Specifying Failures and Recoveries in PACSR *

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

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

Abstract

The paper presents PACSR, a probabilistic extension of a real-time process algebra ACSR. 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 such as processors, memory modules, and communication links, or any other reusable resource of limited capacity. Here, we invest resources with an ability to *fail* and associate, with every resource, a probability of its failure. The resulting formalism allows us to perform probabilistic analysis of real-time system specifications in the presence of resource failures. An attractive feature of PACSR is the ability to express failure-recovery actions easily.

We perform probabilistic reachability analysis for PACSR specifications that allows us to compute the probability of occurrence of an undesirable event. We illustrate PACSR specification and analysis by means of a telecommunications example.

1 Introduction

Process algebras such as CCS [11] have proved to be effective for specification and analysis of distributed systems. Numerous real-time [18, 12, 8] and probabilistic [16] extensions of process algebras exist. We propose an approach that allows one to perform probabilistic analysis for real-time systems.

A common high-level view of a distributed real-time system is that its components 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 [10] 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.

*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).

The notion of a resource, which is important in specification of real-time systems, is even more critical to capture the probabilistic aspects of real-time systems behavior. A major source of behavioral variations in a process is failure of physical devices, such as processors, memory units, and communication links, that the process utilizes during its execution. 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. With each resource used by the system, we associate a fixed 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.

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.

A related approach that combines probabilistic specification with the notion of time is presented in [6]. The main distinguishing features of PACSR are the notion of resources and their use to capture probabilistic data, and the use of priorities to control communication and resource access.

A synchronous probabilistic process algebra WCCS is presented in [15]. There, each choice is assigned a weight. Weights are treated as priorities of the corresponding computation path. Furthermore, weights provide for probabilistic analysis of the specification. Time is measured in WCCS by counting the number of actions performed by a process, and no high-level temporal constructs such as timeouts are provided.

In [13], 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, semantics of PACSR processes are given via *labeled concurrent Markov chains* [17]. This model has also been employed in [6], and variations of it appeared in [13, 4].

We employ probabilistic reachability as means of analysis of PACSR specifications. The method allows us to perform quantitative analysis of safety properties by computing the probability of observing an undesirable event. Another popular method of analysis is probabilistic model checking [7, 2, 4].

The rest of the paper is organized as follows: In the next section we present the syntax of PACSR and then we proceed with its semantics in Section 3. In Section 4, we discuss probabilistic reachability for PACSR terms. In Section 5, we present an application of PACSR for the analysis of a probabilistic system. We conclude with some final remarks

and discussion of future work.

2 The Syntax of PACSR

2.1 Actions

PACSR extends the process algebra ACSR with probability by enriching the notion of resource, associating each resource with a probability. This probability captures the rate at which the resource may fail. PACSR has three types of actions: timed actions, events and probabilistic actions. We discuss these below:

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. Finally, we write R for $Res \cup \overline{Res}$. An action that consumes one tick of time is drawn from the domain $\boldsymbol{P}(\mathsf{R} \times \mathbb{N})$ 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 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 $\mathcal{D}_R$ to denote the domain of timed actions and we let A, B, to range over $\mathcal{D}_R$. We define $\rho(A)$ to be the set of the resources used by action A, for example $\rho(\{(r_1,p_1),(\overline{r_2},p_2)\}) = \{r_1,\overline{r_2}\}$.

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 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 range over the domain of events.

Probabilistic actions. As mentioned earlier, in PACSR we associate each resource, with a probability capturing 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 probabilistic aspects that are captured by probabilistic actions. 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 may fail, in which case the process deadlocks. This will be discussed in more 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.

$$\begin{array}{rcl} P & ::= & \mathrm{NIL} \mid A : P \mid (a, n).\, P \mid P + P \mid P \| P \mid \\ & & P \stackrel{a}{\triangle}_t (P, P, P) \mid P \backslash F \mid [P]_I \mid P \backslash\backslash I \mid rec\ X.P \mid X \end{array}$$

The process NIL represents the inactive process. There are two prefix operators, corresponding to the two types of actions. The first, $A : P$, executes a resource-consuming action during the first time unit and proceeds to process P. On the other hand $(a, n).\, P$, executes the instantaneous event (a, n) and proceeds to P. Sometimes, when it is not relevant for the discussion, we omit the priority of an event in a process. 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 instantaneous events, and they synchronize on timed actions. The scope construct, $P \stackrel{a}{\triangle}_t (Q, R, S)$, binds the process P by a temporal scope and incorporates the notions of timeout and interrupts. We call t the *time bound*, where $t \in \boldsymbol{N} \cup \{\infty\}$ 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 the time bound t by executing an event labeled $\overline{a}$, where $a \in \mathsf{L}$, then control is delegated to process 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 $[P]_I$, $I \subseteq \mathcal{R}$, 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 so as to avoid tedious side conditions. In this way, bound resources in a process are assumed to be different from each other and from the other 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 without any modifications, as well as use non-probabilistic analysis of PACSR processes (without failure recovery actions).

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 (either up or down) on which its behavior immediately depends:

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

Definition 2.1 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 to take account of action priorities.

We begin with the unprioritized semantics of PACSR processes. A *configuration* is a pair of the form $(P, W) \in \mathsf{Proc} \times \mathcal{W}(\mathsf{R})$, representing a PACSR process P in world W. We write S for the set of configurations. The semantics is given with the aid of two labeled transition systems, whose states are configurations and transitions capture probabilistic and nondeterministic transition relations, respectively.

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 the 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 (ActT)). Nondeterministic transitions may be performed from configurations that contain all necessary knowledge regarding the state of resources. With this in mind we partition S into the following two sets:

$S_n = \{(P, W) \in S \mid res(\mathsf{imr}(P)) - res(W) = \emptyset\}$, 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 $\longmapsto$ is written $(P, W) \stackrel{p}{\longmapsto} (P', W')$, denoting 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 action α and become (P', W').

The probabilistic transition relation is given by the following rule:

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

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 $\mathsf{p}(Z_2)$. 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:

$$\begin{aligned}
(P, \emptyset) &\stackrel{1/6}{\longmapsto} (P, \{r_1, r_2\}) \\
(P, \emptyset) &\stackrel{1/6}{\longmapsto} (P, \{\overline{r_1}, r_2\}) \\
(P, \emptyset) &\stackrel{1/3}{\longmapsto} (P, \{r_1, \overline{r_2}\}) \\
(P, \emptyset) &\stackrel{1/3}{\longmapsto} (P, \{\overline{r_1}, \overline{r_2}\})
\end{aligned}$$

Lemma 3.1 For all $s \in S_p$, $\Sigma\{\!|p \mid (s, p, s') \in \longmapsto |\!\} = 1$.

The nondeterministic transition relation is given in Table 1. Note in particular, rules (ActT) and (ActI): instantaneous events preserve the world of a configuration while timed actions re-initialize it to $\emptyset$. Thus, by rule (ActT) we have

$$(P, \{r_1, \overline{r_2}\}) \stackrel{\{(r_1,2),(\overline{r_2},3)\}}{\longrightarrow} (Q, \emptyset)$$

whereas $(P, \{r_1, r_2\})$, $(P, \{\overline{r_1}, r_2\})$, $(P, \{\overline{r_1}, \overline{r_2}\})$ have no transitions.

The prioritized transition system is based on the notion of *preemption* and it extends the unprioritized semantics by refining the nondeterministic transition relation $\longrightarrow$ to take account of priorities. 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. We refer to [10] for the precise definition. The prioritized nondeterministic transition system is obtained from the unprioritized one by prunning away preemptable transitions:

Definition 3.2 The 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 a couple of examples.

(ActT)	$(A : P, B) \xrightarrow{A} (P, \emptyset)$, provided $\rho(A) \subseteq B$	(ActI)	$((a, n).P, B) \xrightarrow{(a,n)} (P, B)$
(ChoiceL)	$\dfrac{(P_1, B) \xrightarrow{\alpha} (P, B')}{(P_1 + P_2, B) \xrightarrow{\alpha} (P, B')}$	(ChoiceR)	$\dfrac{(P_2, B) \xrightarrow{\alpha} (P, B')}{(P_1 + P_2, B) \xrightarrow{\alpha} (P, B')}$
(ParT)	$\dfrac{(P_1, B) \xrightarrow{A_1} (P_1', B'),\ (P_2, B) \xrightarrow{A_2} (P_2', B')}{(P_1 \| P_2, B) \xrightarrow{A_1 \cup A_2} (P_1' \| P_2', B')}$, $\rho(A_1) \cap \rho(A_2) = \emptyset$		
(ParIL)	$\dfrac{(P_1, B) \xrightarrow{(a,n)} (P_1', B')}{(P_1 \| P_2, B) \xrightarrow{(a,n)} (P_1' \| P_2', B')}$	(ParIR)	$\dfrac{(P_2, B) \xrightarrow{(a,n)} (P_2', B')}{(P_1 \| P_2, B) \xrightarrow{(a,n)} (P_1 \| P_2', B')}$
(ParI)	$\dfrac{(P_1, B) \xrightarrow{(a,n)} (P_2', B'),\ \ (P_2, B) \xrightarrow{(\overline{a},m)} (P_2', B')}{(P_1 \| P_2, B) \xrightarrow{(\tau, n+m)} (P_1' \| P_2', B')}$		
(ResI)	$\dfrac{(P, B) \xrightarrow{A} (P', B'),\ \ A' = \{(r, n) \in A \mid r \notin I\}}{(P \backslash\backslash I, B) \xrightarrow{A'} (P' \backslash\backslash I, B')}$	(ResF)	$\dfrac{(P, B) \xrightarrow{\alpha} (P', B'),\ \ l(a) \notin F}{(P \backslash F, B) \xrightarrow{\alpha} (P' \backslash F, B')}$
(CloseT)	$\dfrac{(P, B) \xrightarrow{A_1} (P', B'), A_2 = \{(r, 0) \mid r \in B \cap (I \cap \overline{I})\}}{([P]_I, B) \xrightarrow{A_1 \cup A_2} ([P']_I, B')}$	(CloseI)	$\dfrac{(P, B) \xrightarrow{(a,n)} (P', B')}{([P]_I, B) \xrightarrow{(a,n)} ([P']_I, B')}$
(ScopeCI)	$\dfrac{(P, B) \xrightarrow{(a,n)} (P', B'),\ \ \overline{a} \neq b,\ \ t > 0}{(P \stackrel{b}{\triangle}_t (Q, R, S), B) \xrightarrow{(a,n)} (P' \stackrel{b}{\triangle}_t (Q, R, S), B')}$	(ScopeE)	$\dfrac{(P, B) \xrightarrow{(\overline{b},n)} (P', B'),\ \ t > 0}{(P \stackrel{b}{\triangle}_t (Q, R, S), B) \xrightarrow{(\tau,n)} (Q, B')}$
(ScopeCT)	$\dfrac{(P, B) \xrightarrow{A} (P', B'),\ \ t > 0}{(P \stackrel{b}{\triangle}_t (Q, R, S), B) \xrightarrow{A} (P' \stackrel{b}{\triangle}_{t-1} (Q, R, S), B')}$	(ScopeT)	$\dfrac{(R, B) \xrightarrow{\alpha} (R', B'),\ \ t = 0}{(P \stackrel{b}{\triangle}_t (Q, R, S), B) \xrightarrow{\alpha} (R', B')}$
(ScopeI)	$\dfrac{(S, B) \xrightarrow{\alpha} (S', B'),\ \ t > 0}{(P \stackrel{b}{\triangle}_t (Q, R, S), B) \xrightarrow{\alpha} (S', B')}$	(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

Example 1: A faulty channel The following process describes a faulty channel which, 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**.

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

where $\mathsf{p}(\mathbf{channel}) = 0.99$. 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 other states are non-deterministic.

Example 2: A Fault-Tolerant Computer System In this example we consider a fault-tolerant system, consisting of a number of processors each associated with a watchdog responsible for monitoring the processor and notifying a controller in case of its failure. The controller then reactivates the processor via a restart procedure. The PACSR specification

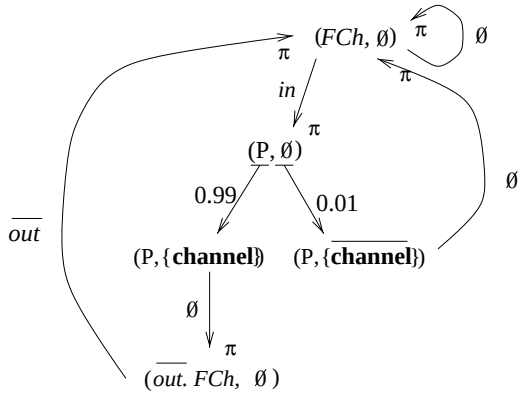

Figure 1: Transition system of process FCh

of the system is given below as the parallel composition Sys, where

$$Sys \stackrel{\text{def}}{=} \Pi_{i \in I}((P_i \| W_i) \| C) \backslash \{w_i, a_i, b_i, res_i\}_{i \in I}$$

and

$$\begin{aligned}
P_i &\stackrel{\text{def}}{=} P' \triangle_n^{b_i} (Crash_i, \overline{w_i}.\, P_i, \text{NIL}) \\
P_i' &\stackrel{\text{def}}{=} [\mathbf{cpu_i}] : P_i' + [\overline{\mathbf{cpu_i}}] : \overline{b_i}.\, \text{NIL} \\
Crash_i &\stackrel{\text{def}}{=} \emptyset : Crash_i + res_i.\, \overline{w_i}.\, P_i \\
W_i &\stackrel{\text{def}}{=} I \triangle_n^{b} (\text{NIL}, \overline{a_i}.\, W_i', w_i.\, W_i) \\
W_i' &\stackrel{\text{def}}{=} \emptyset : W_i' + w_i.\, W_i \\
C &\stackrel{\text{def}}{=} \Sigma_i\, a_i.\, \overline{res_i}.\, C + \emptyset : C.
\end{aligned}$$

Process P_i represents a processor. To model the possibility of resource failure and recovery, we employ for each P_i a resource cpu_i, where $\mathsf{pr}(cpu_i)$ is the probability that the processor fails during a time unit. Thus, as long as resource cpu_i is up, process P_i emits the signal w_i every n time units. On the other hand, if cpu_i fails, the process enters state $Crash_i$, where it remains down (idles) until the fault is detected and dealt with by the controller. When the processor is restarted via channel res_i, it emits a signal via channel w_i and resumes its initial state.

Process W_i represents the watchdog of processor P_i, where I is the idling process, $I \stackrel{\text{def}}{=} \emptyset : I$. Every n time units W_i is ready to receive a message via the processor it monitors via name b_i. If such a message fails to arrive, it notifies the controller via channel a_i and enters state W_i'. In this state, it waits to be notified via w_i that the processor has been restarted. Thus, no alarms are generated between the failure and recovery of the processor.

Finally, process C represents the controller that, on receiving an alarm a_i from one of the watchdogs, sends a restart command to the appropriate processor via channel res_i.

4 Reachability Analysis for PACSR

In this section we consider reachability analysis of PACSR processes. In particular given a process P, we compute the probability that P reaches a set of desired states.

We begin by presenting the definition and some background material for the structure that we use as the model for our reachability analysis: *Labeled Concurrent Markov Chains.*

Definition 4.1 A *Labeled Concurrent Markov Chain* (LCMC) is a tuple

$$\langle S_n, S_p, Act, \longrightarrow_n, \longrightarrow\!\!\!\!\rightarrow_p, s_0 \rangle,$$

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, $\longrightarrow\!\!\!\!\rightarrow_p \subset S_p \times (0,1] \times S_n$ 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 straightforward to see that the SOS rules of PACSR yield transition systems that define LCMC's.

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 $\langle S_n, S_p, Act, \longrightarrow_n, \longrightarrow\!\!\!\!\rightarrow_p, s_0 \rangle$ 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 LCMC's arise by resolving the nondeterministic and probabilistic choices: a *computation* in $T = \langle S_n, S_p, Act, \longrightarrow_n, \longrightarrow\!\!\!\!\rightarrow_p, s_0 \rangle$ 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 $s_i \in S$, $\ell_i \in Act \cup [0,1]$ and $(s_i, \ell_i, s_{i+1}) \in \longrightarrow_p \cup \longrightarrow_n$, for all $0 \leq i$. 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 $\mathsf{time}\,(c) = \#(\ell_1 \ldots \ell_k \restriction \mathcal{D}_R)$, and $\mathsf{last}\, c = s_k$. Note that, by the definition of $\mathsf{Pcomp}(T)$, $\mathsf{last}\, c$ is a nondeterministic state.

To define probability measures on computations, it is necessary to resolve the nondeterminism present. To achieve this, the notion of a scheduler (or adversary) has been employed [17, 6, 14]. 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 $\mathsf{sched} : \mathsf{Pcomp}(T) \mapsto \longrightarrow_n$, such that if $pc \in \mathsf{Pcomp}(T)$ and $\mathsf{sched}(pc) = (s, \alpha, s')$, then $s = \mathsf{last}\, pc$. We use $\mathsf{Sched}(T)$ to denote the set of all schedulers of T. □

Note that $\mathsf{Sched}(T)$ 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, $\sigma(s_0 \ell_1 \ldots \ell_i s_i) = (s_i, \ell_i, s_{i+1})$.

Each scheduler σ induces a probability space [5] on $\mathsf{Scomp}(T, \sigma)$. Let $\mathsf{Scomp}_{fin}(T, \sigma)$ be the set of all partial computations that are a prefix of some $c \in \mathsf{Scomp}(T, \sigma)$, and let $\mathcal{A}^\sigma(T)$ be the sigma-algebra generated by the basic cylinders $C(\omega) = \{c \in \mathsf{Scomp}(T, \sigma) \mid \omega$

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

$$\mathcal{P}(C(\omega)) = \Pi\{\!|\ell_i \in [0,1] \mid 1 \le i \le k|\!\}.$$

We are interested in performing reachability analysis for PACSR processes. In particular we would like to reason about the probability that a PACSR process P reaches a state in which certain actions Φ are enabled. Given a scheduler σ of P we denote this by $\mathsf{Pr}(P \rightsquigarrow \Phi, \sigma)$. Additionally, in order to be able to capture the real-time aspect of PACSR specifications, we offer a time-bounded version of this operator, which we denote by $\mathsf{Pr}(P \rightsquigarrow \Phi, t, \sigma)$. In order to compute these probabilities we introduce the following definitions.

Let $\Phi \subseteq \mathit{Act}$, $t \in \boldsymbol{N}$, and $\sigma \in \mathsf{Sched}(T)$. We define

$$\begin{aligned}
\mathsf{FPaths}(T,\Phi) &= \{c \in \mathsf{Pcomp}(T) \mid \mathsf{last}\, c \xrightarrow{\alpha}_n, \text{ where } \alpha \in \Phi\},\\
\mathsf{FPaths}'(T,\Phi,t) &= \{c \in \mathsf{FPaths}(T,\Phi) \mid \mathsf{time}\,(c) \le t\},\\
\mathsf{SPaths}(T,\Phi,\mathcal{M},\sigma) &= \{c \in \mathsf{Scomp}(T,\sigma) \mid c = c_1c_2, \text{ where } c_1 \in \mathsf{FPaths}(T,\Phi)\},\\
\mathsf{SPaths}'(T,\Phi,\mathcal{M},t,\sigma) &= \{c \in \mathsf{Scomp}(T,\sigma) \mid c = c_1c_2, \text{ where } c_1 \in \mathsf{FPaths}'(T,\Phi,t)\}.
\end{aligned}$$

Thus, $\mathsf{FPaths}(T,\Phi)$ denotes the set of partial computations of T that lead to a state which is capable of performing an action in Φ, while $\mathsf{FPaths}'(T,\Phi,t)$ denotes the subset of such computations that take at most t units of time. Moreover, $\mathsf{SPaths}(T,\Phi,\sigma)$ denotes the set of (infinite) computations in $\mathsf{Scomp}(T,\sigma)$ which are extensions of computations in $\mathsf{FPaths}(T,\Phi,\mathcal{M})$, and similarly for $\mathsf{SPaths}'(T,\Phi,t,\sigma)$. It is easy to see that these sets are measurable in $\mathcal{A}^\sigma(T)$ as, for example, $\mathsf{SPaths}(T,\Phi,\sigma) = \bigcup_\omega C(\omega)$, where $\omega \in \mathsf{FPaths}(T,\Phi) \cap \mathsf{Scomp}_{fin}(T,\sigma)$. The probabilities $\mathsf{Pr}(T,\Phi,\sigma,\epsilon) = \mathcal{P}(\mathsf{SPaths}(T,\Phi,\sigma))$, $\mathsf{Pr}'(T,\Phi,t,\sigma,\epsilon) = \mathcal{P}(\mathsf{SPaths}'(T,\Phi,t,\sigma))$ are given as the smallest solutions to the following sets of equations:

$$\mathsf{Pr}(P,\Phi,\sigma,c) = \begin{cases} 1, & \text{if } P \in S_n, \sigma(c) = (P,\alpha,Q), \alpha \in \Phi\\ \Sigma_Q\, \mathsf{pr}(P,Q) \cdot \mathsf{Pr}(Q,\Phi,\sigma,c\,\mathsf{pr}(P,Q)\,Q), & \text{if } P \in S_p\\ \mathsf{Pr}(Q,\Phi,\sigma,c\,\alpha\,Q), & \text{if } P \in S_n, \sigma(c) = (P,\alpha,Q), \alpha \notin \Phi\\ 0, & \text{otherwise}\end{cases}$$

$$\mathsf{Pr}'(P,\Phi,t,\sigma,c) = \begin{cases} 1, & \text{if } P \in S_n, \sigma(c) = (P,\alpha,Q), \alpha \in \Phi\\ \Sigma_Q\, \mathsf{pr}(P,Q) \cdot \mathsf{Pr}'(Q,\Phi,t,\sigma,c\,\mathsf{pr}(P,Q)\,Q), & \text{if } P \in S_p\\ \mathsf{Pr}'(Q,\Phi,t,\sigma,c\,\alpha\,Q), & \text{if } P \in S_n, \sigma(c) = (P,\alpha,Q), \alpha \notin \Phi \cup \mathcal{D}_R\\ \mathsf{Pr}'(Q,\Phi,t-1,\sigma,c\,\alpha\,Q), & \text{if } P \in S_n, \sigma(c) = (P,\alpha,Q), \alpha \in \mathcal{D}_R - \Phi\\ 0, & \text{otherwise}\end{cases}$$

Thus $\mathsf{Pr}(P,\Phi,\sigma,\epsilon)$ and $\mathsf{Pr}'(P,\Phi,t,\sigma,\epsilon)$ denote the desired $\mathsf{Pr}(P \rightsquigarrow \Phi,\sigma)$ and $\mathsf{Pr}'(P \rightsquigarrow \Phi,t,\sigma)$ respectively.

Given a system P and the set of probabilities $\{\mathsf{Pr}(P \rightsquigarrow \Phi,\sigma) \mid \sigma \in \mathsf{Sched}(P)\}$, the threshold values of this set, that is $\max_{\sigma \in \mathsf{Sched}\,P} \mathsf{Pr}(P \rightsquigarrow \Phi,\sigma)$ and $\min_{\sigma \in \mathsf{Sched}\,P} \mathsf{Pr}(P \rightsquigarrow \Phi,\sigma)$, are of the greatest interest for the verification of the system. The maximum probability being significant in the case where Φ captures undesirable system behavior and the minimum probability if the opposite is true. The same holds for the probabilities $\mathsf{Pr}(P \rightsquigarrow \Phi,t,\sigma)$. We continue by explaining how $\max_{\sigma \in \mathsf{Sched}\,P} \mathsf{Pr}(P \rightsquigarrow \Phi,\sigma)$ can be computed. The rest of the probabilities can be computed similarly.

The maximum value of $\mathsf{Pr}(s \rightsquigarrow \Phi, \sigma)$ over all schedulers is computed as the value of the variable X^s_Φ in the solution for the following set of equations:

$$X^s_\Phi = \begin{cases} \sum_{s \xrightarrow{\pi}_p s'} \pi \cdot X^{s'}_\Phi, & s \in S_p \\ \max(\{X^{s'}_\Phi \mid s \xrightarrow{\alpha}_n s'\}), & \alpha \notin \Phi \cup \mathcal{D}_R \\ 1, & s \xrightarrow{\alpha}_n s', \alpha \in \Phi \\ 0, & \text{otherwise} \end{cases}$$

We can find a solution for this set of equations by solving a linear programming problem, in a way similar to [4]. More precisely, for all equations of the form $X = \max\{X_1, \ldots X_n\}$, we introduce, the set of inequations $X \geq X_i$. Consequently, our aim is to minimize the function $\sum_{s \in S} X^s_\Phi$. Using algorithms based on the ellipsoid method, this problem can be solved in time polynomial to the number of variables (see, e.g. [9]).

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 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.

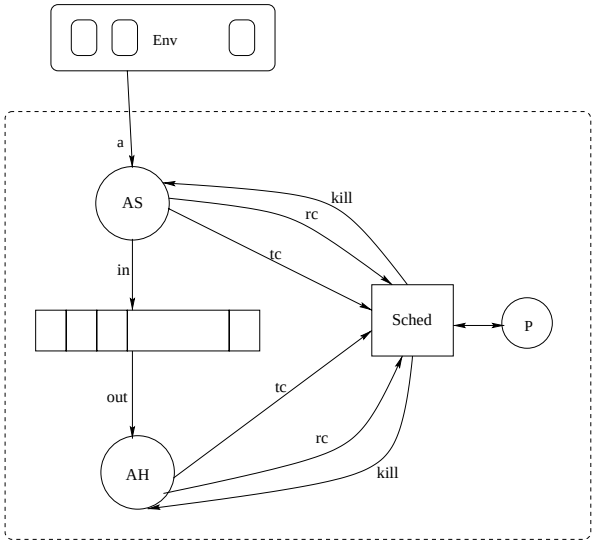

Figure 2: The structure of the application

Specification. The structure of the 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 monitor itself consists of two processes: the alarm sampler, AS, which periodically samples alarms and places them in a bounded-size buffer, and the alarms handler, AH, which removes alarms from the buffer and processes them. 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 a process 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, 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 that we analyze is the reachability of a state where overflow in the alarm buffer is possible: we compute the maximum probability of overflow in the alarm buffer. 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). Thus on comparing the systems we are able to show that one is better than the other, in terms of the probability of buffer overflow.

The specification of is represented by the following collection of processes:

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

The system in its initial state is represented by the 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 a 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. Upon the failure of resource r_i, the 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, each representing the buffer with i alarms. The capacity of the buffer, n, is another parameter of the specification. Each process B_i, except B_n, can accept a new alarm and become B_{i+1}. 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 information on 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. 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, thus making the scheduler begin the next scheduling cycle. Finally, the scheduler is responsible for killing the process in question should it exceed the maximum-allocated time, 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. Note that 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, begins by checking how many alarms exist 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 slot. Thus $\mathbf{n}(i) = \max(i, a_{max})$, with a_{max} being a parameter of the specification. $\mathbf{pt}(i)$ is the time it takes to process i alarms. Note that there is no need for AH to accept *kill* signal from the scheduler, since we limit the number of alarms and the handler always completes within the allocated slot.

Finally, process P represents a low-priority background process having the 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 $p = \mathsf{pr}(r)$.

Verification. We considered two versions of the system. In both cases the probability of an alarm is 0.9. The first version 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, we assumed that the component runs on a 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 buffer size has been doubled, i.e.the buffer can hold 6 alarms and the workload of the component has been increased by allowing up to two alarms per time unit.

We have checked reachability of the set of states, S_{of}, from which overflow is possible,

Time units	S_1 (probability)	S_2 (probability)
10	$2.02\,10^{-6}$	$2.18\,10^{-10}$
20	$5.11\,10^{-6}$	$5.46\,10^{-10}$
30	$9.21\,10^{-6}$	$9.83\,10^{-10}$
40	$1.23\,10^{-5}$	$1.31\,10^{-9}$
50	$1.54\,10^{-5}$	$1.64\,10^{-9}$
60	$1.95\,10^{-5}$	$2.08\,10^{-9}$
70	$2.26\,10^{-5}$	$2.40\,10^{-9}$
80	$2.56\,10^{-5}$	$2.73\,10^{-9}$
90	$2.97\,10^{-5}$	$3.17\,10^{-9}$
100	$3.28\,10^{-5}$	$3.50\,10^{-9}$

Table 2: Results of analysis

that is states where action $\overline{overflow}$ is enabled. We have done this for various values of t. Table 2 shows, for each t, the probability of reaching a state in S_{of}, within t time units. We can see that for all checked intervals the faster version of the system performs better than the other one, despite the increased workload.

6 Conclusions and Future Work

We have presented PACSR, a process-algebraic formalism for specification of resource-oriented real-time systems. The formalism allows one to model resource failures and perform probabilistic analysis of the system's behavior. We have performed probabilistic reachability analysis of PACSR that allows us to compute the probability of occurrence of undesirable events. 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 working to implement PACSR as part of the PARAGON toolset [3], designed to handle large-scale specifications. At the same time, we are extending the analysis method from simple probabilistic reachability to model checking for a probabilistic temporal logic.

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] 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.

[3] 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.

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

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

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

[7] H. Hansson and B. Jonsson. A logic for reasoning about time and probability. *Formal Aspects of Computing*, 6:512–535, 1994.

[8] M. Hennessy and T. Regan. A process algebra for timed systems. Technical Report 5/91, CS, University of Sussex, 1991.

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

[10] 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.

[11] R. Milner. *Communication and Concurrency*. Prentice Hall Intl., 1989.

[12] F. Moller and C. Tofts. A temporal calculus of communicating systems. In *Proceedings of CONCUR'90*. LNCS 458, 1990.

[13] 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.

[14] 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.

[15] C. Tofts. Processes with probabilities, priorities and time. *Formal Aspects of Computing*, 4:536–564, 1994.

[16] R. J. van Glabbeek, S. A. Smolka, and B. Steffen. Reactive, generative and stratified models of probabilistic processes. *Information and Computation*, 121(1):59–80, 15 Aug. 1995.

[17] 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.

[18] W. Yi. *A Calculus of Real Time Systems*. PhD thesis, Chalmers University of Technology, 1991.