Chapter 1

PARS: A Process Algebraic Approach to Resources and Schedulers

1.1 Introduction

Scheduling theory has a rich and long history. In addition, process algebras have been studied as a formal theory of system design and verification since the early 1980's. However, these two separate worlds have not been connected until recent years and still the connection is not yet complete. In other words, using the models and algorithms of scheduling theory in a process algebraic design is still involved with many theoretical and practical complications. In this chapter, building upon previous attempts in this direction, we propose a process algebra, called *PARS* for Process Algebra with Resources and Schedulers, for the design of scheduled real-time systems. Previous attempts to incorporate scheduling algorithms in process algebra either did not have an explicit notion of schedulers [5, 15, 16] (thus, coding the scheduling policy in the process specification) or scheduling is treated for restricted cases that only support single-processor scheduling [6, 12].

Our approach to modeling scheduled systems is depicted in Figure 1.1. In this approach, process specification (including aspects such as causal relations of actions, their timing and resource requirements) is separated from the specification of schedulers. Then one can apply schedulers to processes to obtain scheduled systems and further compose scheduled systems together. A distinguishing feature of our process algebra is the possibility of specifying schedulers as process terms (similar to resource-consuming processes). Another advantage of the proposed approach is the separation between process specification and scheduler specification that provides a separation of concerns, allows for specifying generic scheduling strategies and makes it possible to apply schedulers to systems at different levels of abstraction. Common to most process algebraic frameworks for resources, the proposed framework provides the possibility of extending standard schedulability analysis to the formal verification process.

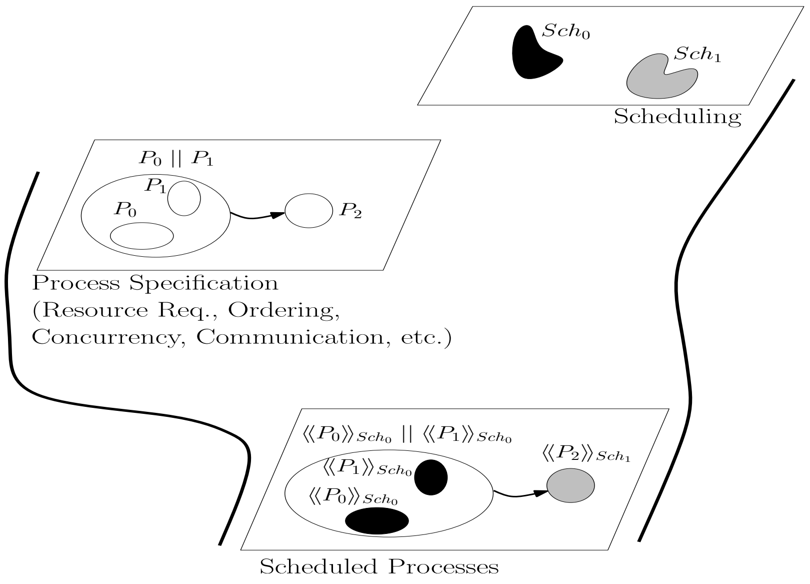
Sch_0
Sch_1
Scheduling
$P_0 \mid\mid P_1$
P_1
P_0
P_2
Process Specification
(Resource Req., Ordering,
Concurrency, Communication, etc.)
$\langle\langle P_0 \rangle\rangle_{Sch_0} \mid\mid \langle\langle P_1 \rangle\rangle_{Sch_0}$
$\langle\langle P_1 \rangle\rangle_{Sch_0}$
$\langle\langle P_0 \rangle\rangle_{Sch_0}$
$\langle\langle P_2 \rangle\rangle_{Sch_1}$
Scheduled Processes

FIGURE 1.1: Schematic view of the *PARS* approach

Related Work Several theories of process algebra with resources have been proposed recently. Our approach is mainly based on dense-time ACSR of [5]. ACSR [16, 15] is a process algebra enriched with possibilities to specify priorities and resources. Several extensions to ACSR have been proposed over time for which [16] provides a summary. The main shortcoming of this process algebra is the absence of an explicit scheduling concept. In this approach, the scheduling strategy is coded by means of priorities inside the process specification domain. Due to absence of a resource provision model, some other restrictions are also imposed on resource demands of processes. For example, two parallel processes are not allowed to call for the same resource or they deadlock.

Our work has also been inspired by [6]. In [6], a process algebraic approach to resource modeling is presented and application of scheduling to process terms is investigated. This approach has an advantage over that of ACSR in that scheduling is separated from the process specification domain. However, firstly, there is no structure or guideline to define schedulers in this language (as [16] puts it, the approach looks like defining a new language semantics for each scheduling strategy) and secondly, the scheduling is restricted to a single resource (single CPU) concept.

Scheduling algebra of [14] defines a process algebra that has processes with interval timing. In order to have an efficient scheduling, actions are supposed to be scheduled without delay or only after another action terminates (so-called *anchor points*). The semantics of the process algebra takes care of defining and extending anchor points over process structure. Since scheduling algebra abstracts from resources, the notion of scheduling is also very abstract and comes short of specifying examples of scheduling strategies such as those that we specify in the remainder of this chapter.

RTSL of [12] defines a discrete-time process algebra for scheduling analysis of single processor systems. The basic process language allows for specifying tasks as sequential processes and a system language takes care of composition of tasks (using parallel composition) and selecting the higher priority active tasks. Furthermore, the approach studies the issue of exception handling in case of missed deadlines. Similar to [6], there is no need for an explicit notion of resources in RTSL, since the only shared resource is the single CPU. The restriction of tasks, in this approach, to sequential processes makes the language less expressive than ours (for example, in the process language a periodic task whose execution time is larger than its period cannot be specified). Also, coding the scheduling policy in terms of a priority function may make specification of scheduling more cumbersome (similar to [6]).

Asynchrony in timed parallel composition (interleaving of relative timed-transitions) has received little interest in timed process algebras. Semantics of parallel composition in ATP [20] and different versions of timed-ACP [2], timed-CCS [8, 10] and timed-CSP [11] all enforce synchronization of timed transitions such that both parallel components evolve in time. The *cIPA* of [1] is among a few timed process algebras that contain a notion of timed asynchrony. In this process algebra non-synchronizing actions are forced to make asynchronous (interleaving) time transitions and synchronizing actions are specified to perform synchronous (concurrent) time transition. We do not see this distinction necessary since non-synchronizing actions may find enough resources to execute in true concurrency and synchronizing actions may be forced to make interleaving time transitions due to the use of shared resources (e.g., scheduling two synchronizing actions on a single CPU). In other words, making the resource model explicit and separating it from process specification allows us to delay such kinds of design decisions and reflect them in the scheduling strategy and scheduled systems semantics.

A related (but different) issue in this regard is *laziness* and *eagerness* of actions (see [9] for a detailed account of the issue) that can lead to similar semantics as what we call an *abstract parallel composition*, which implements timed asynchrony. In general, in presence of only lazy actions, the difference between asynchronous and synchronous time (called abstract and *strict*, respectively in this paper) parallel compositions vanishes and the two types of composition behave the same. However, in our case actions are not absolutely lazy and do not fit in the general framework proposed in [9]. They can only idle if another process in their abstract parallel context can perform an action. We abstract from this implicit idling by allowing time transitions to be done in an asynchronous (interleaving) as well as synchronous (concurrent) manner. This abstraction comes handy when taking resource contention into account (where actions may be prevented from executing concurrently due to resource contention). Strict, i.e., synchronous time, parallel composition is differentiated from abstract, i.e., asynchronous time, parallel composition in this context by separating the resource concerns of its two arguments and forcing parallelism between them.

This chapter can be considered as a continuation of our previous work regarding separation of concerns in the design of embedded systems (e.g., in [17, 18]). There, we propose separation of functionality, timing and coordination aspects which are represented here in the process specification language. In the *PARS* approach, we add the aspect of scheduling to the above set by assuming a (composed) model of timed functionality and coordination underneath our theory.

The main ideas of the *PARS* approach (a separation of processes and schedulers) are used in the CARAT tool for analyzing performance and resource use of component-based embedded systems [4]. Given a set of component models, the CARAT tool builds a system model that closely resembles a *PARS* model. Subsequently, different analysis techniques can be applied to that system model. In order to reduce the analysis time of system models, mostly scenario-based analyses have been performed to get indications of best case and worst case behaviors of a system [3].

This chapter is a revised and extended version of the extended abstract that appeared as [19]. The semantics' of the process algebraic formalism presented in this chapter are improved substantially compared to those in [19], by introducing worst-case execution time and deadline predicates. Consequently, a congruence result for strong bisimilarity is obtained here, which was impossible to obtain in [19]. We also give a number of sound axioms with respect to strong bisimilarity for our process algebraic formalism and give more elaborate examples.

Structure of the chapter The chapter is organized as follows. We define the syntax and semantics of *PARS* in three parts. In Section 1.2, we build a process algebra with asynchronous relative dense time (i.e., with the possibility of interleaving timing transition) for process specifications that have a notion of resource consumption. As explained before, we consider time asynchrony relevant and helpful in this context, since it models possible delays due to resource contentions (such as implementation of multiple parallel compositions on a single or a few CPUs). Then, in Section 1.3, a similar process algebraic theory is developed for schedulers as resource providers. Subsequently, in Section 1.4, application of a scheduler to a process and composition of scheduled systems is defined. To illustrate the usage of our method, we give some examples from the literature in each section. Finally, Section 1.5 concludes the results and presents future research directions.

1.2 Process Specification in *PARS*

A specification in *PARS* consists of three parts: a process specification which represents the usual process algebraic design of the system together with resource requirements of its basic actions, a scheduler specification which specifies availability of resources and policies for providing different resources, and system specification which applies schedulers to processes and composes scheduled systems.

In our framework, resources are represented by a set R. The amount of resources required by a basic action is modeled by a function $\rho : R \to \mathbb{R}^{\geq 0}$. The resource requirement is assumed to be constant at any point of time during the action execution. However, extending this to a function of time, i.e., action duration, is a straightforward extension of our theory. The resources provided by schedulers are modeled using a function $\overline{\rho} : R \to \mathbb{R}^{\leq 0}$. Active tasks that require or provide resources are represented by multisets of such tasks in the semantics. We assume that scheduling strategies can address process identifiers, the execution time of processes, and their deadlines as typical and most common parameters for scheduling. Nevertheless, the extension of our theory with other parameters is orthogonal to our theory.

As a notational convention, we refer to the set of all multisets as $\mathbb{M}$ (we assume that the type of elements in the multiset is clear from the context). To represent a multiset extensionally (using its elements) we use the notation $[a, b, c, \ldots]$. The empty multiset is denoted by $\emptyset$ and $+$ and $-$ are overloaded to represent addition and subtraction of multisets, respectively.

The syntax of process specification in *PARS* is presented in Figure 1.2. It resembles a relative dense-time process algebra (such as relative dense-time ACP of [2]) with empty process ($\epsilon(0)$) and deadlock (δ). The main difference with such a theory is the attachment of resource requirements to basic actions (most process algebras abstract from resource requirements by assuming abundant availability of shared resources).

$$
\begin{aligned}
P ::= {} & \delta \mid p(t) \mid P \,;\, P \mid P \mid\mid P \mid P \mid\mid\mid P \mid P + P \mid \\
& \sigma_t(P) \mid \mu X.P(X) \mid \textstyle\int_{x \in T} P(x) \mid \partial_{Act}(P) \mid id : P
\end{aligned}
$$

$$
\begin{gathered}
p \in (A \times \rho) \cup \{\epsilon\},\ t \in \mathbb{R}^{\geq 0},\ X \text{ is a recursive variable,} \\
Act \subseteq A,\ x \in V_t,\ T \subseteq \mathbb{R}^{\geq 0},\ id \in \mathbb{N}
\end{gathered}
$$

FIGURE 1.2: Syntax of *PARS*, Part 1: Process Specification

Basic action $\epsilon(t)$ represents a non-action or idling lasting for t time which does not require any resource. Other basic actions $(a, \rho)(t)$ are pairs of actions from the set A together with the respective resource requirement function ρ and the timing t during which the required resources should be available for the action.

Example 1.1 Portable tasks
Suppose that the task a can be run on different platforms, either a RISC processor on which it will take 2 units of time and 100 units of memory (during those 2 time units) or on a CISC processor for which it will require 4 units of time and 70 units of memory (over the 4 time units). This gives rise to using the basic actions

$$(a, \{\mathit{RISC} \mapsto 1, \mathit{Mem} \mapsto 100\})(2)$$

and

$$(a, \{\mathit{CISC} \mapsto 1, \mathit{Mem} \mapsto 70\})(4) \ .$$

$\square$

Terms $P \ ; \ P$, $P \ || \ P$, $P \ ||| \ P$, $P + P$ represent sequential composition, abstract parallel composition, strict parallel composition, and nondeterministic choice, respectively. Abstract parallel composition refers to cases where the ordering (and possible preemption) of actions has to be decided by a scheduling strategy. In practice, it is used when two processes have no causal dependency and thus *can* run in parallel but they do not necessarily have to. Particularly, when not enough resources are provided, two components that are composed using the abstract parallel composition may be scheduled sequentially. Strict parallel composition is similar to standard parallel composition in timed process algebra in that it forces concurrent execution of the two operands. Resource-consuming processes composed by this operator should be provided with enough resources to run concurrently or they will all deadlock. Both strict and abstract parallel composition allow for synchronization of actions using the synchronization function γ, i.e., when $\gamma(a, b)$ is defined, actions a and b may synchronize resulting in $\gamma(a, b)$.

Example 1.2 Portable tasks
The following is a description of the two possibilities of executing task a from the previous example.

$$P \doteq (a, \{\mathit{RISC} \mapsto 1, \mathit{Mem} \mapsto 100\})(2) + (a, \{\mathit{CISC} \mapsto 1, \mathit{Mem} \mapsto 70\})(4)$$

$\square$

Example 1.3 (Abstract and strict parallel composition)
Consider the following two tasks a and b which can be run independently from each other.

$$P \doteq (a, \{CPU \mapsto 1\})(2) \mid\mid (b, \{CPU \mapsto 1\})(3)$$

If there is only one CPU available, then the two tasks must be scheduled sequentially and thus take 5 time units to run. If two CPU's are available, then process may complete in 3 time units.

Consider now the following task, which comprises two concurrent sub-tasks a and b that respectively need a CPU and a co-processor to run. The taks cannot commence (or continue) if not both resources are provided to the task.

$$P \doteq (a, \{CPU \mapsto 1\})(2) \mid\mid\mid (b, \{COPROC \mapsto 1\})(1)$$

$\Box$

Deadline operator $\sigma_t(P)$ specifies that process P should terminate within t units of time or it will deadlock. Recursion is specified explicitly using the expression $\mu X.P(X)$ where free variable X may occur in process P and is bound by μX. The term $\int_{x \in T} P(x)$ specifies continuous choice of timing variable x over set T. Similar to recursion, variable x is bound in term P by operator $\int_{x \in T}$. To prevent process P from performing particular actions in set Act (in particular, to force communication among two parallel parties), encapsulation term $\partial_{Act}(P)$ is used. Process terms are decorated with identifiers (natural numbers, following the idea of [6]) using the $id : P$ construct. Identifiers serve to group processes for scheduling purposes. Note that an atomic action is neither required to have an identifier, nor need its identifier be unique. In other words, there is a (possibly empty) set of identifiers attached to each process term and thus related to its comprising actions.

Precedence of binding among binary composition operators is ordered as ;, |||, ||, + where ; binds the strongest and + the weakest. Unary operators are followed by a pair of parentheses or they bind to the largest possible term. In this paper, we are only concerned with closed terms (processes that do not have free recursion or timing variables).

To show how the process specification language is to be used, we specify a few common patterns in scheduling literature [7].

Example 1.4 Periodic tasks
First, we specify a periodic task, consisting of an action a with resource requirements ρ, execution time t, and period t'.

$$P_1 \doteq \mu X.(a, \rho)(t) \mid\mid\mid \epsilon(t') \ ; X$$

Note that strict parallel composition is used here to denote that the arrival of a new task happens concurrently with the execution of an already arrived task. The $\epsilon(t')$ operator in combination with strict parallel composition enforce the duration of the period to exactly t'. The use of parallel composition allows an execution time t greater than the period t'.

Suppose that the exact execution time of a is not known, but that the execution time is within an interval I, then this is specified as follows.

$$P_2 \doteq \mu X. \left(\int_{x \in I} (a, \rho)(x) \right) \mid\mid\mid \epsilon(t') \ ; X$$

The same technique can be applied to give the scheduling of a new task a variable arrival phase within an interval. Throughout the rest of the paper, for intervals I, we use the syntactic shorthand $p(I)$ instead of $\int_{x \in I} p(x)$.

Note that in the above examples, the newly arrived task is set in strict parallel composition with the old tasks, which means that simultaneously enabled tasks should be executed concurrently. If this is not desirable, one can use a synchronization scheme, such as the one used in the following process P, to signal the arrival of the task and further put the arrived tasks in abstract parallel composition. This yields a system in which the task arrivals are strictly periodic while tasks execution is subject to resource availability.

$$\begin{aligned}
&P \doteq \partial_{\{sSignal, rSignal\}} X \mid\mid\mid Y \\
&X \doteq (\epsilon(t') \ ; (sSignal, \emptyset)(0)) \ ; X \\
&Y \doteq (rSignal, \emptyset)(0) \ ; ((a, \rho)(t) \mid\mid Y) \\
&\gamma(rSignal, sSignal) \doteq \gamma(sSignal, rSignal) \doteq signal
\end{aligned}$$

□

Example 1.5 Aperiodic tasks
Specification of aperiodic tasks follows a pattern similar to the specification of periodic tasks with the difference that, their period of arrival is not known:

$$S \doteq \mu X.(b, \rho')(t) \mid\mid\mid \epsilon([0, \infty)) \ ; X$$

If the process specification of the system consists of periodic user level tasks and aperiodic system level tasks (e.g., system interrupts) that are to be scheduled with different policies, the specification goes as follows:

$$SysProc \doteq System : (S) \mid\mid User : (P_1)$$

where *System* and *User* are distinct identifiers for these two types of tasks, and where P_1 is as specified in Example 1.4. To prevent the system from deadlocking when the resources are not available, we can compose the system with an idling process as follows:

$$\begin{aligned}
&Idle \doteq \mu X. \epsilon([0, \infty)) \ ; X \\
&SysProc_{Id} \doteq Idle \mid\mid SysProc
\end{aligned}$$

□

Semantics of process specification is given in Figures 1.3, 1.4, 1.5 and 1.6 in the style of Structural Operational Semantics of [22]. In this semantics, states are process terms from the syntax (together with an auxiliary operator defined next). Corresponding to the possible events of spending time on actions and committing them, there are two types of transitions in the semantics. First, time passage (by spending time on resources or idling) $\overset{M,t}{\rightarrow}$ ($t \in \mathbb{R}^{>0}$), where M is the multiset that represents actions participating in the transition and the amount of resources required by each. Elements of M are of the form (ids, ρ), where ids is a set of identifiers related to the action having resource requirements ρ. Each $\widetilde{id} \in ids$ is of the form (id, t_1, t'_1) where id is the syntactic identifiers of the process and t_1 and t'_1 are, respectively, the last deadline and the worst case execution of the corresponding process as specified later in Figures 1.5 and 1.6. The second type of transitions are action transitions $\overset{a}{\rightarrow}$ ($a \in A$) that happen when an action has spent enough time on its required resources such that the remaining time for the action is zero (e.g., rule **(A1)** in Figure 1.3). We decided not to combine resource requirements of different actions and keep them separate in a multiset since they may be provided (according to their respective process identifiers) by different scheduling policies. We use $\overset{\chi}{\rightarrow}$ as an acronym for either of the two transitions. Without making it explicit in the semantics, we assume maximal progress of actions in that the system can only progress in time whenever no action transition is possible. This assumption can be made explicit by an extra condition, namely absence of action transition, in the premises of all timed-transitions. However, we do not reflect this idea in the formal semantics for sake of readability. Predicate $P\surd$ refers to possibility of successful termination of P. The semantics of process specification is the smallest transition relation (union of the time and action transition relations) satisfying the rules of Figures 1.3, 1.4, 1.5 and 1.6.

In Figure 1.3, rules **(I0)** and **(I1)** specify termination and a time transition, respectively. In rule **(I1)**, $\overline{0}$ is an acronym for the function mapping all resources to zero. Rules **(A0)** and **(A1)** specify how an atomic action can spend its time on resources and after that commit its action, respectively. Rules **(S0)**-**(S2)** present the semantics of sequential composition. Rules **(C0)**-**(C1)** provide a semantics for nondeterministic choice. Our choice operator does not have the property of time-determinism (i.e., passage of time cannot determine choices). The reason is that in *PARS*, spending time on resources can reveal the decision taken for the non-deterministic choice. Semantics of the deadline operator is defined by **(D0)**-**(D2)**. Note that there is no rule for the case $\sigma_0(P)$ where process P can only do a time step. Absence of a semantic rule for such a case means that this process deadlocks (i.e., missing a deadline will result in a deadlock). Semantics of the encapsulation operator is defined in rules **(E0)**-**(E2)**. These rules state that the encapsulation operator prevents process P from performing actions in *Act*. This can be quite useful in forcing parallel processes to synchronize on certain actions (see e.g., [2]). Rules **(R0)**-**(R1)** and **(CC0)**-**(CC1)** specify the semantics of recursion and

$$\textbf{(I0)}\frac{}{\epsilon(0)\surd} \qquad \textbf{(I1)}\frac{t' \leq t}{\epsilon(t) \xrightarrow{[(\emptyset,\bar{0})],t'} \epsilon(t-t')}$$

$$\textbf{(A0)}\frac{t' \leq t}{(a,\rho)(t) \xrightarrow{[(\emptyset,\rho)],t'} (a,\rho)(t-t')} \qquad \textbf{(A1)}\frac{}{(a,\rho)(0) \xrightarrow{a} \epsilon(0)}$$

$$\textbf{(S0)}\frac{P \xrightarrow{\chi} P'}{P\,;Q \xrightarrow{\chi} P'\,;Q} \qquad \textbf{(S1)}\frac{P\surd \quad Q \xrightarrow{\chi} Q'}{P\,;Q \xrightarrow{\chi} Q'} \qquad \textbf{(S2)}\frac{P\surd \quad Q\surd}{P\,;Q\surd}$$

$$\textbf{(C0)}\frac{P \xrightarrow{\chi} P'}{\begin{array}{c} P+Q \xrightarrow{\chi} P' \\ Q+P \xrightarrow{\chi} P' \end{array}} \qquad \textbf{(C1)}\frac{P\surd}{\begin{array}{c} P+Q\surd \\ Q+P\surd \end{array}}$$

$$\textbf{(D0)}\frac{P \xrightarrow{M,t} P' \quad t \leq t_0}{\sigma_{t_0}(P) \xrightarrow{M,t} \sigma_{t_0-t}(P')} \qquad \textbf{(D1)}\frac{P \xrightarrow{a} P'}{\sigma_{t_0}(P) \xrightarrow{a} \sigma_{t_0}(P')} \qquad \textbf{(D2)}\frac{P\surd}{\sigma_{t_0}(P)\surd}$$

$$\textbf{(E0)}\frac{P \xrightarrow{a} P' \quad a \notin Act}{\partial_{Act}(P) \xrightarrow{a} \partial_{Act}(P')} \qquad \textbf{(E1)}\frac{P \xrightarrow{M,t} P'}{\partial_{Act}(P) \xrightarrow{M,t} \partial_{Act}(P')} \qquad \textbf{(E2)}\frac{P\surd}{\partial_{Act}(P)\surd}$$

$$\textbf{(R0)}\frac{P(\mu X.P(X)) \xrightarrow{\chi} P'}{\mu X.P(X) \xrightarrow{\chi} P'} \qquad \textbf{(R1)}\frac{P(\mu X.P(X))\surd}{\mu X.P(X)\surd}$$

$$\textbf{(CC0)}\frac{y_t \in T \quad P(y_t) \xrightarrow{\chi} P'}{\int_{x\in T} P(x) \xrightarrow{\chi} P'} \qquad \textbf{(CC1)}\frac{y_t \in T \quad P(y_t)\surd}{\int_{x\in T} P(x)\surd}$$

$$\textbf{(Id0)}\frac{P \dagger_{t_1} \quad \mho_{t'_1}(P) \quad P \xrightarrow{M,t} P'}{id : P \xrightarrow{M\oplus(id,t_1,t'_1),t} id : P'} \qquad \textbf{(Id1)}\frac{P \xrightarrow{a} P'}{id : P \xrightarrow{a} id : P'} \qquad \textbf{(Id2)}\frac{P\surd}{id : P\surd}$$

$$a \in A,\ t,t' \in \mathbb{R}^{>0}, t_0 \in \mathbb{R}^{\geq 0}, t_1, t'_1 \in \mathbb{R}^{\geq 0} \cup \{\infty\}, \chi \in (\mathbb{M} \times \mathbb{R}^{>0}) \cup A$$

FIGURE 1.3: Semantics of *PARS*, Part 1(a): Process Specification, Sequential Subset

continuous choice, respectively. A recursive process can perform a transition, if the unfolded processes can do so. Note that in the continuous choice, the choice is made as soon as the bound term makes a transition. Rules **(Id0)-(Id2)** specify the semantics of id by adding $\widetilde{id}$ to the multiset in the transition, where $\widetilde{id}$ is the tuple (id, t_1, t'_1) consisting of the syntactic id, deadline, and worst case execution time of the process (see Figures 1.5 and 1.6 for the definitions of deadline and worst case execution time, respectively). The operator $\oplus : \mathbb{M} \times Type(\widetilde{id}) \to \mathbb{M}$, used in the semantic rule **(Id0)**, intuitively merges the new identifier $\widetilde{id}$ with the existing identifiers (associated to a particular resource requirement information) and is formally defined as follows.

$$\begin{aligned}
&\emptyset \oplus \widetilde{id} \doteq \emptyset \\
&([(ids, \rho)] + M) \oplus \widetilde{id} \doteq [(ids \cup \{\widetilde{id}\}, \rho)] + (M \oplus \widetilde{id})
\end{aligned}$$

In Figure 1.4, abstract parallel composition is specified by rules **(P0)-(P4)** and strict parallel composition is defined by rules **(SP0)-(SP3)**. In rule **(P0)**, $t \gg Q$ is an auxiliary operator (called deadline shift) that is used to specify that Q is getting t units of time closer to its deadline. The semantics of this operator is formally defined by means of the rules **(DS0)-(DS2)**. The semantics for deadline shift takes into account only the deadline of active actions. Active actions are actions that can introduce a task at the moment or already have introduced one. This is in line with the intuition that in scheduling theory only *ready* actions can take part in scheduling and other actions have to wait for their causal predecessors to commit. Function $\gamma(a, b)$ in rules **(P3)** and **(SP2)** specifies the result of a synchronized communication between a and b.

The semantics of abstract parallel composition deviates from standard semantics of parallelism in timed process algebras in that it allows for asynchronous passage of time by the two parties (rule **(P0)**). This reflects the fact that depending on availability of resources and due to scheduling, concurrent execution of tasks can be preempted and serialized at any moment of time.

The latest deadline predicate $P\dagger_t$ is defined by the deduction rules in Figure 1.5. The latest deadline is supposed to indicate the longest period, during which the process requirements should be met or the process will certainly miss a deadline (specified by some σ_t operator) and thus, will turn into a deadlocking situation. Deduction rules defining the concept of the latest deadline for δ, $\epsilon(t)$ and $(a, \rho)(t)$ are self-explanatory; they are all defined to be infinity because they do not contain any deadline operator. In case of sequential composition, we make a case distinction between the case where the first argument does not terminate and where it does. In the former case, the latest deadline of the process is due to its first argument. In the latter case, both arguments may have deadlines and both may continue their execution; thus, the argument which has a later deadline determines the latest possible deadline. The deadline of a process of the form $P + Q$ is determined by the later deadline between that of P and of Q. The deadline of $\sigma_t(P)$ is determined

$$\textbf{(P0)}\ \frac{P \xrightarrow{M,t} P'}{\begin{array}{c} P \parallel Q \xrightarrow{M,t} P' \parallel t \gg Q \\ Q \parallel P \xrightarrow{M,t} t \gg Q \parallel P' \end{array}} \qquad \textbf{(P1)}\ \frac{P \xrightarrow{a} P'}{\begin{array}{c} P \parallel Q \xrightarrow{a} P' \parallel Q \\ Q \parallel P \xrightarrow{a} Q \parallel P' \end{array}}$$

$$\textbf{(P2)}\ \frac{P \xrightarrow{M,t} P' \quad Q \xrightarrow{M',t} Q'}{P \parallel Q \xrightarrow{M+M',t} P' \parallel Q'} \qquad \textbf{(P3)}\ \frac{P \xrightarrow{a} P' \quad Q \xrightarrow{b} Q' \quad \gamma(a,b) = c}{P \parallel Q \xrightarrow{c} P' \parallel Q'}$$

$$\textbf{(SP0)}\ \frac{P \xrightarrow{M,t} P' \quad Q \xrightarrow{M',t} Q'}{P \mid\mid\mid Q \xrightarrow{M+M',t} P' \mid\mid\mid Q'} \qquad \textbf{(SP1)}\ \frac{P \xrightarrow{a} P'}{\begin{array}{c} P \mid\mid\mid Q \xrightarrow{a} P' \mid\mid\mid Q \\ Q \mid\mid\mid P \xrightarrow{a} Q \mid\mid\mid P' \end{array}}$$

$$\textbf{(SP2)}\ \frac{P \xrightarrow{a} P' \quad Q \xrightarrow{b} Q' \quad \gamma(a,b) = c}{P \mid\mid\mid Q \xrightarrow{c} P' \mid\mid\mid Q'}$$

$$\textbf{(DS0)}\ \frac{P \xrightarrow{a} P' \quad P \dagger_{t_0} \quad t \leq t_0}{t \gg P \xrightarrow{a} P'} \qquad \textbf{(DS1)}\ \frac{P \xrightarrow{M,t'} P' \quad P \dagger_{t_0} \quad t' \leq t_0 - t}{t \gg P \xrightarrow{M,t'} t \gg P'}$$

$$\textbf{(DS2)}\ \frac{P\surd}{t \gg P\surd}$$

$$\textbf{(P4)}\ \frac{P\surd \quad Q\surd}{P \parallel Q\surd} \qquad \textbf{(SP3)}\ \frac{P\surd \quad Q\surd}{P \mid\mid\mid Q\surd}$$

$a \in A,\ t, t' \in I\!R^{>0}, t_0 \in I\!R^{\geq 0}, \chi \in (I\!M \times I\!R^{>0}) \cup A$

FIGURE 1.4: Semantics of *PARS*, Part 1(b): Process Specification, Parallel operators

by the minimum of t and the latest deadline of P (since P can wait no more than both t and its latest deadline). Process $t' \gg P$ has already spent t' of its time, so its latest deadline is shifted by t'. Missing a deadline in either of the parallel components results in a missed deadline in the composite process; thus, deadline of both $P \mid\mid Q$ and $P \mid\mid\mid Q$ is defined as the minimum of the deadlines of their arguments. The latest deadline of a recursive process is determined by unfolding its definition. For a continuous choice, the deadline of the process is defined as the supremum, i.e., the least upper bound of the set of deadlines of the alternative choices. Note that the maximum of such deadlines may not exist, e.g., if the deadlines form an open interval. Neither encapsulation, nor adding an identifer, influence our estimation of the latest deadline.

$$\frac{}{\delta\dagger_\infty} \quad \frac{}{\epsilon(t)\dagger_\infty} \quad \frac{}{(a,\rho)(t)\dagger_\infty}$$

$$\frac{\neg P\surd \quad P\dagger_t}{P \,;\, Q\dagger_t} \quad \frac{P\surd \quad P\dagger_t \quad Q\dagger_{t'}}{P \,;\, Q\dagger_{\max(t,t')}} \quad \frac{P\dagger_t \quad Q\dagger_{t'}}{P + Q\dagger_{\max(t,t')}}$$

$$\frac{P\dagger_t}{\sigma_{t'}(P)\dagger_{\min(t,t')}} \quad \frac{P\dagger_t}{t' \gg P\dagger_{t-t'}} \quad \frac{P\dagger_t \quad Q\dagger_{t'}}{P \mid\mid Q\dagger_{\min(t,t')}} \quad \frac{P\dagger_t \quad Q\dagger_{t'}}{P \mid\mid\mid Q\dagger_{\min(t,t')}}$$

$$\frac{P(\mu X.P(X))\dagger_t}{\mu X.P(X)\dagger_t} \quad \frac{}{\int_{x\in T} P(x)\dagger_{\mathrm{Sup}\{t_y \mid P(y)\dagger_{t_y} \wedge y \in T\}}} \quad \frac{P\dagger_t}{\partial_{Act}(P)\dagger_t} \quad \frac{P\dagger_t}{id : P\dagger_t}$$

FIGURE 1.5: Semantics of *PARS*, Part 1(c): Process Specification, Deadlines $(t, t' \in I\!R^{\geq 0} \cup \{\infty\})$

The Worst Case Execution Time (WCET) predicate $\mho_t(P)$ is defined by the deduction rules in Figure 1.6. As the names suggest the intuition behind worst case execution time is to be an upper bound estimation of the longest computation time of a process. Most of the deduction rules are self-explanatory; WCET is generally defined by taking the minimum of the deadline of the process and its maximal execution time. The following lemma shows that WCET of each process is less than or equal to its *latest* deadline.

LEMMA 1.1

For each process P and $t \in I\!R^{\geq 0} \cup \{\infty\}$, if $\mho_t(P)$ holds, then there exists $t' \in I\!R^{\geq 0} \cup \{\infty\}$ such that $P\dagger_{t'}$ and $t \leq t'$.

PROOF By a case distinction on the last deduction rule in the structure of the proof for $\mho_t(P)$. The lemma holds vacuously for δ, $\epsilon(t)$ and $(a, \rho)(t)$ since for all of them, the latest deadline is ∞. For other deduction rules, there exists a premise of the form $\dagger_{t'} P$ and t is of the form $\min(t', e)$, for some expression e. Thus, it holds that $t \leq t'$. ∎

$$\frac{}{\mho_0(\delta)} \qquad \frac{}{\mho_t(\epsilon(t))} \qquad \frac{}{\mho_t((a,\rho)(t))} \qquad \frac{\mho_t(P) \quad \mho_{t'}(Q) \quad (P\,;Q)\dagger_{t''}}{\mho_{\min(t'',t+t')}(P\,;Q)}$$

$$\frac{\mho_t(P) \quad \mho_{t'}(Q) \quad (P+Q)\dagger_{t''}}{\mho_{\min(t'',\max(t,t'))}(P+Q)} \qquad \frac{\mho_t(P) \quad \sigma_{t'}(P)\dagger_{t''}}{\mho_{\min(t'',t)}(\sigma_{t'}(P))} \qquad \frac{\mho_t(P) \quad t' \gg P\dagger_{t''}}{\mho_{\min(t'',t)}(t' \gg P)}$$

$$\frac{\mho_t(P) \quad \mho_{t'}(Q) \quad (P \mid\mid Q)\dagger_{t''}}{\mho_{\min(t'',t+t')}(P \mid\mid Q)} \qquad \frac{\mho_t(P) \quad \mho_{t'}(Q) \quad (P \mid\mid\mid Q)\dagger_{t''}}{\mho_{\min(t'',t+t')}(P \mid\mid\mid Q)}$$

$$\frac{\mho_t(P(\mu X.P(X))) \quad (\mu X.P(X))\dagger_{t'}}{\mho_{\min(t',t)}(\mu X.P(X))} \qquad \frac{\{\mho_{t_y}(P(y)) \mid y \in T\} \quad (\int_{x\in T} P(x))\dagger_{t'}}{\mho_{\min(t',\mathrm{Sup}_{y\in T}(t_y))}(\int_{x\in T} P(x))}$$

$$\frac{\mho_t(P)}{\mho_t(\partial_{Act}(P))} \qquad \frac{\mho_t(P)}{\mho_t(id : P)}$$

FIGURE 1.6: Semantics of *PARS*, Part 1(d): Process Specification, Worst Case Execution Times ($t, t', t_y \in I\!R^{\geq 0} \cup \{\infty\}$)

The deadline and WCET are just meant to be estimations of process measures, and any other well-defined performance measure on processes can replace or extend the semantics.

In order to compare processes, e.g., to compare a specification with its implementation, it is customary to define a notion of equivalence or pre-order among processes. In this chapter, we adapt the notion of strong bisimilarity to our setting which provides an appropriate theoretical starting point. Other weaker notions of equality and pre-order can be analogously adopted to our settings.

DEFINITION 1.1 *(Strong Bisimulation) A symmetric relation R on process terms is a strong bisimulation for resource requiring processes if and only if for all pairs $(P, Q) \in R$:*

1. $P \xrightarrow{\chi} P' \Rightarrow \exists_{Q'} Q \xrightarrow{\chi} Q' \wedge (P', Q') \in R$

2. $P\surd \Rightarrow Q\surd$

3. $P\dagger_t \Rightarrow Q\dagger_t$

4. $\mho_t(P) \Rightarrow \mho_t(Q)$

Two processes P and Q are called strongly bisimilar, denoted by $P \underline{\leftrightarrow} Q$ if and only if there exists a strong bisimulation relation R such that $(P, Q) \in R$.

THEOREM 1.1 Congruence of strong bisimilarity for the process language

Strong bisimilarity, as defined in Definition 1.1, is a congruence with respect to all operators in our process specification language.

PROOF The deduction rules are in the PANTH format of [23]. Furthermore, by counting the number of symbols in each predicate / left-hand-side of each transition, we can define a stratification measure which does not increase from the conclusion to the positive premises and decreases from conclusion to negative premises. Thus, the set of SOS rules for our process language specification are complete, i.e., they univocally define a transition relation. Hence, it follows from the meta-theorem of [23] that our notion of strong bisimilarity is a congruence [23]. $\square$

Below we present some properties of the operators introduced in this section. The notation $FV(P)$ denotes the free variables of process P. The proofs are tedious but straightforward and therefore omitted.

THEOREM 1.2 Properties of processes
For arbitrary processes P, Q, and R

$$\begin{array}{lll}
P + P & \underline{\leftrightarrow} & P \\
P + Q & \underline{\leftrightarrow} & Q + P \\
(P + Q) + R & \underline{\leftrightarrow} & P + (Q + R) \\
P + \sigma_0(\delta) & \underline{\leftrightarrow} & P \\
\\
(P + Q) \mathbin{;} R & \underline{\leftrightarrow} & (P \mathbin{;} R) + (Q \mathbin{;} R) \\
\\
\sigma_t(\sigma_{t'}(P)) & \underline{\leftrightarrow} & \sigma_{\min(t,t')}(P) \\
\sigma_t(P + Q) & \underline{\leftrightarrow} & \sigma_t(P) + \sigma_t(Q) \\
\sigma_t(P \mathbin{;} Q) & \underline{\leftrightarrow} & \sigma_t(\sigma_t(P) \mathbin{;} \sigma_t(Q)) \\
\sigma_0(\epsilon(0)) \mathbin{;} P & \underline{\leftrightarrow} & P \\
\\
\mu X.P(X) & \underline{\leftrightarrow} & P(\mu X.P(X)) \\
\\
\int_{x\in\varnothing} P(x) & \underline{\leftrightarrow} & \sigma_0(\delta) \\
\int_{x\in T} P(x) & \underline{\leftrightarrow} & P(t) + \int_{x\in T} P(x) \qquad (t \in T) \\
\int_{x\in T} P(x) & \underline{\leftrightarrow} & P(x) \qquad (x \notin FV(P(x))) \\
\int_{x\in T} (P(x) + Q(x)) & \underline{\leftrightarrow} & \int_{x\in T} P(x) + \int_{x\in T} Q(x) \\
\\
P \mathbin{||} Q & \underline{\leftrightarrow} & Q \mathbin{||} P \\
(P \mathbin{||} Q) \mathbin{||} R & \underline{\leftrightarrow} & P \mathbin{||} (Q \mathbin{||} R) \\
P \mathbin{||} \epsilon(0) & \underline{\leftrightarrow} & P \\
\\
P \mathbin{|||} Q & \underline{\leftrightarrow} & Q \mathbin{|||} P \\
(P \mathbin{|||} Q) \mathbin{|||} R & \underline{\leftrightarrow} & P \mathbin{|||} (Q \mathbin{|||} R) \\
\\
t \gg t' \gg P & \underline{\leftrightarrow} & (t + t') \gg P
\end{array}$$

Due to the fact that the definitions of the deadline predicate and the worst case execution time are just approximations many of the properties that are quite standard for the operators in standard process algebra are not valid in this setting. An example is the associativity of sequential composition.

1.3 Scheduler Specification

The syntax of scheduler specification (Sc) is similar to process specification and is specified in Figure 1.7. Basic actions of schedulers are predicates ($Pred$) mentioning appropriate processes to be provided with resources and the amount of resources ($\overline{\rho} : R \to \mathbb{R}^{\leq 0}$) provided during the specified time.

Note that resource provisions are denoted by functions from resources to non-positive integers so that they can cancel the resource requirements when confronted with them in the next section. The predicate can refer to the syntactic identifiers, deadline or worst-case execution time for processes. In the syntax of *Pred*, Id is a variable from set V_i (with a distinguished member $\underline{Id}$ and typical members $Id_0, Id_1, etc.$). $\underline{Id}$ and Id_i refer to the semantic identifier of the particular process receiving the specified resource and its environment processes, respectively. Following the structure of a semantic identifier, Id is a tuple containing syntactic identifier ($Id.id$), deadline ($Id.Dl$) and execution time ($Id.WCET$). As in the process language, the language for predicates can be extended to other metrics of processes. Since we aimed at separating the process specification aspects from scheduling aspects, we did not include constructs such as resource consuming actions and identifiers in our schedulers language.

A couple of new operators are added to the ones in the process specification language. The preemptive precedence operator $\triangleright$ gives precedence to the right-hand-side term (with the possibility of the right-hand side taking over the execution of left-hand side at any point) and continuous preemptive precedence $\rangle\!\rangle_{t \in T}$ which gives precedence to the choice of least possible t. Note that this need not be the least element of T (which may not even exist) but rather it is the predicate with the least possible t that can match a resource requirement. (If no such least possible t exists the result of application of the scheduler to the process should be a deadlock.) The following examples illustrate the use of these operators.

Example 1.6 Precedence operator

Consider the process specification of Example 1.5, where the system consists of two types of processes: User processes and system processes. Suppose that system processes always have a priority over user processes in using a single CPU. The following scheduler specifies a general scheduling policy that observes the above priorities:

$$\begin{aligned} PrSch \doteq\ & (\underline{Id}.id = User, CPU \mapsto -1)([0,\infty)) \triangleright \\ & (\underline{Id}.id = System, CPU \mapsto -1)([0,\infty)) \end{aligned}$$

□

Example 1.7 Continuous precedence operator

Assume that our scheduling strategy assigns the only available CPU to the process with shortest deadline for the worst case execution time of the process. The following specification provides us with such a scheduler:

$$CntPrSch \doteq \rangle\!\rangle_{t \in I\!R^{\geq 0}} (\underline{Id}.Dl = t \wedge \underline{Id}.WCET = t', \{CPU \mapsto -1\})(t')$$

In the above scheduler the continuous preemptive precedence operator $\rangle_{t \in \mathbb{R}^{\geq 0}}$ in combination with $\underline{Id}.Dl = t$ enforces the process with earliest deadline to be chosen. $\square$

The non-preemptive counter-parts of the above operators $\rhd^{\mathrm{n}}$ and $\rangle^{\mathrm{n}}_{t \in T}$ have the same intuition but they do not allow taking over of one side if the other side has already decided to start. The timing variables bound by continuous choice or generalized precedence operators can be used in predicates (as timing constants) as well as in process timings. For simplicity, we only allow comparison to time points and single identifier in the predicate part and we only introduce continuous precedence operators with precedence for lower time points. Enriching the first-order language of predicates and introducing continuous precedence operators for higher time values are possible extensions of the language.

$$
\begin{aligned}
Sc \;&::= \delta \mid s(t) \mid Sc\,;\,Sc \mid Sc \mid\mid Sc \mid Sc \mid\mid\mid Sc \mid Sc + Sc \mid \\
&\qquad \textstyle\int_{x \in T} P(x) \mid Sc \rhd Sc \mid Sc \rhd^{\mathrm{n}} Sc \mid \rangle_{t \in T} Sc(t) \mid \rangle^{\mathrm{n}}_{t \in T} Sc(t) \mid \mu X.Sc(X) \\
Pred &::= Id.id\ Op\ Num \mid Id.Dl\ Op\ time \mid Id.WCET\ Op\ time \mid \\
&\qquad Pred \wedge Pred \mid Pred \vee Pred \mid true \\
Op \;&::= < \mid = \mid > \\
&\qquad s \in Pred \times \overline{\rho}, t \in \mathbb{R}^{\geq 0}_{\infty},\ x \in V_t,\ time \in V_t \cup \mathbb{R}^{\geq 0},\ Id \in V_i,\ Num \in \mathbb{N}
\end{aligned}
$$

FIGURE 1.7: Syntax of *PARS*, Part 2: Scheduler Specification

The semantic rules for our scheduler specification language are given in Figure 1.8. We omitted the semantic rules for operators in common with process specification since they are very similar to those specified in process specification semantics. Rules for the operators common to the process specification language given in Figures 1.3 and 1.4 should be copied here with the following two provisos. Firstly, abstract parallel composition in schedulers has a simpler semantics, since schedulers do not have a deadline operator. Namely, rule **(P'0)** should be replaced with the following rule.

$$\textbf{(P'0)}\,\frac{P \overset{M,t}{\rightarrow} P'}{\begin{array}{c} P \mid\mid Q \overset{M,t}{\rightarrow} P' \mid\mid Q \\ Q \mid\mid P \overset{M,t}{\rightarrow} Q \mid\mid P' \end{array}}$$

Note that, due to this change, the auxiliary operator $t \gg P$ does not appear in the semantics of schedulers. Secondly, for simplicity, we did not include action prefixing in the syntax of the language for schedulers. Consequently, rules concerning action transitions need not be copied to the semantics of schedulers. Actions transitions can indeed be useful in modelling interactions within schedulers and among schedulers and processes but we omitted both for the sake of simpler presentation and keeping the orthogonality between the process and the scheduler language. Note that breaking the orthogonality and merging the two languages opens up the possibility of modeling more complex schedulers which need to communicate with each other and with processes or need to receive a resource and then distribute it afterwards, e.g., servers in a deferable server scheme [7].

The transition relation in the semantics is of the form $\overset{M,t}{\rightarrow}$, where M is a multiset containing predicates about processes that can receive a certain amount of resources during time t. Elements of multiset M are of the form $(pred, npred, \overline{\rho})$ where $pred$ is the positive predicate that the process receiving resources should satisfy, $npred$ is the negative predicate that the process should falsify and $\overline{\rho}$ is the function representing the amount of different resources offered to such a process. In this level, we assume no information about the resource requiring process that the scheduler is to be confronted with. Thus, the resource grant predicates specify the criteria that processes receiving resources should satisfy (being able to match the predicate) and the criteria they should falsify (not being able to perform higher precedence transitions). For example, the positive predicate $\underline{Id}.Id = User$ specifies that the process receiving the resource should have $User$ as its syntactic identifier and the predicate $\underline{Id}.Dl = t$ specifies that the latest deadline of the receiving process should be t. Once the same predicates appear in the negative side, they mean that no process with identifier $User$ or no process with deadline t is currently able to receive the resource.

Rules **(ScA0)** and **(ScA1)** specify semantics of atomic scheduler actions. Rules **(Pr0)-(Pr2)** specify the semantics for precedence operator. In these rules, $M \vee_{neg} pred$ stands for disjunction of negative predicates in all elements of M with predicate $pred$:

$$\begin{array}{ll} [(pred_0, npred_0, \overline{\rho})] \vee_{neg} pred & \doteq [(pred_0, npred_0 \vee pred, \overline{\rho})] \\ (M + [(pred_0, npred_0, \overline{\rho})]) \vee_{neg} pred & \doteq (M \vee_{neg} pred) + \\ & \quad [(pred_0, npred_0 \vee pred, \overline{\rho})] \end{array}$$

$$\textbf{(ScA0)}\,\frac{}{(p,\overline{\rho})(0)\surd} \qquad \textbf{(ScA1)}\,\frac{t' \leq t}{(p,\overline{\rho})(t) \overset{[(p,\mathit{false},\overline{\rho})],t'}{\longrightarrow} (p,\overline{\rho})(t-t')}$$

$$\textbf{(Pr0)}\,\frac{P \overset{M,t}{\rightarrow} P'}{P \rhd Q \overset{M \vee_{neg} enabled(Q),t}{\longrightarrow} P' \rhd Q}$$

$$\textbf{(Pr1)}\,\frac{Q \overset{M,t}{\rightarrow} Q'}{P \rhd Q \overset{M,t}{\rightarrow} P \rhd Q'} \qquad \textbf{(Pr2)}\,\frac{P\surd \quad Q\surd}{P \rhd Q\surd}$$

$$\textbf{(NPr0)}\,\frac{P \overset{M,t}{\rightarrow} P'}{P \rhd^{\mathrm{n}} Q \overset{M \vee_{neg} enabled(Q),t}{\longrightarrow} P'}$$

$$\textbf{(NPr1)}\,\frac{Q \overset{M,t}{\rightarrow} Q'}{P \rhd^{\mathrm{n}} Q \overset{M,t}{\rightarrow} Q'} \qquad \textbf{(NPr2)}\,\frac{P\surd \quad Q\surd}{P \rhd^{\mathrm{n}} Q\surd}$$

$$\textbf{(CPr0)}\,\frac{P(t'') \overset{M,t'}{\rightarrow} P'(t'') \quad t'' \in T}{\rangle_{t \in T} P(t) \overset{M \vee_{neg} \exists_t t \in \lfloor T \rfloor_{t''} \wedge enabled(P(t)),t'}{\longrightarrow} \rangle_{t \in T} P'(t)}$$

$$\textbf{(CPr1)}\,\frac{t' \in T \quad P(t')\surd}{\rangle_{t \in T} P(t)\surd}$$

$$\textbf{(NCPr0)}\,\frac{P(t') \overset{M,t}{\rightarrow} P' \quad t' \in T}{\rangle^{\mathrm{n}}_{t \in T} P \overset{M \vee_{neg} \exists_t t \in \lfloor T \rfloor_{t'} \wedge enabled(P(t)),t}{\longrightarrow} P'} \qquad \textbf{(NCPr1)}\,\frac{t' \in T \quad P(t')\surd}{\rangle^{\mathrm{n}}_{t \in T} P(t)\surd}$$

FIGURE 1.8: Semantics of *PARS*, Part 2: Scheduler Specification

In the same semantic rules, enabled-ness of a process term is used as a negative predicate to assure that a lower priority process cannot make a transition when a higher priority one is able to do so. This notion is formally defined as follows:

$$\begin{array}{l}
\mathit{enabled}(\delta) \doteq \mathit{false} \\
\mathit{enabled}(((\mathit{pred}, \overline{\rho}), t)) \doteq \mathit{pred} \\
\mathit{enabled}(P \,;\, Q) \doteq \begin{cases} \mathit{enabled}(P) \vee \mathit{enabled}(Q) & \mathit{if}\ P\surd \wedge P \rightarrow \\ \mathit{enabled}(P) & \mathit{if}\ \neg(P\surd) \\ \mathit{enabled}(Q) & \mathit{if}\ P\surd \wedge P \nrightarrow \end{cases} \\
\mathit{enabled}(P \,||\, Q) \doteq \mathit{enabled}(P \,|||\, Q) \doteq \mathit{enabled}(P + Q) \doteq \\
\mathit{enabled}(P \rhd Q) \doteq \mathit{enabled}(P \rhd^{\mathrm{n}} Q) \doteq \mathit{enabled}(P) \vee \mathit{enabled}(Q) \\
\mathit{enabled}(\int_{x \in T} P(x)) \doteq \\
\mathit{enabled}(\rangle_{x \in T} P(x)) \doteq \mathit{enabled}(\rangle^{\mathrm{n}}_{x \in T} P(x)) \doteq \exists_{x \in T}\, \mathit{enabled}(P(x)) \\
\mathit{enabled}(\mu X.P(X)) \doteq \mathit{enabled}(P(\mu X.P(X)))
\end{array}$$

In the above definition $P \rightarrow$ stands for the possibility of performing a transition $P \rightarrow P'$ for some P' and $P \nrightarrow$ is the negation of it. Note that using $P \nrightarrow$ and $\neg(P\surd)$ in this definition introduces negative premises to our semantics indirectly. But this is harmless to well-definedness of our semantics since a standard stratification can be found for it. To illustrate the semantics of precedence operator, we give the following example:

Example 1.8

Consider the scheduler specification of Example 1.6. This specification generates a transition system that allows an arbitrary time transition with positive predicate $\underline{\mathit{Id}}.\mathit{id} = \mathit{System}$. However, according to the rule **(Pr0)**, for transitions with positive predicate $\underline{\mathit{Id}}.\mathit{id} = \mathit{User}$, the predicate of $t \in [0, \infty) \wedge \underline{\mathit{Id}}.\mathit{id} = \mathit{System}$ is added as a negative predicate, as well. Intuitively, this should mean that CPU is provided to a user process if no system process is able to take that transition. Of course, part of this intuition remains to be formalized by the semantics of applying schedulers to processes. $\square$

Rules **(NPr0)-(NPr2)** specify the semantics for the non-preemptive precedence operator. The only difference between these rules and their preemptive counterparts, i.e., **(Pr0)-(Pr2)** is that after making a transition by one of the two arguments, the other argument disappears and thus the argument making the transition can no more be preempted. Rules **(CPr0)-(CPr1)** and **(NCPr0)-(NCPr1)** present the semantic rules for preemptive and non-preemptive continuous precedence operators, respectively.

In rules **(CPr0)** and **(NCPr0)** operator $\lfloor T \rfloor_t$ is defined as follows:

$$\lfloor T \rfloor_t \doteq \{t' | t' \in T \wedge t' < t\}$$

Example 1.9 Specifying scheduling strategies
We specify a few generic scheduling strategies for a single processor to show the usage of the scheduler specification language:

- Non-preemptive Round-Robin scheduling: Consider a scheduling strategy where a single processor is going to be granted to processes non-preemptively in the increasing order of their identifiers (from 0 to n). The following scheduler specifies this scheduling strategy:

$$\begin{aligned} Sch_{NP-RR} \doteq\ & (\underline{Id} = 0, \{CPU \mapsto -1\})[0,\infty)\ ;\\ & (\underline{Id} = 1, \{CPU \mapsto -1\})[0,\infty)\ ;\\ & \ldots\ ;\ (\underline{Id} = n, \{CPU \mapsto -1\})[0,\infty) \end{aligned}$$

- Rate Monotonic (RM) scheduling: Consider the following process specification $SysProc$ of several periodic processes composed by the abstract parallel composition operator:

$$\begin{aligned} & SysProc \doteq ||_{i=0}^{n}\ P_i\\ & P_i \doteq \mu X.(2i+1) :(Q)\ |||\ ((2i) : (\epsilon(t'))\ ;\ X) \end{aligned}$$

In the above specification, even identifiers refer to the period of the tasks and odd identifiers refer to the tasks themselves. The following scheduler, specifies the preemptive rate monotonic strategy, where processes with shortest period (higher rate), have a priority in receiving CPU time:

$$\begin{aligned} RMSch(k,t) \doteq\ & (\underline{Id}.id = 2k+1 \wedge Id_0 = 2k \wedge Id_0.WCET = t,\\ & \{CPU \mapsto -1\})([0,\infty))\\ RMSch \doteq\ & \rangle_{t \in I\!R^{\geq 0}} RMSch(0,t) + \ldots + RMSch(n,t) \end{aligned}$$

Identifier $\underline{Id}$ refers to the process receiving the resource (i.e., a task defined by the process Q). Identifier Id_0 is an arbitrary identifier referring to the corresponding period of the task. Hence, $Id_0.WCET$ refers to the period of the task denoted by $\underline{Id}$.

- Earliest Deadline First (EDF) scheduling: Consider the previous pattern of processes, then the following expression specifies preemptive earliest deadline first scheduling:

$$\begin{aligned} EDFSch(k,t) \doteq\ & (\underline{Id}.id = 2k+1 \wedge Id_0 = 2k \wedge Id_0.Dl = t,\\ & \{CPU \mapsto -1\})([0,\infty))\\ EDFSch \doteq\ & \rangle_{t \in I\!R^{\geq 0}} EDFSch(0,t) + \ldots + EDFSch(n,t) \end{aligned}$$

$\Box$

Common to the formalism for process specification, one can use the notion of strong bisimilarity to relate schedulers. The definition of strong bisimulation

/ bisimilarity for schedulers remains the same as Definition 1.1; only items 3 and 4 in this definition should be dropped since schedulers do not have a notion of deadline and WCET.

THEOREM 1.3 Congruence of strong bisimilarity for the scheduler language

Strong bisimilarity, is a congruence with respect to all operators in our scheduler specification language.

PROOF Again, all our deduction rules for schedulers (including the simplified rule **(P'0)** and those borrowed from Figures 1.3 and 1.4) are in the PANTH format of [23]. The rules specified for the scheduler are also stratified by the same stratification measure used in the proof of Theorem 1.1. Thus, the set of deduction rules is complete and our notion of strong bisimilarity is a congruence following the meta-theorem of [23]. □

Since many of the operators for specifying schedulers are similar to those used for describing processes, they enjoy similar properties. Since schedulers do not have deadline and worst case execution time predicates associated with them, there are additional properties.

THEOREM 1.4 Properties of schedulers
For arbitrary schedulers P, Q, and R

$$
\begin{array}{lcll}
P + P & \underline{\leftrightarrow} & P & \\
P + Q & \underline{\leftrightarrow} & Q + P & \\
(P + Q) + R & \underline{\leftrightarrow} & P + (Q + R) & \\
P + \delta & \underline{\leftrightarrow} & P & \\
 & & & \\
\delta \;;\; P & \underline{\leftrightarrow} & \delta & \\
\epsilon(0) \;;\; P & \underline{\leftrightarrow} & P & \\
(P + Q) \;;\; R & \underline{\leftrightarrow} & (P \;;\; R) + (Q \;;\; R) & \\
(P \;;\; Q) \;;\; R & \underline{\leftrightarrow} & P \;;\; (Q \;;\; R) & \\
 & & & \\
\mu X.Sc(X) & \underline{\leftrightarrow} & Sc(\mu X.Sc(X)) & \\
 & & & \\
\int_{x \in \varnothing} P(x) & \underline{\leftrightarrow} & \delta & \\
\int_{x \in T} P(x) & \underline{\leftrightarrow} & P(t) + \int_{x \in T} P(x) & (t \in T) \\
\int_{x \in T} P(x) & \underline{\leftrightarrow} & P(x) & (x \notin FV(P(x))) \\
\int_{x \in T} (P(x) + Q(x)) & \underline{\leftrightarrow} & \int_{x \in T} P(x) + \int_{x \in T} Q(x) & \\
\left(\int_{x \in T} P(x)\right) \;;\; Q & \underline{\leftrightarrow} & \int_{x \in T} (P(x) \;;\; Q) & (x \notin FV(Q)) \\
 & & & \\
P \mid\mid Q & \underline{\leftrightarrow} & Q \mid\mid P & \\
(P \mid\mid Q) \mid\mid R & \underline{\leftrightarrow} & P \mid\mid (Q \mid\mid R) & \\
P \mid\mid \epsilon(0) & \underline{\leftrightarrow} & P & \\
P \mid\mid \delta & \underline{\leftrightarrow} & P \;;\; \delta & \\
 & & & \\
P \mid\mid\mid Q & \underline{\leftrightarrow} & Q \mid\mid\mid P & \\
(P \mid\mid\mid Q) \mid\mid\mid R & \underline{\leftrightarrow} & P \mid\mid\mid (Q \mid\mid\mid R) &
\end{array}
$$

1.4 Applying Schedulers to Processes

Scheduled systems are processes resulting from applying a number of schedulers to processes. The syntax of scheduled systems is presented in Figure 1.9. In this syntax, P and Sc refer to the syntactic class of processes and schedulers presented in the previous sections, respectively. Term $\langle\!\langle Sys \rangle\!\rangle_{Sc}$ denotes applying scheduler Sc to the system Sys and $\partial_R(Sys)$ is used to close a system specification and prevent it from acquiring resources in R.

The semantics of new operators for scheduled systems is defined in Figure 1.10. In this semantics, the transition relation is the same as the transition relation in the process specification semantics of Section 1.2. Since a process is a system by definition, all semantic rules of that section carry over to the semantics of systems. Moreover, as in schedule specification phase, we re-use

$$Sys ::= P \mid \langle\!\langle Sys \rangle\!\rangle_{Sc} \mid Sys \,;\, Sys \mid Sys \,||\, Sys \mid Sys \,|||\, Sys \mid Sys + Sys \mid$$
$$\partial_R(Sys) \mid \sigma_t(Sys) \mid \mu X.Sys(X) \mid id : Sys \mid t \gg Sys$$

FIGURE 1.9: Syntax of *PARS*, Part 3: Syntax of Scheduled Systems

the rules of Section 1.2 in a more general sense in order to cover the semantics of sequential, abstract and strict parallel composition, non-deterministic choice of systems, and deadline shift operator.

$$\textbf{(Sys0)}\,\frac{P \overset{M,t}{\rightarrow} P' \quad Sch \overset{M',t}{\rightarrow} Sch'}{\langle\!\langle P \rangle\!\rangle_{Sch} \overset{apply_P(M,M'),t}{\rightarrow} \langle\!\langle P' \rangle\!\rangle_{Sch'}}$$

$$\textbf{(Sys1)}\,\frac{P \overset{a}{\rightarrow} P'}{\langle\!\langle P \rangle\!\rangle_{Sch} \overset{a}{\rightarrow} \langle\!\langle P' \rangle\!\rangle_{Sch}} \qquad \textbf{(Sys2)}\,\frac{P\surd}{\langle\!\langle P \rangle\!\rangle_{Sch}\surd}$$

$$\textbf{(ER0)}\,\frac{Sys \overset{M,t}{\rightarrow} Sys' \quad \forall_{(ids,\rho)\in M, r\in R}\,\rho(r) = 0}{\partial_R(Sys) \overset{M,t}{\rightarrow} \partial_R(Sys')}$$

$$\textbf{(ER1)}\,\frac{Sys \overset{a}{\rightarrow} Sys'}{\partial_R(Sys) \overset{a}{\rightarrow} \partial_R(Sys')} \qquad \textbf{(ER2)}\,\frac{Sys\surd}{\partial_R(Sys)\surd}$$

FIGURE 1.10: Semantics of *PARS*, Part 3(a): Scheduled System Specification

To extend the semantics of process specification to the system specification, we need to define deadline and worst case execution time predicates on the newly defined operators. These operator and functions are defined in Figure 1.11.

The application operator $\langle\!\langle P \rangle\!\rangle_{Sch}$ is defined by semantic rules **(Sys0)-(Sys2)** in Figure 1.11. Rules **(ER0)-(ER2)** represent encapsulation of resource usage. In semantic rule **(Sys0)**, the application operator $apply_P : \mathbb{M} \times \mathbb{M} \rightarrow \mathbb{M}$ is meant to apply a multiset of resource providing predicates (second parameter)

$$\frac{P\dagger_t}{\langle\!\langle P\rangle\!\rangle_{Sch}\dagger_t} \quad \frac{P\dagger_t}{\partial_R(P)\dagger_t} \quad \frac{\mho_t(P) \quad \langle\!\langle P\rangle\!\rangle_{Sch}\dagger_{t'}}{\mho_{\min(t,t')}(\langle\!\langle P\rangle\!\rangle_{Sch})} \quad \frac{\mho_t(P) \quad \partial_R(P)\dagger_{t'}}{\mho_{\min(t,t')}(\partial_R(P))}$$

FIGURE 1.11: Semantics of *PARS*, Part 3(b): Scheduled System Specification, Deadlines and Worst Case Execution Times

to a multiset of resource requiring tasks (first parameter).

$$apply_P(M, [(pred, npred, \overline{\rho})] + M') \doteq$$

$$apply_P(applyTask_P(M, [(pred, npred, \overline{\rho})], \emptyset), M')$$

$$applyTask_P(\emptyset, [(pred, npred, \overline{\rho})], M) \doteq \emptyset$$

$$applyTask_P([(ids, \rho)], \emptyset, M) \doteq [(ids, \rho)]$$

$$applyTask_P([(ids, \rho)] + M, [(pred, npred, \overline{\rho})], M') \doteq$$

$$\begin{cases} [(ids, max(\overline{0}, \rho + \overline{\rho})] + & \textit{if } pred(ids, M + M') \wedge \\ applyTask_P(M, [pred, npred, \min(\overline{0}, \overline{\rho} + \rho)], & \neg npred(ids, M + M')) \wedge \\ \qquad M' + [(ids, \rho)] & \neg engage(P, M, M' + [(ids, \rho)], \\ & (pred, npred, \overline{\rho})) \\ [ids, \rho] + applyTask_P(M, [pred, npred, \overline{\rho} - \rho], & \textit{otherwise} \\ \qquad M' + [(ids, \rho)]) & \end{cases}$$

The intuition behind this definition is to apply the provided resources to the requirements while satisfying positive predicates and falsifying negative predicates. This is done by taking an arbitrary resource providing predicate, applying it to the resource requiring multisets (by checking its applicability to each task, i.e., pair of identifiers and resource requirements) and proceeding with the rest. In the above definition, $pred(ids, M + M')$ means that there exists a mapping from identifiers of the predicate (containing particularly a mapping from $\underline{Id}$ to a member of ids) that satisfies predicate $pred$. Expressions $min(\overline{0}, \overline{\rho})$ and $max(\overline{0}, \rho)$ are taking point-wise minimum and maximum of $\overline{\rho}(r)$ and $\rho(r)$ with 0, respectively. The predicate *engage* is meant to check that there is no transition from P that can potentially engage with the resources provided by $\overline{\rho}$ and satisfy the negative predicates in the current context. This predicate is defined formally as follows:

$$\begin{aligned}
&engage(P, M, M', (pred, npred, \overline{\rho})) \doteq \\
&\qquad \exists_{M'', P', ids', \widetilde{id}', \rho'} P \overset{M'',t}{\rightarrow} P' \wedge M' \subseteq M'' \wedge \\
&\qquad (ids', \rho') \in M'' - M' \wedge \rho' \bowtie \overline{\rho} \wedge \widetilde{id}' \in ids' \wedge npred(\widetilde{id}') \\
&\rho' \bowtie \overline{\rho} \doteq \exists_r \rho'(r) > 0 \wedge \overline{\rho}(r) < 0
\end{aligned}$$

Note that generally $apply_P$ is not a function and its resulting multiset may depend on the ordering of selecting and applying predicates and tasks. By definition, for all such outcomes, there exists a corresponding transition in the semantics.

THEOREM 1.5 Congruence of strong bisimilarity for the system language
Strong bisimilarity is a congruence with respect to all operators of our scheduled systems language.

PROOF The deduction rules are in the PANTH format and are stratifiable. Therefore, congruence of strong bisimilarity follows [23]. □

To better illustrate the semantics, we give a few examples of system scheduling in the remainder.

Example 1.10 EDF scheduling
Consider the following process specification and three different schedulers:

$$\begin{aligned}
&SysProc_{Id} \doteq 1 : (\sigma_1([(CPU \mapsto 1), (Mem \mapsto 50)](1))) \parallel \\
&\qquad\qquad 2 : (\sigma_2((CPU \mapsto 1)(Mem \mapsto 50)(2))) \\
&NP - EDF \doteq \mu X. \rhd^{\mathrm{n}}_{t \in I\!R^{\geq 0}} (\underline{Id}.Dl = t)([(CPU \mapsto -2), (Mem \mapsto -100)])(2) \\
&EDF_1 \doteq \mu X. \rhd_{t \in I\!R^{\geq 0}} (\underline{Id}.Dl = t)[(CPU \mapsto -2), (Mem \mapsto -100)](2) \\
&EDF_2 \doteq \mu X. \rhd_{t \in I\!R^{\geq 0}} ((\underline{Id}.Dl = t)[(CPU \mapsto -1), (Mem \mapsto -50)](2)) \mid\mid\mid \\
&\qquad\qquad \rhd_{t \in I\!R^{\geq 0}} ((\underline{Id}.Dl = t)[(CPU \mapsto -1), (Mem \mapsto -50)](2))
\end{aligned}$$

It is interesting to observe that according to the semantics of Figure 1.10, in the system $\partial_{Mem}(\langle\!\langle SysProc_{Id} \rangle\!\rangle_{NP-EDF})$ the only possible run follows the following scenario: The scheduler grants both available processors and the whole 100 units of memory for 3 units of time to process with identifier 1 since this is the active process with the least deadline. However, this causes the deadline of the tasks 2 to be shifted for 2 units of time (according to the semantics of parallel composition in Figure 1.4) and thus, process 2 will deadlock, after commitment of process 1.

For the system $\partial_{Mem}(\langle\!\langle SysProc_{Id} \rangle\!\rangle_{EDF_1})$, however, the scenario is different. The scheduler can start providing all available resources to task 1 for one unit of time but after that (since the choice of least deadline remains there) available resources will not be wasted anymore and will be given to process 2. However, the process misses its deadline anyway, since it needs 2 units of time and has a deadline of 1.

In contrary, the system $\partial_{Mem}(\langle\!\langle SysProc_{Id} \rangle\!\rangle_{EDF_2})$ allows for a successful run. In this case at the first time unit each of the two processes can receive a CPU and 50 units of memory. This is due to the fact that after providing the required resources of process 1 by one of the schedulers, the other scheduler may assign its resources to process 2 (see definition of operator $apply_P$ and in particular definition of *engage*). It follows from the semantics that after applying one resource offer to process 1 the whole process cannot engage in a resource interaction with a deadline of less than 2 and thus process 2 can receive its required resource.

The above behavior is in-line with the intuition of scheduler specification, as well. Scheduler $NP - EDF$ specifies a non-preemptive scheduler and thus cannot change its resource grant behavior after making the initial decision. Scheduler EDF_1 suggests that both processes and 100 units of memory should be granted to the process(es) that have the least deadline and thus, disallows other processes with higher deadlines from exploiting the remaining resources. Finally, scheduler EDF_2 specifies that two processes with the two least deadlines may benefit from the provided processor. $\square$

1.5 Conclusions

In this paper, we proposed a process algebra with support for specification of resources requirements and provisions. Our contribution to the current real-time and/or resource-based process algebraic formalisms can be summarized as follows:

1. Defining a dense and asynchronous timed process algebra with resource consuming processes
2. Providing a (similar) process algebraic language with basic constructs for defining resource providing processes (schedulers with multiple resources)
3. Defining hierarchical application of schedulers to processes and composing scheduled systems

The theory presented in this paper can be completed/extended in several ways. Among those, axiomatizing *PARS* is one of the most important ones in our list. We have presented some sound axioms for the process and scheduler specification part of *PARS* (which are considerably different from axioms of similar process algebras due to its special properties and constructs such as presence of the abstract parallel composition operator). However, a full axiomatization remains a challenge. As it can be seen in this chapter, the three phases of specifications share a major part of the semantics, thus, bringing the three levels of specification closer (for example, allowing for interaction among processes and schedulers or allowing for resource consuming schedulers) can be beneficial. Such a combination leads to more expressiveness (in that complicated interactions of scheduler and processes can be captured concisely), but is against our design decision to separate the world of schedulers from the world of processes. Furthermore, applying the proposed theory in practice calls for simplification, optimization for implementation and tooling in the future.

Another interesting extension of our work may be the integration with the algebraic framework for the compositional computation of trade-offs as developed in [13], which uses the same concepts of requested and granted resources. Such an extension may allow for trade-off analysis in the current algebraic setting.

Acknowledgments. Reinder Brill provided helpful insights on scheduling theory. Useful comments of the anonymous reviewers and the editors are also gratefully acknowledged.

References

[1] L. Aceto and D. Murphy. Timing and causality in process algebra. *Acta Informatica*, 33(4):317–350, 1996.

[2] J. C. M. Baeten and C. A. Middelburg. *Process Algebra with Timing.* EATCS Monographs. Springer-Verlag, Berlin, Germany, 2002.

[3] E. Bondarev, J. Muskens, P. H. N. de With, M. R. V. Chaudron, and J. Lukkien. Predicting real-time properties of component assemblies: A scenario-simulation approach. In *EUROMICRO*, pages 40–47. IEEE Computer Society, 2004.

[4] E. R. V. Bondarev, M. R. V. Chaudron, and P. H. N. de With. CARAT: a toolkit for design and performance analysis of component-based embedded systems. In *DATE*, pages 1024–1029, 2007.

[5] P. Brémond-Grégoire and I. Lee. A process algebra of communicating shared resources with dense time and priorities. *Theoretical Computer Science*, 189(1–2):179–219, 1997.

[6] M. Buchholtz, J. Andersen, and H. H. Loevengreen. Towards a process algebra for shared processors. In *Proceedings of Second Workshop on Models for Time-Critical Systems (MTCS'01)*, volume 52 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science, 2002.

[7] G. C. Buttazzo. *Hard Real-Time Computing Systems.* Kluwer Academic Publishers, Boston, Massachusetts, 2000. Third printing.

[8] F. Corradini, D. D'Ortenzio, and P. Inverardi. On the relationships among four timed process algebras. *Fundamenta Informaticae*, 38(4):377–395, 1999.

[9] F. Corradini, G. Ferrari, and P. Marco. Eager, busy-waiting and lazy actions in timed computation. In *Proceedings of Express'97 (Santa Margherita Ligure, Italy)*, Electronic Notes in Theoretical Computer Science, pages 133–150, 1997.

[10] M. Daniels. Modelling real-time behavior with an interval time calculus. In J. Vytopil, editor, *Proceedings of Formal Techniques in Real-Time and Fault-Tolerant Systems, Second International Symposium*, volume 571 of *Lecture Notes in Computer Science*, pages 53–71, Nijmegen, The Netherlands. Springer-Verlag, Berlin, Germany, 1991.

[11] J. Davies and S. Schneider. A brief history of Timed CSP. *Theoretical Computer Science*, 138(2):243–271, Feb. 1995.

[12] A. N. Fredette and R. Cleaveland. RTSL: A language for real-time schedulability analysis. In *Proceedings of the Real-Time Systems Symposium*, pages 274–283. IEEE Computer Society Press, Los Alamitos, CA, USA, 1993.

[13] M. C. W. Geile, T. Basten, B. D. Theelen, R. H. J. M. Otten. An algebra of Pareto points. *Fundamenta Informaticae*, 78(1):35–74, 2007.

[14] R. van Glabbeek and P. Rittgen. Scheduling algebra. In A. M. Haeberer, editor, *Proceedings of 7th International Conference on Algebraic Methodology And Software Technology (AMAST 99)*, volume 1548 of *Lecture Notes in Computer Science*, pages 278–292, Amazonia, Brazil. Springer-Verlag, Berlin, Germany, 1999.

[15] I. Lee, J.-Y. Choi, H. H. Kwak, A. Philippou, and O. Sokolsky. A family of resource-bound real-time process algebras. In *Proceedings of 21st International Conference on Formal Techniques for Networked and Distributed Systems (FORTE'01)*, pages 443–458. Kluwer Academic Publishers, August 2001.

[16] I. Lee, A. Philippou, and O. Sokolsky. Resources in process algebra. *Journal of Logic and Algebraic Programming*, 72:98–122, 2007.

[17] M.R. Mousavi, T. Basten, M. A. Reniers, M. R. V. Chaudron, and G. Russello. Separating functionality, behavior and timing in the design of reactive systems: (GAMMA + coordination) + time. Technical Report 02-09, Department of Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands, 2002.

[18] M.R. Mousavi, M. A. Reniers, T. Basten, and M. R. V. Chaudron. Separation of concerns in the formal design of real-time shared dataspace systems. In *Proceedings of the 3rd International Conference on Application of Concurrency to System Design (ACSD'03), Guimarães, Portugal.* IEEE Computer Society Press, Los Alamitos, CA, USA, 2003.

[19] M.R. Mousavi, M. A. Reniers, T. Basten, and M. R. V. Chaudron. Pars: A process algebra with resources and schedulers. In K. G. Larsen and P. Niebert, editors, *Formal Modeling and Analysis of Timed Systems: First International Workshop, FORMATS 2003, Marseille, France, September 6-7, 2003. Revised Papers*, volume 2791 of *Lecture Notes in Computer Science*, pages 134–150. Springer, 2003.

[20] X. Nicollin and J. Sifakis. The algebra of timed processes ATP: theory and application. *Information and Computation*, 114(1):131–178, Oct. 1994.

[21] G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, Aarhus, Denmark, Sept. 1981.

[22] G. D. Plotkin. A structural approach to operational semantics. *Journal of Logic and Algebraic Progamming (JLAP)*, 60:17–139, 2004. This article first appeared as [21].

[23] C. Verhoef. A congruence theorem for structured operational semantics with predicates and negative premises. *Nordic Journal of Computing*, 2(2):274–302, 1995.