finite set of serially reusable resources drawn from the set Res. We also consider set $\overline{Res}$ that contains, for each $r \in Res$, an element $\overline{r}$, representing the *failed* resource r. We write R for $Res \cup \overline{Res}$. An action that consumes one tick of time is drawn from the domain $\boldsymbol{P}(\mathsf{R} \times \boldsymbol{N})$ with the restriction that each resource is represented at most once. For

example the singleton action $\{(r,p)\}$ denotes the use of some resource $r \in Res$ at priority level p. Such an action cannot happen if r has failed. On the other hand, action $\{(\overline{r},q)\}$ takes place with priority q given that resource r has failed. This construct is useful for specifying recovery from failures. The action $\emptyset$ represents idling for one unit of time, since no resource is consumed.

We let $\mathcal{D}_R$ denote the domain of timed actions and we let A, B range over $\mathcal{D}_R$. We define $\rho(A)$ to be the set of the resources used by action A; for example $\rho(\{(r_1,p_1),(\overline{r_2},p_2)\}) = \{r_1,\overline{r_2}\}$.

Resource Probabilities In PACSR we associate each resource with a probability specifying the rate at which the resource may fail. In particular, for all $r \in Res$ we denote by $\mathsf{p}(r) \in [0,1]$ the probability of resource r being up, while $\mathsf{p}(\overline{r}) = 1 - \mathsf{p}(r)$ denotes the probability of r failing. Thus, the behavior of a resource-consuming process has certain probabilistic aspects to it which are reflected in the operational semantics of PACSR. For example, consider process $\{(cpu,1)\}:$ NIL, where resource *cpu* has probability of failure $1/3$, i.e. $\mathsf{p}(cpu) = 2/3$. Then with probability $2/3$, resource *cpu* is available and thus the process may consume it and become inactive, while with probability $1/3$ the resource fails and the process deadlocks. This is discussed in detail in Section 3.

2.2 Processes

We let P, Q range over PACSR 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 PACSR processes.

$$\begin{array}{rl} P ::= & \text{NIL} \mid (a,n).\,P \mid A:P \mid P+P \mid P\|P \mid \\ & P \,\triangle_t^a\, (P,P,P) \mid P\backslash F \mid [P]_I \mid P\backslash\backslash I \mid rec\ X.P \mid X \end{array}$$

The 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. The second, $A:P$, executes a resource-consuming action during the first time unit and proceeds to process P. The process $P+Q$ represents a nondeterministic choice between the two summands. The 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.

The scope construct, $P \,\triangle_t^a\, (Q,R,S)$, binds the process P by a temporal scope and incorporates the notions of timeout and interrupts. We call t the *time bound*, where $t \in \mathbb{N} \cup \{\infty\}$ and require that P may execute for a maximum of t time units. The scope may be exited in one of three ways: First, if P terminates successfully within t time-units by executing an event labeled $\overline{a}$, where $a \in \mathsf{L}$, then control is delegated to Q, the success-handler. On the other hand, if P fails to terminate within time t then control proceeds to R. Finally, throughout execution of this process construct, P may be interrupted by process S. 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. $P \backslash\backslash I$ hides the identity of resources in I so that they are not visible on the interface with the environment. Finally, the process $rec\ X.P$ denotes standard recursion. We write Proc for the set of PACSR processes.

The operator $P \backslash\backslash I$ binds all free occurrences of the resources of I in P. This binder gives rise to the sets of *free* and *bound resources* of a process P. In what follows, we work up to α-conversion on resources. In this way, bound resources in a process are assumed to be different from each other and from the free resources, and α-equivalent processes are assumed to have the same transitions.

Note that the syntax of PACSR processes is the same as that of ACSR. The only extension concerns the appearance of failed resources in timed actions. This allows us to perform probabilistic analysis of existing ACSR specifications, and non-probabilistic analysis of PACSR specifications.

The informal account of behavior just given is made precise via a family of rules that define the labeled transition relations $\longrightarrow_\pi$ and $\longmapsto$ 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 PACSR process with the set of resources on which its behavior immediately depends:

$$\begin{array}{ll}
\mathsf{imr}(\mathrm{NIL}) = \emptyset & \mathsf{imr}(P_1 \| P_2) = \mathsf{imr}(P_1) \cup \mathsf{imr}(P_2) \\
\mathsf{imr}(a.P) = \emptyset & \mathsf{imr}(P \backslash F) = \mathsf{imr}(P) \\
\mathsf{imr}(A : P) = \rho(A) & \mathsf{imr}([P]_I) = \mathsf{imr}(P) \cup I \\
\mathsf{imr}(P_1 + P_2) = \mathsf{imr}(P_1) \cup \mathsf{imr}(P_2) & \mathsf{imr}(P \backslash\backslash I) = \mathsf{imr}(P) \\
\mathsf{imr}(P \stackrel{a}{\triangle}_t (Q, R, S)) = \begin{cases} \mathsf{imr}(P + S), \text{ if } t > 0 \\ \mathsf{imr}(R), \text{ if } t = 0 \end{cases} & \mathsf{imr}(rec\ X.P) = \mathsf{imr}(P)
\end{array}$$

Definition 1. Let $Z = \{c_1, \ldots, c_n\} \subseteq \mathsf{R}$. We write

- $\mathsf{p}(Z) = \Pi_{1 \leq i \leq n} \mathsf{p}(c_i)$,
- $\mathcal{W}(Z) = \{Z' \subseteq Z \cup \overline{Z} \mid x \in Z' \text{ iff } \overline{x} \notin Z'\}$, and
- $res(Z) = \{r \in Res \mid r \in Z \text{ or } \overline{r} \in Z\}$. □

Thus $\mathcal{W}(Z)$ denotes the set of all possible worlds involving the set of resources Z, that is the set of all combinations of the resources in Z being up or down. For example, $\mathcal{W}(\{r_1, \overline{r_2}\}) = \{\{\overline{r_1}, \overline{r_2}\}, \{\overline{r_1}, r_2\}, \{r_1, \overline{r_2}\}, \{r_1, r_2\}\}$. Note that $\mathsf{p}(\emptyset) = 1$ and $\mathcal{W}(\emptyset) = \{\emptyset\}$.

3 Operational Semantics

The semantics of PACSR processes is given in two steps. At the first level, a transition system 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. A *configuration* is a pair of the form $(P, W) \in \mathsf{Proc} \times 2^{\mathsf{R}}$, representing a PACSR process P in world W. We write S for the set of 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 PACSR process P, we begin with the configuration $(P, \emptyset)$. As computation proceeds, probabilistic transitions are performed to determine the status of resources which are 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. Timed actions erase all previous knowledge of the configuration's world (see law (Act2)). Nondeterministic transitions may be performed from configurations that contain all necessary knowledge regarding the state of resources. 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 $\longmapsto \subset S_p \times [0,1] \times S_n$ be the probabilistic transition relation. A triple in $\longmapsto$, written $(P, W) \stackrel{p}{\longmapsto} (P', W')$, 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 $(P, W) \stackrel{\alpha}{\longrightarrow} (P', W')$, 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:

$$(\mathsf{PROB})\ \frac{(P, W) \in S_p, Z_1 = res(\mathsf{imr}(P)) - res(W), Z_2 \in \mathcal{W}(Z_1)}{(P, W) \stackrel{\mathsf{p}(Z_2)}{\longmapsto} (P, W \cup Z_2)}$$

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 $\mathsf{p}(Z_2)$. Note that configuration (P, W) evolves into $(P, W \cup Z_2)$ which is, by definition, a nondeterministic configuration.

For example, given resources r_1 and r_2 such that $\mathsf{p}(r_1) = 1/2$ and $\mathsf{p}(r_2) = 1/3$, $P \stackrel{\text{def}}{=} \{(r_1, 2), (\overline{r_2}, 3)\} : Q$ has exactly the following transitions:

$$(P, \emptyset) \stackrel{1/6}{\longmapsto} (P, \{r_1, r_2\}) \qquad (P, \emptyset) \stackrel{1/6}{\longmapsto} (P, \{\overline{r_1}, r_2\})$$
$$(P, \emptyset) \stackrel{1/3}{\longmapsto} (P, \{r_1, \overline{r_2}\}) \qquad (P, \emptyset) \stackrel{1/3}{\longmapsto} (P, \{\overline{r_1}, \overline{r_2}\})$$

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

The nondeterministic transition relation is given in Table 1. Note that the symmetric versions of rules (Sum) and (Par1) have been omitted. Consider in particular rules (Act1) and (Act2): instantaneous events preserve the world of a configuration while timed actions re-initialize it to $\emptyset$. Thus, by rule (Act2) we have $(P, \{r_1, \overline{r_2}\}) \stackrel{\{(r_1,2),(\overline{r_2},3)\}}{\longrightarrow} (Q, \emptyset)$, whereas $(P, \{r_1, r_2\})$, $(P, \{\overline{r_1}, r_2\})$, and $(P, \{\overline{r_1}, \overline{r_2}\})$ have no transitions. Except for the appearance of worlds in configurations, the rules of Table 1 are essentially identical to the ones for ACSR. It is worth pointing out that all processes in a parallel composition need to synchronize on a timed action (Par3).

(Act1) $((a,n).P, B) \stackrel{(a,n)}{\longrightarrow} (P, B)$ (Act2) $(A : P, B) \stackrel{A}{\longrightarrow} (P, \emptyset)$, if $\rho(A) \subseteq B$

(Sum) $\dfrac{(P_1, B) \stackrel{\alpha}{\longrightarrow} (P, B')}{(P_1 + P_2, B) \stackrel{\alpha}{\longrightarrow} (P, B')}$ (Par1) $\dfrac{(P_1, B) \stackrel{(a,n)}{\longrightarrow} (P_1', B')}{(P_1 \| P_2, B) \stackrel{(a,n)}{\longrightarrow} (P_1' \| P_2, B')}$

(Par2) $\dfrac{(P_1, B) \stackrel{(a,n)}{\longrightarrow} (P_2', B'),\ \ (P_2, B) \stackrel{(\overline{a},m)}{\longrightarrow} (P_2', B')}{(P_1 \| P_2, B) \stackrel{(\tau,n+m)}{\longrightarrow} (P_1' \| P_2', B')}$

(Par3) $\dfrac{(P_1, B) \stackrel{A_1}{\longrightarrow} (P_1', B'),\ (P_2, B) \stackrel{A_2}{\longrightarrow} (P_2', B')}{(P_1 \| P_2, B) \stackrel{A_1 \cup A_2}{\longrightarrow} (P_1' \| P_2', B')}$, $\rho(A_1) \cap \rho(A_2) = \emptyset$

(Res1) $\dfrac{(P, B) \stackrel{A}{\longrightarrow} (P', B'),\ \ A' = \{(r,n) \in A \mid r \notin I\}}{(P \backslash\backslash I, B) \stackrel{A'}{\longrightarrow} (P' \backslash\backslash I, B')}$

(Res2) $\dfrac{(P, B) \stackrel{\alpha}{\longrightarrow} (P', B'),\ \ l(a) \notin F}{(P \backslash F, B) \stackrel{\alpha}{\longrightarrow} (P' \backslash F, B')}$ (Cl1) $\dfrac{(P, B) \stackrel{(a,n)}{\longrightarrow} (P', B')}{([P]_I, B) \stackrel{(a,n)}{\longrightarrow} ([P']_I, B')}$

(Cl2) $\dfrac{(P, B) \stackrel{A_1}{\longrightarrow} (P', B'), A_2 = \{(r,0) \mid r \in B \cap (I \cup \overline{I})\}}{([P]_I, B) \stackrel{A_1 \cup A_2}{\longrightarrow} ([P']_I, B')}$

(Sc1) $\dfrac{(P, B) \stackrel{(a,n)}{\longrightarrow} (P', B'),\ \ \overline{a} \neq b,\ \ t > 0}{(P \stackrel{b}{\triangle}_t (Q, R, S), B) \stackrel{(a,n)}{\longrightarrow} (P' \stackrel{b}{\triangle}_t (Q, R, S), B')}$

(Sc2) $\dfrac{(P, B) \stackrel{(\overline{b},n)}{\longrightarrow} (P', B'),\ \ t > 0}{(P \stackrel{b}{\triangle}_t (Q, R, S), B) \stackrel{(\tau,n)}{\longrightarrow} (Q, B')}$ (Sc3) $\dfrac{(R, B) \stackrel{\alpha}{\longrightarrow} (R', B'),\ \ t = 0}{(P \stackrel{b}{\triangle}_t (Q, R, S), B) \stackrel{\alpha}{\longrightarrow} (R', B')}$

(Sc4) $\dfrac{(P, B) \stackrel{A}{\longrightarrow} (P', B'),\ \ t > 0}{(P \stackrel{b}{\triangle}_t (Q, R, S), B) \stackrel{A}{\longrightarrow} (P' \stackrel{b}{\triangle}_{t-1} (Q, R, S), B')}$

(Sc5) $\dfrac{(S, B) \stackrel{\alpha}{\longrightarrow} (S', B'),\ \ t > 0}{(P \stackrel{b}{\triangle}_t (Q, R, S), B) \stackrel{\alpha}{\longrightarrow} (S', B')}$ (Rec) $\dfrac{(P[rec\ X.P/X], B) \stackrel{\alpha}{\longrightarrow} (P', B')}{(rec\ X.P, B) \stackrel{\alpha}{\longrightarrow} (P', B')}$

Table 1. The nondeterministic relation

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 systems associated with the relations $\longmapsto$ and $\longrightarrow_\pi$, 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, the idle action $\emptyset$ is preemptable by all other timed actions. The basic idea behind $\longrightarrow_\pi$ is that a nondeterminstic transition of the form $(P, W) \stackrel{\alpha}{\longrightarrow}_\pi (P', W')$ is permitted if and only if there are no higher-priority transitions enabled in (P, W), that is $\beta \prec \alpha$ for all β enabled in (P, W). Thus we have that the prioritized nondeterministic transition system is obtained from the unprioritized one by pruning away preemptable transitions.

Definition 2. The prioritized labeled transition system $\longrightarrow_\pi$ is defined as follows: $(P, W) \stackrel{\alpha}{\longrightarrow}_\pi (P', W')$ if and only if (1) $(P, W) \stackrel{\alpha}{\longrightarrow} (P', W')$ is an unprioritized nondeterministic transition, and (2) there is no unprioritized transition $(P, W) \stackrel{\beta}{\longrightarrow} (P'', W'')$ such that $\alpha \prec \beta$. □

We conclude this section with an example. The following process describes a faulty channel that, on receipt of an input, may either produce an output with probability 0.99 or lose the message with probability 0.01, depending on the state of resource **channel**, where $\mathsf{p}(\mathbf{channel}) = 0.99$.

$$FCh \stackrel{\text{def}}{=} (in.\, P \;+\; \emptyset : FCh) \backslash\backslash \{\mathbf{channel}\}$$
$$P \stackrel{\text{def}}{=} \{\mathbf{channel}\}.\, \overline{out}.\, FCh \;+\; \{\overline{\mathbf{channel}}\}.\, FCh\,.$$

Figure 1 exhibits the transition system of process FCh in world $\emptyset$, that is, without initial knowledge about the status of resource **channel**. Note that state $(P, \emptyset)$ is probabilistic, while all other states are non-deterministic.

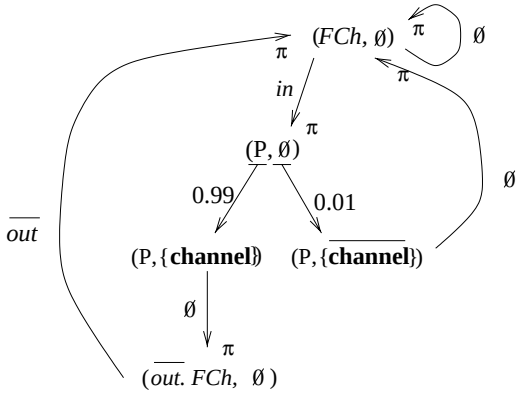

Fig. 1. Transition system of process FCh

4 Model Checking for PACSR

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 PACSR specifications, we introduce in this section a probabilistic temporal logic that allows one to associate probabilistic constraints with fragments of behaviors. The associated model-checking algorithm, also presented in this section, is used to check that these constraints are satisfied. Behavioral fragments of interest are expressed in terms of regular expressions over *Act*, the set of observable actions.

Before presenting the logic, we introduce the structure that we use as the model for formulas of the logic and introduce some notation. Logical formulas are interpreted with respect to a given *Labeled Concurrent Markov Chain*.

Definition 3. 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. □

It is easy to see that the operational semantics of PACSR yields transition systems that are LCMCs.

In what follows, we let α, β range over Act and ℓ over $Act \cup [0,1]$. In addition, when it is clear from the context, we will simply refer to an LCMC $\langle S_n, S_p, Act, \longrightarrow_n, \longrightarrow_p, s_0\rangle$ by s_0. Given $s, s' \in S$, $\mathsf{pr}(s,s')$ denotes the probability that s may perform at most one probabilistic transition to become s':

$$\mathsf{pr}(s,s') = \begin{cases} 1, & \text{if } s = s', s \in S_n \\ \pi, & \text{if } s \xrightarrow{\pi}_p s' \\ 0, & \text{otherwise} \end{cases}$$

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 $s_i \in S$, $\ell_{i+1} \in Act \cup [0,1]$ 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 $c = s_0\ell_1 \ldots \ell_k s_k \in \mathsf{Pcomp}(T)$, we define $\mathsf{trace}\,(c) = \ell_1 \ldots \ell_k \restriction Act - \{\tau\}$, $\mathsf{states}\,(c) = \{s_1, \ldots s_k\}$, $\mathsf{time}\,(c) = \#(\ell_1 \ldots \ell_k \restriction \mathcal{D}_R)$, $\mathsf{init}\,c = s_0 \ldots s_{k-1}$ and $\mathsf{last}\,c = s_k$.

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 [22, 12, 19]. A scheduler is an entity that, given a partial computation ending in a nondeterministic state, chooses the next transition to be executed.

Definition 4. 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 σ induces a probability space [11] on $\mathsf{Scomp}(T, \sigma)$. 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 $\mathcal{P}(C(\omega)) = \Pi\{\!|\ell_i \in [0,1] \mid 1 \leq i \leq k|\!\}$.

4.1 Probabilistic HML with until

We now introduce our logic for PACSR which is based on the Hennessy-Milner Logic (HML) with *until* [9]. Our logic extends the work of [9] in two ways. First it allows for quantitative analysis of probabilistic properties of a system by associating a probabilistic condition with the *until* operator. The condition takes the form of $\leq p$ or $\geq p$ for a constant p. Intuitively, *until* expresses a property of an execution of the system, which we expect to hold with a probability satisfying the condition of the operator.

The second extension allows us to parameterize *until* operators with regular expressions over event names, instead of a single name. Using this construct, we can express, with a single temporal operator, a property of an execution that contains a series of events, rather than only one event. In the non-probabilistic setting, there is no need for such extension, since one can always express this property by using several nested *until* operators. In the probabilistic setting, however, nesting of operators would preclude us from associating a single probabilistic condition with the whole execution.

Additionally, in order to be able to capture real-time aspects of PACSR specifications, we offer a time-bounded version of the *until* operator.

Definition 5. *(Probabilistic HML with until)* The syntax of $\mathcal{L}^{pr}_{HMLu}$ is defined by the following grammar, where f, f' range over $\mathcal{L}^{pr}_{HMLu}$-formulas, Φ is a regular expression over Act, and $\bowtie \in \{\leq, \geq\}$:

$$f \;::=\; tt \;\mid\; \neg f \;\mid\; f \wedge f' \;\mid\; f\langle\Phi\rangle_{\bowtie p} f' \;\mid\; f\langle\Phi\rangle^{t}_{\bowtie p} f'$$

□

In order to present the semantics of the logic, we introduce the following definitions. Let $\Phi \subseteq Act^*$, $\mathcal{M}, A \subseteq S$, and $\sigma \in \mathsf{Sched}(T)$. We define

$$
\begin{array}{lcl}
\mathsf{FPaths}_A(T,\Phi,\mathcal{M}) & = & \{c \in \mathsf{Pcomp}(T) \mid \mathsf{last}\, c \in \mathcal{M}, \mathsf{trace}\,(c) \in \Phi, \mathsf{states}\,(\mathsf{init}\,(c)) \subseteq A\},\\
\mathsf{FPaths}'_A(T,\Phi,t,\mathcal{M}) & = & \{c \in \mathsf{FPaths}_A(T,\Phi,\mathcal{M}) \mid \mathsf{time}\,(c) \leq t\},\\
\mathsf{SPaths}_A(T,\Phi,\mathcal{M},\sigma) & = & \{c \in \mathsf{Scomp}(T,\sigma) \mid c = c_1c_2, c_1 \in \mathsf{FPaths}_A(T,\Phi,\mathcal{M})\},\\
\mathsf{SPaths}'_A(T,\Phi,\mathcal{M},t,\sigma) & = & \{c \in \mathsf{Scomp}(T,\sigma) \mid c = c_1c_2, c_1 \in \mathsf{FPaths}'_A(T,\Phi,t,\mathcal{M})\}.
\end{array}
$$

Thus, $\mathsf{FPaths}_A(T,\Phi,\mathcal{M})$ denotes the set of partial computations of T that lead to a state in $\mathcal{M}$ via a sequence of actions in Φ and pass only via states in A, while $\mathsf{FPaths}'_A(T,\Phi,t,\mathcal{M})$ denotes the subset of such computations that take at most t units of time. Moreover, $\mathsf{SPaths}_A(T,\Phi,\mathcal{M},\sigma)$ denotes the set of (complete) computations in $\mathsf{Scomp}(T,\sigma)$ that are extensions of partial computations in $\mathsf{FPaths}_A(T,\Phi,\mathcal{M})$, and similarly for $\mathsf{SPaths}'_A(T,\Phi,\mathcal{M},t,\sigma)$. It is easy to see that these sets are measurable in $\mathcal{A}^\sigma(T)$ as, for example, $\mathsf{SPaths}_A(T,\Phi,\mathcal{M},\sigma) = \bigcup_\omega C(\omega)$, where $\omega \in \mathsf{FPaths}_A(T,\Phi,\mathcal{M}) \cap \mathsf{Scomp}_{fin}(T,\sigma)$.

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

$$
\mathsf{Pr}_A(P,\Phi,\mathcal{M},\sigma,c) = \begin{cases}
1 & \text{if } \varepsilon \in \Phi, P \in \mathcal{M}\\
\Sigma_Q\ \mathsf{pr}(P,Q) \cdot \mathsf{Pr}_A(Q,\Phi,\mathcal{M},\sigma,c\,\mathsf{pr}(P,Q)\,Q) & \text{if } P \in S_p \cap A\\
\mathsf{Pr}_A(Q,\Phi-\alpha,\mathcal{M},\sigma,c\,\alpha\,Q) & \text{if } P \in S_n \cap A, \sigma(c) = (P,\alpha,Q)\\
0 & \text{otherwise}
\end{cases}
$$

where $\Phi - \alpha$ is $\{\phi \mid \alpha\phi \in \Phi\}$ if $\alpha \neq \tau$ and Φ, otherwise. Thus $\mathsf{Pr}_A(P,\Phi,\mathcal{M},\sigma,s_0)$ denotes the probability of performing from P, under scheduler σ, a sequence in Φ to reach a state in $\mathcal{M}$ while passing only via states in A. Probability $\mathsf{Pr}'_A(T,\Phi,\mathcal{M},t,\sigma,T) = \mathcal{P}(\mathsf{SPaths}'_A(T,\Phi,\mathcal{M},t,\sigma))$ denotes the probability of achieving the same effect within t time units and it can be similarly computed.

Finally, the satisfaction relation $\models\ \subseteq (S_n \cup S_p) \times \mathcal{L}^{pr}_{HMLu}$ is defined inductively as follows. Let $T = (S_n, S_p, Act, \longrightarrow_n, \longrightarrow_p, s_0)$, be an LCMC. Then:

$s \models tt$	always	
$s \models \neg f$	*iff*	$s \not\models f$
$s \models f \wedge f'$	*iff*	$s \models f$ and $s \models f'$
$s \models f\langle\Phi\rangle_{\bowtie p} f'$	*iff*	there is $\sigma \in \mathsf{Sched}(s)$ such that $\mathsf{Pr}_A(s,\Phi,B,\sigma,s) \bowtie p$, where $A = \{s' \mid s' \models f\}$ and $B = \{s' \mid s' \models f'\}$
$s \models f\,\langle\Phi\rangle^t_{\bowtie p} f'$	*iff*	there is $\sigma \in \mathsf{Sched}(s)$ such that $\mathsf{Pr}'_A(s,\Phi,B,t,\sigma,s) \bowtie p$, where $A = \{s' \mid s' \models f\}$, $B = \{s' \mid s' \models f'\}$

4.2 The Model-Checking Algorithm

Let *closure*(f) denote the set of formulas $\{\ f' \cup \neg f' \mid f'$ is a subformula of $f\ \}$. Our model-checking algorithm is similar to the CTL model-checking algorithm of [8]. In order to check that LCMC T satisfies some formula $f \in \mathcal{L}^{pr}_{HMLu}$, the algorithm labels each state s of T with a set $F \subseteq$ *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 subformulas of f and proceeds to more complex subformulas. 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 $f\langle\Phi\rangle_{\leq p} f'$ ($f\langle\Phi\rangle_{\geq p} f'$), we compute the maximum (minimum) probability of the specified behavior. The maximum value of $\mathsf{Pr}_A(s, \Phi, B, \sigma, s)$ over all σ is computed as the value of the variable $X^s_{f\langle\Phi\rangle f'}$ in the smallest solution of the following set of equations:

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

A solution for this set of equations can be computed by solving a linear programming problem, in a manner similar to [6]. More precisely, for all equations of the form $X = \max\{X_1, \ldots X_n\}$, we introduce, the set of inequations $X \geq X_i$. Our aim is to minimize the function $\sum_{s \in S} X^s_{f\langle\phi\rangle f'} + X^s_{f\langle\varepsilon\rangle f'}$. Using algorithms based on the ellipsoid method, this problem can be solved in time polynomial in the number of variables (see, e.g. [15]).

The efficiency of the algorithm can be improved in many obvious ways. In particular, a symbolic version of the algorithm along the lines of [3] is possible.

5 A Telecommunications Application

In this section we present an application of PACSR for the specification and analysis of a probabilistic system. The example was inspired by the specification of a telecommunications switching system presented in [1]. The system is comprised of a number of interacting concurrent processes with real-time constraints. As we will demonstrate, PACSR enables a natural description of the system in question, while the notion of priorities and their semantical treatment makes the implementation of the scheduling algorithm straightforward.

Specification. The structure of the system specification is shown in Figure 2. The subsystem in the dashed box is the monitor, which handles malfunctions in other components of the switch by processing *alarms*. Alarms are modeled as originating in the environment of the monitor (the solid-lined box).

The monitor consists of two processes: the alarm sampler which periodically samples alarms and places them in a bounded-size buffer, and the alarm handler which removes and processes alarms from the buffer. Process P represents low-priority background computation performed on the same processor.

All processes in the system have fixed priorities, the alarm sampler having the highest priority. Scheduling is non-preemptive and respects process priority. Thus, whenever processes are ready to be scheduled the scheduler passes control to the process with the highest priority. Once a process takes control, it is allowed to run for some maximum allocated time. If it is not completed by the deadline, it is killed by the operating system.

There are two sources of probabilistic behavior in the system. First, alarms are delivered after a hardware failure is detected by some component. We represent each device as a resource which a certain probability of failure. Additionally,

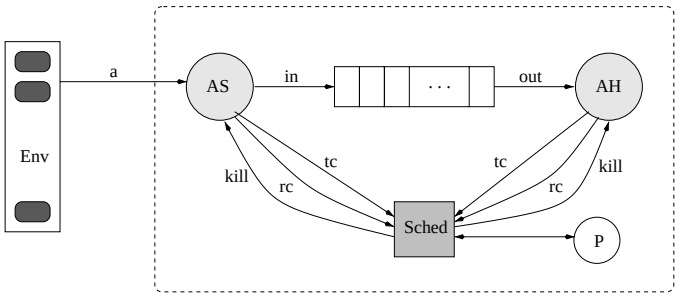

Fig. 2. Structure of the application

according to the scheduling requirements, all processes must relinquish control within a maximum allocated time. However, in reality this is often not the case. Thus, to analyze the system adequately, we take into account the probability of processes exceeding their allocated time-slice by assuming that the execution time of such processes is geometrically distributed.

Finally, the correctness requirement for the alarm handler is a probabilistic property: the probability of overflow in the alarm buffer should not exceed a given value. In our analysis of the model, we experimented with two instantiations of the specification involving different values for the various constants (e.g., the buffer size, and the various probabilities). Consequently, on comparing the two systems, we were able to show that one is better than the other, in terms of the probability of buffer overflow. The specification consists of the collection of processes in the following table.

$$
\begin{array}{ll}
Sys & \stackrel{\text{def}}{=} (Env \| B_0 \| \emptyset : Sched \| AS \| AH \| P) \backslash F \backslash\backslash I \\
Env & \stackrel{\text{def}}{=} \Pi_{1 \le i \le \mathbf{N}}\, P_i \\
P_i & \stackrel{\text{def}}{=} \{r_i\} : P_i \;+\; \{\overline{r_i}\} : (P_i \| Q_i) \\
Q_i & \stackrel{\text{def}}{=} \overline{a}.\,\mathrm{NIL} + \emptyset : Q_i \\
B_0 & \stackrel{\text{def}}{=} in.\, B_1 \;+\; \emptyset : B_0 \\
B_i & \stackrel{\text{def}}{=} in.\, B_{i+1} \;+\; \Sigma_{1 \le j \le i} d_j.\, B_{i-j} \;+\; \emptyset : B_i \;+\; \overline{out_i}.\, B_i \\
B_n & \stackrel{\text{def}}{=} in.\, \overline{overflow}.\,\mathrm{NIL} \;+\; \Sigma_{1 \le j \le n} d_j.\, B_{n-j} \;+\; \emptyset : B_n \;+\; \overline{out_n}.\, B_n. \\
Sched & \stackrel{\text{def}}{=} (tc, 1).\, \emptyset^{\infty} \;\triangle^{g}_{\mathbf{t_{max}}}\; (\mathrm{NIL}, \overline{kill}.\, Sched, rc.\, Sched) \;+\; \emptyset : Sched \\
AS & \stackrel{\text{def}}{=} AS' \| (\emptyset^{p} : AS) \\
AS' & \stackrel{\text{def}}{=} (\overline{tc}, 2) :\; AS'' \;+\; \emptyset : AS' \\
AS'' & \stackrel{\text{def}}{=} a.\, \overline{in}.\, AS'' \;+\; \emptyset : \overline{rc}.\,\mathrm{NIL} \\
AH & \stackrel{\text{def}}{=} \Sigma_i\; out_i.\, AH_{\mathbf{n}(i)} \;+\; \emptyset : AH \\
AH_i & \stackrel{\text{def}}{=} (\overline{tc}, 1) : AH_i^{A} \;+\; \emptyset : AH \\
AH_i^{A} & \stackrel{\text{def}}{=} \emptyset^{\mathbf{pt}(i)} : \overline{d_i}.\; \overline{rc}.\, AH \\
P & \stackrel{\text{def}}{=} (\overline{tc}, 0) :\; P' \;\triangle^{h}_{\infty}\; (\mathrm{NIL}, \mathrm{NIL}, kill.\, P) \;+\; \emptyset : P \\
P' & \stackrel{\text{def}}{=} (\{r\} : P' \;+\; \{\overline{r}\} : \overline{h}.\, \overline{rc}.\, P) \backslash\backslash \{r\}
\end{array}
$$

The system in its initial state is represented by process Sys, where $F = \{a, up, tc, rc, g, h, in, kill\} \cup \{out_i\} \cup \{d_i\}$, $I = \{r_i\} \cup \{r\}$, and Env represents the environment, B_0 the (empty) buffer, $Sched$ the scheduler, AS and AH the alarm sampler and the alarm handler processes, respectively, and P the low-priority background process.

The environment Env, responsible for providing alarms, is modeled as the parallel composition of processes P_i each of which consumes a resource r_i. We assume that the probability of failure is the same for all r_i. When resource r_i fails, an alarm is sent by process Q_i. For the purpose of the example, we do not distinguish between different alarms and record only the fact of their arrival. The number of processes P_i that determines the maximum number of alarms that can arrive within one time unit, is one of the parameters of the specification.

The buffer is given by a collection of processes B_i. The capacity of the buffer, n, is another parameter of the specification. An attempt to write to B_n, the process representing a full buffer, will result in the emission of the signal $\overline{overflow}$. Each process B_i, except B_0, can output the number of alarms it has using signal out_i, and also pass j ($j \leq i$) alarms to the handler by means of signal d_j, becoming B_{i-j}.

Scheduler $Sched$ allocates the next time slot to processes according to their priorities, by means of channel tc (tc stands for "take control"). Note that, while various components might attempt to access tc, the prioritized semantics of PACSR ensures that the highest-priority process will succeed. Processes signal their completion by means of signal rc ("relinquish control"), thus forcing the scheduler to begin the next scheduling cycle. Finally, the scheduler is responsible for killing a process should it exceed the maximum-allocated time, $\mathbf{t_{max}}$.

The alarm sampler AS is a periodic process with period p. Every p time-units it attempts to take control and sample all available alarms. The alarm sampler receives alarms emitted by the environment via a and passes them to the buffer via in. It only executes for a single time-unit and on completing execution it relinquishes control by signaling on rc. The alarm handler AH, upon being scheduled, checks how many alarms are in the buffer. If the buffer is not empty, it takes as many alarms from the buffer as it can process in its allocated time slice. Thus $\mathbf{n}(i) = \min(i, a_{max})$, with a_{max} being a parameter of the specification. $\mathbf{pt}(i)$ is the time it takes to process i alarms.

Finally, process P represents a low-priority background process having scheduling priority 0. To model variations in its execution time, we employ resource r, failures of which represent termination of P. Therefore, P's execution time is geometrically distributed with parameter $\mathsf{pr}(r)$.

Verification. We considered two versions of the system. In both cases the probability of an alarm is 0.9. The first version, S_1, features the possibility of at most one alarm per time unit and a buffer of size 3. The alarm handler can process two alarms per time slot, and each alarm requires one unit of processing time (*i.e.*, $\mathbf{pt}(i) = i$). For the second version, S_2, we assumed that the monitor runs on faster hardware. Therefore, the handler can now process four alarms per time slot, and $\mathbf{pt}(i) = i/2$, appropriately rounded. At the same time, the workload of

Time units	S_1 (false)	S_2 (true)
10	2×10^{-6}	3×10^{-10}
20	5×10^{-6}	6×10^{-10}
30	9×10^{-6}	1.0×10^{-9}
40	1.2×10^{-5}	1.3×10^{-9}
50	1.5×10^{-5}	1.6×10^{-9}
60	1.9×10^{-5}	2.1×10^{-9}
70	2.2×10^{-5}	2.4×10^{-9}
80	2.5×10^{-5}	2.8×10^{-9}
90	2.9×10^{-5}	3.1×10^{-9}
100	3.2×10^{-5}	3.5×10^{-9}

Table 2. Results of analysis

the component has been increased by allowing up to two alarms per time unit and the size of the buffer has been doubled.

For both versions of the system we checked the property $tt\langle\overline{overflow}\rangle^t_{\leq q} tt$ for various values of t and q. Table 2 shows, for a range of t, the largest value of q for which the property fails for S_1, and the smallest value of q that makes the property true for S_2. It can be seen that S_2 consistently outperforms S_1.

6 Conclusions and Future Work

We have presented PACSR, a process algebra for specification of resource-oriented real-time systems. The formalism allows one to model resource failures and perform probabilistic analysis of a system's behavior. A temporal logic for expressing high-level probabilistic properties of PACSR specifications was introduced. A simple model-checking algorithm was also given. We illustrated the utility of the proposed approach using a telecommunications application.

Analysis of the example given in the paper has been performed manually. We are currently implementing PACSR as part of the PARAGON toolset [5], designed to handle large-scale specifications.

References

1. R. Alur, L. Jagadeesan, J. Kott, and J. V. Olnhausen. Model-checking of real-time systems: a telecommunications application. In *Proceedings of the International Conference on Software Engineering*, 1997.
2. J. Baeten, J. Bergstra, and S. Smolka. Axiomatizing probabilistic processes: ACP with generative probabilities. *Information and Computation*, 121(2):234–255, Sept. 1995.
3. C. Baier, E. Clarke, V. Hartonas-Garmhausen, M. Kwiatkowska, and M. 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.

4. C. Baier and M. Kwiatkowska. Automatic verification of liveness properties of randomized systems (extended abstract). In *Proceedings of the* 14^{th} *Annual ACM Symposium on Principles of Distributed Computing,* Santa Barbara, California, Aug. 1997.
5. H. Ben-Abdallah, D. Clarke, I. Lee, and O. Sokolsky. PARAGON: A Paradigm for the Specification, Verification, and Testing of Real-Time Systems. In *IEEE Aerospace Conference*, pages 469–488, Feb 1-8 1997.
6. 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.
7. E. Clarke and E. Emerson. *Design and Synthesis of Synchronization Skeletons Using Branching Time Temporal Logic.* LNCS 131, 1981.
8. E. Clarke, E. Emerson, and A. P. Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications. *ACM Trans. Prog. Lang. Syst.*, 8(2), 1986.
9. R. De Nicola and F. Vaandrager. Three logics for branching bisimulation. In *Proceedings of LICS '90.* IEEE Computer Society Press, 1990.
10. A. Giacalone, C. Jou, and S. Smolka. Algebraic reasoning for probabilistic concurrent systems. In *Proceedings of Working Conference on Programming Concepts and Methods*, Sea of Gallilee, Israel, Apr. 1990. IFIP TC 2, North-Holland.
11. P. Halmos. *Measure Theory.* Springer Verlag, 1950.
12. H. Hansson. *Time and Probability in Formal Design of Distributed Systems.* PhD thesis, Department of Computer Systems, Uppsala University, 1991. DoCS 91/27.
13. H. Hansson and B. Jonsson. A logic for reasoning about time and probability. *Formal Aspects of Computing*, 6:512–535, 1994.
14. P. Iyer and M. Narasimha. 'almost always' and 'definitely sometime' are not enough: Probabilistic quantifiers and probabilistic model checking. Technical Report TR-96-16, Department of Computer Science, North Carolina State University, July 1996.
15. H. Karloff. *Linear Programming.* Progress in Theoretical Computer Science. Birkhauser, 1991.
16. J.-P. Katoen, R. Langerak, and D. Latella. Modeling systems by probabilistic process algebra: An event structures approach. In *Proceedings of FORTE '92 – Fifth International Conference on Formal Description Techniques*, pages 255–270, Oct. 1993.
17. 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.
18. R. Segala. *Modelling and Verification of Randomized Distributed Real-Time Systems.* PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1995.
19. 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.
20. K. Seidel. *Probabilistic CSP.* PhD thesis, Oxford University, 1992.
21. C. Tofts. Processes with probabilities, priorities and time. *Formal Aspects of Computing*, 4:536–564, 1994.
22. M. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proceedings* 26^{th} *Annual Symposium on Foundations of Computer Science*, pages 327–338. IEEE, 1985.