RTSL: A Formal Language for Real-Time Schedulability Analysis

Andre N. Fredette† Rance Cleaveland‡

ABSTRACT

This paper aims to close the gap between real-time schedulability analysis approaches and formal specification and verification techniques for real-time. We develop a generalized approach to schedulability analysis that is mathematically founded in a process algebra called RTSL. RTSL is used to describe the functional behavior, timing behavior, timing constraints (or deadlines), and scheduling discipline for real-time systems. Within the framework of our approach, we formally model the most commonly used real-time scheduling disciplines and provide a capability for modeling others. The formal semantics of RTSL uses the scheduling discipline as a parameter and allows the reachable state space of any system to be automatically generated and searched for timing exceptions. We provide a generalized schedulability analysis algorithm to perform this state-based analysis.

1. Introduction

A *real-time system* is one which has time constraints, normally called deadlines, the violation of which can lead to a degradation of system performance or complete failure. The problem of *real-time schedulability analysis* involves establishing that a set of concurrent processes will always meet its deadlines when executed under a particular scheduling discipline on a given system. In this paper we will introduce a generalized model for real-time schedulability analysis that relies on the examination of the reachable states of a system. To facilitate this approach, we use techniques developed in the area of process algebra.

An approach to real-time schedulability analysis consists of three components: a process model, a scheduling discipline and a schedulability analysis algorithm. The process model specifies how the processes may act and interact, as well as the types of time constraints allowed. The scheduling discipline defines how the scheduler chooses among processes competing for processing time and other shared resources. Finally, the schedulability analysis algorithm may be used to determine whether a set of processes will meet its deadlines when executed under the given scheduling discipline. Examples of real-time scheduling approaches include fixed priority schemes such as *rate monotonic* scheduling (RMS) and dynamic priority schemes such as *earliest deadline* scheduling (EDS) and *least slack*. When using RMS, priorities are assigned to periodic processes as a monotonic function of the period length with shorter periods implying higher priorities. RMS analysis techniques were given in [12,13] for independent processes and then extended to

---

† Department of Electrical Engineering and Computer Science, United States Military Academy, West Point, NY 10996. Most of the work described in this paper was performed while author was a Ph.D. student at North Carolina State University.

‡ Department of Computer Science, North Carolina State University, Raleigh, NC 27695. Author supported by NSF/DARPA grant CCR-9014775, NSF Grant CCR 9120995, ONR Young Investigator Award N00019-92-J-1582, and NSF Young Investigator Award CCR-9257963.

processes which share resources via priority inheritance protocols in [11,17,18]. Limitations of existing RMS theory include an inability to handle direct synchronization between active processes and more than one time constraint per process. Except for the class of independent periodic processes, the EDS schedulability analysis algorithms typically consist of some form of exhaustive simulation to determine whether any execution sequence can result in a missed deadline [7,8,13,15,19]. The different process models allow various forms of inter-process communication; however, there are still limitations on these synchronizations and the time constraints.

An alternative form of analysis for real-time systems can be found in the field of formal specification and verification. In recent years, a number of formal models have been developed to reason about time in computer systems. Four major categories are Real-Time Logics [2], Timed Automata/Transition Systems [1,2,9] and Timed Process Algebras [4,6,16]. In general, these formalisms represent attempts to extend existing formal computation theory to capture the notion of time and as such support richer process models than those considered in the scheduling literature. However, a general limitation of existing formal methods intended for real-time (with the exception of [6]) is the lack of support for schedulability analysis. One problem is the common *maximal parallelism* assumption, that every process occupies its own processor, which obviates the need for schedulability analysis [4]. Other models either do not support concurrency or do not consider the effects of specific scheduling disciplines [1,2,9,16]. CCSR [6] represents a promising step in applying formal methods to real-time schedulability analysis. However, only discrete time systems and fixed priority scheduling are considered, and there are still limitations in the inter-process synchronization allowed.

In summary, various schedulability analysis approaches have been proposed; however, each model depends on a specific, and often limited, process model which is typically described informally. These informal descriptions can lead to difficulty in recognizing subtle differences between the models. Formal approaches for reasoning about timing properties of concurrent systems exist, but do not adequately address the concerns of schedulability analysis. These premises suggest that schedulability analysis technology can benefit from a formal examination using the techniques that have been developed for the formal specification and verification of programs.

2. A Generalized Model for Real-Time Schedulability Analysis

In this section, we characterize schedulability analysis as reachability analysis on the state space of a real-time system. This generalized approach has three components: a specification language, a scheduling discipline, and a reachability analysis algorithm. The specification language, called *Real-Time Specification Language* (RTSL), is used to specify the functional behavior, timing behavior, timing constraints (or deadlines), and the scheduling discipline for uniprocessor real-time systems. Once a system is specified and a scheduling discipline is chosen, the operational semantics of RTSL formally defines the state space of the system. Timing information is encoded in the states of the system allowing one to determine whether any deadlines have been missed.

Reachability analysis can be performed on the state space to evaluate the schedulability of the system. If no states in which a deadline is missed are reachable, then the system is schedulable. The most common scheduling disciplines (discussed in Section 1) have been modeled and, since the scheduling discipline is a modular component of the formal model, it is easy to add others. An additional benefit of the separation of scheduling discipline and process model is that it allows for the analysis of the same real-time system using various scheduling disciplines. The following sections present the syntax and semantics for discrete-time RTSL, formal models of real-time scheduling disciplines and considerations for the discrete-time analysis of real-time computer systems.

2.1. The Syntax of RTSL

RTSL has been designed to capture the notions that a real-time program is one which takes time to execute and has constraints on its execution time. We first must specify a time domain $\mathbb{D}$; in this paper we consider only discrete time and take $\mathbb{D} = \mathbb{N}$, the set of natural numbers. We describe an analogous continuous time model in [5]. The syntax and semantics of RTSL are based loosely on Robin Milner's CCS [14]. Processes are built from *actions* using various process constructors. Actions may be thought of as the instructions that processes execute. The functional behavior of a process can be described by the sequences of actions that it can perform. The RTSL action set contains actions of the form τ^d for $d > 0$ (which represents d units of *internal computation*), χ^d for $d \in \{0,1\}$ (which represents d units of *idling*), and a and $\bar{a}$ (which represent *complementary synchronization actions*). Intuitively, a process P wishing to perform an "a" must synchronize with a process Q ready to execute an "$\bar{a}$". This notion is very similar to synchronous message passing. We can intuitively think of "a" as a message channel with no buffer. So, in the above example, P is receiving a message on channel a and Q is sending a message on channel a. A duration function d indicates how much computation time this synchronization requires; for any synchronization action a, we require $d(a) > 0$.

This treatment of actions is a departure from the standard process algebra convention of letting actions be durationless events as indicated in [Nicollin91a]. We make this departure so that it is more natural to apply scheduling disciplines to the actions that processes perform.

Instead of using the duration function to identify the duration of synchronization actions, we could use the superscript convention as is done for internal actions; however, this would complicate the treatment of synchronizations. The question that arises is what to do when complementary synchronization actions are assigned different durations. Consider some of the options if P could do an a^3 and Q could do an $\bar{a}^2$. One option would be to let P and Q synchronize for either the maximum (3) or minimum (2) of the durations. Another alternative would be to let the synchronization take 2 time units and then make P do the internal action τ^1 following the completion of the synchronization. A third option would be to say that they are different actions and not let P and Q synchronize on channel a. All three of these options will

complicate the formal semantics of RTSL since it will have to check both the action and its duration to determine how to handle the synchronization. The alternative we choose is essentially the same as the third option. Any two complementary actions "a" and "$\bar{a}$" are assigned the same duration given by d(a). Varying execution times can be modeled by providing a different action for each possible execution time. For example, if a synchronization on channel a can take either 2 or 3 time units, we can define two actions a_2 and a_3 with $d(a_2)=2$ and $d(a_3)=3$.

Within the RTSL framework, a user may specify sequential processes with the Process Description Language (PDL). The syntax for PDL is given by the BNF grammar in Figure 2.1 below.

Proc ::= **nil** | **exc** | **X** | $\tau^{\mathbf{d}}$ | **a** | Proc + Proc | Proc ⊕ Proc | Proc**;**Proc | **μX.**Proc | **sw(**Proc**,d)** | **fw(**Proc**,d)** | **e(**Proc**,d)** | Proc **h** Proc

Figure 2.1. Syntax for RTSL Process Description Language (PDL)

Any finite string in PDL represents a user definable *process*. PDL has two pre-defined processes **nil** and **exc**. **nil** is the *terminated process* that can idle, but do no other action. The *exception process* **exc** cannot execute any action and is used to identify unsafe states in a process. **X** represents the name of a *process variable*. Process variables can be used in conjunction with the *recursion operator*, **μX**, to represent ongoing behavior in a process. For example, **μX.a;X** is the process that can perform **a** actions ad infinitum. This same behavior can alternatively be represented by using the following short hand notation: $C \stackrel{def}{=} a;C$, where C is a *process constant* and a;C is the *defining equation* for the process constant C. $\tau^{\mathbf{d}}$ represents a process that can perform d time units of internal computation, and then terminate. **a** represents a process that can synchronize on **a** and then terminate.

All other processes are built from these basic processes using the constructs shown in the PDL grammar. The basic process constructors are *external choice* (+), *internal choice* (⊕) and *sequential composition* (;). + and ⊕ are used to represent nondeterministic branches in the execution of a process. The process **a+b** can start either a synchronization on channel **a** or channel **b**, depending on which complementary action is available. + is called an external choice operator because the branch selection is dependent upon the actions offered by other processes in the system. As soon as **a+b** starts an **a**, it can no longer do the **b**. The operation of + is very similar to that of the Ada select statement. ⊕ is different from + in that the choice is made internally by the process. **a⊕b** might be used to represent an abstraction of the code segment "if N>k then do **a**, else do **b**." The sequential composition of processes P and Q (P;Q) behaves as P until P terminates and then behaves as Q. Thus, after the process $\mathbf{(a+b);\tau^4}$ completes either an **a** action or **b** action, it does 4 units of internal computation.

Time constraints are specified for processes using the *time constructs* **sw**, **fw** and **e**. In **sw**(P,d), **fw**(P,d) and **e**(P,d) the d represents a deadline for the enclosed process. This deadline is

measured relative to the time at which the enclosed process is first ready to execute. **sw** is the *start within* operator. **sw**(P,d) must perform the first action of P within d time units after it is ready; otherwise, an exception is generated. **fw** is the *finish within* operator. **fw**(P,d) behaves as P as long as it finishes within d time units after it starts; otherwise, an exception is generated. **e** is the *exactly* operator. **e** is similar to **w** except that **e**(P,d) takes exactly d time units to complete. If P finishes in d' time units where d' < d, then **e**(P,d) idles for d-d' time units. χd actions do not appear in PDL; however, **e** can be used to represent a specified delay. The process **e**(**nil**,d) idles for exactly d time units. As long as an enclosed process meets the time constraint, **sw** and **fw** have no effect on its behavior. **e**, on the other hand, actually imposes timing behavior on the enclosed process.

If an exception is generated by one of the time constructs or by the exception process, the process is said to be in an *exception state*. An *exception handler* may be defined using the **h** operator. P**h**Q behaves as P as long as no exception occurs in P. If an exception occurs in P, then P**h**Q behaves as Q. If an exception occurs in a process that has no exception handler, then the process terminates in an exception state. The goal of our generalized approach to real-time schedulability analysis is to determine whether any un-handled exceptions are possible in a system of concurrently executing processes. If an un-handled exception is possible, then the system is not schedulable.

In addition to the processes definable in PDL, the full RTSL grammar includes the following *auxiliary processes*. *Named processes* have the form Process_Name::P, where P is a process. *Synchronizations in progress* have the form a^d which represents an action a after it has been started, but has d units of processing time remaining. *Synchronizing pairs* are represented by terms of the form $P\langle a^d \rangle Q$, where P and Q are named processes. In $P\langle a^d \rangle Q$, processes P and Q are in the middle of a synchronization on action a, with d units of processing time required for the completion of the synchronization. Partially completed synchronization actions and synchronizing pairs may not be specified by a user, but may occur in the execution of a real-time system. *Proc*$^-$ is the set of all user definable named processes. *Proc* is the set of all RTSL processes which includes the members of Proc$^-$ together with synchronizations in progress and synchronizing pairs

The following definitions give a more precise meaning to the intuition that has been provided so far.

Definition 2.1:

- $\mathbb{D}$ is the duration domain. We let the lower case letter d range over the duration domain.
- *Act* is the set of all synchronization actions, we let the lower case letters a, b and c range over the set Act. For any $a \in$ Act, $\bar{a}$ represents its complement and also belongs to Act. We let α^d range over the set $\{a^d, \tau^d, \chi^d \mid d \in \mathbb{D}\}$.
- The *duration function* d: Act$\rightarrow\mathbb{D}$, provides a duration for every action in Act. For all $a \in$ Act, $d(a) = d(\bar{a}) > 0$.
- *ProcName* is a countable set of all *process names*. We use uppercase letters and words with the first letter capitalized to represent process names.
- *Proc*$^-$ is the set of all processes definable in PDL, together with *named processes* which have the form Name::Process, where Name is a process name and Process is defined in PDL.

Definition 2.2: *Aux* is the set of all *auxiliary processes* of the following two types:

- *synchronization in progress*: a^d. The appearance of a^d indicates that the synchronization action a has been started and d units of processing time are required to complete the synchronization.
- *synchronizing pair*: $P\langle a^d\rangle Q$, where $\{P,Q\} \subseteq$ Proc$^-$, $a \in$ Act and $d \in \mathbb{D}$. In $P\langle a^d\rangle Q$, processes P and Q are in the middle of a synchronization on action a, with d units of processing time required for the completion of the synchronization.

Definition 2.3: *Proc* = Proc$^-$ $\cup$ Aux is the set of all RTSL processes. We generally let the letters P,Q,R and S range over the set Proc; however, process names, process constants and process terms are sometimes used synonymously.

The following shorthand notation for the nondeterministic choice operator is often useful:

Definition 2.4: An *indexed sum* $\sum_{i\in I} P_i$ is shorthand notation for the process $P_{i_1} + P_{i_2} + \cdots + P_{i_n}$ for each $i_j \in$ I. The set I is finite.

Common types of processes that RTSL can be used to specify include periodic, sporadic, and server. For example, a generic periodic process P with period t can represented as $\mu X.e(P,t);X$ or with the shorthand notation $C \stackrel{def}{=} e(P,t);C$.

Variable length execution times for computation and synchronization can also be represented. The basic synchronization primitive can be used to implement various forms of inter-process communication and cooperation. The **sw**, **fw** and **e** operators allow the specification of time constraints for the completion of a whole process, as well as for sub-components of a process.

Process constants provide very useful capabilities that are worth mentioning. First of all, process constants can be used to define what might be thought of as macros. RTSL has only one action (τ^d) to represent internal computation. Particular code segments can be given more descriptive names by using process constants. For example, if Procedure A, that takes 10 time units to complete, is executed in several places in a process, it can be defined as $ProcA \stackrel{def}{=} \tau^{10}$. Then,

whenever Procedure A is called in any process, the process constant ProcA can be used instead of τ^{10}. This is also useful for code segments with varying execution times. For example, if Procedure A could take from 8 to 10 time units, it would be defined by $ProcA \stackrel{def}{=} \tau^8+\tau^9+\tau^{10}$ or $ProcA \stackrel{def}{=} \sum_{d\in\{8,9,10\}} \tau^d$. Synchronizations can also be defined in this manner. Consider a send and receive on channel b with a set of execution times given by I. $SendA \stackrel{def}{=} \sum_{i\in I} \bar{a}_i$ and $ReceiveA \stackrel{def}{=} \sum_{i\in I} a_i$, with $d(a_i)=i$.

A real-time system is specified by a set of named processes and a priority function as defined below:

Definition 2.5: A *real-time system* is specified by a tuple $S = \langle \mathcal{P}, \Pi_{sd} \rangle$ where

- The *process set* $\mathcal{P} \subseteq 2^{Proc}$ is a finite set of uniquely named processes specified in RTSL.
- The *priority-function (or* Π *-function)* $\Pi_{sd} : 2^{Proc} \rightarrow 2^{Proc}$ is a function that returns the set of highest priority processes for any system state. For any state σ, $\Pi(\sigma) \subseteq \sigma$.

The Π-function, which is used to model scheduling disciplines, is covered in more detail in Section 2.4. For technical reasons, all of the processes in $\mathcal{P}$ must be named; however, in our examples we will often abuse this requirement and omit the process names for the sake of simplicity. To use RTSL specifications in a reachability-based analysis approach, it is necessary to define precisely how the processes behave. This definition appears in the form of an operational (or transitional) semantics. The semantics we present defines how both individual processes behave, and how processes may cooperate in a system.

2.2. A Discrete-Time Process-Level Semantics for RTSL

The *transition relation* $\rightarrow$ defines the transitions available to processes. In what follows we let α^d represent actions of the form τ^d, χ^d and a^d. Intuitively, $P \xrightarrow{\alpha^1} Q$ means P may execute action α for one time unit and become Q. The behavior of processes is formally defined by a transitional semantics which is given by a set of transition rules of the form:

$$\frac{\text{Premises}}{\text{Conclusions}} (\text{Side Conditions})$$

If the premises and side conditions are true of a process, then one may conclude that a particular transition or predicate is valid. The semantics uses the termination and exception predicates defined in Figure 2.2 below.

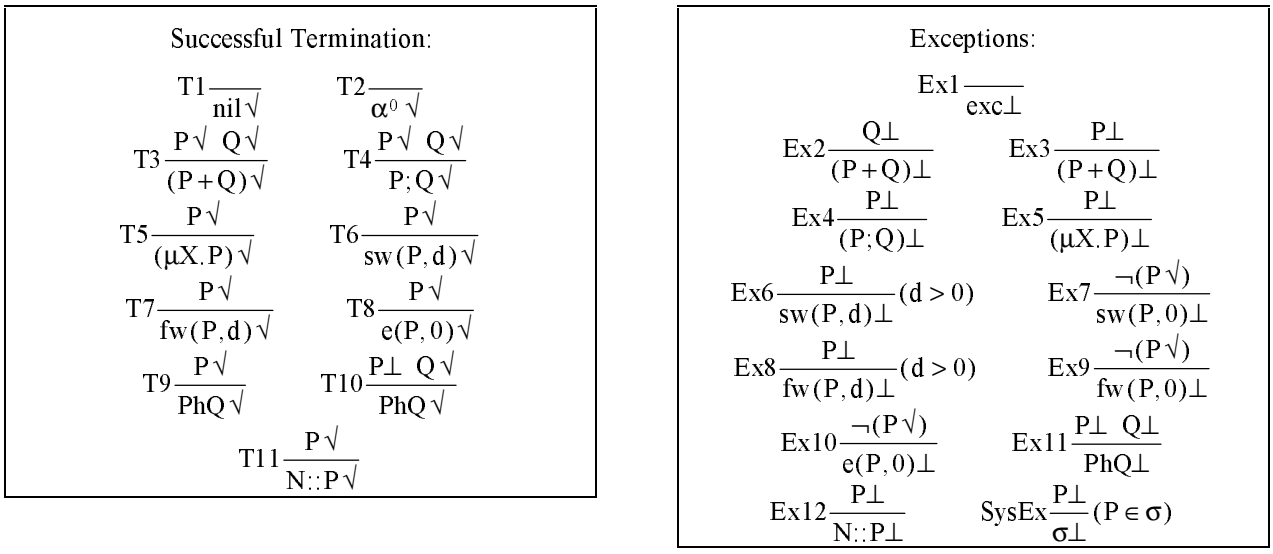


Figure 2.2. RTSL Termination and Exception Predicates

In Figure 2.2, $\surd$ is a predicate that is true for successfully terminated processes, while $\perp$ is a predicate holding of processes in an exception state.

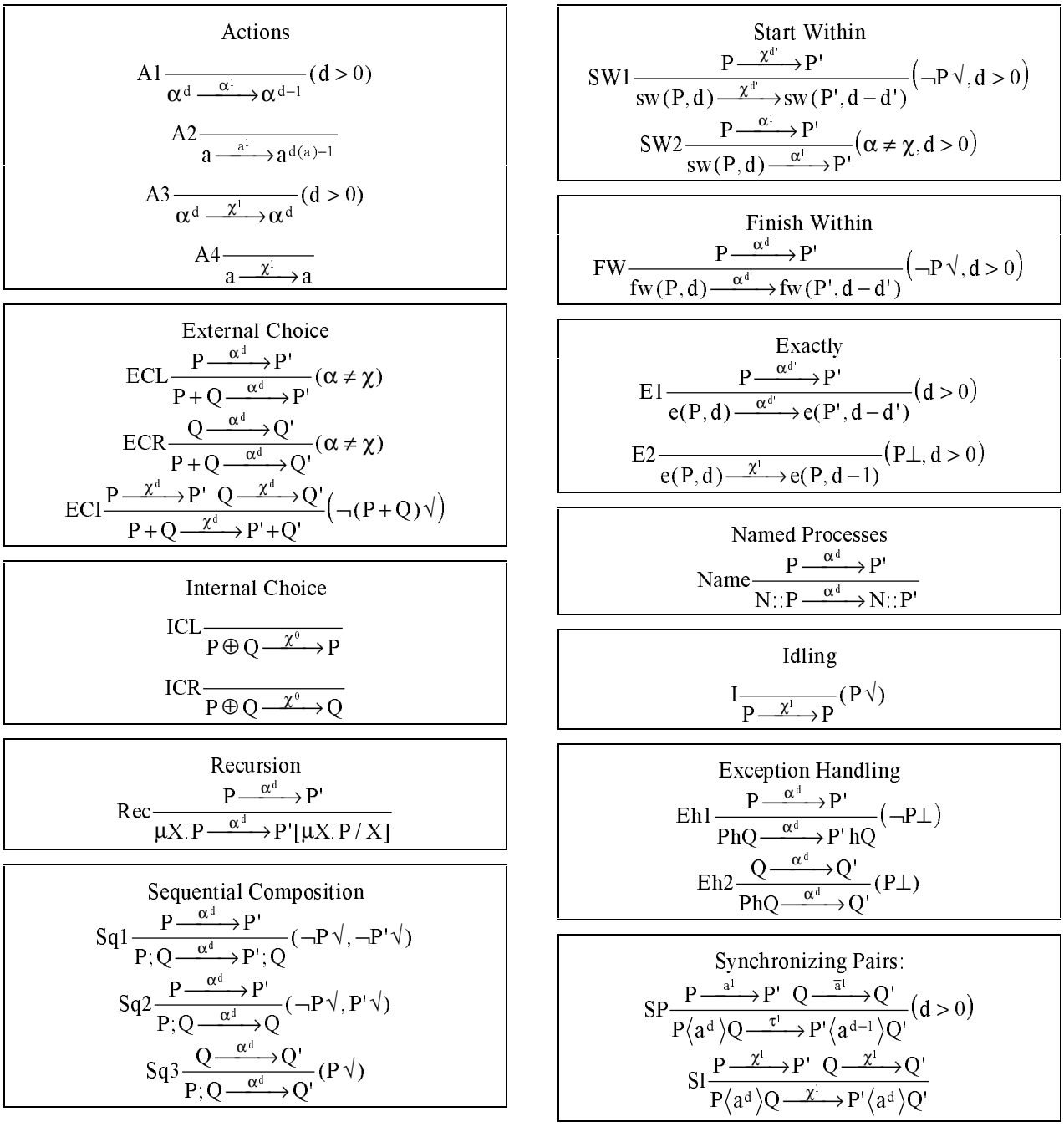
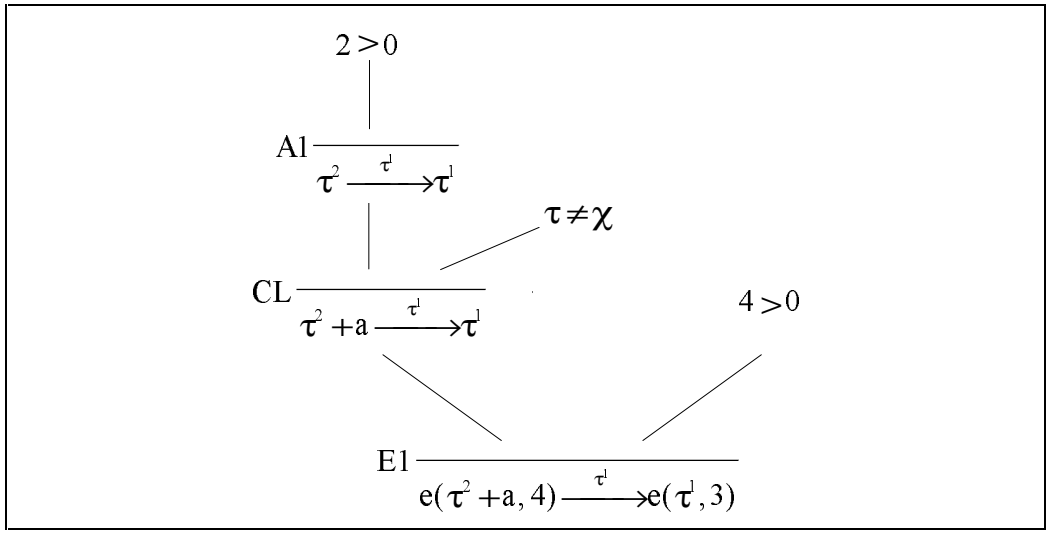


Figure 2.3. Discrete-Time RTSL Process-Level Semantics

The rules in Figures 2.2 and 2.3 are inference rules defined recursively on the structure of processes. Rule A1 is an example of a rule which requires no premises. According to rule A1, if d>0, the process α^d can perform one unit of α and then evolve to the process α^{d-1}. Rule FW is an example of a time construct rule. FW says that given a process of the form fw(P,d), if P is not terminated ($\neg P\surd$), and there is time left (d>0) and P can perform the action α^1 and evolve to P',

α^1 v ' 1. W y s
s by s g g by l
y s . F x l ll g g vs ll g s s
vl : $\tau^2 + 4 \xrightarrow{\tau^1} \tau^1 3$



Fg 2.4. I D g T s $\tau^2 + 4 \xrightarrow{\tau^1} \tau^1 3$

Rl s 3 4 E2 I ll l l g ss A g l χ^d
b ul ss s u b s s l ys F
x l l 5 s ss l ys x yl 5 u s T s s Fg 2.3
delay determinism y s bl .

D 2.6: T *delay determinism* y s s ll s
$\forall\{ ' \ "\} \subseteq \xrightarrow{\chi^1} ' \wedge \xrightarrow{\chi^1} " \Rightarrow ' = "$ = sy q l y

T s s s b v v ul ss A -l y s v
ss s u yl x u g ss T ul b v v ul ss
s ul by b v ss ss . T ss -l v l
b v s s x s .

2.3. A Discrete-Time System-Level Semantics for RTSL

Sy s s by s ss l s q l $\mathcal{P}$. I
ll s l σ g v s ys s s A s - ys v vs *tick*
g b l l k D g k u u τ^1 b l . T u
k b l ss x u g l ss
y z g T ss s g u u s ll *active*
ss s. T ss ys ll *inactive* ss T s s
v ss s y s *idle* χ^1 l ss s s . I ss s
y u u g k ll ss l. T *CanIdle* σ s u s
ss ll ss s l u s T u *Idle* σ u s

s ss σ' b σ' s ss σ s l u s C Il Il yll bl . O us yl C I1 Il s s =1

D 2.7: F σ ⊆ 2^Proc ∈ 𝔻

- C I1 σ ∀ ∈σ ∃ ! $\xrightarrow{\chi^d}$ '
- I 1 : $2^{\text{Proc}} \times \mathbb{D} \to 2^{\text{Proc}}$ Il σ ={'| ∈σ ∧ $\xrightarrow{\chi^d}$ '}

T s s v lb ys s s by ss l v l s s Fg 2.6. T s s v lb ys s σ u s s v lb ss σ. T s ul g s l s ss ss sl x u ss u g y k T y u Π σ s s g s y ss s σ. F σ Π σ s u s ss s ul g s l . S Π u s lb s b S 2.4.

A ys s g u H D 2.10 s ll k y ys z g s b sl u. S z g s sl u ys z l s ss vl ys z su. I yl b su x us b s ss T x s s by u lz u N D 2.8. I x l ss su l ys z ss Q N s ll Q N s Q ss z g l s x ss T Q s x l x b l yl s Q ll b x s .

H s b b b ys z g s u k u lz b . T u s v Fg 2.5 s v s u s. A s lg g k x s ul b us us, s v s lb . T x s s Q su. T s s b us by ul Ex10 b ⊥ Q⊥ v Q⊥. Rul Ex11 sb T 6 l s s.

T Ou s

T 1 $\frac{P\tilde{\perp}}{(P;Q)\tilde{\perp}}$ T 2 $\frac{P\tilde{\perp}}{\text{sw}(P,d)\tilde{\perp}}(d>0)$ T 3 $\frac{P\tilde{\perp}}{\text{fw}(P,d)\tilde{\perp}}(d>0)$

T 4 $\frac{\neg(P\surd)}{\text{fw}(P,0)\tilde{\perp}}$ T 5 $\frac{\neg(P\surd)}{\text{e}(P,0)\tilde{\perp}}$ T 6 $\frac{P\tilde{\perp}}{\text{PhQ}\tilde{\perp}}$ T 7 $\frac{P\tilde{\perp}}{\text{N::}P\tilde{\perp}}$

Fg 2.5. RS T Ou s

D 2.8: N → s *neutralization function* :

- N d = x
- N Q = N Q
- N Q =N Q
- N = N
- N = N

- N N : = N N

D 2.9: H_p: $\to 2^{Proc}$ s *process heating function* :

- H_p $=\{\}$ $\neq$ $_1\langle\ ^d\rangle$ $_2$;
- H_p $\langle\ ^d\rangle Q$ $=\{$ $\langle\ ^d\rangle Q$ >0 $\wedge\neg$ $\tilde{\perp}$ $\wedge\neg Q\tilde{\perp}$;
- H_p $\langle\ ^d\rangle Q$ $=\{$ $NQ\}$ >0 $\wedge$ $\tilde{\perp}$ $\wedge\neg Q\tilde{\perp}$;
- H_p $\langle\ ^d\rangle Q$ $=\{N$ Q >0 $\wedge\neg$ $\tilde{\perp}$ $\wedge Q\tilde{\perp}$;
- H_p $\langle\ ^d\rangle Q$ $=\{$ Q >0 $\wedge$ $\tilde{\perp}$ $\wedge Q\tilde{\perp}$;
- H_p $\langle\ ^0\rangle Q$ $=\{$ Q

D 2.10: H $2^{Proc}\to 2^{Proc}$ s *system heating function* :

$$H\sigma = \bigcup_{P\in\sigma} H_p$$

I ss Q ys z g k s lvlb ss s v yll l ys z . O ss s d ss s l y $^{-d}$. W sl k ss v Q b s s g u v svl b by u C1 C2 d y b s q yl s ss g b ls b y ub s s x l s F s s s H N $\tilde{\perp}$ yl v l g g s u s yb u .

T s - s lvls s RSL s s Fg 2.6 b l .

$$\text{Sync}\ \frac{P\xrightarrow{a^1}P'\quad Q\xrightarrow{\bar{a}^1}Q'\quad \text{CanIdle}(\sigma-\{P,Q\},1)\quad \sigma\overset{i^0}{\not\longrightarrow}}{\sigma\xrightarrow{\tau^1}H(\{P'\langle a^{d(a)-1}\rangle Q'\}\cup \text{Idle}(\sigma-\{P,Q\},1))}\begin{pmatrix}\neg\sigma\perp,\ \{P,Q\}\subseteq\sigma,\\ \{P,Q\}\cap\Pi(\sigma)\neq\varnothing\end{pmatrix}$$

$$\text{Comp}\ \frac{P\xrightarrow{\tau^1}P'\quad \text{CanIdle}(\sigma-\{P\},1)\quad \sigma\overset{i^0}{\not\longrightarrow}}{\sigma\xrightarrow{\tau^1}H(\{P'\}\cup \text{Idle}(\sigma-\{P\},1))}(\neg\sigma\perp,\ P\in\Pi(\sigma))$$

$$\text{Idle}\ \frac{\text{CanIdle}(\sigma,1)\quad \sigma\overset{i^0}{\not\longrightarrow}}{\sigma\xrightarrow{\chi^1}H(\text{Idle}(\sigma,1))}(\neg\sigma\perp,\ \Pi(\sigma)=\varnothing)$$

$$\text{SysIC}\ \frac{P\xrightarrow{i^0}P'}{\sigma\xrightarrow{i^0}\{P'\}\cup(\sigma-P)}(\neg\sigma\perp,\ P\in\sigma)$$

Fg 2.6. D s T RSL s L v l S s

T ss l v l s us b l sv l u s bu v s bs s u u . T u sbl s ss l v l s . T ss s ys z by u S ss u l u by u C ll ss l u by u Il l b sl by u SysIC. Rl s S C s *active* s s l s ss x u s u Il s s *inactive* s ll ss l .

W ll x S u .l By u g $\neg\sigma\perp$ s s S s ll s C Il us s u ss s x s . T { Q}$\subseteq\sigma$ q s ss b g s b s ss s ; u { Q}$\cap\Pi$ $\sigma\neq\varnothing$ q s l s ss s

g g s y ss s . T ss $\xrightarrow{a^1}$ ' Q $\xrightarrow{\bar{a}^1}$ Q'
q Q s l y s z .s T ls s
C I l σ{ Q} q s ll ss s σ b b l u .
T ly s y s u l s s. B u ly s s
I l σ{ Q} s u q s y R v y R σ{ Q}. T d
S y u s s bv s s l s σ s σ k
s ss s σ s vl s .

T H g u H s l g s s U g g u
ll s u s l s l-v l s s u z x l g

F x l ss l v l s s s s σ = {; ¯Q τ²}
Π σ = { ; } l T ll g g s s s
{ ; ¯;Q τ²} $\xrightarrow{\tau^1}$ { Q τ²}.

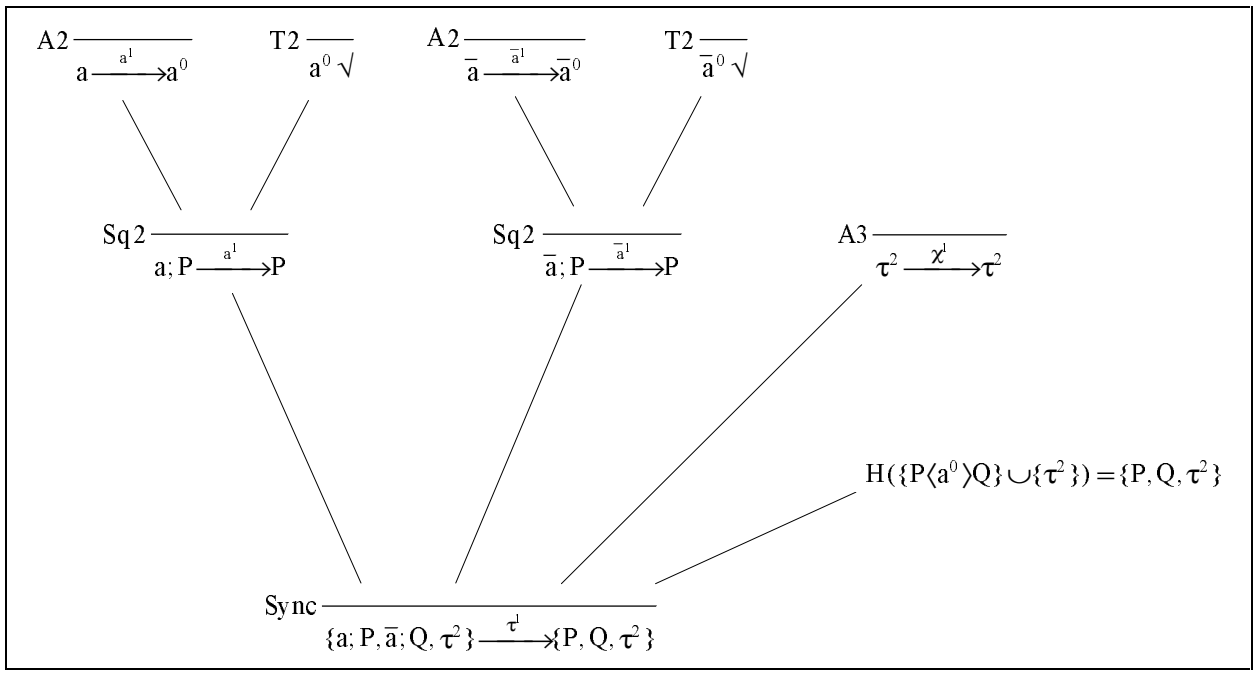


Fig 2.7. I D g T s { ; ¯;Q τ²} $\xrightarrow{\tau^1}$ { Q τ²}

U g ss l v l s s b v y -l s S b l vl
s b b s ll *timed transition system*.

D 2.11: A *timed transition system (TTS)* s u-ul ⟨Σ T σ_0 E⟩ :

- Σ s s *states*;
- T ⊆ Σ×{τ^1 χ^1}×Σ s *transition relation*;
- σ_0 ∈ Σ s *start state*;
- E ⊆ Σ s s *exception states*.

T s TTS s b s l b l s s [C v l 9] A u g
s s [H z g 9] b s s s u TTS s
s l y I TTS RTSL s s s s s s s

s s s $\mathcal{P}$ s s s by ss l v l l s s gv
Fg 2.6.

D 2.12: T b v y -l ys $S = \langle \mathcal{P}\ \Pi_{sd} \rangle$ b s b by s *corresponding timed transition system* $M_S = \langle \Sigma\ T\ \sigma_0\ E \rangle$:

- $\sigma_0 = \mathcal{P}$;
- $\Sigma \subseteq 2^{Proc}$ $T \subseteq 2^{Proc} \times \{\tau^1\ \chi^0\ \chi^1\} \times 2^{Proc}$ usvyl s ll s
  1. $\sigma_0 \in \Sigma$;
  2. $\sigma \in \Sigma$ $\sigma \xrightarrow{\alpha^d} \sigma'$ $\langle \sigma\ \alpha^d\ \sigma' \rangle \in T$ $\sigma' \in \Sigma$;
- $E = \{\sigma \mid \sigma \in \Sigma \wedge \sigma\!\perp\}$.

F y -l ys $S = \langle \mathcal{P}\ \Pi_{sd} \rangle$ s g TTS $M_S = \langle \Sigma\ T\ \sigma_0\ E \rangle$ b s ubl y ysls u s bl y ysls s s M_S. I x s E s b s s $\mathcal{P}$ S s s ulb. I b us l M_S S ss ub yl $E = \emptyset$.

D 2.13: A -l ys $S = \langle \mathcal{P}\ \Pi_{sd} \rangle$ s *schedulable* $E = \emptyset$ s s g TTS $M_S = \langle \Sigma\ T\ \mathcal{P}\ E \rangle$.

Gv -l ys s $S = \langle \mathcal{P}\ \Pi_{sd} \rangle$ s s g LTS $M_S = \langle \Sigma\ T\ \sigma_0\ E \rangle$ b ul yz s u ysby us g ll g g .

```
        u S ub σ0 : STATE : BOOLEAN
var
    Σ E A yz : SET STATES = ∅;      /* Σ E σ0 s TS */
    S Sa Se : SET STATES;
    σ σ' : STATE;
begin
    Σ = Σ ∪ {σ0};
    if σ0⊥ then E = E ∪ {σ0}
    else A yz = A yz ∪ {σ0} end if;
    while A yz ≠ ∅ loop
        s σ ∈ A yz ;
      A yz = A yz - {σ};
      S = {σ' | σ --α^d--> σ' ∧ σ ∉ Σ};   /* s s b σ */
      Σ = Σ ∪ S;                          /* s s Σ */
      Se = {σ' | σ' ∈ S ∧ σ'⊥};           /* s s S */
      Sa = S - Se;                        /* s s S yl */
      E = E ∪ Se;
      A yz = A yz ∪ Sa;
    end loop
    return E = ∅ ; /*ys ss ub s s*/
end S ul b ;
```

Fg 2.8. G lz Ds T S ubl y A ys Ag

T bv g gs y s s s Σ E A yz . Σ s g b l s
s s TS E s s x s s A yd ls y v y b
yz . T g s k s s s x s . I Σ A yz
lz {σ0} W l A yz s ll s u xl s g vs
s σ A yz u s S s s s b σ l y Σ
s Σ. S s x s s -x s s T
x s s E -x s s A yz . W l
b s s v b xl l s ys ss ulb s
x s s E s y I s ss y sv s s v lb s
b us s l y by s s

I s bv g g b s b. I ll s y s
s s ulb l y g b sg y s s g s
s s x s s s v . A bv g ll s
lu s g s s . W l s s by q
ul g l s s *depth first search* g b us u bgg g
u ss I us s ls s s xl s s v
x s s s v sq s s s l x s
lb s k yb gg u ss

T l s ll g s ug s l s v sbs qu
s :s Fx y E l s D l S yI .l D s
L s S k yC l g l b u F 9

2.4. Formal Models of Real-Time Scheduling Disciplines

A s ug s l b v s sg s s .s T
ss -l s ulb l y ys s su ll s s gv
s v s y ss s ss v l g us
s ub vl s ss s .s T us
s ug s l s ss y v *priority order* $\leq_{sd} \subseteq \times$. T y
s l s l v s ss ys . T l
s ug s l s ys ys s g s y y
ss ys s σ s ug s l s gv by Π *-function* Π_{sd} σ
.

F y s yl s s v lb g s y ss y b
k . O s u y s y b z g s y y
ss I u wl s s *ready* ss y x u . B
s s ug s l s y ss s s
l s s yll bl .

D 2.14: R y σ ∈ σ ∧
- ∃ ' . $\xrightarrow{\tau^d}$ '
- ∃ ' . $\xrightarrow{a^d}$ ' ∧ ∃ Q∈σ{ } Q . Q$\xrightarrow{\bar{a}^d}$Q'.

D 2.15: F y s *sort* s s s ll ys z s yb yp y s v vs s s s l s [M l 8].

D 2.16: *Complements* s s s :s
- F y s l s yp ¯.
- ¯s' l = s ql .
- F ys sA ⊆ A Ā ={¯ | ∈ A}.

2.4.1. Fixed Priority

T s ls s ug s l l s *fixed priority* s ug s l . I x ys lg s s sg x y u- . T s s g gu x u ys . T s ug s l s s x l x ys ug s ss C 2

D 2.17: T *fixed priority order* ≤fp ⊆ × s s ll s
- F ∈ $\mathcal{P}$ y ∈ ℕ u l ub s s s . v y ∈ $\mathcal{P}$ s ℕ
- ⟨ d⟩Q = x Q}
- ≤fp Q ≤ Q.

T x y Π-u b s ll s

D 2.18: Πfp σ ={ | ∈σ∧R y σ ∧∀Q∈σ{ } y Q σ ⇒ Q ≤fp }

2.4.2. Earliest Deadline

Earliest deadline s ug s y y s ug s l s u s l . W l s l s ug s s s g y s l l . I l l s yl l s b RTSL y b l s sg s I lu s l s b gu l s s b s F x l s 1 5 + 2 0 l b v g y g b s s l l ? W l l s v y b l s l bu gg lu b s . O s s x u b s s s s ks y u b . I s - l s s k y b s ll b s x s s s l s us l .l T l s s yl yp *deadline function* lgv 2.19 b l .

D 2.19: :l →𝔻 s *deadline function* :

- l A = l
- l = ∞ ∈ { l x α^d X
- l Q = { l l Q }
- l ⊕Q = { l l Q }
- l Q = l ¬ √
  l Q = lQ √
- l μX = l
- ls = { l }
- l = { l }
- l = { l }
- l Q = l ¬ ⊥
  l Q = lQ ⊥
- l ⟨ ^{d}⟩Q = { l l Q }

D 2.20: T *earliest deadline priority order* $\leq_{ed}$ ⊆ × s s l s $\leq_{ed}$ Q lQ ≥ l .

D 2.21: Π_{ed} σ ={ | ∈σ∧R y σ ∧∀Q∈σ{ } yQ σ ⇒ Q $\leq_{ed}$ }

2.4.3. Basic Priority Inheritance Protocol

T Bs y I l I s [S 90b] ss u y s g s l s s T s s s y z s l y g s s u bu y b l b k g yb x b y s u s s su s S u bl y y s s s y s -l s s s s s y s s yb v .

T g l b y s g y s H g s b k by l y s L L x u s H s y u l L s s v s g s b k g H. y s s lk lg l s k g y s H g g s q k s s b .

T s y s b y s v s s g y s z . T s s *synchronization partner* u .

D 2.22: s: × 2^{Proc} × A →2^{Proc} s *synchronization partner* u .

- s σ =Q|Q ∈ σ - ∧ ¯ ∈ s Q}

s σ u s s ll s s s σ y x u l y ¯ u y s l . I s s u l l y s u s s u b s v s s y y s z s v s s s v s s u y s s y s z .s I s l s y D 2.22 l s g l s y s z s

ss y g. W s sl yll s ss b
lb k ly s

D 2.23: b ⊆ × × 2^{Proc} s *blocked-by* l : bQ σ

- ∈ σ
- ∃ ' d . $\xrightarrow{a\ d}$ '
- σ'=s σ
- ∀R ∈ σ' ¬ ∃ R d' .R $\xrightarrow{\bar{a}\ d'}$ R'
- Q ∈ σ'

T l y l s s y yl y
l s ; v g l s ss lu qs y
sv l s v s s v ss y b b v g gv
s v s sl l b RTSL. T s b l s b s u s
W y vs s g l s s s
ss u .

W y l s s ss x u s g s y ll
ss s lb k g B s y s s v g l
s s v ls lb k ly l y
gv ss .

D 2.24: F y -l sy S = ⟨$\mathcal{P}$ Π_{pi}⟩ *basic priority inheritance priority order* $\leq_{pi}$ ⊆ × s s ll s:

- E ∈ $\mathcal{P}$ s ssg bs y ∈ ℙ s v g s g u b
  g g y
- v_ = x{ Q | Q bb$^+$ } ∪ { } bb$^+$ s s v ls
  lb k ly l .
- v_ ⟨ d⟩Q = x{ v_ v_ Q}
- $\leq_{pi}$ Q v_ ≤ v_ Q.

T bs y Π-u s s ll s

D 2.25: Π_{pi} σ ={ | ∈σ∧R y σ ∧∀Q∈σ{ } y Q σ ⇒ Q $\leq_{pi}$ }

2.5. An Example Application

A s l x l ll b s v u v l u y s q
ks I s x l ll l D b s v g s $\mathbb{Z}^+$. E
b $\mathbb{Z}^+$ s s g u l s ll s u - b x u
. T x l s sss ss a b ll g
su - s s

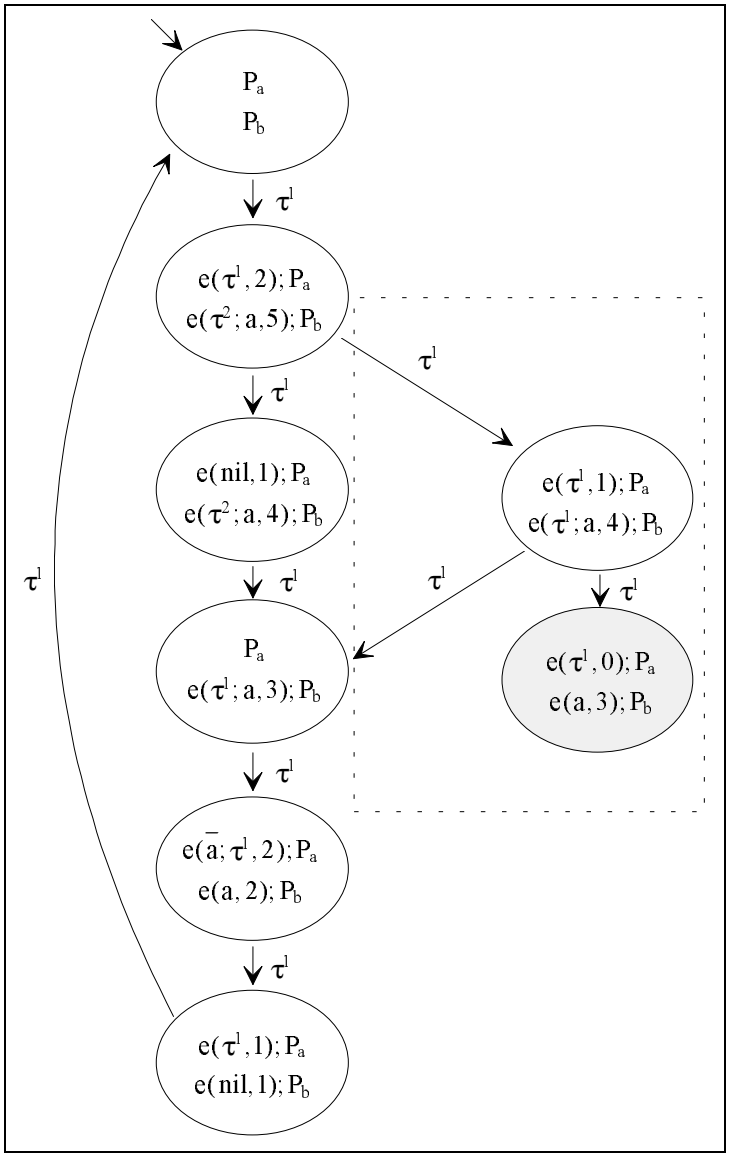
a: v y3u s
s sg l ;
u u ;
;

b: v y6u s
v sg l
;
u s u ;
v sg l
;
;

I s s g sg ks
u RTSL
s s ss
u b :

- a $\stackrel{\text{def}}{=}$ $\bar{\ }$;τ^1 3; a
- b $\stackrel{\text{def}}{=}$;τ^2; 6; b
- $\dashv$
- S= $\langle$ {A: a B b} Π_{sd} $\rangle$

W s s s a
b s s
x y s u g s l
b ss v s
y s s . T TTS s
Fg 2.9 u b u s
s. T u q s s A
B g
sv s . N s s s x s τ^1 0 ; a $\perp$ b s a s
l . S s s b x s s s u b. I y
sg s g s b $\leq$ a s u ub s s Fg
2.9 s s s s s s b x T s s x s s
s s u b. I L S=$\langle \Sigma\ T\ \mathcal{P}\ E \rangle$ b TTS s Fg 2.9



Fg 2.9. TTS Ex l b

$\langle \Sigma, T, \mathcal{P}, E \rangle$ be the modified TT , it is interesting to note that $\Sigma \subsetneq \Sigma$, $T \subsetneq T$, and $E \subsetneq E$. These relationships illustrate the filtering effects of scheduling disciplines. When more restrictive the scheduling disciplines are used, fewer system states are possible.

ome general notes on the application of state-based analysis techniques are useful here. In the worst case, the problem commonly referred to as the *state explosion problem* may arise: the number of system states can be exponential in the number of processes. However, we hypothesize that the filtering effects of scheduling disciplines and the regular structure of many real-time systems will tend to control the problem in practice. We plan to test this hypothesis experimentally as soon as an analysis tool based on RT L has been developed. The complexity of the analysis of discrete-time systems is also a function of the duration of the unit time quantum. While a smaller time quantum results in less wasted time, this time efficiency is achieved at the expense of an increase in complexity. If the time quantum is large enough so that every action takes one unit of time, then the complexity of the timed analysis will be comparable to the untimed case. It has also been shown that the size of systems that can be analyzed can be significantly increased by using clever techniques for representing states and searching the state space [10]. The analysis of a system with over 250,000 states is reported in [10]. Furthermore, by using binary decision diagrams to represent state systems symbolically instead of explicitly, the analysis of systems with more than 10^{20} states is conceivable [3]. Finally, with the formal model outlined in this paper, we believe that it will be easier to develop and analyze complexity reducing analysis techniques such as are presented in [20].

3. Conclusions

This dissertation aims to close the gap between real-time schedulability analysis approaches and formal specification and verification techniques for real-time. This goal is accomplished by developing a generalized approach to schedulability analysis that is mathematically founded in a process algebra called RT L. RT L is used to describe the functional behavior, timing behavior, timing constraints (or deadlines), and scheduling discipline for real-time systems. Within the framework of our approach, we formally model the most commonly used real-time scheduling disciplines and provide a capability for modeling others. The formal semantics of RT L uses the scheduling discipline as a parameter and allows the reachable state space of any system to be automatically generated and searched for timing exceptions. We provide a generalized schedulability analysis algorithm to perform this state-based analysis.

The formalization of the general concept of real-time scheduling theory is enlightening in several ways. The RT L approach is less restrictive than existing real-time schedulability analysis approaches about the kinds of processes and time constraints that can be specified and analyzed. Within RT L, time constraints can be specified for a whole process, as well as for any component of a process. Exception handlers can also be specified and analyzed. RT L has a basic synchronization primitive with which most general forms of inter-process cooperation can be modeled. We show how to model standard synchronous message passing and semaphores among others. The ability to specify more general types of processes provides greater flexibility to real-

time system developers while maintaining the desired property of verifiability of schedulability. The modularity of the scheduling discipline in RT L provides a common process model upon which various scheduling techniques may be tested. Finally, RT L can serve as a theoretical foundation for developing and analyzing new and existing real-time scheduling approaches.

This research makes two contributions to the area of formal methods for real-time systems. The first is the formal characterization of real-time scheduling disciplines and the inclusion of the scheduling discipline as parameter to the system-level semantics. To accomplish this task, it was necessary to develop a model that is compatible with the types of processes and scheduling disciplines commonly used in real-time applications.

As in any state-based analysis approach, the primary limitation of the techniques presented in this dissertation is complexity. In the worst case, the size of a real-time transition system is exponential in the size of the processes. A benefit of including the scheduling discipline in the formal model is that it acts as a filter on set of all possible execution sequences. This filtering effect tends to reduce the size of the resulting transition system. The regular structure of many real-time systems, such as periodic processes, will also tend to make the size of the state space more manageable. Even with these factors, however, this type of analysis is probably more appropriate for small to medium scale systems.

Future work includes plans to implement an analysis tool based on RT L. With such a tool we can further evaluate how the approach works for real systems. We also plan to extend RT L for use with multiprocessor computer systems. Although this extension would be fairly simple for discrete time systems, the analysis of continuous time multiprocessor systems still presents many challenging problems. The addition of language constructs such as prioritized and internal choice and the option to assign priorities to transitions will also add to the utility of RT L.

Another potentially fruitful area of research is the development of approximations to the reachability analysis described in this dissertation which are designed to reduce the complexity of the analysis. It may be possible, for example, to develop sufficient conditions for the absence of exception states in a system that do not require the analysis of the global state space. Also in this area, we expect to be able to identify sub-classes of real-time systems for which the complexity of the analysis can be significantly reduced. uch a class of systems is one for which only upper-bounds on execution time are necessary for analysis.

Finally, we believe RT L can serve as the formal basis for the development of a real-time programming language within which all implementable systems are verifiable.

4. References

[1] R. Alur and D. L. Dill, "Automata for Modeling Real-Time ystems," in Proceedings of ICALP 90: Automata, Languages, and Programming, M. .Paterson ed., Lecture Notes in Computer cience 443, pp. 322-335, pringer-Verlag, 1990.

[2] Rajeev Alur and Thomas A. Henzinger, "Logics and Models of Real Time: A urvey," in *Proceedings of the REX Workshop "Real-Time: Theory in Practice"*, June 1991.

[3] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill and L. J. Hwang, " ymbolic Model Checking: 10^{20} tates and Beyond," in *Proceedings of 5th Annual IEEE Symposium on Logic in Computer Science*, pp. 428-439, IEEE Computer ociety Press, June 4-7, 1990.

[4] J. Davies and . chneider, *An Introduction to Timed CSP*. Technical Report PRG-75, Oxford University Computing Laboratory, UK, August 1989.

[5] Andre N. Fredette, *A Generalized Approach to the Analysis of Real-Time Computer Systems*. Ph.D. Dissertation, North Carolina tate University, March 1993.

[6] Richard Gerber and Insup Lee, "A Layered Approach to Automating the Verification of Real-Time ystems," *IEEE Transactions on Software Engineering*, 18(9), eptember 1992, pp. 768-784.

[7] E. W. Giering and T. P. Baker, "Toward the Deterministic cheduling of Ada Tasks," in *Proceedings of the IEEE Real-Time Systems Symposium*, pp. 31-40, C Press, Los Alamitos, California, 1989.

[8] Wolfgang A. Halang and Alexander D. toyenko, *Constructing Predictable Real Time Systems*. Kluwer Academic Publishers, Boston, 1991.

[9] Thomas A. Henzinger, Zohar Manna and Amir Pnueli, "Timed Transition ystems," in *Proceedings of the REX Workshop "Real-Time: Theory in Practice"*, June 1991.

[10] Gerard J. Holzmann, Patrice Godefroid and Didier Pirottin, "Coverage Preserving Reduction trategies for Reachability Analysis," in *Proceedings of the 12th International Symposium on Protocol Specification, Testing and Verification*, June 22-25, 1992.

[11] Mark H. Klein and Thomas Ralya, *An Analysis of Input/Output Paradigms for Real-Time Systems*. Technical Report CMU/ EI-90-TR-19, oftware Engineering Institute, July 1990.

[12] John Lehoczky, Lui ha and Ye Ding, "The Rate Monotonic cheduling Algorithm: Exact Characterization and Average Case Behavior," in *Proceedings of the IEEE Real-Time Systems Symposium*, pp. 166-171, C Press, Los Alamitos, California, 1989.

[13] C. L. Liu and J. W. Layland, " cheduling Algorithms for Multiprogramming in a Hard Real Time Environment," *JACM*, 20(1), 1973, pp. 46-61.

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

[15] A. K. Mok, *Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment*. Ph.D. Thesis, MIT, May 1983.

[16] Xavier Nicollin and Joseph ifakis, "An Overview and ynthesis on Timed Process Algebras," in *Proceedings of the 1991 Computer-Aided Verification Conference*, 1991.

[17] Lui ha and John B Goodenough, "Real-Time cheduling Theory and Ada," *Computer*, April 1990, pp. 53-62.

[18] Lui ha, Ragunathan Rajkumar and John P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time ynchronization," *IEEE Transactions on Computers*, 39(9), eptember 1990, pp. 1175-1185.

[19] Alexander D. toyenko, V. Carl Hamacher and Richard C. Holt, "Analyzing Hard-Real-Time Programs for Guaranteed chedulability," *IEEE Transactions on Software Engineering*, 17(8), August 1991, pp. 737-749.

[20] Alexander D. toyenko and Thomas J. Marlowe, “Polynomial-Time Transformations and chedulability Analysis of Parallel Real-Time cheduling Algorithms,” *Real-Time Systems*, Vol. 4, No. 4, December 1992, pp. 307-329.