Available online at www.sciencedirect.com
ScienceDirect
The Journal of Logic and Algebraic Programming 72 (2007) 98–122
THE JOURNAL OF LOGIC AND ALGEBRAIC PROGRAMMING
www.elsevier.com/locate/jlap

Resources in process algebra ☆

Insup Lee [a], Anna Philippou [b,*], Oleg Sokolsky [a]

[a] *Department of Computer and Information Science, University of Pennsylvania, 200 South 33rd Street, Philadelphia, PA, USA*
[b] *Department of Computer Science, University of Cyprus, P.O. Box 20537, Nicosia, Cyprus*

Abstract

The algebra of communicating shared resources (ACSR) is a timed process algebra which extends classical process algebras with the notion of a *resource*. It takes the view that the timing behavior of a real-time system depends not only on delays due to process synchronization, but also on the availability of shared resources. Thus, ACSR employs resources as a basic primitive and it represents a real-time system as a collection of concurrent processes which may communicate with each other by means of instantaneous events and compete for the usage of shared resources. Resources are used to model physical devices such as processors, memory modules, communication links, or any other reusable resource of limited capacity. Additionally, they provide a convenient abstraction mechanism for capturing a variety of aspects of system behavior.

In this paper we give an overview of ACSR and its probabilistic extension, PACSR, where resources can fail with associated failure probabilities. We present associated analysis techniques for performing qualitative analysis (such as schedulability analysis) and quantitative analysis (such as resource utilization analysis) of process-algebraic descriptions. We also discuss mappings between probabilistic and non-probabilistic models, which allow us to use analysis techniques from one algebra on models from the other.

Keywords: Process algebra; Real-time systems; Schedulability analysis; Resource modeling

1. Introduction

Modeling timing aspects of system behavior has a long history in process-algebraic formalisms. In this paper, we advocate the use of resources in the modeling of real-time systems as a means of arriving at simpler and more faithful models.

Process algebras, such as CCS [26], CSP [17], and ACP [6], have been developed to describe and analyze communicating, concurrently executing systems. They are based on the premises that the two most essential notions in understanding complex dynamic systems are concurrency and communication [26]. The Algebra of Communicating Shared Resources (ACSR [21]) introduced by Lee et al., is a timed process algebra which can be regarded as an extension of CCS. The timing behavior of a real-time system depends not only on delays due to process synchronization,

☆ This research was supported in part by ARO DAAD19-01-1-0473, ARO W911NF-05-1-0182, NSF CCR-0209024, NSF CNS-0509327, and NSF CNS-0509143.
* Corresponding author.
E-mail addresses: lee@cis.upenn.edu (I. Lee), annap@cs.ucy.ac.cy (A. Philippou), sokolsky@saul.cis.upenn.edu (O. Sokolsky).

1567-8326/$ - see front matter © 2007 Elsevier Inc. All rights reserved.
doi:10.1016/j.jlap.2007.02.005

but also on the availability of shared resources. Most real-time process algebras adequately capture delays due to process synchronization; however, they abstract out resource-specific details by assuming idealistic operating environments. On the other hand, scheduling and resource-allocation algorithms used for real-time systems ignore the effect of process synchronization except for simple precedence relations between processes. The ACSR algebra provides a formal framework that combines the areas of process algebra and real-time scheduling and, thus, can help us to reason about systems that are sensitive to deadlines, process interaction and resource availability.

The computation model of ACSR is based on the view that a real-time system consists of a set of communicating processes that use shared resources for execution, and synchronize with one another. The notion of real time in ACSR is quantitative and discrete, and is accommodated using the concept of timed actions. Executing a timed action requires access to a set of serially reusable resources and takes one unit of time. Idling of a process is treated as a special timed action that consumes no resources. The execution of a timed action is subject to the availability of the resources used in it. The contention for resources is arbitrated according to the priorities of competing actions. To ensure the uniform progression of time, processes execute timed actions synchronously. Similar to CCS, the execution of an event is instantaneous and never consumes any resource. The notion of communication is modeled using events through the execution of complementary events, which are then converted into an internal event. Processes execute events asynchronously except when two processes synchronize through matching events. Priorities are also used to direct the choice when several events are possible at the same time. Thus, the concurrency model of ACSR includes interleaving semantics for events as well as lock-step parallelism for timed actions.

Naturally, the proposed framework inherits all the attractive features of process-algebraic approaches. To begin with, it facilitates the modular and hierarchical specification of real-time systems. It provides a small set of operators that can be used to build a large specification in a bottom-up fashion. It supports a number of constructs that are unique to ACSR which provide the exception, timeout and interrupt features and abstraction mechanisms for resource-related information. Furthermore, ACSR is accompanied by equivalence relations that are congruence relations. This feature enables on one hand the hierarchical and stepwise development of large-scale systems and, on the other hand, the verification of complex systems by reasoning about their parts.

The notion of a resource, which is already important in the specification of real-time systems, additionally provides a convenient abstraction mechanism for probabilistic aspects of system 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. This direction of work was investigated in the context of the process algebra PACSR [30], where the ACSR framework was extended with the possibility of resource failures which happen with a given probability. 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 interesting effect of associating probabilities with resources, is that the specification of a process does not involve probabilities directly. Failure probabilities of individual resources are defined separately and are used only during analysis. This makes specifications 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.

In addition to PACSR, ACSR has been extended into a family of other process algebras. Extensions and variations include GCSR [5] that allows the visual representation of ACSR processes, Dense-time ACSR [9] that includes a more general notion of time, ACSR-VP [19] that includes a value-passing capability, and P^2ACSR [35] that allows to specify power-constrained systems. MCSR [23], an extension of ACSR with multicapacity resources, allows us to consider memory constraints. The PARAGON toolset [34] provides tool support for modeling and analysis using these formalisms.

In this paper we review some of the main results obtained in this line of work and we demonstrate the methodologies obtained for the specification and analysis of real-time systems. In particular, we provide a comprehensive presentation of ACSR and its probabilistic descendant PACSR from the perspective of modeling and reasoning about resource-constrained real-time systems. In doing this, we highlight the design choices behind the main language features and subtle interactions that exist between them which are important for system specification and verification. In the context of ACSR, we discuss bisimulation-checking and we introduce an HML logic with *until* featuring regular expressions over observables as parameters. Furthermore, we discuss a methodology for performing schedulability analysis of real-time systems and we provide a compositional result which allows us to trace the source of undesirable deadlocks

in system models. In the context of PACSR, we give emphasis on performing quantitative analysis of process-algebraic descriptions by model-checking logical properties. Finally, we propose property-preserving mappings between ACSR and PACSR. These mappings highlight the natural relationship between the two process algebras and allow us to apply analysis techniques of one to the other. Interestingly, these mappings preserve bisimulation equivalence, which implies that two probabilistic systems are equivalent if their non-probabilistic projections exhibit bisimilar behavior, and *vice versa*.

Related work in the area of resource handling in embedded real-time systems falls into two categories. On the one hand, the importance of the issue has been long realized by practitioners and a number of model-based, albeit informal, approaches have been published. We mention [25,33,18,4,2] among many others. This modeling approach is primarily concerned with high-level performance evaluations. Models in this category address concrete resources, such as processors, communication channels and shared data. Modeling is done in terms of formulae that relate the rate of resource use to the response time of a real-time task. The models are used either in a simulation environment to predict quality-of-service properties of the real-time system or dynamically as part of access control mechanisms. For example, the authors of [25] consider the system that coordinates the use of real-time communication channels with the availability of processor resource.

By contrast, several formal approaches have emerged that aim at scheduling of sets of tasks under constraints. Models in this category aim at a much more detailed behavioral representation of the system behavior, and are intended to be analysed using state-space exploration techniques such as model checking. For the most part, these approaches consider only timing constraints and do not introduce the notion of a resource, implicitly considering the processor as the only shared resource in the system. For example, the authors of [15] propose a formalism that allows us to model preemption in asynchronous real-time systems. In [7], the authors limit themselves to fixed-priority scheduling approaches, which allow them not to consider preemption directly, accounting for it in the worst-case computation time. The formalism of [10] provides a general scheme for handling preemption of processes due to resource contention, but the scheduling rules have to be encoded by modifying transition rules in the formalism (effectively creating a custom formalism for each scheduling policy). A different approach is taken in [1], where the authors view the scheduling activity as control and use controller synthesis techniques to model scheduled real-time systems. Another approach based on timed automata is presented in [3], where priorities are encoded using bounded integer variables. In all of these approaches, adding other kinds of resources requires a major extension to the formalism. PAMR [28] and PARS [27] are process algebras that, like the ACSR family, employ resources as language primitives. In [27], a dense-time process algebra is defined for performing schedulability analysis of real-time systems where two separate theories are given for specifying resource-consuming processes (e.g. tasks) and resource providers (schedulers). Compared to our approach, the need to specify schedulers explicitly and encode process priorities into schedulers in PARS results in lower-level models, which we expect will be harder to analyze. To the best of our knowledge, no tool support exists for PARS. In a radically different approach, PAMR [28] does not consider the timing aspects of resource sharing. Instead, the process algebra captures the utility that processes derive from resource use. Processes with different utility functions can exchange resources with each other in order to maximize the overall utility of the system. The authors of [12] approach sharing of consumable resources, such as power, from a game-theoretic perspective. The notion of a *resource interface* captures resource consumption by a process and assumptions about resource consumption by the environment of the process. Compatible interfaces can operate together without violating each other's assumptions. In this case, as well, timing of resource use is not explicitly considered.

The rest of the paper is organized as follows. Section 2 describes the basic computation model of ACSR and overviews its syntax and semantics. It also describes a simple scheduling example. Section 3 explains PACSR and extends the same scheduling example with probabilistic resource failure. Section 4 discusses mappings between ACSR and PACSR, and, finally, Section 5 concludes the paper.

2. ACSR

In the ACSR algebra there are two types of actions: those which consume time and those which are instantaneous. The time-consuming actions represent one "tick" of a global clock. These actions may also represent the consumption of resources, e.g. CPUs, devices, memory, batteries in the system configuration. In contrast, the instantaneous actions provide a synchronization mechanism between concurrent processes.

Timed actions. We consider a system to be composed of a finite set of serially reusable resources, denoted by $\mathcal{R}$. An action that consumes one "tick" of time is drawn from the domain $\boldsymbol{P}(\mathcal{R} \times \mathbb{N})$ with the restriction that each resource be represented at most once. As an example, the singleton action $\{(r, p)\}$ denotes the use of some resource $r \in \mathcal{R}$ running at priority level p. The action $\emptyset$ represents idling for one time unit since no resource is consumed.

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

Instantaneous events. Instantaneous actions, called *events*, provide the basic synchronization mechanism in the process algebra. We assume a set of channels L. An event is denoted by a pair (a, p), where a is the *label* of the event, and $p \in \mathbb{N}$ is its *priority*. Labels are drawn from the set $\mathcal{L} = \{a?, a! \mid a \in L\} \cup \{\tau\}$. We say that $a?$ and $a!$ are *inverse* labels. As in CCS, the special identity label τ arises when two events with inverse labels are executed in parallel.

We use $\mathcal{D}_E$ to denote the domain of events, and let e range over $\mathcal{D}_E$. We use $l(e)$ to represent the label of the event e. The entire domain of actions is $Act = \mathcal{D}_R \cup \mathcal{D}_E$, and we let α and β range over Act.

2.1. Syntax and semantics

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

$$P ::= \text{NIL} \mid (a, n).\, P \mid A{:}P \mid P + P \mid P \| P \mid P \backslash F \mid [P]_I \mid P \,\triangle_t^a\, (P, P, P) \mid b \rhd P \mid C$$

We write Proc for the set of ACSR processes. The above operators are given precise meaning via a family of rules that define the labeled transition relations on processes. The semantics is defined in two steps. First, we develop the *unconstrained* transition system, where a transition is denoted as $P \xrightarrow{\alpha} P'$. Within "$\rightarrow$" no priority arbitration is made between actions. We subsequently refine "$\rightarrow$" to define the prioritized transition system, "$\rightarrow_\pi$". The non-prioritized transition relation is given in Table 1. (Note that the symmetric rules of (Sum), (Par1) and (Par2) have been omitted.)

We proceed to discuss each of the operators and their associated rules. The process NIL represents the inactive process. It has no rules associated with it thus it cannot perform any steps. The process $(a, n).\, P$ executes the instantaneous event (a, n) and proceeds to P. The process $A{:}P$ executes a resource-consuming action during the first time unit and proceeds to P. The process $P + Q$ represents a non-deterministic 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. Specifically, rule (Par2) represents event synchronization that transforms matching observable events into an internal event τ. Note that priorities of both events are involved in computing the priority of the τ event. Different functions can be used to compute the resulting priority. This function has to be symmetric in its arguments to reflect that synchronization is symmetric; further, the function has to be monotonic in both arguments and satisfy $f(m, n) \geqslant m, n$. The function $\max(m, n)$ could have been also used instead of addition. As stipulated by rule (Par3), for a timed action to take place, all concurrent components must simultaneously engage in a timed action thereby ensuring the uniform passage of time. Note that the side condition of the rule requires that at most one process may use a resource during any time step. In $P \backslash F$, where $F \subseteq L$, the scope of channels in F is restricted to process P and, thus, components of P may use these labels to interact with one another but not with P's environment.

The resource closure operator, $[P]_I$, $I \subseteq \mathcal{R}$, describes a method for restricting the scope of resources in I, within process P. Specifically, when a process P is embedded in a closed context such as $[P]_I$, we ensure that there is no further sharing of the resources in I. For every time-consuming action A performed by P utilizing less than the full resource set I, the action is augmented with $(r, 0)$ pairs for each resource $r \in I - \rho(A)$. Instantaneous events are not affected. As we will discuss below the use of this operator in system models is important for the correct application of the prioritized transition relation. As an example consider process $P \stackrel{\text{def}}{=} \emptyset : P_1 + \{(cpu, 1)\} : P_2$, $I = \{cpu\}$. Then:

$$[P]_I \xrightarrow{\{(cpu,0)\}} [P_1]_I, \quad [P]_I \xrightarrow{\{(cpu,1)\}} [P_2]_I$$

The scope construct, $P \triangle_t^a\, (Q, R, S)$, binds 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

Table 1
The non-prioritized relation

(Act1)	$e.P \xrightarrow{e} P$	(Act2)	$A : P \xrightarrow{A} P$
(Sum)	$\dfrac{P_1 \xrightarrow{\alpha} P}{P_1 + P_2 \xrightarrow{\alpha} P}$	(Par1)	$\dfrac{P_1 \xrightarrow{e} P_1'}{P_1 \| P_2 \xrightarrow{e} P_1' \| P_2}$
(Par2)	$\dfrac{P_1 \xrightarrow{(a?,n)} P_1',\ P_2 \xrightarrow{(a!,m)} P_2'}{P_1 \| P_2 \xrightarrow{(\tau,n+m)} P_1' \| P_2'}$	(Cond)	$\dfrac{P \xrightarrow{\alpha} P'}{true \triangleright P \xrightarrow{\alpha} P'}$
(Par3)	$\dfrac{P_1 \xrightarrow{A_1} P_1',\ P_2 \xrightarrow{A_2} P_2'}{P_1 \| P_2 \xrightarrow{A_1 \cup A_2} P_1' \| P_2'},\quad \rho(A_1) \cap \rho(A_2) = \emptyset$		
(Res1)	$\dfrac{P \xrightarrow{e} P',\ l(e) \notin F}{P \backslash F \xrightarrow{e} P' \backslash F}$	(Res2)	$\dfrac{P \xrightarrow{A} P'}{P \backslash F \xrightarrow{A} P' \backslash F}$
(Cl1)	$\dfrac{P \xrightarrow{A_1} P',\ A_2 = \{(r,0) \mid r \in I - \rho(A_1)\}}{[P]_I \xrightarrow{A_1 \cup A_2} [P']_I}$	(Cl2)	$\dfrac{P \xrightarrow{e} P'}{[P]_I \xrightarrow{e} [P']_I}$
(Sc1)	$\dfrac{P \xrightarrow{e} P',\ l(e) \neq b!,\ t > 0}{P \overset{b}{\triangle}_t (Q, R, S) \xrightarrow{e} P' \overset{b}{\triangle}_t (Q, R, S)}$	(Sc2)	$\dfrac{P \xrightarrow{(b!,n)} P',\ t > 0}{P \overset{b}{\triangle}_t (Q, R, S) \xrightarrow{(\tau,n)} Q}$
(Sc3)	$\dfrac{P \xrightarrow{A} P',\ t > 0}{P \overset{b}{\triangle}_t (Q, R, S) \xrightarrow{A} P' \overset{b}{\triangle}_{t-1} (Q, R, S)}$	(Sc4)	$\dfrac{R \xrightarrow{\alpha} R',\ t = 0}{P \overset{b}{\triangle}_t (Q, R, S) \xrightarrow{\alpha} R'}$
(Sc5)	$\dfrac{S \xrightarrow{\alpha} S',\ t > 0}{P \overset{b}{\triangle}_t (Q, R, S) \xrightarrow{\alpha} S'}$	(Rec)	$\dfrac{P \xrightarrow{\alpha} P', C \overset{\text{def}}{=} P}{C \xrightarrow{\alpha} P'}$

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 $a!$, where $a \in L$, then control is delegated to Q, the success-handler. Else, if P fails to terminate within time t then control proceeds to R. Finally, throughout execution of this process, P may be interrupted by process S. As an example consider the task specification $T \overset{\text{def}}{=} R \overset{a}{\triangle}_{10} (SH, EH, IN)$ where

$$R \overset{\text{def}}{=} (in?, 1).(a!, 2).\text{NIL} + \emptyset : R$$
$$SH \overset{\text{def}}{=} (ack!, 1).T$$
$$EH \overset{\text{def}}{=} (nack!, 1).T$$
$$IN \overset{\text{def}}{=} (kill?, 3).\text{NIL}$$

This task awaits for an input request to arrive for a 10 time-unit period. If such an event takes place the process signals the arrival on channel a and, subsequently, the success handler process, SH, acknowledges the event. If the deadline elapses without the appearance of the event, the task signals the lack of input on channel $nack$. Finally, at any point during its computation, the task may receive a signal on channel $kill$ and halt its computation. According to the rules for scope, process $R \overset{a}{\triangle}_{10} (SH, EH, IN)$ may engage in the following actions:

$$R \overset{a}{\triangle}_{10} (SH, EH, IN) \xrightarrow{\emptyset} R \overset{a}{\triangle}_{9} (SH, EH, IN)$$

$$R \triangle^{a}_{10} (SH, EH, IN) \xrightarrow{(in?,1)} ((a!, 2).\mathrm{NIL}) \triangle^{a}_{10} (SH, EH, IN)$$
$$R \triangle^{a}_{10} (SH, EH, IN) \xrightarrow{(kill?,3)} \mathrm{NIL}$$

Furthermore, note that:

$$R \triangle^{a}_{0} (SH, EH, IN) \xrightarrow{(nack!,1)} T$$
$$((a!, 2).\mathrm{NIL}) \triangle^{a}_{10} (SH, EH, IN) \xrightarrow{(\tau,2)} SH$$

Process $b \rhd P$ represents the conditional process: it performs as P if boolean expression b evaluates to *true* and as NIL otherwise. Process constant C with process definition $C \stackrel{\text{def}}{=} P$ allows standard recursion.

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

$$P_t = t < 2 \rhd (a_t, t).\, P_{t+1}, t \in \{0 \ldots 2\}$$

is equivalent to the following three processes:

$$P_0 = (a_0, 0).\, P_1, \quad P_1 = (a_1, 1).\, P_2, \quad P_2 = \mathrm{NIL}$$

The prioritized transition system is based on *preemption*, which incorporates our treatment of priority. This is based on a transitive, irreflexive, binary relation on actions, $\prec$, called the *preemption relation*. If $\alpha \prec \beta$, for two actions α and β, we say that α *is preempted by* β. Then, in any process, if there is a choice between executing either α or β, β will always be executed. We refer to [8] for the precise definition of $\prec$. Here, we briefly describe the three cases for which $\alpha \prec \beta$ is deemed to be true by the definition.

- The first case is for two timed actions α and β which compete for common resources. Here, it must be that the preempting action β employs all of its resources at priority level at least the same as α. Also, β must use at least one resource at a higher level. It is still permitted for α to contain resources not in β but all such resources must be employed at priority level 0. Otherwise, the two timed actions are incomparable. Note that β cannot preempt an action α consuming a *strict subset* of its resources at the same or lower level. This is necessary for preserving the compositionality of the parallel operator. For instance, $\{(r_1, 2), (r_2, 0)\} \prec \{(r_1, 7)\}$ but $\{(r_1, 2), (r_2, 1)\} \not\prec \{(r_1, 7)\}$ and $\{(r_1, 2)\} \not\prec \{(r_1, 7), (r_2, 1)\}$.
- The second case is for two events with the same label. Here, an event may be preempted by another event with the same label but a higher priority. For example, $(\tau, 1) \prec (\tau, 2)$, $(a, 2) \prec (a, 5)$, and $(a, 1) \not\prec (b, 2)$ if $a \neq b$.
- The third case is when an event and a timed action are comparable under "$\prec$". Here, if $n > 0$ in an event (τ, n), we let the event preempt any timed action. For instance, $\{(r_1, 2), (r_2, 5)\} \prec (\tau, 2)$, but $\{(r_1, 2), (r_2, 5)\} \not\prec (\tau, 0)$. This case ensures that interactions happen as soon as both parties are ready. The case of zero priority is treated as a special case to enable the modeling of timing uncertainty.

We define the prioritized transition system "$\rightarrow_\pi$", which simply refines "$\rightarrow$" to account for preemption.

Definition 1. The labeled transition system "$\rightarrow_\pi$" is defined as follows: $P \xrightarrow{\alpha}_\pi P'$ if and only if (1) $P \xrightarrow{\alpha} P'$ is an unprioritized transition, and (2) there is no unprioritized transition $P \xrightarrow{\beta} P''$ such that $\alpha \prec \beta$.

We conclude this section by discussing some important characteristics of the language:

Concurrency semantics. Beginning with instantaneous events, we may see that ACSR adopts the CCS-style of communication, that is, processes may execute such events asynchronously and independently with the exception of two processes synchronizing on complementary events, leading to an internal event taking place. On the other hand, to ensure the uniform progress of time, timed transitions are *synchronous*, that is, for a timed action to take place in a composition of parallel processes, all components must simultaneously engage in a timed action (possibly idling). This is made explicit in rule (Par3). Note that the side condition of the rule requires that at most one process may use a resource during any time step. A consequence of this side condition is that whenever two, or more, concurrent processes are competing for the use of the same resource and neither is willing to engage in alternative behaviour,

then the system is deadlocked. This is because, by (Par3), no timed action is allowed to take place. This fact plays a significant role in the algebra as it is exploited for performing schedulability analysis.

Deadlock vs successful termination. In timed process algebras it is often convenient to distinguish between a completed process and a deadlocked process. The semantic difference between the two is that the completed process cannot perform any actions but allows time to progress whereas the deadlocked process does not. The ACSR process NIL corresponds to the deadlocked process, whereas the completed process can be defined as $Idle \stackrel{\text{def}}{=} \emptyset : Idle$. By using the deadlocked process NIL, it is possible to model abnormal conditions. Note that presence of the deadlocked process as a component of a parallel composition causes a timelock in the system due to the synchronous nature of the parallel composition with respect to time passing.

Time passage. An important observation to make about the semantic rules is that the only "source" of time progress in the language is the action-prefix operator. That is, time progress has to be explicitly encoded into the model by the designer. This feature forces the designer to think carefully about time progress and may result in fewer unexpected behaviors in a complex model. For example, consider two tasks competing for the use of a resource *cpu*, and, suppose that the first needs to urgently employ the resource (or else it misses its deadline) while the second is willing to wait indefinitely until the resource becomes available. These two tasks can be modeled as follows:

$$T_1 \stackrel{\text{def}}{=} \{(cpu, 1)\} : Idle, \qquad T_2 \stackrel{\text{def}}{=} \{(cpu, 1)\} : Idle + \emptyset : T_2$$

and, by rule (Par3), $T_1 \| T_2 \xrightarrow{\{(cpu,1)\}} Idle \| T_2$. On the other hand, if both tasks were urgent in using the resource, then the idling option $\emptyset : T_2$ would be absent from the definition of T_2, and the composition of the two processes would be a deadlocked system. This close correspondence between the presence of deadlocks and tasks missing their deadline is taken advantage of for schedulability analysis of ACSR processes.

Maximal progress. It is often the case that a process has the option between idling indefinitely or performing some other instantaneous or timed step. Although such idling behaviors may emerge in system models, they are generally considered unrealistic and should be avoided. Most timed process algebras include some means of ensuring progress in an execution. In ACSR, this is achieved by a combination of the closure operator and priority-based preemption relation. In particular, we may see that by closing a system by the set of its resources we enforce progress to be made.

For example, consider process T_2 above. We may observe that this process can choose to idle indefinitely even if resource *cpu* becomes available. This behavior, however, does not comply with our intention of defining a process that may delay *until* resource *cpu* becomes available in which case it makes progress by consuming the resource. If we consider the system $Sys \stackrel{\text{def}}{=} [T_1 \parallel T_2]_{\{cpu\}}$ we initially obtain the behavior $Sys \xrightarrow{(cpu,1)}_{\pi} [Idle \parallel T_2]_{\{cpu\}}$. Subsequently, this process enables the transitions $[Idle \parallel T_2]_{\{cpu\}} \xrightarrow{(cpu,0)} [Idle \parallel T_2]_{\{cpu\}}$ and $[Idle \parallel T_2]_{\{cpu\}} \xrightarrow{(cpu,1)} [Idle \parallel Idle]_{\{cpu\}}$. In the prioritized transition system, the former transition is pruned by the preemption relation, allowing only the latter progress-making transition.

2.2. Analysis of real-time systems in ACSR

ACSR models can be analyzed in several ways. Similar to other behavioral formalisms, equivalence checking and model checking are common ways of establishing functional and timing correctness. In the former case, a detailed model is checked for equivalence with a more abstract model that represents system requirements. In the latter case, system requirements are expressed as formulae in a temporal logic and a model-checking algorithm is used to verify that the model satisfies these formulae.

In addition, ACSR allows us to perform schedulability analysis of a real-time system model. Resource-sharing execution of concurrent processes or threads in a real-time system is typically controlled by a scheduler that follows a particular scheduling discipline, such as Rate Monotonic, RM, and Earliest Deadline First, EDF. Different properties of scheduling algorithms may result in violations of timing constraints of a system under one scheduling discipline, while another scheduling discipline may succeed. Schedulability analysis determines whether the set of processes in a real-time system can be scheduled, by any scheduler, or with respect to a given scheduling discipline.

2.2.1. Resource-sensitive process equivalences

Equivalence between ACSR processes is based on the concept of *bisimulation* [29,26] which compares the computation trees of two processes. Two processes are bisimilar if, for each step of one, there is a matching, possibly multiple, step of the other, leading to bisimilar states. Below, we introduce three well-known such relations on which we base our study. First, we recall some useful definitions. We say that Q is a *derivative* of P, if there are $\alpha_1, \ldots, \alpha_n \in Act$, $n \geqslant 0$, such that $P \xrightarrow{\alpha_1}_\pi \cdots \xrightarrow{\alpha_n}_\pi Q$, in which case we also write $P \xrightarrow{\alpha_1 \ldots \alpha_n}_\pi Q$. Moreover, given $\alpha \in Act$ we write $\Rightarrow_\pi$ for the reflexive and transitive closure of $\xrightarrow{\tau}_\pi$, $\stackrel{\alpha}{\Rightarrow}_\pi$ for the composition $\Rightarrow_\pi \xrightarrow{\alpha}_\pi \Rightarrow_\pi$, and $\stackrel{\hat{\alpha}}{\Rightarrow}_\pi$ for $\Rightarrow_\pi$ if $\alpha = \tau$ and $\stackrel{\alpha}{\Rightarrow}_\pi$ otherwise.

Definition 2

(1) *Strong bisimilarity* is the largest symmetric relation, denoted by $\sim$, such that, if $P \sim Q$ and $P \xrightarrow{\alpha}_\pi P'$, there exists Q' such that $Q \xrightarrow{\alpha}_\pi Q'$ and $P' \sim Q'$.

(2) *Weak bisimilarity* is the largest symmetric relation, denoted by $\approx$, such that, if $P \approx Q$ and $P \xrightarrow{\alpha}_\pi P'$, there exists Q' such that $Q \stackrel{\hat{\alpha}}{\Rightarrow}_\pi Q'$ and $P' \approx Q'$.

(3) *Branching bisimilarity* is the largest symmetric relation, denoted by $\simeq$, such that, if $P \simeq Q$ and $P \xrightarrow{\alpha}_\pi P'$, either (1) $\alpha = \tau$ and $P' \simeq Q$, or (2) there exist Q', Q'' such that $Q \Rightarrow_\pi Q'' \xrightarrow{\alpha}_\pi Q'$ and $P \simeq Q''$, $P' \simeq Q'$.

Strong bisimilarity is an equivalence relation and a congruence with respect to the ACSR operators [8]. Weak and branching bisimilarities are equivalence relations though not congruences for ACSR. We may obtain a weak bisimulation congruence and a branching bisimulation congruence in the usual way by appropriately handling initial actions of processes [26].

Algorithms for checking strong and weak bisimulation for finite-state ACSR processes have been implemented in the VERSA toolset, thus allowing the verification of ACSR specifications. We refer the interested reader to [22] for examples of using weak bisimulation for the verification of railroad-crossing systems.

2.2.2. Model checking of ACSR processes

We have defined a temporal logic for expressing properties of ACSR processes and a model-checking algorithm to determine whether a finite-state ACSR process satisfies a given formula.

We use an extension of the Hennessy–Milner logic (HML) with *until*, which was proposed in [14]. The *until* operator of [14] is parameterized by a single observable event. When one wants to express a complex temporal behavior that involves a number of events, it is necessary to resort to multiple nested *until* operators, which makes the formula hard to read. In order to improve the usability of the logic, we introduce an extended *until* operator that is parameterized by a *regular expression*. The regular expression represents the set of observable behaviors that are admissible along a path within the scope of the *until* operator. Discussion in Section 3 revisits the design decisions made in the definition of this logic and provides additional justification for the regular expressions in the *until* operator. We point out that the introduced logic, similarly to the logic of [14], characterizes *branching bisimulation*. However, we do not go into the details in this paper.

Observables. Formulae of the logic will be interpreted over labeled transition systems generated by ACSR processes. Formulae, therefore, will refer to the labels of the transition systems, that is, events and actions. However, events and actions carry with them the values of their dynamic attributes, which are not meaningful in the logical context. Therefore, primitive constructs used in the logical formulae are event labels, and sets of resources, as action labels. Given an event e, we write $\mathsf{obs}(e) = \ell(e)$ and, given a timed action $A = \{(r_1, p_1), \ldots, (r_n, p_n)\}$, we write $\mathsf{obs}(A) = \{r_1, \ldots, r_n\}$.

Regular expressions. We use the standard definition of regular expressions using the following grammar:

$$\Phi ::= l \mid V \mid \Phi\Phi \mid \Phi + \Phi \mid \Phi*,$$

where $l \in \mathcal{L}$, $V \subseteq \mathcal{R}$. As usual, we understand a regular expression as a set of strings in the alphabet of event and action labels. Operators are concatenation, union, and Kleene star. A *derivative* of Φ is a regular expression Φ' such that whenever a string $\sigma' \in \Phi'$, there exists a string σ such that $\sigma\sigma' \in \Phi$.

The logic. The syntax of the logic $\mathcal{L}_{HMLu}$ is given by the following grammar.

$$f \quad ::= \quad tt \quad \mid \quad \neg f \quad \mid \quad f \wedge f' \quad \mid \quad f\langle\Phi\rangle f' \quad \mid \quad f\langle\Phi\rangle^t f'.$$

The atomic proposition tt represents a trivial property that is always true. Logical connectives have the classical interpretations. The logic introduces two kinds of *until* operators which are used to specify properties of a path through a transition system. Both operators are parameterized with a regular expression that specifies the observable behavior expected along the path. In addition, the second *until* operator introduces a time bound on the length of the path to enable us to specify quantitative timing properties of the system.

The semantics for $\mathcal{L}_{HMLu}$ is given with respect to a given labeled transition system $T = (S, Act, \longrightarrow_\pi, s_0)$. In order to define the semantic function, we introduce the following definitions. A *computation* in T is a sequence $c = s_0\,\alpha_1\,s_1 \ldots \alpha_n\,s_n$, such that $s_i \in S, \alpha_{i+1} \in Act$ and $(s_i, \alpha_{i+1}, s_{i+1}) \in \longrightarrow_\pi$, for all $0 \leqslant i < n$. We define $\mathsf{trace}\,(c) = \mathsf{obs}(\alpha_1) \ldots \mathsf{obs}(\alpha_n)\restriction (\mathcal{L} \cup 2^{\mathcal{R}} - \{\tau\})$, $\mathsf{states}\,(c) = \{s_0, \ldots s_n\}$, $\mathsf{time}\,(c) = \#(\alpha_1 \ldots \alpha_n \restriction \mathcal{D}_R)$, $\mathsf{first}\,(c) = s_0$, $\mathsf{init}\,(c) = s_0 \ldots s_{n-1}$, and $\mathsf{last}\,(c) = s_n$. The operator $w\restriction W$ denotes a projection of the sequence w that retains only those elements of w that belong to the set W. Thus, $\mathsf{trace}\,(c)$ is the observable content of a path, $\mathsf{states}\,(c)$ are the states traversed by c, $\mathsf{time}\,(c)$ is the duration of the path, that is, the number of timed actions along the path, and $\mathsf{init}\,(c)$ is the path truncated by the last transition.

The semantic function $\models\ \subseteq S \times \mathcal{L}_{HMLu}$ is defined inductively as follows:

$s \models tt$	always	
$s \models \neg f$	*iff*	$s \not\models f$
$s \models f \wedge f'$	*iff*	$s \models f$ and $s \models f'$
$s \models f\,\langle\Phi\rangle\,f'$	*iff*	there is a path c such that $\mathsf{trace}\,(c) \in \Phi$, $\mathsf{first}\,(c) = s$, $\forall s' \in \mathsf{states}\,(\mathsf{init}\,(c)) \models f$, and $\mathsf{last}\,(c) \models f'$
$s \models f\,\langle\Phi\rangle^t\,f'$	*iff*	there is a path c such that $\mathsf{trace}\,(c) \in \Phi$, $\mathsf{first}\,(c) = s$, $\forall s' \in \mathsf{states}\,(\mathsf{init}\,(c)) \models f$, $\mathsf{last}\,(c) \models f'$ and $\mathsf{time}\,(c) \leqslant t$

We have developed a model-checking algorithm that determines whether a finite-state ACSR process P satisfies a given $\mathcal{L}_{HMLu}$ formula f. The algorithm follows the approach of [13] for CTL model checking. We label the states of P with the values of each syntactic subformula of f according to the semantic function for the logic. The only non-trivial case is the *until* operator $f_1\langle\Phi\rangle f_2$, which, effectively, computes the product of P with the regular expression Φ and explores it in a depth-first manner, traversing nodes (P', Φ') where P' is already labeled with f_2, and Φ' is a derivative of Φ. Traversal of a path terminates when a node (P'', Φ'') is reached, such that P'' is labeled with f_2 and Φ'' accepts the empty string.

Compared to the original definition of HML with *until* [14], parameterized *until* operators can express a property of an execution that contains a series of events, rather than only one event. We believe that this extension makes the logic easier to use. As an example, consider a process that receives messages from a communication channel, processes them for one time unit, and sends acknowledgements back to the sender. The nominal execution pattern, then, is $\Phi_{nom} = (recv?\{cpu\}ack!\emptyset*)*$. In addition, the process would have to be concerned with exceptional conditions such as timeouts, preemptions from higher-priority processes, etc. Reasoning about the nominal behavior of such process would involve formulas of the form $f\langle recv?\{cpu\}ack!\rangle f'$. Such a formula can of course be rewritten to use single-event *until* operators by nesting: $f\langle recv?\rangle(f\langle\{cpu\}\rangle(f\langle recv?\rangle f')$, which is significantly harder to comprehend visually.

Furthermore, the use of Kleene star in regular expressions allows us to increase the expressive power of the logic. The formula $f\langle\Phi_{nom}\rangle f'$ reasons about continuous nominal behavior, and cannot be expressed using a finite formula in the original HML with *until*.

At the same time, the number of derivatives of Φ that need to be explored during model checking is, in the worst case, exponential in the size of Φ, whereas checking the original HML with *until* is linear in the size of the formula. We note, however, that a $\mathcal{L}_{HMLu}$ formula can be exponentially smaller than the equivalent formula with single-event nested *until* operators.

2.3. Example: EDF scheduling

In the sequel, we will use a simple example from the area of schedulability analysis. The example describes a set of periodic tasks scheduled according to the Earliest-Deadline-First [24] scheduling policy. This policy assigns dynamic

priorities to the tasks according to the proximity of their deadlines. We assume here that the deadline of each task is equal to its period. That is, once a task is dispatched, it needs to complete its execution before it is dispatched again. The ACSR representation of the EDF scheduling algorithm was first presented in [11].

An instance of the scheduling problem contains a set of tasks t_i, with period p_i and worst-case execution time e_i. The task set is modeled as a collection of processes $Task_1, \ldots, Task_n$, one for each periodic task in the set. The process $Task_i$ contains two concurrent subprocesses: T_i and $Dispatch_i$. The auxiliary process $Dispatch_i$ handles periodic invocations of the task t_i and detects missed deadlines. The process T_i captures the state of the task t_i, which can be awaiting the next invocation, be executing on the processor resource, or be preempted by a higher priority task. Both tasks share the same processor, modeled by the resource cpu. No other tasks use the processor.

The process T_i idles until it is awakened by the $start_i$ event and then starts competing for the processor. At each time unit, the task may either get access to the processor or, if it is preempted by a higher-priority task, it idles until the next time unit. Once the necessary amount (i.e., e_i) of execution time is accumulated, the task returns to the initial state and waits for the next period. The process $Dispatch_i$ sends the $start_i$ event to T_i every p_i time units. If T_i has not completed its execution by then, meaning that the deadline is missed, it cannot accept the $start_i$ event and the dispatcher deadlocks.

The complete specification is shown in Fig. 1, where, for simplicity, we employ the notation $A^n : P$ for the process that makes n consecutive executions of action A before proceeding to process P. In the specification of a task, i is the task number and j the accumulated execution time. The priority of task i after having accumulated j units of execution time is $d_{max} - (p_i - j)$, where $d_{max} = \max_{1 \leqslant i \leqslant n} (p_i)$. Note that closure of the resource cpu is applied. This is for ensuring progress in the model, as discussed in the context of the ACSR prioritized transition system.

Let us make some important observations regarding this specification. We begin by noting that process *System* is composed of a number of sequential processes T_i and $Dispatch_i$. Process T_i can be considered to be a patient process, in that all its derivatives enable an idling action, thus are willing to allow time to pass, if necessary. On the other hand, process $Dispatch_i$ is patient in all its derivatives, expect the initial state which is urgent to perform event $(start!_i, i)$, that is, to dispatch an instance of the task every p_i time units. If process T_i is in its initial state, then the task dispatch will successfully take place, otherwise, process T_i is still executing a previous task invocation and the system will deadlock signifying that the deadline of an invocation has been missed. In other words, the model of the system is constructed so that a missed deadline induces a deadlock. This is achieved by associating an urgent event at the time of each task deadline. This approach can be applied in a variety of contexts for the purpose of performing schedulability analysis.

ACSR analysis techniques allow us to verify the schedulability of a system of tasks for fixed values of parameters e_i and p_i. According to the philosophy of ACSR modeling, the correctness criterion for a system to be schedulable is that the corresponding process executes forever, in other words, it does not deadlock. The following result pinpoints the source of deadlocks in a special class of systems, including process *System* above. Specifically, it states that, given a parallel composition of a set of sequential processes, some of which being patient in their execution while others occasionally wishing to fire urgent events, a deadlock occurs exactly when an urgent event cannot be executed due to a participating process being unable to engage in it. This result can be applied to the ACSR models of a wide range of real-time systems, since it captures the essence of the ACSR design and verification methodology and, in particular, the association of schedulability analysis with deadlock detection.

$$
\begin{aligned}
System &\stackrel{\text{def}}{=} [Task_1 \| \ldots \| Task_n]_{\{cpu\}} \\
Task_i &\stackrel{\text{def}}{=} (T_i \| Dispatch_i) \backslash \{start_i\} && i = \{1..n\} \\
Dispatch_i &\stackrel{\text{def}}{=} (start_i!, i).\emptyset^{p_i} : Dispatch_i && i = \{1..n\} \\
T_i &\stackrel{\text{def}}{=} (start_i?, 0).P_{i,0} + \emptyset : T_i && i = \{1..n\} \\
P_{i,j} &\stackrel{\text{def}}{=} j < e_i \rhd (\emptyset : P_{i,j} \\
&\qquad\qquad + \{(cpu, d_{max} - (p_i - j))\} : P_{i,j+1}) \\
&+ \quad j = e_i \rhd T_i && i = \{1..n\}, j = \{0..e_i\}
\end{aligned}
$$

Fig. 1. ACSR specification of an EDF scheduling problem.

First, let us introduce some useful notation. In the sequel, we will write $P \xrightarrow{\alpha}_\pi$ if there is some process P' such that $P \xrightarrow{\alpha}_\pi P'$ and $P \not\xrightarrow{\alpha}_\pi$ if there is no process P' such that $P \xrightarrow{\alpha}_\pi P'$. Finally, we write $\Pi_{i\in I} P_i$, where $I = \{i_1, \ldots, i_n\}$, for $P_{i_1} \| \ldots \| P_{i_n}$.

Theorem 3. *Consider a process P such that $P \stackrel{\text{def}}{=} [(\Pi_{i\in I} P_i)\backslash F]_U$ for $F \subseteq L$, $U \subseteq \mathcal{R}$ and where, for all i, P_i contains no parallel composition operator, and $I = I_1 \cup I_2$ where,*

- *for all $i \in I_1$ and for all derivatives Q_i of P_i, $Q_i \xrightarrow{\emptyset}_\pi$ or $Q_i \xrightarrow{\alpha}_\pi$, $\alpha \in \mathcal{D}_E$, $\ell(\alpha) \notin F$, and*
- *for all $i \in I_2$ and for all derivatives Q_i of P_i, either (1) $Q_i \xrightarrow{\emptyset}_\pi$ or (2) $Q_i \xrightarrow{\alpha_i}_\pi$, $\ell(\alpha_i) \in F$, and $Q_i \not\xrightarrow{\beta}_\pi$ for all $\beta \in \text{Act} - \{\alpha_i\}$.*

Then, for all derivatives Q of P, if $Q \not\longrightarrow_\pi$ then $Q = [(\Pi_{i\in I} Q_i)\backslash F]_U$ and $Q_i \xrightarrow{\alpha_i}_\pi$ for some $i \in I_2$.

Proof. Consider a process $P \stackrel{\text{def}}{=} [(\Pi_{i\in I} P_i)\backslash F]_U$ satisfying the conditions of the theorem. We will prove that any derivative Q of P is such that $Q = [(\Pi_{i\in I} Q_i)\backslash F]_U$, where no Q_i contains a parallel composition operator, and I can be partitioned into sets I_1 and I_2 satisfying the conditions of the theorem. The proof will be carried out by induction on the length, n, of the transition $P \xrightarrow{w}_\pi Q$, $w \in Act^*$.

Clearly, the claim holds for $n = 0$. Suppose that it holds for $n = k - 1$ and that $P \xrightarrow{w}_\pi Q' \xrightarrow{\alpha}_\pi Q$ is a transition of size n. By the induction hypothesis, $Q' = [(\Pi_{i\in I} Q'_i)\backslash F]_U$ satisfies the conditions of the theorem. Consider the transition $Q' \xrightarrow{\alpha}_\pi Q$. Three cases exist:

- $\alpha \in \mathcal{D}_R$. This implies that, for all $i \in I$, $Q'_i \xrightarrow{A_i}_\pi Q_i$, for some Q_i, $Q = [(\Pi_{i\in I} Q_i)\backslash F]_U$ and $\alpha = \bigcup_{i\in I} A_i$. It is straightforward to see that no Q_i contains a parallel composition operator and that, since each Q_i is a derivative of Q'_i, the conditions of the theorem are satisfied.
- $\alpha = \tau$. This implies that there exist $j, k \in I$, such that $Q'_j \xrightarrow{\alpha_j}_\pi Q_j$ and $Q'_k \xrightarrow{\alpha_k}_\pi Q_k$, where $\ell(\alpha_j)$ and $\ell(\alpha_k)$ are inverse labels, and

  $$Q = [(\Pi_{i\in I-\{j,k\}} Q'_i \parallel Q_j \parallel Q_k)\backslash F]_U.$$

  It is straightforward to see that no Q'_i, Q_i, contains a parallel composition operator and to check that conditions of the theorem are satisfied.
- $\alpha \in \mathcal{D}_E$. This implies that there exists $j \in I$, such that $Q'_j \xrightarrow{\alpha_j}_\pi Q_j$, $Q = [(\Pi_{i\in I-\{j\}} Q'_i \parallel Q_j)\backslash F]_U$ and the proof follows easily.

So consider an arbitrary derivative Q of P and suppose that $Q \not\longrightarrow_\pi$. Since $Q = [(\Pi_{i\in I} Q_i)\backslash F]_U$ satisfies the conditions of the theorem, and $Q \not\xrightarrow{\alpha}_\pi$, $\alpha \in Act$, it must be that some $Q_i \not\xrightarrow{\emptyset}_\pi$, $i \in I_2$. This implies that $Q_i \xrightarrow{\alpha_i}_\pi$ and the result follows. □

We now return to the schedulability of system *System* above. The main step in reaching the desired result is to associate the source of deadlocks, as characterized by the previous theorem, to the violation of task deadlines.

Proposition 4. *System is schedulable if and only if it contains no deadlocks.*

Proof. First, we observe that if *System* contains no deadlocks then the associated real-time system is schedulable: task activations take place as planned and no deadlines are missed.

To prove the opposite direction, we show that if the system contains a deadlock then *System* is not schedulable. Consider system *System*. Note that, although in its current form this process does not satisfy the conditions of Theorem 3, by using the ACSR axiom system [8], we may easily rewrite this process to an equivalent process which does, that is,

$$System = [\ (\Pi_{i=1\ldots n}\ \ (T_i\ \ \|\ \ Dispatch_i))\ \backslash\ F\ ]_{\{cpu\}}$$

where $F = \{start_i \mid 1 \leqslant i \leqslant n\}$.

It is straightforward to verify that the above process satisfies the conditions of Theorem 3, and, further, I_2 contains all processes $Dispatch_i$, with $\alpha_i = (start_i, i)$. Consequently, by the same theorem, if a deadlock arises in *System*, the event $start_i$ is enabled in some process $Dispatch_i$ but not in the respective T_i process. This implies that the task has not

yet accumulated the required execution time while its deadline has elapsed. Thus, the system is not schedulable which completes the proof. □

Consequently, the presented model can be instantiated to a specific task set, and its schedulability can be checked by performing reachability analysis on the state space of the process to search for deadlock states.

As a concrete instance of this problem, we consider a set of three tasks with $p_1 = 8$, $p_2 = 10$, $p_3 = 14$, $e_1 = 3$, $e_2 = 3$, $e_3 = 1$. Theoretical results from [24] show that a set of tasks is schedulable if the utilization of the task set, $U_i = \Sigma_{i\in\{1\ldots n\}} e_i/p_i$ does not exceed 1. The task set from our example satisfies this criterion and, by checking the resulting process for the absence of deadlocks, we can indeed verify that all deadlines are met. On the other hand, the set $e_1 = 2$, $p_1 = 3$, $e_2 = 1$, $p_2 = 2$ has a deadlock and thus is not schedulable.

The same type for analysis can be applied for any task model no matter the scheduling discipline or the presence of offsets, task dependencies and other behavioral variations, including scenaria for which no schedulability tests exist. Once the model is faithfully captured as an ACSR process, schedulability analysis can be automatically decided by searching for deadlocks within the system. For example, consider a set of periodic tasks as introduced in Fig. 1, extended with *data dependencies*. When a task T_1 supplies data for T_2, an instance of T_2 cannot begin its execution until a preceding instance of T_1 completes its execution, producing fresh data. To the best of our knowledge, there is no exact schedulability test for this task model. Encoding of this task model into ACSR has been introduced in [31]. It satisfies the conditions of Theorem 3, and all results of this section are immediately applicable for the schedulability analysis of such task sets.

Schedulability analysis of systems such as the one above can be alternatively carried out by bisimulation checking or model checking. For bisimulation checking, one needs to construct the specification of the system as an ACSR process. In the example above, the requirement being that the process executes forever, we would write $Spec \stackrel{\text{def}}{=} \emptyset : Spec$ and check that $System \approx Spec$. Of course, compared to deadlock-detection this approach is inefficient. Nonetheless, the general approach is viable in the context of value-passing ACSR, where bisimulation plays a central role for performing schedulability analysis: system models may contain a number of unspecified parameters, and the purpose of the analysis performed is to specify values for these parameters that make the system schedulable. To achieve this, symbolic bisimulation is employed between the parameterized system and the process that idles forever, and, with the aid of integer programming, appropriate ranges for the parameters that make the system schedulable are computed [19].

On the other hand, model checking the schedulability of a system can be carried out by the inclusion of special actions in the model that signify missed deadlines. Consequently, one may check whether these actions may eventually take place, and, if so, conclude that the system is not schedulable. In the example above, this would involve rewriting the dispatcher process as follows:

$$\begin{aligned} Dispatch_i \stackrel{\text{def}}{=}\ & (start_i!, i).\emptyset^{p_i} : Dispatch_i \\ & + (\tau, 0).(miss!, i).\text{NIL} \quad i = \{1..n\} \end{aligned}$$

The new summand involves an internal action taking place at priority 0. As specified by the preemption relation, this action may take place only if no other action is enabled within the system. That is, if the deadline of process T_i is missed and the $start_i$ even cannot be accepted, this second summand is enabled and the *miss* event is fired. Thus, the correctness requirement in the new system is the property $\neg(tt\langle\{cpu\}* \ miss!\rangle tt)$.

3. Probabilistic ACSR

PACSR (Probabilistic ACSR) extends the process algebra ACSR by associating each resource with a probability. This probability captures the rate at which the resource may fail. Instantaneous events in PACSR are identical to those of ACSR; timed actions can now account for resource failure, as discussed below.

Timed actions. As in ACSR, we assume that a system contains a finite set of serially reusable resources drawn from the set $\mathcal{R}$. We also consider set $\overline{\mathcal{R}}$ that contains, for each $r \in \mathcal{R}$, an element $\overline{r}$ representing the *failed* resource r. We write R for $\mathcal{R} \cup \overline{\mathcal{R}}$. Actions are constructed as in ACSR, but now can contain both normal and failed resources. So, in PACSR, the action $\{(r, p)\}$, $r \in \mathcal{R}$, 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.

Resource probabilities. In PACSR we associate each resource with a probability at which the resource may fail. In particular, for all $r \in \mathcal{R}$ 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 the process $\{(cpu, 1)\}$:NIL, where resource cpu has probability of failure $1/3$, i.e., $\mathsf{p}(\overline{cpu}) = 1/3$. 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.

Probabilistic processes. The syntax of PACSR processes is the same as that of ACSR. The only extension concerns the appearance of failed resources in timed actions. Thus, it is possible on one hand to assign failure probabilities to resources of existing ACSR specifications and perform probabilistic analysis on them, and, on the other hand, to ignore failure probabilities and apply non-probabilistic analysis of PACSR specifications.

Before we present the semantics 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:

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

Definition 5. Let $Z = \{r_1, \ldots, r_n\} \subseteq \mathsf{R}$. We write
- $\overline{Z} = \{\overline{r} \mid r \in Z\}$,
- $\mathsf{p}(Z) = \Pi_{1 \leqslant i \leqslant n} \mathsf{p}(r_i)$,
- $\mathcal{W}(Z) = \{Z' \subseteq Z \cup \overline{Z} \mid r \in Z' \text{ iff } \overline{r} \notin Z'\}$, and
- $\mathsf{res}(Z) = \{r \in \mathcal{R} \mid r \in Z \text{ or } \overline{r} \in Z\}$.

We say that $\mathcal{W}(Z)$ contains 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 $\mathcal{W}(\emptyset) = \{\emptyset\}$.

3.1. Operational semantics

As with ACSR, the semantics of PACSR processes is given in two steps. At the first level, a transition system captures the non-deterministic 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), 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 non-deterministic. 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 (PAct2)). Non-deterministic 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:

$\mathsf{S}_n = \{(P, W) \in \mathsf{S} \mid \mathsf{res}(\mathsf{imr}(P)) - \mathsf{res}(W) = \emptyset\}$, the set of non-deterministic configurations, and
$\mathsf{S}_p = \{(P, W) \in \mathsf{S} \mid \mathsf{res}(\mathsf{imr}(P)) - \mathsf{res}(W) \neq \emptyset\}$, the set of probabilistic configurations.

Let $\twoheadrightarrow_p \subset \mathsf{S}_p \times [0, 1] \times \mathsf{S}_n$ be the probabilistic transition relation. A triple in $\twoheadrightarrow_p$, written $(P, W) \stackrel{\pi}{\twoheadrightarrow}_p (P', W')$, denotes that process P in world W may become P' and enter world W' with probability π. Furthermore, let $\twoheadrightarrow_n \subset$

$\mathsf{S}_n \times Act \times \mathsf{S}$ be the non-deterministic transition relation. A triple in $\twoheadrightarrow_n$ is written as $(P, W) \overset{\alpha}{\twoheadrightarrow}_n (P', W')$, capturing that process P in world W may non-deterministically perform α and become (P', W').

The probabilistic transition relation is given by the following rule:

$$\text{(PROB)} \quad \frac{(P, W) \in \mathsf{S}_p,\ Z_1 = \mathsf{res}(\mathsf{imr}(P)) - \mathsf{res}(W),\ Z_2 \in \mathcal{W}(Z_1)}{(P, W) \overset{\mathsf{p}(Z_2)}{\twoheadrightarrow}_p (P, W \cup Z_2)}$$

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

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

$$(P, \emptyset) \overset{1/6}{\twoheadrightarrow}_p (P, \{r_1, r_2\}) \qquad (P, \emptyset) \overset{1/6}{\twoheadrightarrow}_p (P, \{\overline{r_1}, r_2\})$$
$$(P, \emptyset) \overset{1/3}{\twoheadrightarrow}_p (P, \{r_1, \overline{r_2}\}) \qquad (P, \emptyset) \overset{1/3}{\twoheadrightarrow}_p (P, \{\overline{r_1}, \overline{r_2}\})$$

Lemma 6. *For all* $s \in \mathsf{S}_p$, $\Sigma\{\!|\, p \mid (s, p, s') \in \twoheadrightarrow_p \,|\!\} = 1$, *where* $\{\!|$ *and* $|\!\}$ *are multiset brackets and the summation over the empty multiset is* 1.

The non-deterministic transition relation for PACSR is given similarly to ACSR. Note, however, that states in the transition system are now configurations, that is, pairs of processes and associated worlds, and not simple processes. In Table 2, we present some representative rules. (PAct1) and (PAct2) contain the essence of the semantics extension specifying the usage of failed and non-failed resources and the treatment of resource worlds. In (PAct1) we may see that instantaneous events preserve the world of a configuration, while (PAct2) specifies that timed actions re-initialize the world to $\emptyset$. Further, for an action to take place, all its resources must be available in the configuration's world. Thus, by rule (PAct2), in the example above we have $(P, \{r_1, \overline{r_2}\}) \overset{\{(r_1,2),(\overline{r_2},3)\}}{\twoheadrightarrow_n} (Q, \emptyset)$, whereas $(P, \{r_1, r_2\})$, $(P, \{\overline{r_1}, r_2\})$, and $(P, \{\overline{r_1}, \overline{r_2}\})$ have no transitions. The remaining rules can be obtained from those of ACSR by simply assigning worlds to processes in the style of rules (PSum1) and (PPar1). Finally, the prioritized non-deterministic relation of PACSR, $\twoheadrightarrow_\pi$, is derived by application of the preemption relation $\prec$ as for ACSR.

Note that the probabilistic resource failure mechanism implemented in PACSR can be used to describe a number of probabilistic phenomena. For example, we may model persistent failure of a resource using modes as shown in the process $P \overset{\text{def}}{=} \{(r, 1)\}{:}P' + \{(\overline{r}, 1)\}{:}Q$, where, upon the failure of resource r, the process enters mode Q where resource r is replaced by a failed resource. It is also straightforward to model delays following the geometric distribution, with the aid of a single resource, and event arrival following the binomial distribution, with the aid of n resources, where n is the size of the support of the distribution. However, it is much more complicated to model/simulate other

Table 2
PACSR non-deterministic relation

(PAct1)	$(e.P, W) \overset{e}{\twoheadrightarrow}_n (P, W)$	
(PAct2)	$(A{:}P, W) \overset{A}{\twoheadrightarrow}_n (P, \emptyset)$	if $\rho(A) \subseteq W$
(PSum1)	$\dfrac{(P_1, W) \overset{\alpha}{\twoheadrightarrow}_n (P, W')}{(P_1 + P_2, W) \overset{\alpha}{\twoheadrightarrow}_n (P, W')}$	
(PPar1)	$\dfrac{(P_1, W) \overset{e}{\twoheadrightarrow}_n (P_1', W')}{(P_1 \| P_2, W) \overset{e}{\twoheadrightarrow}_n (P_1' \| P_2, W')}$	

discrete distributions such as the Poisson distribution. This is due to the fact that the only available tool for producing probabilistic behavior are the Bernoulli distributions associated with resources, the two possible outcomes being *alive* and *failed*. A dense-time PACSR variant, where we may be able to introduce more complicated probability distributions directly, is a promising direction of future research.

3.2. Probabilistic analysis techniques

In this section we discuss possible analysis that can be performed on PACSR specifications. We begin by presenting the formal model underlying PACSR processes which is that of *labeled concurrent Markov chains* [36].

Definition 7. A *labeled concurrent Markov chain* (LCMC) is a tuple $\langle \mathsf{S}_n, \mathsf{S}_p, Act, \twoheadrightarrow_\pi, \twoheadrightarrow_p, s_0\rangle$, where S_n is the set of non-deterministic states, S_p is the set of probabilistic states, Act is the set of labels, $\twoheadrightarrow_\pi \subset \mathsf{S}_n \times Act \times (\mathsf{S}_n \cup \mathsf{S}_p)$ is the non-deterministic transition relation, $\twoheadrightarrow_p \subset \mathsf{S}_p \times (0, 1] \times \mathsf{S}_n$ is the probabilistic transition relation, satisfying $\Sigma_{(s,p,t)\in\twoheadrightarrow_p} p = 1$ for all $s \in \mathsf{S}_p$, and $s_0 \in \mathsf{S}_n \cup \mathsf{S}_p$ is the initial state.

We may see that the operational semantics of PACSR yields transition systems that are LCMCs. Analysis of PACSR processes is carried out on the underlying LCMC. In what follows, we let ℓ range over $Act \cup [0, 1]$ and write S for $\mathsf{S}_n \cup \mathsf{S}_p$.

A *computation* in $T = \langle \mathsf{S}_n, \mathsf{S}_p, Act, \twoheadrightarrow_\pi, \twoheadrightarrow_p, s_0\rangle$ is either a finite sequence $c = s_0\, \ell_1\, s_1 \ldots \ell_k\, s_k$, or an infinite sequence $c = s_0\, \ell_1\, s_1 \ldots \ell_k\, s_k \ldots$, such that $s_i \in \mathsf{S}_n \cup \mathsf{S}_p$, $\ell_{i+1} \in Act \cup [0, 1]$ and $(s_i, \ell_{i+1}, s_{i+1}) \in \twoheadrightarrow_p \cup \twoheadrightarrow_\pi$, for all $0 \leqslant i$. Given a computation $c = s_0\, \ell_1\, s_1 \ldots \ell_k\, s_k$, we write $s_0 \overset{w}{\twoheadrightarrow} s_k$, where $w = \ell_1 \ldots \ell_k$.

To define probability measures on computations of an LCMC the non-determinism present must be resolved. To achieve this, the notion of a *scheduler* has been employed [36,16]. A scheduler σ is an entity that, given a computation ending in a non-deterministic state, chooses the next transition to be executed. This gives rise to computation trees that can be viewed as labeled Markov chains. Each path through a computation tree is a *scheduled computation* of the LCMC and can be assigned a probability by taking a product of the probabilistic labels along the path. See [30] for the details.

3.2.1. Equivalence checking

Equivalence between PACSR is based on the concept of probabilistic bisimulation [20,16]. First, we introduce a useful notation.

Definition 8. For $s \in \mathsf{S}$ and $\mathcal{M} \subseteq \mathsf{S}$, we define

$$\mu(s, \mathcal{M}) = \begin{cases} \Sigma_{s'\in\mathcal{M}} \{\!|\, p \mid (s, p, s') \in \twoheadrightarrow_p \,|\!\}, & \text{if } s \in \mathsf{S}_p \\ 1, \quad \text{if } s = s', s \in \mathsf{S}_n \\ 0, \quad \text{otherwise.} \end{cases}$$

Thus, $\mu(s, \mathcal{M})$ denotes the cumulative probability of state s probabilistically reaching states in $\mathcal{M}$, taken to be 1 if s is a non-deterministic state in $\mathcal{M}$.

Strong probabilistic bisimulation is then defined as follows:

Definition 9. *Probabilistic strong bisimulation* is the largest symmetric relation denoted by $\sim_p$, such that, whenever $s \sim_p t$,

(1) for all $\alpha \in Act$, if $s, t \in \mathsf{S}_n$ and $s \overset{\alpha}{\twoheadrightarrow}_\pi s'$ then $t \overset{\alpha}{\twoheadrightarrow}_\pi t'$ and $s' \sim_p t'$;
(2) for all $\mathcal{M} \in \mathsf{S}/\sim_p$, $\mu(s, \mathcal{M}) = \mu(t, \mathcal{M})$, where $\mathsf{S}/\sim_p$ is the set of equivalence classes of S over $\sim_p$.

Thus, two states are strongly bisimilar to each other if they can reach all equivalence classes of strong bisimilarity with the same probability and they can simulate each other's behavior. The above definition is almost identical to the one proposed in [16], where an *alternating* model is considered, However, with a slight reformulation of the definition of $\mu(s, \mathcal{M})$, Definition 9 allows for pairs of probabilistic and non-deterministic systems to be considered bisimulation equivalent.

It can be shown that $\sim$ is a congruence with respect to the PACSR operators.

We have also defined a probabilistic weak bisimulation [32] and a probabilistic branching bisimulation for the model, which allows us to compare observable behaviors of PACSR processes similarly to the case of ACSR. In addition, probabilistic information embedded in the probabilistic transitions allows us to perform quantitative analysis of PACSR specifications. In particular, we can compute the probability of reaching a given state or a deadlocked state.

3.2.2. Model checking of PACSR processes

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 define a probabilistic version of the logic $\mathcal{L}_{HMLu}$ defined in Section 2.2. The two *until* operators are extended with probabilistic conditions, thus providing for quantitative analysis of a PACSR specification. The condition takes the form of $\leqslant p$ or $\geqslant p$ for a constant p. Thus, *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 addition of probabilistic constraints is in fact the source of the need to parameterize the *until* operators with regular expressions, which we have introduced in $\mathcal{L}_{HMLu}$. Compared to the original definition of HML with *until* defined in [14], parameterized *until* operators can express a property of an execution that contains a series of events, rather than only one event. Unlike the non-probabilistic setting, where one can often express the same property by using several nested *until* operators, in the probabilistic setting such extension appears to be necessary. Nesting of *until* operators would preclude us from associating a single probabilistic condition with the whole execution. 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. It can be shown that the resulting logic characterizes probabilistic branching bisimulation.

PACSR observables are taken to range over the observable content of PACSR events and timed actions, and regular expressions are then defined in the expected manner. The syntax of $\mathcal{L}^{pr}_{HMLu}$ is defined by the following grammar, where f, f' range over $\mathcal{L}^{pr}_{HMLu}$-formulae, Φ is a regular expression over the set of PACSR observables, and $\bowtie \in \{\leqslant, \geqslant\}$:

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

$\mathcal{L}^{pr}_{HMLu}$-formulae are interpreted over states of LCMCs. Informally, formulae of the form $f\langle\Phi\rangle f'$ state that there is some execution and some integer l such that f holds for the first $l-1$ steps and f' becomes true in the lth step and the observable behavior of the l-step execution involves some behavior from Φ. The subscript $\bowtie p$ denotes that the probability of paths fulfilling the formula is $\bowtie p$ and the use of superscript t denotes that the paths of interest are only those that achieve the goal in at most t time units. For instance, formula $tt\langle(\mathcal{L} \cup 2^{\mathcal{R}})*\rangle^{4}_{\geqslant 1} f$ expresses that there is some execution of the system for which f becomes true within the first four time units, with probability 1.

In order to present the semantics of the two *until* operators, we need to compute the probabilities that certain behaviors occur. Consider for example the formula $f\langle\Phi\rangle^{t}_{\bowtie p} f'$. Given two sets of states A and B of an LCMC T and a scheduler σ we consider the following set of computations of T. The computations are scheduled by σ and lead to a state in B via a trace with observable content in Φ, with intermediate states in A, and take at most time t. It can be shown that this set of computations is measurable in the probability space of T. We denote its probability $\mathsf{Pr}_A(T, \Phi, B, t, \sigma)$. Similarly, we define $\mathsf{Pr}_A(T, \Phi, B, \sigma)$ as the probability measure of computations scheduled by scheduler σ that lead to a state in B via intermediate states in A and observable content in Φ. Both of these probabilities can be computed as the solution of a set of linear equations. See [30] for the details.

The satisfaction relation $\models \subseteq (\mathsf{S}_n \cup \mathsf{S}_p) \times \mathcal{L}^{pr}_{HMLu}$, stating when an LCMC state s satisfies a given formula, is defined inductively as follows, where we write $\mathsf{Sched}(s)$ for the set of schedulers of the LCMC s.

$$
\begin{array}{rcll}
System & \stackrel{\text{def}}{=} & [Task_1 \| \ldots \| Task_n]_{\{cpu\}} & \\
Task_i & \stackrel{\text{def}}{=} & (T_i \| Dispatch_i) \backslash \{start_i\} & i = \{1..n\} \\
Dispatch_i & \stackrel{\text{def}}{=} & (start_i!, i).\emptyset^{p_i} : Dispatch_i & \\
 & & +(\tau, 0).(miss!, 1).\mathrm{NIL} & i = \{1..n\} \\
T_i & \stackrel{\text{def}}{=} & (start_i?, 0).P_{i,0} + \emptyset : T_i & i = \{1..n\} \\
P_{i,j} & \stackrel{\text{def}}{=} & j < e_i \rightarrow (\emptyset : P_{i,j} & \\
 & & \quad +\{(cpu, d_{max} - (p_i - j))\} : P_{i,j+1} & \\
 & & \quad +\{(\overline{\mathbf{cpu}}, \mathbf{d_{max}} - (\mathbf{p_i} - \mathbf{j}))\} : \mathbf{P_{i,j}}) & \\
 & + & j = e_i \rightarrow T_i & i = \{1..n\}, j = \{0..e_i\}
\end{array}
$$

Fig. 2. EDF scheduling problem with processor failures.

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

A model-checking algorithm for $\mathcal{L}^{pr}_{HMLu}$, suitable for finite-state PACSR specifications, was proposed in [30]. It follows the outline of the algorithm for $\mathcal{L}_{HMLu}$, with the exception that labeling for the *until* operators requires us to solve a linear programming problem. We refer the reader to the same paper for an application of the model-checking technique to a telecommunications application.

3.3. Example

We illustrate the utility of PACSR in the analysis of fault-tolerance properties by slightly extending the example of Section 2.3. Note that we employ the version of the dispatchers that emit special actions to flash missed deadlines with the intention of exploiting these messages for model checking the system. We consider the same set of tasks running on a processor with an intermittent fault. At any time unit, the processor may be running, in which case the higher-priority task executes normally, or it may be down, in which case none of the tasks execute. We modify the specification of a task by extending it with a failure-recovery mechanism (shown in bold) which specifies that whenever resource *cpu* has a failure the execution time of the task is not increased (Fig. 2).

We apply the probabilistic analysis to the task set with: $e_1 = 2$, $p_1 = 5$, $e_2 = 1$, $p_2 = 2$. Even though the task set is schedulable under perfect conditions, in the presence of failures the tasks may still miss their deadlines. Given the probability of a processor failure, we can compute the probability that a deadline is missed. The properties we check have the form $\neg(true\langle\{cpu\} * miss!\rangle_{\leqslant\alpha} true)$. The following list of pairs show results of the experiments we ran. The first element of each pair is the *cpu* failure probability and the second is the greatest value of the probability α for which the property holds, which corresponds to the probability that a deadline is missed: $\{(0, 0), (0.1, 0.003), (0.2, 0.130), (0.25, 0.339), (0.3, 0.585)\}$.

4. Mappings between ACSR and PACSR

In this section we study the relation between the two process algebras by providing mappings between them. These mappings confirm the natural relationship between the process algebras and allow us to isolate properties preserved between the two. They are defined by structural induction at both the process level and the logic level of the languages and they are shown to preserve bisimulation between the two formalisms.

4.1. From PACSR to ACSR

In this section, we study a mapping from the PACSR to the ACSR formalism. This mapping allows us to conclude the type of ACSR analysis techniques that can be applied on PACSR models and move between the two frameworks to check both probabilistic requirements as well as requirements expressed in terms of ACSR. For example, we observe how schedulability analysis techniques originating from ACSR can be applied to PACSR specifications. Furthermore, we establish mappings between formulae of $\mathcal{L}_{HMLu}$ and $\mathcal{L}^{pr}_{HMLu}$ and we conclude how properties expressed in the former can be checked on PACSR processes.

We begin by defining a mapping between PACSR and ACSR processes. This mappings acts on the set of PACSR configurations and makes use of different functions for probabilistic and non-deterministic configurations.

Definition 10. We define $\llbracket \cdot \rrbracket : \mathsf{S} \mapsto \mathsf{Proc}$ by

$$\llbracket (P, W) \rrbracket = \begin{cases} \llbracket (P, W) \rrbracket_p, & \text{if } (P, W) \in \mathsf{S}_p \\ \llbracket (P, W) \rrbracket_n, & \text{if } (P, W) \in \mathsf{S}_n \end{cases}$$

where $\llbracket \cdot \rrbracket_p : \mathsf{S}_p \mapsto \mathsf{Proc}$ is defined by

$$\llbracket (P, W) \rrbracket_p = \sum_{W' \in \mathcal{W}(\mathsf{imr}(P) - \mathsf{res}(W))} \tau.\, \llbracket (P, W \cup W') \rrbracket_n$$

and $\llbracket \cdot \rrbracket_n : \mathsf{S}_n \mapsto \mathsf{Proc}$ is defined inductively as follows:

$$\begin{aligned}
\llbracket (\mathrm{NIL}, W) \rrbracket_n &= \mathrm{NIL} \\
\llbracket ((a, n).P, W) \rrbracket_n &= (a, n).\llbracket (P, W) \rrbracket \\
\llbracket (A{:}P, W) \rrbracket_n &= \begin{cases} A : \llbracket (P, \emptyset) \rrbracket, & \text{if } \rho(A) \subseteq W \\ \mathrm{NIL}, & \text{otherwise} \end{cases} \\
\llbracket (P + Q, W) \rrbracket_n &= \llbracket (P, W) \rrbracket_n + \llbracket (Q, W) \rrbracket_n \\
\llbracket (P \| Q, W) \rrbracket_n &= \llbracket (P, W) \rrbracket_n \| \llbracket (Q, W) \rrbracket_n \\
\llbracket (P \backslash F, W) \rrbracket_n &= \llbracket (P, W) \rrbracket_n \backslash F \\
\llbracket ([P]_I, W) \rrbracket_n &= [\llbracket (P, W) \rrbracket_n]_I \\
\llbracket (P \mathbin{\triangle^a_t} (Q, R, S), W) \rrbracket_n &= \llbracket (P, W) \rrbracket_n \\
&\qquad \mathbin{\triangle^a_t} (\llbracket (Q, W) \rrbracket_n, \llbracket (R, W) \rrbracket_n, \llbracket (S), W \rrbracket_n) \\
\llbracket (b \rhd P, W) \rrbracket_n &= b \rhd \llbracket (P, W) \rrbracket_n \\
\llbracket (C, W) \rrbracket_n &= \llbracket (P, W) \rrbracket_n, \quad \text{where } C \stackrel{\text{def}}{=} P
\end{aligned}$$

Thus, mapping $\llbracket \cdot \rrbracket$ maps PACSR configurations into ACSR processes as follows: Function $\llbracket \cdot \rrbracket_p$ abstracts away the probabilistic behavior of a probabilistic process by replacing probabilistic transitions of a probabilistic configurations by internal actions, while $\llbracket \cdot \rrbracket_n$ preserves non-deterministic actions. This is made precise by the following proposition.

Proposition 11

(1) *If* $(P, W) \in \mathsf{S}_p$, *then*

- *If* $(P, W) \xrightarrow{p}\!\!\!\!\rightarrow_p (P', W')$ *then* $\llbracket (P, W) \rrbracket_p \xrightarrow{\tau}_\pi \llbracket (P', W') \rrbracket_n$, *and*
- *If* $\llbracket (P, W) \rrbracket_p \xrightarrow{\alpha}_\pi Q$, *then* $\alpha = \tau$ *and there exists* W' *such that* $(P, W) \xrightarrow{p}\!\!\!\!\rightarrow_p (P, W')$, *where* $\llbracket (P, W') \rrbracket_n = Q$.

(2) *If* $(P, W) \in \mathsf{S}_n$, *then*

- *If* $(P, W) \xrightarrow{\alpha}\!\!\!\!\rightarrow_\pi (P', W')$ *then* $\llbracket (P, W) \rrbracket_n \xrightarrow{\alpha}_\pi \llbracket (P', W') \rrbracket$, *and*
- *If* $\llbracket (P, W) \rrbracket_n \xrightarrow{\alpha}_\pi Q$, *then there exists* (P', W') *such that* $(P, W) \xrightarrow{\alpha}\!\!\!\!\rightarrow_\pi (P', W')$, *where* $\llbracket (P', W') \rrbracket = Q$.

Proof. The first clause follows straightforwardly from the definition of $\llbracket \cdot \rrbracket_p$. Note that, if $\llbracket (P, W) \rrbracket_p \xrightarrow{\tau}_\pi P'$, then it must be that $P' = \llbracket (P, W \cup W') \rrbracket_p$ for some $W' \in \mathcal{W}(\mathsf{imr}(P) - \mathsf{res}(W))$ and, by the PACSR semantics, $(P, W) \overset{p}{\twoheadrightarrow}_p (P, W \cup W')$, where $p = \mathsf{pr}(W')$.

The proof of the second clause follows by structural induction of P. Clearly, for $P = (a, n).Q$ and $P = A : Q$, the base cases, the result follows. The most interesting of the remaining cases is the one concerning the parallel composition operator which we now consider. To begin with, suppose $(P \| Q, W) \overset{\alpha}{\twoheadrightarrow}_\pi (R, W')$. Three cases exist:

- $\alpha \in \mathcal{D}_E$ and $(P, W) \overset{\alpha}{\twoheadrightarrow}_\pi (P', W)$. By the induction hypothesis, $\llbracket (P, W) \rrbracket_n \xrightarrow{\alpha}_\pi \llbracket (P', W) \rrbracket_n$ and, since $\llbracket (P \| Q, W) \rrbracket_n = \llbracket (P, W) \rrbracket_n \| \llbracket (Q, W) \rrbracket_n$, we conclude that

$$\llbracket (P \| Q, W) \rrbracket_n \xrightarrow{\alpha}_\pi \llbracket (P', W) \rrbracket_n \| \llbracket (Q, W) \rrbracket_n = \llbracket (P' \| Q, W) \rrbracket_n.$$

- $\alpha \in \mathcal{D}_R$ and $(P, W) \overset{A_1}{\twoheadrightarrow}_\pi (P', \emptyset)$, $(Q, W) \overset{A_2}{\twoheadrightarrow}_\pi (Q', \emptyset)$, $\alpha = A_1 \cup A_2$. By the induction hypothesis, $\llbracket (P, W) \rrbracket_n \xrightarrow{A_1}_\pi \llbracket (P', \emptyset) \rrbracket_n$, $\llbracket (Q, W) \rrbracket_n \xrightarrow{A_2}_\pi \llbracket (Q', \emptyset) \rrbracket_n$ and, thus, since $\llbracket (P \| Q, W) \rrbracket_n = \llbracket (P, W) \rrbracket_n \| \llbracket (Q, W) \rrbracket_n$,

$$\llbracket (P \| Q, W) \rrbracket_n \xrightarrow{\alpha}_\pi \llbracket (P', \emptyset) \rrbracket_n \| \llbracket (Q', \emptyset) \rrbracket_n = \llbracket (P' \| Q', \emptyset) \rrbracket_n.$$

- $\alpha = \tau$ and $(P, W) \overset{(a?,m)}{\twoheadrightarrow}_\pi (P', W)$, $(Q, W) \overset{(a!,n)}{\twoheadrightarrow}_\pi (Q', W)$. By the induction hypothesis, $\llbracket (P, W) \rrbracket_n \xrightarrow{(a?,m)}_\pi \llbracket (P', W) \rrbracket_n$, $\llbracket (Q, W) \rrbracket_n \xrightarrow{(a!,n)}_\pi \llbracket (Q', W) \rrbracket_n$ and, thus, since $\llbracket (P \| Q, W) \rrbracket_n = \llbracket (P, W) \rrbracket_n \| \llbracket (Q, W) \rrbracket_n$

$$\llbracket (P \| Q, W) \rrbracket_n \xrightarrow{\tau}_\pi \llbracket (P', W) \rrbracket_n \| \llbracket (Q', W) \rrbracket_n = \llbracket (P' \| Q', W) \rrbracket_n.$$

The proof of the other direction uses similar arguments. □

A corollary of the above proposition follows:

Corollary 12. *If* $(P, W) \in \mathsf{S}$ *then*

$$(P, W) \overset{w}{\twoheadrightarrow} (P', W') \text{ if and only if } \llbracket (P, W) \rrbracket \xrightarrow{w'}_\pi \llbracket (P', W') \rrbracket$$

where w' *is obtained from* w *by replacing all probabilistic labels with* τ *actions.*

As a consequence of this result, we may conclude that a PACSR process deadlocks if and only if its ACSR mapping does so. Therefore, the ACSR methodology for performing schedulability analysis, including Theorem 3, can be also carried out in the PACSR setting.

We now answer the question of which properties of a PACSR process P are preserved by $\llbracket P \rrbracket$. Let us write $\mathcal{L}^p_{HMLu}$ for the fragment of $\mathcal{L}^{pr}_{HMLu}$ that contains neither negation nor a subscript of the form $\leqslant p$. We begin by defining correspondences between logics $\mathcal{L}^{pr}_{HMLu}$ and $\mathcal{L}_{HMLu}$. Specifically, for $f \in \mathcal{L}^{pr}_{HMLu}$ let $\llbracket f \rrbracket$ be the formula g which is identical to f but with all probabilistic conditions removed. Further, for $f \in \mathcal{L}_{HMLu}$ let $\|f\|$ be the formula g which is identical to f but with every *until* operator decorated by the probabilistic condition > 0. We have the following result:

Proposition 13. *For any PACSR process* P *and* $f \in \mathcal{L}^p_{HMLu}$, *if* $P \models f$ *then* $\llbracket P \rrbracket \models \llbracket f \rrbracket$.

Proof. Consider a PACSR process P. The proof is carried out by induction on the structure of the formula.

- $f = tt$. Clearly, $P \models tt$ and $\llbracket P \rrbracket \models \llbracket f \rrbracket$.
- $f = f_1 \wedge f_2$. By the induction hypothesis,

$$\begin{aligned} P \models f &\Rightarrow P \models f_1 \ \wedge \ P \models f_2 \\ &\Rightarrow \llbracket P \rrbracket \models \llbracket f_1 \rrbracket \ \wedge \ \llbracket P \rrbracket \models \llbracket f_2 \rrbracket \\ &\Rightarrow \llbracket P \rrbracket \models \llbracket f_1 \wedge f_2 \rrbracket \end{aligned}$$

as required.

- $f = f_1\langle\Phi\rangle_{\geqslant p} f_2$. Suppose that $P \models f$. Then there exists $\sigma \in \mathsf{Sched}(P)$ such that $\mathsf{Pr}_A(P, \Phi, B, \sigma, P) \geqslant p$, where $A = \{s' \mid s' \models f\}$ and $B = \{s' \mid s' \models f'\}$. This implies that there exists computation c such that $\mathsf{trace}\,(c) \in \Phi$, $\mathsf{first}\,(c) = (P, \emptyset)$, and for all $s' \in \mathsf{states}\,(\mathsf{init}\,(c))$, $s' \models f$, and $\mathsf{last}\,(c) \models f'$. By the induction hypothesis and Corollary 12, we may conclude that $\llbracket P \rrbracket \models \llbracket f \rrbracket$.
- $f = f_1\langle\Phi\rangle^t_{\geqslant p} f_2$. The proof follows similarly to the previous case. □

Note that this result fails to hold for formulae containing negation as exhibited by the following counter-example. Consider formula $f = tt\langle a\rangle_{\geqslant 0.5} tt$ which expresses that label a occurs with probability at least 0.5. Suppose that $P \models \neg f$ and that P can perform a with probability 0.4, that is, there exist computations that may perform a but their probability measure does not add up to 0.5. Nonetheless, it is not the case that $\llbracket P \rrbracket \models \llbracket \neg f \rrbracket$, since this would entail that P is incapable of performing a, which is clearly not true.

The following proposition is particularly important for the analysis of PACSR properties, since it allows us to perform analysis of a PACSR process P with respect to a non-probabilistic logical specification.

Proposition 14. *For any PACSR process P and $g \in \mathcal{L}_{HMLu}$, $P \models \|g\|$ if and only if $\llbracket P \rrbracket \models g$.*

Proof. Consider a PACSR process P and a $\mathcal{L}_{HMLu}$ property g. The proof is carried out by induction on the structure of g.

- $g = tt$. Clearly, $P \models \|tt\|$ and $\llbracket P \rrbracket \models tt$.
- $g = \neg f$. Since $P \models g$, it is not the case that $P \models f$ and, by the induction hypothesis, $\llbracket P \rrbracket \models g$.
- $g = g_1 \wedge g_2$. By the induction hypothesis,

$$\begin{aligned} P \models g &\Leftrightarrow P \models \|g_1\| \ \wedge \ P \models \|g_2\| \\ &\Leftrightarrow \llbracket P \rrbracket \models g_1 \ \wedge \ \llbracket P \rrbracket \models g_2 \\ &\Leftrightarrow \llbracket P \rrbracket \models g_1 \wedge g_2 \end{aligned}$$

  as required.
- $g = g_1\langle\Phi\rangle g_2$, $\|g\| = \|g_1\|\langle\Phi\rangle_{>0}\|g_2\|$. If $P \models \|g\|$, then there exists $\sigma \in \mathsf{Sched}(P)$ such that $\mathsf{Pr}_A(P, \Phi, B, \sigma, P) > 0$, where $A = \{s' \mid s' \models \|g_1\|\}$ and $B = \{s' \mid s' \models \|g_2\|\}$. This is equivalent to the existence of a computation c such that $\mathsf{trace}\,(c) \in \Phi$, $\mathsf{first}\,(c) = (P, \emptyset)$, and for all $s' \in \mathsf{states}\,(\mathsf{init}\,(c))$, $s' \models \|g_1\|$, and $\mathsf{last}\,(c) \models \|g_2\|$. By the induction hypothesis and Corollary 12, this is equivalent to $\llbracket P \rrbracket \models g$.
- $g = g_1\langle\Phi\rangle^t g_2$, $\|g\| = \|g_1\|\langle\Phi\rangle^t_{>0}\|g_2\|$. This is similar to the previous case. □

We will next prove that $\llbracket \cdot \rrbracket$ preserves bisimilarity of processes. Specifically, we will show that for any two PACSR processes P and Q, $(P, W) \sim_p (Q, W)$ if and only if $\llbracket (P, W) \rrbracket \sim \llbracket (Q, W) \rrbracket$. An important observation in achieving the result is that, if $P \sim Q$ then $\mathsf{imr}(P) = \mathsf{imr}(Q)$. This is due to the fact that, if $r \in \mathsf{imr}(P)$, then $P \xrightarrow{A}_\pi P'$ for some process P' and some action A with $r \in \rho(A)$ (this can be verified by a trivial structural induction on P). Thus, if P and Q are bisimilar, $Q \xrightarrow{A}_\pi$ and $r \in \mathsf{imr}(Q)$. A symmetric result holds for strong probabilistic bisimulation, i.e., if $(P, W) \sim_p (Q, W)$, then $\mathsf{imr}(P) = \mathsf{imr}(Q)$.

Theorem 15. *For every pair of PACSR processes P and Q, $(P, W) \sim_p (Q, W)$ if and only if $\llbracket (P, W) \rrbracket \sim \llbracket (Q, W) \rrbracket$.*

Proof. Suppose that $(P, W) \sim_p (Q, W)$ and consider the relation

$$\mathcal{R} = \{(\llbracket (P, W) \rrbracket, \llbracket (Q, W) \rrbracket) \mid (P, W) \sim_p (Q, W)\}.$$

Two cases exist:

- Suppose $(P, W) \in \mathsf{S}_n$. Then, if $\llbracket (P, W) \rrbracket \xrightarrow{\alpha}_\pi R$, by Proposition 11, $(P, W) \xrightarrow{\alpha}\!\!\!\!\rightarrow_\pi (P', W')$, where $R = \llbracket (P', W') \rrbracket$. Then, since $(P, W) \sim_p (Q, W)$, $(Q, W) \xrightarrow{\alpha}\!\!\!\!\rightarrow_\pi (Q', W')$, where $(P', W') \sim_p (Q', W')$. By, Proposition 11, $\llbracket (Q, W) \rrbracket \xrightarrow{\alpha}_\pi \llbracket (Q', W') \rrbracket$ with $(\llbracket (P', W') \rrbracket, \llbracket (Q', W') \rrbracket \in \mathcal{R}$ as required.

- Suppose $(P, W) \in \mathsf{S}_p$. Then, if $\llbracket (P, W) \rrbracket \xrightarrow{\tau}_\pi R$, by Proposition 11, $(P, W) \overset{p}{\twoheadrightarrow}_p (P, W')$, where $R = \llbracket (P, W') \rrbracket$. Then, since $(P, W) \sim_p (Q, W)$, $(Q, W) \overset{p'}{\twoheadrightarrow}_p (Q, W')$, where $(P, W') \sim_p (Q, W')$, and, again by Proposition 11, $\llbracket (Q, W) \rrbracket \xrightarrow{\tau}_\pi \llbracket (Q, W') \rrbracket$ and $(\llbracket (P, W') \rrbracket, \llbracket (Q, W') \rrbracket) \in \mathcal{R}$ as required.
  Consequently, $\mathcal{R}$ is a strong bisimulation as required.

  For the other way round suppose that $\llbracket (P, W) \rrbracket \sim \llbracket (Q, W) \rrbracket$ and consider the relation:

$$\mathcal{R} = \{((P, W), (Q, W)) \mid \llbracket (P, W) \rrbracket \sim \llbracket (Q, W) \rrbracket\}.$$

Two cases exist:

- Suppose $(P, W) \in \mathsf{S}_n$. Then, if $(P, W) \overset{\alpha}{\twoheadrightarrow}_\pi (P', W')$, by Proposition 11, $\llbracket (P, W) \rrbracket \xrightarrow{\alpha}_\pi \llbracket (P', W') \rrbracket$. Since $\llbracket (P, W) \rrbracket \sim \llbracket (Q, W) \rrbracket$ we have that $\llbracket (Q, W) \rrbracket \xrightarrow{\alpha}_\pi R$ and $\llbracket (P', W') \rrbracket \sim R$. Application of Proposition 11 yields $(Q, W') \overset{\alpha}{\twoheadrightarrow}_\pi (Q', W')$, where $\llbracket (Q', W') \rrbracket = R$. Thus, $((P', W'), (Q', W')) \in \mathcal{R}$ as required.
- Suppose $(P, W) \in \mathsf{S}_p$ and suppose $\mathcal{M}$ is an equivalence class of $\mathcal{R}$. Then, if $(P, W) \overset{p}{\twoheadrightarrow}_p (P, W') \in \mathcal{M}$, by Proposition 11, $\llbracket (P, W) \rrbracket \xrightarrow{\tau}_\pi \llbracket (P, W') \rrbracket$. Since $\llbracket (P, W) \rrbracket \sim \llbracket (Q, W) \rrbracket$, $\llbracket (Q, W) \rrbracket \xrightarrow{\tau}_\pi \llbracket (Q, W') \rrbracket$, and by Proposition 11, $(P, W) \overset{p}{\twoheadrightarrow}_p (P, W')$. Since this holds for every member of $\mathcal{M}$, we conclude that $\mu((P, W), \mathcal{M}) = \mu((Q, W), \mathcal{M})$, as required.
  Consequently, $\mathcal{R}$ is a probabilistic strong bisimulation which completes the proof. □

This result is especially interesting for the following reasons. First, it justifies our choice of mapping from PACSR to ACSR and it highlights the natural relationship between the two. Further, it implies the rather surprising result that, two probabilistic systems are equivalent if and only if their non-probabilistic projections exhibit bisimilar behavior. This can be interpreted as follows: two PACSR processes are bisimilar irrespectively of the probability of failure of the resources on which they operate given, of course, that they operate in the same resource environment.

4.2. From ACSR to PACSR

For completeness, we also consider the reverse mapping that transforms ACSR processes into PACSR processes. We define $[\![\cdot]\!] : \mathsf{Proc} \mapsto \mathsf{S}$ by mapping each ACSR process P to the PACSR configuration $(P, \emptyset)$ and assigning to each resource of P probability of failure equal to 0. That is, none of the resources in the resulting PACSR process ever fails. Interestingly, the set of logical formulae preserved by the new mapping is quite different compared to the mapping considered in Section 4.1.

The following proposition shows that the proposed mapping preserves the basic branching of an ACSR process but introduces a single probabilistic action with probability label 1 before each state featuring a timed action.

Proposition 16

(1) *If* $\mathsf{imr}(P) = \emptyset$, *then*
- *if* $P \xrightarrow{\alpha}_\pi Q$ *then* $[\![P]\!] \overset{\alpha}{\twoheadrightarrow}_\pi [\![Q]\!]$, *and*
- *if* $[\![P]\!] \overset{\alpha}{\twoheadrightarrow}_\pi (R, W)$, *then* $P \xrightarrow{\alpha}_\pi Q$ *with* $(R, W) = [\![Q]\!]$.

(2) *If* $\mathsf{imr}(P) \neq \emptyset$, *then*
- *if* $P \xrightarrow{\alpha}_\pi Q$ *then* $[\![P]\!] \overset{1}{\twoheadrightarrow}_p \overset{\alpha}{\twoheadrightarrow}_\pi [\![Q]\!]$, *and*
- *if* $[\![P]\!] \overset{p}{\twoheadrightarrow}_p (Q, W)$, *then* $p = 1$, *and* $(Q, W) \overset{\alpha}{\twoheadrightarrow}_\pi (Q', W')$ *if and only if* $P \xrightarrow{\alpha}_\pi P'$ *with* $(Q', W') = [\![P']\!]$.

Proof. The proof follows directly from the mapping and the PACSR semantics. □

A corollary of the above proposition follows:

Corollary 17

- *If* $P \xrightarrow{w}_{\pi} Q$ *then* $[\![P]\!] \xrightarrow{w'}\!\!\!\!\rightarrow_{\pi} [\![Q]\!]$, *where* w' *is obtained from* w *by adding a probabilistic label with value* 1 *before every timed action.*
- *If* $[\![P]\!] \xrightarrow{w}\!\!\!\!\rightarrow_{\pi} [\![Q]\!]$ *then* $P \xrightarrow{w'}_{\pi} Q$, *where* w' *is obtained from* w *by removing all probabilistic labels.*

We now turn to deciding which properties of an ACSR process P are preserved by $[\![P]\!]$. We define correspondences between logics $\mathcal{L}^{pr}_{HMLu}$ and $\mathcal{L}_{HMLu}$. Specifically, for $f \in \mathcal{L}_{HMLu}$, let $[\![f]\!]$ be the formula g which is identical to f but with every *until* operator decorated by the probabilistic transition $\geqslant 1$. Further, for $f \in \mathcal{L}^{pr}_{HMLu}$, we define $\lceil\!\lceil f \rceil\!\rceil$ as the $\mathcal{L}_{HMLu}$ formula obtained from f by replacing (1) each subformula $g = g_1\langle\Phi\rangle_{\leqslant p} g_2$ by tt if $p = 1$ and by $\neg(\lceil\!\lceil g_1 \rceil\!\rceil\langle\Phi\rangle\lceil\!\lceil g_2 \rceil\!\rceil)$ otherwise, and (2) each subformula $g = g_1\langle\Phi\rangle_{\geqslant p} g_2$ by tt, if $p = 0$ and $\lceil\!\lceil g_1 \rceil\!\rceil\langle\Phi\rangle\lceil\!\lceil g_2 \rceil\!\rceil$, otherwise. We have the following results:

Proposition 18. *For any ACSR process* P *and* $f \in \mathcal{L}_{HMLu}$, $P \models f$ *if and only if* $[\![P]\!] \models [\![f]\!]$.

Proof. Consider an ACSR process P. The proof is carried out by induction on the structure of the formula.

- The cases $g = tt$, $g = \neg f$ and $g = g_1 \wedge g_2$ follow trivially.
- $f = f_1\langle\Phi\rangle f_2$, $[\![f]\!] = [\![f_1]\!]\langle\Phi\rangle_{\geqslant 1}[\![f_2]\!]$. If $P \models f$, then there exists computation c such that $\mathsf{trace}(c) \in \Phi$, $\mathsf{first}(c) = P$, and for all $s' \in \mathsf{states}(\mathsf{init}(c))$, $s' \models f$, and $\mathsf{last}(c) \models f'$. By Corollary 17 we may conclude that a similar computation with probability measure equal to 1 exists for $[\![P]\!]$. Then, by the induction hypothesis, we have that $[\![P]\!] \models [\![f]\!]$. The opposite direction similarly holds.
- $f = f_1\langle\Phi\rangle^t f_2$. The proof is similar to that of the previous case. □

Thus, any $\mathcal{L}_{HMLu}$ property satisfied by an ACSR process P is also satisfied by its PACSR mapping with probability 1. This is intuitive, since all computations of P survive in P with probability of occurrence equal to 1.

On the other hand, an $\mathcal{L}^{pr}_{HMLu}$ property f satisfied by the mapping of P is abstracted into $\mathcal{L}_{HMLu}$ as follows: any subformula f' of f with probabilistic condition < 1 is in fact satisfied neither by process P nor by its abstraction, since, as we have seen above, all properties are either satisfied, with probability 1, or not satisfied. Finally, if a subformula has a probabilistic condition $\geqslant p$, $p \neq 0$, then it is satisfied in both processes. We prove this below.

Proposition 19. *For any ACSR process* P *and* $g \in \mathcal{L}^{pr}_{HMLu}$, $P \models \lceil\!\lceil g \rceil\!\rceil$ *if and only if* $[\![P]\!] \models g$.

Proof. Consider an ACSR process P and a $\mathcal{L}^{pr}_{HMLu}$ property g. The proof is carried out by induction on the structure of g.

- The cases $g = tt$, $g = \neg f$ and $g = g_1 \wedge g_2$ follow trivially.
- $g = g_1\langle\Phi\rangle_{\geqslant p} g_2$, $p \neq 0$. If $[\![P]\!] \models g$, then there exists computation c such that $\mathsf{trace}(c) \in \Phi$, $\mathsf{first}(c) = (P, \emptyset)$, and for all $s' \in \mathsf{states}(\mathsf{init}(c))$, $s' \models f$, and $\mathsf{last}(c) \models f'$. By the nature of the mapping, all probabilistic labels on this transition are equal to 1. Thus, the probability measure of the computation is equal to 1 and in fact we may conclude that $[\![P]\!] \models g_1\langle\Phi\rangle_{\geqslant 1} g_2$. By Corollary 17 above we may conclude that a similar computation, but with all probabilistic labels removed, exists for P. Then, by the induction hypothesis, we have that $[\![P]\!] \models [\![f]\!]$. The opposite direction similarly holds.
- $g = g_1\langle\Phi\rangle_{\leqslant p} g_2$, $p \neq 1$. Suppose that $[\![P]\!] \models g$ and that there exists computation c such that $\mathsf{trace}(c) \in \Phi$, $\mathsf{first}(c) = (P, \emptyset)$, and for all $s' \in \mathsf{states}(\mathsf{init}(c))$, $s' \models f$, and $\mathsf{last}(c) \models f'$. By the nature of the mapping, all probabilistic labels on this transition are equal to 1. Thus, the probability measure of the computation is equal to 1 and we obtain a contradiction to the assumption that $[\![P]\!] \models g$. By Corollary 17 and the induction hypothesis, we may conclude that there exists no computation c from P such that $\mathsf{trace}(c) \in \Phi$, $\mathsf{first}(c) = P$, and for all $s' \in \mathsf{states}(\mathsf{init}(c))$, $s' \models f$, and $\mathsf{last}(c) \models f'$. Therefore, $P \models \neg\lceil\!\lceil g \rceil\!\rceil$, as required. The opposite direction similarly holds.
- The cases $g = g_1\langle\Phi\rangle^t_{\geqslant p} g_2$ and $g = g_1\langle\Phi\rangle^t_{<p} g_2$ follow similarly to the two previous cases. □

We will next prove that $[\![\cdot]\!]$ preserves bisimilarity of processes. Specifically, we will show that for any two ACSR processes P and Q, $P \sim Q$ if and only if $[\![P]\!] \sim_p [\![Q]\!]$.

Proposition 20. *For every pair of ACSR processes P and Q, $P \sim Q$ if and only if $[\![P]\!] \sim_p [\![Q]\!]$.*

Proof. Suppose that $P \sim Q$ and consider the relation

$$\mathcal{R} = \{((P, W), (Q, W)) \mid P \sim Q\}.$$

Two cases exist:

- Suppose $(P, W) \in \mathsf{S}_n$. Then, if $(P, W) \stackrel{\alpha}{\longrightarrow\!\!\!\!\!\rightarrow}_\pi (P', W')$, it also holds that $P \stackrel{\alpha}{\longrightarrow}_\pi P'$ and, since $P \sim Q$, $Q \stackrel{\alpha}{\longrightarrow}_\pi Q'$, where $P' \sim Q'$, yielding $(Q, W) \stackrel{\alpha}{\longrightarrow\!\!\!\!\!\rightarrow}_\pi (Q', W')$ with $((P', W'), (Q', W')) \in \mathcal{R}$ as required.
- Suppose $(P, W) \in \mathsf{S}_p$. Then, since all resources in P have probability of failure 0, $(P, W) \stackrel{1}{\longrightarrow\!\!\!\!\!\rightarrow}_p (P, W \cup W')$, where $W' = \mathsf{imr}(P) - \mathsf{res}(W)$. Since $P \sim Q$, $\mathsf{imr}(P) = \mathsf{imr}(Q)$ and, thus, $(Q, W) \stackrel{1}{\longrightarrow\!\!\!\!\!\rightarrow}_p (Q, W \cup W')$ and $((P, W \cup W'), (Q, \cup W')) \in \mathcal{R}$ as required.

  Consequently, $\mathcal{R}$ is a probabilistic strong bisimulation and since $[\![P]\!] = (P, \emptyset)$ $[\![Q]\!] = (Q, \emptyset)$ the result follows.

  Now, consider the other way round and suppose that $[\![P]\!] \sim_p [\![Q]\!]$. Consider the relation:

$$\mathcal{R} = \{(P, Q) \mid [\![P]\!] \sim_p [\![Q]\!]\}.$$

Two cases exist:

- Suppose $\mathsf{imr}(P) = \emptyset$. Then, if $P \stackrel{\alpha}{\longrightarrow}_\pi P'$, by Proposition 16, $[\![P]\!] \stackrel{\alpha}{\longrightarrow\!\!\!\!\!\rightarrow}_\pi [\![P']\!]$. Since $[\![P]\!] \sim_p [\![Q]\!]$, $[\![Q]\!] \stackrel{\alpha}{\longrightarrow\!\!\!\!\!\rightarrow}_\pi \mathcal{R}$ and $[\![P']\!] \sim_p R$, and again, by Proposition 16, $Q \stackrel{\alpha}{\longrightarrow}_\pi Q'$, where $[\![Q']\!] = R$. Thus, $(P', Q') \in \mathcal{R}$ as required.
- Suppose $\mathsf{imr}(P) \neq \emptyset$. Then, if $P \stackrel{\alpha}{\longrightarrow}_\pi P'$, by Proposition 16, $[\![P]\!] \stackrel{1}{\longrightarrow\!\!\!\!\!\rightarrow}_p R' \stackrel{\alpha}{\longrightarrow\!\!\!\!\!\rightarrow}_\pi R$. Since $[\![P]\!] \sim_p [\![Q]\!]$, $[\![Q]\!] \stackrel{1}{\longrightarrow\!\!\!\!\!\rightarrow}_p S'$ and $R' \sim_p S'$ and $S' \stackrel{\alpha}{\longrightarrow\!\!\!\!\!\rightarrow}_\pi S \sim_p R$. Then, by Proposition 16, $Q \stackrel{\alpha}{\longrightarrow}_\pi Q'$, where $[\![Q']\!] = S$. Thus, $(P', Q') \in \mathcal{R}$ as required.

  Consequently, $\mathcal{R}$ is a strong bisimulation which completes the proof. □

This time the bisimulation preservation of the mapping is not surprising since the translation given by the mapping neither introduces nor abstracts away information of an ACSR process.

5. Conclusions

We have presented two resource-bound real-time process algebras from the ACSR family: the basic algebra ACSR and its probabilistic extension PACSR. ACSR was developed to handle schedulability analysis in a process-algebraic setting by introducing the notion of a shared resource into the formalism. PACSR extends ACSR by elaborating on the nature of resources to support the notion of probabilistic resource failures. We have shown that, in ACSR, schedulability analysis can be applied by recasting the problem of whether a system is schedulable into deadlock detection. On the other hand, PACSR modeling is primarily concerned with quantitative evaluation of system specifications. For example, one might like to compute the probability that a system remains schedulable in the presence of resource failures. The analysis techniques we have proposed are capable of performing such analysis.

In our presentation we have focused on illustrating the basic features of the two formalisms and highlighting the intentions behind various design choices as well as subtle interactions between constructs that play a crucial role for specifying and verifying systems. We have also introduced a compositional result for tracing the source of undesirable deadlocks in system models and, in the context of ACSR, we have introduced an HML logic with *until*, featuring regular expressions over observables as parameters. Finally, we have proposed property-preserving mappings between ACSR and PACSR which highlight the natural relationship between the two and allow us to apply analysis techniques of one to the other.

As mentioned in Section 1, the family of resource-bound real-time process algebras includes several more formalisms. There is a formalism for visual specification of ACSR and a version of ACSR for the dense-time domain.

P^2ACSR introduces additional resource attributes to reason about power consumption. MCSR extends ACSR to handle multicapacity resources, allowing us to handle systems with memory constraints. Finally, ACSR-VP extends ACSR with a value-passing capability during communication and parameterized process definitions. In the ACSR-VP setting, bisimulation plays a central role for performing schedulability analysis: System specifications may contain a number of unspecified parameters, and the purpose of the analysis performed is to specify values for these parameters that make the system schedulable. To achieve this, symbolic bisimulation is employed between the parameterized system and the process that idles forever, and, with the aid of integer programming, appropriate ranges for the parameters that make the system schedulable are computed.

In future work we aim to identify additional classes of resources and develop means of incorporating them into a unified formalism, as well as to provide flexible tool support for model development in the formalism. An interesting extension to the current work is to go beyond serially reusable resources to *consumable* resources, which can be used only a fixed amount of times during a computation and can be possibly replenished after a certain amount of time.

References

[1] K. Altisen, G. Goessler, J. Sifakis, Scheduler modeling based on the controller synthesis paradigm, J. Real-Time Syst., 23 (1–2) (2002) 55–84.
[2] H. Ammar, V. Cortellessa, A. Ibrahim, Modeling resources in a UML-based simulative environment, in: Proceedings of AICCSA'01, IEEE Computer Society Press, 2001, pp. 405–410.
[3] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, W. Yi, TIMES: A tool for schedulability analysis and code generation of real-time systems, in: Proceedings of FORMATS'03, LNCS 2791, 2003, pp. 60–72.
[4] L. Baum, T. Kramp, Towards a uniform modeling technique for resource-usage scenarios, in: Proceedings of PDPTA'99, 1999, pp. 1324–1329.
[5] H. Ben-Abdallah, GCSR: a graphical language for the specification, refinement and analysis of real-time systems. PhD thesis, Department of Computer and Information Science, University of Pennsylvania, 1996.
[6] J.A. Bergstra, J.W. Klop, Algebra of communicating processes with abstraction, J. Theoret. Comput. Sci. 37 (1985) 77–121.
[7] V.A. Braberman, M. Felder, Verification of real-time designs: combining scheduling theory with automatic formal verification, in: Proceedings of ESEC'99, LNCS 1687, 1999, pp. 494–510.
[8] P. Brémond-Grégoire, J.-Y. Choi, I. Lee, A complete axiomatization of finite-state ACSR processes, Inform. and Comput. 138 (2) (1997) 124–159.
[9] P. Brémond-Grégoire, I. Lee, Process algebra of communicating shared resources with dense time and priorities, Theoret. Comput. Sci. 189 (1–2) (1997) 179–219.
[10] M. Buchholtz, J. Andersen, H.H. Loevengreen, Towards a process algebra for shared processors, Electron. Notes Theoret. Comput. Sci. 52 (3) (2001).
[11] J.-Y. Choi, I. Lee, H.-L. Xie, The specification and schedulability analysis of real-time systems using ACSR, in: Proceeding of RTSS'95, IEEE Computer Society Press, 1995, pp. 266–275.
[12] A. Chakrabarti, L. de Alfaro, T.A. Henzinger, M. Stoelinga, Resource interfaces, in: Proceedings of EMSOFT'03, LNCS 2855, 2003, pp. 117–133.
[13] E.M. Clarke, E.A. Emerson, A.P. Sistla, Automatic verification of finite-state concurrent systems using temporal logic specifications, ACM Trans. Program. Lang. Syst. 8 (2) (1986) 244–263.
[14] R. De Nicola, F.W. Vaandrager, Three logics for branching bisimulation, Proceedings of LICS'90, IEEE Computer Society Press, 1990, pp. 118–129.
[15] J. Ermont, F. Boniol, TPAP: an algebra of preemptive processes for verifying real-time systems with shared resources, Electron. Notes Theoret. Comput. Sci. 65 (6) (2002).
[16] H. Hansson, Time and probability in formal design of distributed systems. Ph.D. thesis, Department of Computer Systems, Uppsala University, 1994.
[17] C.A.R. Hoare, Communicating Sequential Processes, Prentice-Hall, 1985.
[18] E. Huh, L. Welch, B. Shirazi, C. Cavanaugh, Heterogeneous resource management for dynamic real-time systems, in: Proceedings of Heterogeneous Computing Workshop 2005, 2000, pp. 287–296.
[19] H.-H. Kwak, I. Lee, A. Philippou, J.Y. Choi, O. Sokolsky, Symbolic schedulability analysis of real-time systems, Proceedings RTSS'98, IEEE Computer Society Press, 1998, pp. 409–419.
[20] K. Larsen, A. Skou, Bisimulation through probabilistic testing, Inform. Comput. 94 (1) (1991) 1–28.
[21] I. Lee, P. Brémond-Grégoire, R. Gerber, A process-algebraic method for the specification and analysis of real-time systems, Proc. IEEE (1994) 158–171.
[22] I. Lee, H. Ben-Abdallah, J.-Y. Choi, A process-algebraic approach to the specification and analysis of resource-bound real-time systems, Formal Methods for Real-Time Computing, John Wiley and Sons, 1996, pp. 167–194.
[23] I. Lee, A. Philippou, O. Sokolsky, A general resource framework for real-time systems, in: Proceedings of RISSEF'02, LNCS 2941, 2002, pp. 234–248.
[24] C.L. Liu, J.W. Layland, Scheduling algorithms for multi-programming in a hard-real-time environment, J. ACM 20 (1) (1973) 46–61.

[25] A. Mehra, A. Indiresan, K.G. Shin, Resource management for real-time communication: Making theory meet practice, in: Proceedings of RTAS'96, IEEE Computer Society Press, 1996, pp. 130–138.
[26] R. Milner, Communication and Concurrency, Prentice-Hall, 1989.
[27] M. Mousavi, M. Reniers, T. Basten, M. Chaudron, PARS: a process algebra with resources and schedulers, in: Proceedings of FORMATS'03, LNCS 2791, 2003, pp. 134–150.
[28] M. Nunez, I. Rodriguez, PAMR: A process algebra for the management of resources in concurrent systems, in: Proceedings of FORTE'01, Kluwer, 2001, pp. 169–184.
[29] D. Park, Concurrency and automata on infinite sequences, in: Proceedings of the 5th GI Conference, LNCS 104, 1981, pp. 167–183.
[30] A. Philippou, R. Cleaveland, I. Lee, S. Smolka, O. Sokolsky, Probabilistic resource failure in real-time process algebra, in: Proceedings of CONCUR'98, LNCS 1466, 1998, pp. 389–404.
[31] A. Philippou, O. Sokolsky, Process-algebraic analysis of timing and schedulability properties, Handbook of Real-Time and Embedded Systems, Chapman and Hall/CRC, in press.
[32] A. Philippou, O. Sokolsky, I. Lee, Weak bisimulation for probabilistic systems, in: Proceedings of CONCUR'00, LNCS 1877, 2000, pp. 334–349.
[33] S. Saewong, R. Rajkumar, Cooperative scheduling of multiple resources, in: Proceedings of RTSS'99, IEEE Computer Society Press, 1999, pp. 90–101.
[34] O. Sokolsky, I. Lee, H. Ben-Abdallah, Specification and analysis of real-time systems with PARAGON, Ann. Software Eng. 7 (1999) 211–234.
[35] O. Sokolsky, A. Philippou, I. Lee, K. Christou, Modeling and analysis of power-aware systems, in: Proceedings of TACAS'03, LNCS 2619, 2003, pp. 409–425.
[36] M. Vardi, Automatic verification of probabilistic concurrent finite-state programs, in: Proceedings of FOCS'85, 1985, pp. 327–338.