# Symbolic Schedulability Analysis of Real-time Systems *

Hee-Hwan Kwak, Insup Lee, and Anna Philippou
Department of Computer and Information Science
University of Pennsylvania, USA
{heekwak,annap}@saul.cis.upenn.edu, lee@cis.upenn.edu

Jin-Young Choi
Department of Computer Science and Engineering
Korea University, Korea
choi@formal.korea.ac.kr

Oleg Sokolsky
Computer Command and Control Company, USA
sokolsky@cccc.com

## Abstract

*We propose a unifying method for analysis of scheduling problems in real-time systems. The method is based on ACSR-VP, a real-time process algebra with value-passing capabilities. We use ACSR-VP to describe an instance of a scheduling problem as a process that has parameters of the problem as free variables. The specification is analyzed by means of a symbolic algorithm. The outcome of the analysis is a set of equations, a solution to which yields the values of the parameters that make the system schedulable. Equations are solved using integer programming or constraint logic programming. The paper presents specifications of two scheduling problems as examples.*

## 1. Introduction

The desire to automate or incorporate intelligent controllers into control systems has lead to rapid growth in the demand for real-time software systems. Moreover, these systems are becoming increasingly complex and require careful design analysis to ensure reliability before implementation. Recently, there has been much work on formal methods for the specification and analysis of real-time systems [8, 10]. Most of the work assumes that various real-time systems attributes, such as execution time, release time, priorities, etc., are fixed *a priori* and the goal is to determine whether a system with all these known attributes would meet required safety properties. One example of safety property is schedulability analysis; that is, to determine whether or not a given set of real-times tasks under a particular scheduling discipline can meet all of its timing constraints.

The pioneering work by Liu and Layland [16] derives schedulability conditions for rate-monotonic scheduling and earliest-deadline-first scheduling. Since then, much work on schedulability analysis has been done which includes various extensions of these results [11, 24, 22, 4, 23, 20, 17, 3]. Each of these extensions expands the applicability of schedulability analysis to real-time task models with different assumptions. In particular, there has been much advance in scheduling theory to address uncertain nature of timing attributes at the design phase of a real-time system. This problem is complicated because it is not sufficient to consider the worst case timing values for schedulability analysis. For example, scheduling anomalies can occur even when there is only one processor and jobs have variable execution times and are nonpreemptable. Also for preemptable jobs with one processor, scheduling anomalies can occur when jobs have arbitrary release times and share resources. These scheduling anomalies make the problem of validating a priority-driven system hard to perform. Clearly, exhaustive simulation or testing is not practical in general except for small systems of practical interest. There have been many different heuristics developed to solve some of these general schedulability analysis prob-

*This research was supported in part by NSF CCR-9415346, NSF CCR-9619910, AFOSR F49620-95-1-0508, AFOSR F49620-96-1-0204 (AASERT), ARO DAAG55-98-1-0393, and ONR N00014-97-1-0505 (MURI).

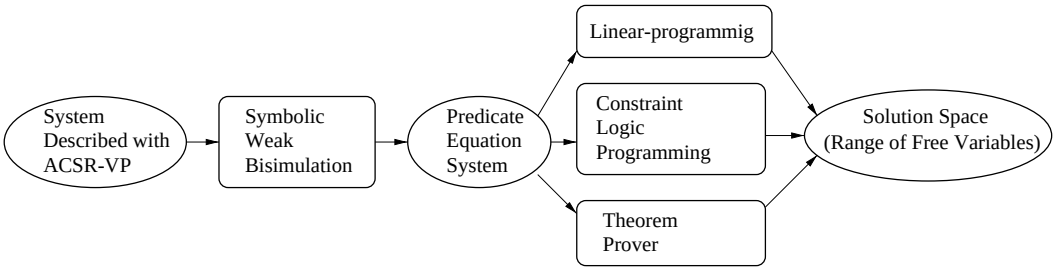



**Figure 1. Overview**

lems. However, each algorithm is problem specific and thus when a problem is modified, one has to develop new heuristics.

In this paper, we describe a framework that allows one to model scheduling analysis problems with variable release and execution times, relative timing constraints, precedence relations, dynamic priorities, multiprocessors etc. Our approach is based on ACSR-VP and symbolic bisimulation algorithm.

ACSR (Algebra of Communicating Shared Resources) [13], is a discrete real-time process algebra. ACSR has several notions, such as resources, static priorities, exceptions, and interrupts, which are essential in modeling real-time systems. ACSR-VP is an extension of ACSR with value-passing and parameterized processes to be able to model real-time systems with variable timing attributes and dynamic priorities. In addition, symbolic bisimulation for ACSR-VP has been defined. ACSR-VP without symbolic bisimulation has been applied to the simple schedulability analysis problem [5], by assuming that all parameters are ground, i.e., constants. However, it is not possible to use the technique described in [5] to solve the general schedulability analysis problem with unknown timing parameters.

Figure 1 shows the overall structure of our approach. We specify a real-time system with unknown timing or priority parameters in ACSR-VP. For the schedulability analysis of the specified system, we check symbolically whether or not it is bisimilar to a process idling forever. The result is a set of predicate equations, which can be solved using widely available linear-programming or constraint-programming techniques. The solution to the set of equations identifies, if exists, under what values of unknown parameters the system becomes schedulable.

The rest of the paper is organized as follows. Sections 2 and 3 overview the theory of the underlying formal method, ACSR-VP, and introduce symbolic bisimulation for ACSR-VP expressions. Section 4 gives specifications of two scheduling problems, namely the *period assignment problem* and the *start-time assignment problem*. Section 5 illustrates analysis of two instances of these problems. We conclude with a summary and an outline of future work in Section 6.

## 2. ACSR-VP

ACSR-VP extends the process algebra ACSR [13] by allowing values to be communicated along communication channels. In this section we present ACSR-VP concentrating on its value-passing capabilities. We refer to the above papers for additional information on ACSR.

We assume a set of variables $X$ ranged over by $x$, $y$, a set of values $V$ ranged over by $v$, and a set of labels $L$ ranged over by $c$, $d$. Moreover, we assume a set $Expr$ of expressions (which includes arithmetic expressions) and we let $BExpr \subset Expr$ be the subset containing boolean expressions. We let $e$ and $b$ range over $Expr$ and $BExpr$ respectively, and we write $\vec{z}$ for a tuple $z_1, \ldots z_n$ of syntactic entities.

ACSR-VP has two types of actions: instantaneous communication and timed resource access. Access to resources and communication channels is governed by priorities. A priority expression $p$ is attached to every communication event and resource access. A partial order on the set of events and actions, the preemption relation, allows one to model preemption of lower-priority activities by higher-priority ones.

Instantaneous actions, called *events*, provide the basic synchronization and communication primitives in the process algebra. An event is denoted as a pair $(i, e_p)$ representing execution of action $i$ at priority $e_p$, where $i$ ranges over $\tau$, the idle action, $c?x$, the input action, and $c!e$, the output action. We use $\mathcal{D}_E$ to denote the domain of events and let $\lambda$ range over events. We use $l(\lambda)$ and $\pi(\lambda)$ to represent the label and priority, respectively, of the event $\lambda$; e.g., $l((c!x, p)) = c!$ and $l((c?x, p)) = c?$. To model resource access, we assume that a system contains a finite set of serially-reusable resources drawn from some set $R$. An action that consumes one tick of time is drawn from the domain $\boldsymbol{P}(R \times Expr)$ with the restriction that each resource is represented at most once. For example the singleton action $\{(r, e_p)\}$ denotes the use of some resource $r \in R$ at priority

level $e_p$. The action $\emptyset$ represents idling for one unit of time, since no resource is consumed. We let $\mathcal{D}_R$ to denote the domain of timed actions with $A$, $B$, to range over $\mathcal{D}_R$. We define $\rho(A)$ to be the set of the resources used by action $A$, for example $\rho(\{(r_1, p_1), (r_2, p_2)\}) = \{r_1, r_2\}$. We also use $\pi_r(A)$ to denote the priority level of the use of the resource $r$ in the action $A$; e.g., $\pi_{r_1}(\{(r_1, p_1), (r_2, p_2)\}) = p_1$, and write $\pi_r(A) = 0$ if $r \notin \rho(A)$. The entire domain of actions is denoted by $\mathcal{D} = \mathcal{D}_R \cup \mathcal{D}_E$, and we let $\alpha$, $\beta$ range over $\mathcal{D}$. We let $P, Q$ range over ACSR-VP processes and we assume a set of process constants ranged over by $C$. The following grammar describes the syntax of ACSR-VP processes:

$$P \quad ::= \quad \text{NIL} \mid A : P \mid \lambda.P \mid P + P \mid P \| P \mid b \to P \mid P \backslash F \mid [P]_I \mid C(\vec{x}).$$

In the input-prefixed process $(c?x, e).P$ the occurrences of variable $x$ is bound. We write $\mathbf{fv}(P)$ for the set of free variables of $P$. Each agent constant $C$ has an associated definition $C(\vec{x}) \stackrel{\text{def}}{=} P$ where $\mathbf{fv}(P) \subseteq \vec{x}$ and $\vec{x}$ are pairwise distinct. We note that in an input prefix $(c?x, e).P$, $e$ should not contain the bound variable $x$, although $x$ may occur in $P$.

An informal explanation of ACSR-VP constructs follows: The process NIL represents the inactive process. There are two prefix operators, corresponding to the two types of actions. The first, $A : P$, executes a resource-consuming action during the first time unit and proceeds to process $P$. On the other hand $\lambda.P$, executes the instantaneous event $\lambda$ and proceeds to $P$. The process $P + Q$ represents a nondeterministic choice between the two summands. The process $P \| Q$ describes the concurrent composition of $P$ and $Q$: the component processes may proceed independently or interact with one another while executing instantaneous events, and they synchronize on timed actions. Process $b \to P$ represents the conditional process: it performs as $P$ if boolean expression $b$ evaluates to *true* and as NIL otherwise. In $P \backslash F$, where $F \subseteq L$, the scope of labels in $F$ is restricted to process $P$: components of $P$ may use these labels to interact with one another but not with $P$'s environment. The construct $[P]_I$, $I \subseteq R$, produces a process that reserves the use of resources in $I$ for itself, extending every action $A$ in $P$ with resources in $I - \rho(A)$ at priority 0.

The semantics of ACSR-VP processes may be provided as a labeled transition system, similarly to that of ACSR. It additionally makes use of the following ideas: Process $(c!e_1, e_2).P$ transmits the value obtained by evaluating expression $e_1$ along channel $c$, with priority the value of expression $e_2$, and then behaves like $P$. Process $(c?x, p).P$ receives a value $v$ from communication channel $c$ and then behaves like $P[v/x]$, that is $P$ with $v$ substituted for variable $x$. In the concurrent composition $(c?x, p_1).P_1 \| (c!v, p_2).P_2$, the two components of the parallel composition may synchronize with each other on channel $c$ resulting in the transmission of value $v$ and producing an event $(\tau, p_1 + p_2)$.

## 3. Semantics and Analysis

### 3.1. Unprioritized Symbolic Graphs with Assignment

Consider the simple ACSR-VP process $P \stackrel{\text{def}}{=} (in?x, 1).(out!x, 1).\text{NIL}$ that receives a value along channel $in$ and then outputs it on channel $out$, and where $x$ ranges over integers. According to traditional methods for providing semantic models for concurrent processes, using transition graphs, process $P$ in infinite branching, as it can engage in the transition $(in?n, 1)$ for every integer $n$. As a result standard techniques for analysis and verification cannot be applied to such processes.

Several approaches have been proposed to deal with this problem for various subclasses of value-passing processes [9, 15, 19, 12]. One of these advocates the use of *symbolic* semantics for providing finite representations of value-passing processes. This is achieved by taking a more conceptual view of value-passing than the one employed above. More specifically consider again process $P$. A description of its behavior can be sufficiently captured by exactly two actions: an input of an integer followed by the ouput of this integer. Based on this idea the notion of symbolic transition graphs [9] and transition graphs with assignment [15] were proposed and shown to capture a considerable class of processes.

In this section we present symbolic graphs with assignment for ACSR-VP processes. As it is not the intention of the paper to present in detail the process-calculus theory of this work, we only give an overview of the model and we refer to [12] for a complete discussion.

### 3.2. Symbolic Transition Graphs with Assignment

A *substitution*, or *assignment* is a function $\theta\colon X \to Expr$, such that $\theta(x) \neq x$ for a finite number of $x \in X$. Given a substitution $\theta$, *domain* of $\theta$ is the set of variables $D(\theta) = \{x \mid \theta(x) \neq x\}$. A substitution whose domain is empty is called the *identity substitution*, and is denoted by $\mathsf{Id}$. When $|D(\theta)| = 1$, we use $[\theta(x)/x]$ for the substitution $\theta$. Given two substitutions $\theta$ and $\sigma$, the *composition* of $\theta$ and $\sigma$ is the substitution denoted by $\theta;\sigma$ such that for every variable $x$, $\theta;\sigma(x) = \sigma(\theta(x))$. We often write $\theta\sigma$ for $\theta;\sigma$.

An SGA is a rooted directed graph where each node $n$ has an associated finite set of free variables $\mathbf{fv}(n)$ and each edge is labeled by a guarded action with assignment. Note that a node in SGA is a ACSR-VP term. Furthermore, we use $\epsilon$ to denote the empty action the purpose of which is explained later.

**Definition 3.1 (SGA)** A Symbolic Graph with Assignment (SGA) for ACSR-VP is a rooted directed graph where each node $n$ has an associated ACSR-VP term and each edge is labeled by boolean, action, assignment, $(b, \alpha, \theta)$ or by boolean, the empty action and assignment, $(b, \epsilon, \theta)$. □

Given an ACSR-VP process, the corresponding SGA can be generated using the rules in Figure 2. Note that the purpose of action $\epsilon$ is to decorate transitions that involve no action, but are nonetheless necessary for registering substitutions, see Rule (3). Transition $P \stackrel{b,\alpha,\theta}{\longmapsto} P'$ denotes that given the truth of boolean expression $b$, $P$ can evolve to $P'$ by performing action $\alpha$ and putting into effect the assignment $\theta$. The interpretation of these rules is straightforward and we explain them by an example: Consider the following process.

$$P(x) \stackrel{\text{def}}{=} (x < 2) \rightarrow (a!1, 1).P'(x + 1)$$
$$P'(x) \stackrel{\text{def}}{=} (x < 3) \rightarrow P(1)$$

Process $P(1)$ can output $a!1$ infinitely many times. Applying the rules above gives rise to the SGA in Figure 3(a).

One possible interpretation of our SGA can be given along the lines of programming languages: Process $P$ can be thought of as a procedure, so that $P(1)$ represents a call of $P$ with actual parameter 1 which is accepted by $P$ with formal parameter $x$ declared in $P$'s body. According to its definition, $P$ checks if $x < 2$ and if this boolean expression holds, $P$ outputs $a!1$ and calls process $P'$ with actual parameter $x + 1$. Process $P'$ then checks the validity of condition $x < 3$. If this is satisfied, process $P'$ calls $P$ with actual parameter 1.

Although introduction of the empty action $\epsilon$ appears useful in constructing SGA's from process terms it is possible to remove them by means of a fixpoint of a normalization function. For example, given the SGA in Figure 3(a), by applying the normalization process the NSGA in Figure 3(b) is obtained.

We say that an a normalized $SGA$ (NSGA) $(N, E, \mapsto)$ is *finite* if $|N|$ is finite. For the remainder of the paper we only consider finite NSGA's.

### 3.3. The prioritized Symbolic Transition System

We have illustrated how ACSR-VP processes can be given finite representations as SGA's via the symbolic transition relation $\mapsto$. However, this relation makes no arbitration between actions with respect to their priorities. To achieve this, we refine the relation $\mapsto$ to obtain the prioritized symbolic transition system $\mapsto_\pi$. This is based on the notion of *preemption* which incorporates our treatment of priority, and in particular on relation $\succ$, the *preemptive relation*, a transitive, irreflexive relation on actions [2]. Then for two actions $\alpha$ and $\beta$, $\alpha \succ \beta$ denotes that $\alpha$ preempts $\beta$, which implies that in any real-time system, if there is a choice between the two actions, $\alpha$ will always be executed. For example $(c?x, 2) \succ (c?x, 1)$ and $\{(r, 2)\} \succ \{(r, 0)\}$.

Extending the notion of preemption in the value-passing setting involves dealing with the presence of free variables in process descriptions. For example, given actions $\alpha = (c?x, y_1)$ and $\beta = (c?x, y_2)$, whether $\alpha \succ \beta$ or $\beta \succ \alpha$ depends on the values to which variables $y_1$ and $y_2$ are instantiated. This idea can easily be incorporated to yield the prioritized transition relation $\mapsto_\pi$. For the precise definition we refer the reader to [12]. We illustrate this with an example. Consider process $P$:

$$P(x) \stackrel{\text{def}}{=} (a?y, 1).P'(x, y)$$
$$P'(x, y) \stackrel{\text{def}}{=} (y \leq 1) \rightarrow (a!(x + y), y).NIL$$
$$+\ (y \leq 2) \rightarrow (a!(x + y), 2).NIL$$

The unprioritized NSGA for $P$ and its prioritized version, $Q$ are shown in Figure 4. Note that transition $P' \stackrel{y\leq 1,(a!(x+y),y),\mathsf{Id}}{\longmapsto} NIL$ is preempted by $P' \stackrel{(y\leq 2,(a!(x+y),2),\mathsf{Id}}{\longmapsto} NIL$ since whenever the former is enabled, the latter is also enabled with a higher priority (that is, whenever $y \leq 1$, $y \leq 2$ and $y < 2$).

### 3.4. Weak Bisimulation

Various methods have been proposed for the verification of concurrent processes. Central among them is observational equivalence that allows to compare an implementation with a specification of a given system. Observational equivalence is based on the idea that two equivalent systems exhibit the same behavior at their interfaces with the environment. This requirement was captured formally through the notion of *bisimulation* [18], a binary relation on the states of systems. Two states are bisimilar if for each single computational step of the one there exists an appropriate matching (multiple) step of the other, leading to bisimilar states.

In this setting, bisimulation for symbolic transition graphs is defined in terms of relations parametrized on boolean expressions, of the form $\simeq^b$, where $p \simeq^b q$ if and only if, for each interpretation satisfying boolean $b$, $p$ and $q$ are bisimilar in the traditional notion. In [12] the authors have proposed weak version of bisimulations for SGA's, that is observational equivalences that abstract away from internal system behavior (both for late and early semantics). Furthermore, algorithms were presented for computing these equivalences. Given two closed processes whose symbolic transition graphs are finite, the algorithm constructs a predicate equation system that corresponds to the most general condition for the two processes to be weakly bisimilar.

$$(1)\quad \frac{-}{\alpha.P \overset{true,\alpha,\mathsf{Id}}{\longmapsto} P} \qquad (2)\quad \frac{C(\vec{v}) \overset{b,\epsilon,\theta}{\longmapsto} C}{\alpha.C(\vec{v}) \overset{b,\alpha,\theta}{\longmapsto} C}$$

$$(3)\quad \frac{-}{C(\vec{v}) \overset{true,\epsilon,\vec{x}:=\vec{v}}{\longmapsto} C}\quad C(\vec{x}) \overset{\text{def}}{=} P \qquad (4)\quad \frac{P \overset{b,\alpha,\theta}{\longmapsto} P'}{\mathsf{C} \overset{b,\alpha,\theta}{\longmapsto} P'}\quad \mathsf{C}(\vec{x}) \overset{\text{def}}{=} P$$

$$(5)\quad \frac{P \overset{b,\alpha,\theta}{\longmapsto} P'}{b' \rightarrow P \overset{b\wedge b',\alpha,\theta}{\longmapsto} P'}$$

$$(6)\quad \frac{P \overset{b,\alpha,\theta}{\longmapsto} P'}{P + Q \overset{b,\alpha,\theta}{\longmapsto} P'} \qquad (7)\quad \frac{P \overset{b,\alpha,\theta}{\longmapsto} P'}{Q + P \overset{b,\alpha,\theta}{\longmapsto} P'}$$

$$(8)\quad \frac{P \overset{b,\lambda,\theta}{\longmapsto} P'}{P\backslash F \overset{b,\lambda,\theta}{\longmapsto} P'\backslash F}\quad \begin{array}{l}\tau \notin F\\ l(\lambda) \notin F\end{array} \qquad (9)\quad \frac{P \overset{b,A,\theta}{\longmapsto} P'}{P\backslash F \overset{b,A,\theta}{\longmapsto} P'\backslash F}$$

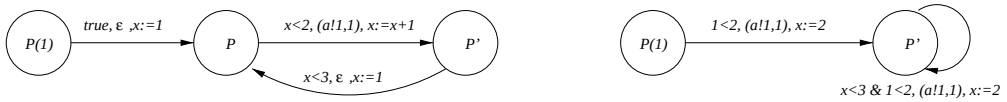
$$(10)\quad \frac{P \overset{b,\lambda,\theta}{\longmapsto} P'}{[P]_I \overset{b,\lambda,\theta}{\longmapsto} [P']_I} \qquad (11)\quad \frac{P \overset{b,A_1,\theta}{\longmapsto} P'}{[P]_I \overset{b,A_1\cup A_2,\theta}{\longmapsto} [P']_I}\quad A_2 = \{(r,0) \mid r \in I - \rho(A_1)\}$$

$$(12)\quad \frac{P \overset{b_1,A_1,\theta_1}{\longmapsto} P' \qquad Q \overset{b_2,A_2,\theta_2}{\longmapsto} Q'}{P\|Q \overset{b_1\wedge b_2,A_1\cup A_2,\theta_1\cup\theta_2}{\longmapsto} P'\|Q'}\quad \rho(A_1)\cap\rho(A_2) = \emptyset$$

$$(13)\quad \frac{P \overset{b,\alpha,\vec{x}:=\vec{e}}{\longmapsto} P'}{P\|Q \overset{b,\alpha,\vec{x},\vec{y}:=\vec{e},\vec{y}}{\longmapsto} P'\|Q}\quad \mathbf{fv}(Q) = \{\vec{y}\} \qquad (14)\quad \frac{P \overset{b,\alpha,\vec{x}:=\vec{e}}{\longmapsto} P'}{Q\|P \overset{b,\alpha,\vec{x},\vec{y}:=\vec{e},\vec{y}}{\longmapsto} Q\|P'}\quad \mathbf{fv}(Q) = \{\vec{y}\}$$

$$(15)\quad \frac{P \overset{b_1,(c?z,e_1),\theta_1}{\longmapsto} P' \qquad Q \overset{b_2,(c!e_2,e_3),\theta_2}{\longmapsto} Q'}{P\|Q \overset{b_1\wedge b_2,(\tau,e_1+e_3),(\theta_1\cup\theta_2);\{z:=e_2\}}{\longmapsto} P'\|Q'}\quad z \notin \mathbf{fv}(P)\cup\mathbf{fv}(Q)$$

**Figure 2. Rules for constructing Symbolic Graphs with Assignment**




(a) (b)

**Figure 3. SGA (a) and Normalized SGA (b)**

Recall process $P(x)$ from section 3.3. Furthermore, consider the following process with bound variable $x'$:

$$R(x') \overset{\text{def}}{=} (a?y', 1).R'(x', y')$$
$$R'(x', y') \overset{\text{def}}{=} (y' \leq 2) \rightarrow (a!(x' + y' + 1), 2).NIL$$

The prioritized NSGA for $R$ is similar to $Q$ with the exception that after receiving a value via channel $a$, $R$ outputs value $x' + y' + 1$. Applying the symbolic bisimulation algorithm for processes $P$ and $R$, we obtain the following predicate equation system.

$$\begin{aligned} X_{00}(x, x') &= \forall z X_{11}(z, x, x') \\ X_{11}(z, x, x') &= z \leq 2 \rightarrow z \leq 2 \wedge x + z = x' + z + 1 \\ &\quad \wedge z \leq 2 \rightarrow z \leq 2 \wedge x' + z + 1 = x + z \end{aligned}$$

This equation system can easily be reduced to the equation $X_{00}(x, x') \equiv x = x' + 1$, which allows us to conclude that $P(x)$ and $R(x')$ are bisimilar if and only if $x = x' + 1$ holds. In general, since we are dealing with a domain of linear expressions, predicate equations obtained from the bisimulation algorithm can be solved using integer programming techniques [21].

## 4. Real-time System Scheduling

In this section, we show how several problems of real-time system scheduling can be specified and analyzed using ACSR-VP. According to [25], real-time scheduling problems can be categorized into the following three groups: priority assignment, execution synchronization, and schedu-


**Figure 4. SGA of $P$ and $Q$**

lability analysis problems. The priority assignment problem requires assigning priorities to jobs so that the system schedulability is maximized. The execution synchronization problem is the problem of deciding when and how to release jobs so that the precedence constraints are satisfied and the system schedulability, as well as other performance concerns, are optimized. Schedulability analysis problem is the problem of verifying that a system is schedulable, given a certain priority assignment method and execution synchronization method.

Classic examples of solutions to these problems include the rate-monotonic priority assignment problem on a single processor [16]. It uses static priority assignment, where the priority of each job is assigned in the inverse order of period; the job with shortest period has the highest priority. Deadline-monotonic priority assignment was proposed by [14], where the system has jobs with arbitrary relative deadlines. Dynamic priority assignment problem has been addressed by earliest-deadline first algorithms.

The same groups of problems can be considered in the presence of end-to-end scheduling constraints. Gerber *et al.* [6] proposed the method to guarantee a system's end-to-end requirements of real-time systems. In [27], Tindell *et al.* attempted to compute upper bounds on the end-to-end response time. They also proposed priority assignment in distributed system where jobs have end-to-end deadlines. In [1], Bettati studied the problem of scheduling a set of jobs with arbitrary release times and end-to-end deadlines.

We propose to address real-time scheduling problems by means of analysis based on ACSR-VP. In this approach, a specific instance of a problem is specified as an ACSR-VP expression and symbolically analyzed. In this paper, we illustrate our approach by giving general solutions to two scheduling problems. The first problem is the *period assignment problem* (Section 4.1). It can be viewed as an variant of schedulability analysis problem. The second problem is the *start-time assignment problem* (Section 4.2). It is a version of the execution synchronization problem with end-to-end scheduling. Our methods of solving these problems are optimal in the sense that if the method can not find the period or start-time assignment, then the system can not be scheduled for any assignment of periods (respectively, start times).

### 4.1. Shortest Job First Scheduling

We define the *period assignment problem* as follows. Consider a set of $n$ preemptable periodic jobs sharing a processor. We apply the shortest job first scheduling algorithm to schedule these $n$ jobs. Each job is characterized by two parameters: execution time and period. We assume that the deadline for each task is the same as its period. Execution times $E_1, \ldots, E_n$, and periods $P_1, \ldots, P_{k-1}, P_{k+1}, \ldots, P_n$ are known. We have to determine the period of the $k^{th}$ job.

We model each job in the set as the following ACSR-VP process:

$$
\begin{aligned}
&Job_i(e_i, p_i, s_i, t_i) \stackrel{\text{def}}{=} \\
&\quad (s_i < e_i) \wedge (t_i < p_i) \rightarrow \\
&\qquad \{(cpu, MAX - e_i)\} : Job_i(e_i, p_i, s_i + 1, t_i + 1) \\
&\qquad + \emptyset : Job_i(e_i, p_i, s_i, t_i + 1) \\
&\quad + (s_i = e_i) \wedge (t_i \leq p_i) \rightarrow Wait(e_i, p_i, t_i) \\
&Wait_i(e_i, p_i, t_i) \stackrel{\text{def}}{=} \\
&\quad (t_i < p_i) \rightarrow \emptyset : Wait_i(e_i, p_i, t_i + 1) \\
&\quad + (t_i = p_i) \rightarrow Job_i(e_i, p_i, 0, 0)
\end{aligned}
$$

Process $Job_i(e_i, p_i, s_i, t_i)$ represents a job with execution time $e_i$ and period $p_i$, which has accumulated $s_i$ units of processing time in the current period. The current period has started $t_i$ time units ago. As long as the job is not finished ($s_i < e_i$) and the current period is not over ($t_i < p_i$), the job competed with other job for access to the CPU, which is represented by resource $cpu$. The priority of the job is $MAX - e_i$, where $MAX = \max(E_i)$. That is, the shortest job has the highest priority. If the job is preempted by a higher-priority process, it idles in that time unit. alternatively, if the job has completed ($s_i = e_i$), it turns into process $Wait_i(e_i, p_i, t_i)$, which idles until the end of the current period and restarts itself.

Assuming that, initially, all jobs are started at time 0, we can capture behavior of the whole system as

$$
\begin{aligned}
SJF(prd) \stackrel{\text{def}}{=} [&Job_1(E_1, P_1, 0, 0) \parallel \cdots \parallel Job_k(E_k, prd, 0, 0) \\
&\parallel \cdots \parallel Job_n(E_n, P_n, 0, 0) \,]_{\{cpu\}}.
\end{aligned}
$$

The free variable $prd$ represents the period of $Job_k$, which has to be determined. Notice that *Closure* operator is used in *SJF(prd)* process to prevent resource *cpu* from being idle when there is a job waiting to be executed.

Note that process $Job_i(e_i, p_i, s_i, t_i)$ will deadlock if it has not finished executing by the end of its period. The

composite process $SJF(prd)$ will also deadlock when one of its constituent processes deadlocks. We can use this property of the specification to determine admissible range for values of $prd$. We can apply the symbolic weak bisimulation algorithm to analyze the equivalence of $SJF(prd)$ and process $\emptyset^{\infty}$, which never deadlocks. This gives us a set of conditions on $prd$. These conditions, when satisfied, will guarantee that $SJF(prd)$ never deadlocks, that is, that no job misses its deadline.

### 4.2. Scheduling with Constraints

In this section, we use ACSR-VP to specify an end-to-end scheduling problem introduced in [7]. We are given a set of jobs running on a single processor, and the order of execution of jobs is fixed. The system is non-preemptable; that is, a job always finishes before the next one is started. The order of job execution is assumed to be fixed and known. Jobs have variable execution times denoted, for $i^{th}$ job, $[e_i^-, e_i^+]$. Additionally, there is a set of constraints on absolute and relative times of initiation and completion of jobs. The goal is to statically determine the range of start times for each job so that there are no conflicts between the jobs and all constraints are satisfied. We call this problem the *start-time assignment problem*.

Constraints that we consider in this problem are linear inequalities over start times and execution times of the jobs. Examples of constraints are "Job $a$ should start no earlier than time $t$ ($s_a \geq t$);" "Job $a$ should be finished before time $t$ ($s_a + e_a \leq t$);" "Job $a$ should be finished within $t$ time units after job $b$ finishes ($s_b + e_b \leq s_a + e_a + t$)." A concrete example of this problem is shown in Section 5.2.

Deriving the start-time assignments for arbitrary constraints is an NP-hard problem [7]. The complexity of a brute-force search is exponential with respect to the bounds of execution times and the number of jobs [26], making this approach impractical for most real-life systems. As in [7], we limit ourselves to constraints with at most two variables. A natural approach for solving this problem is to employ linear programming. However, applying linear-programming techniques to start-time assignment problem directly requires us to encode the scheduling algorithm into the linear constraints.

Our method lets us circumvent this problem. We construct an ACSR-VP specification of the set of jobs together with their constraints. The symbolic semantics of ACSR-VP allows us to produce a predicate equation system that can be solved by well-known techniques such as linear programming or constraint logic programming.

Each $Job_i, i \in \{1, \ldots, n\}$, is specified as follows:

$$
\begin{aligned}
&Job_i(t_i, s_i) \stackrel{\text{def}}{=} \\
&\quad (t_i < s_i) \rightarrow \emptyset : Job_i(t_i + 1, s_i) \\
&\quad + (Start!, n + 1 - i).(t_i = s_i \rightarrow Job_i'(0, t_i, s_i)) \\
&Job_i'(r_i, t_i, s_i) \stackrel{\text{def}}{=} \\
&\quad (r_i < e_i^-) \rightarrow \{(cpu, 1)\} : Job_i'(r_i + 1, t_i + 1, s_i) \\
&\quad + (r_i = e_i^-) \rightarrow Job_i''(0, t_i, s_i) \\
&Job_i''(e_i, t_i, s_i) \stackrel{\text{def}}{=} \\
&\quad (e_i < e_i^+ - e_i^-) \rightarrow \{(cpu, 1)\} : Job_i''(e_i + 1, t_i + 1, s_i) \\
&\quad + (e_i \leq e_i^+ - e_i^-) \rightarrow (Finished!, 1).IDLE
\end{aligned}
$$

The job process uses signals $Start$ and $Finish$ to communicate with the constraint process discussed below. $Job_i$ represents behavior of the job before its start time $s_i$ comes. At that moment, $Job_i$ sends an event $Start$ to synchronize with $Constraint_i$, and becomes $Job_i'$. $Job_i'$ represents the mandatory execution time of the job, that is, until the lower bound on its execution time $e_i^-$ arrives. Then $Job_i'$ becomes $Job_i''$, which continually offers the choice between completing the job by sending event $Finished$ to the corresponding constraint, or continuing the execution until the upper bound of execution time ($e_i^+$) is reached. After the execution is completed, the job becomes idle, represented by ACSR-VP process $IDLE \stackrel{\text{def}}{=} \emptyset : IDLE$.

Constraints are represented by a collection of processes $Constraint_i$, $i \in \{1, \ldots, n\}$. $Constraint_i$ models the state of the system that $Job_{i-1}$ is finished but $Job_i$ is not started yet. Upon event $Start$, it becomes $Constraint_i'$, which models the state of the system that $Job_i$ is in execution. Event $Finished$ turns $Constraint_i'$ into $Constraint_i''$, which represents the state of the system after $Job_i$ is finished. $Constraint_i''$ checks if the timing conditions related to $Job_i$ are satisfied and deadlocks if the condition fails.

$$
\begin{aligned}
Constraint(t) &\stackrel{\text{def}}{=} Constraint_1(t) \\
Constraint_1(t) &\stackrel{\text{def}}{=} (Start?, 1).Constraint_1'(t) \\
&\quad + \emptyset : Constraint_1(t + 1) \\
Constraint_1'(t) &\stackrel{\text{def}}{=} (Finished?, 1).Constraint_1''(t) \\
&\quad + \emptyset : Constraint_1'(t + 1) \\
Constraint_1''(t) &\stackrel{\text{def}}{=} b_1 \rightarrow \emptyset : Constraint_2(t, 1) \\
&\quad \vdots \\
Constraint_n(t, \vec{v}) &\stackrel{\text{def}}{=} (Start?, 1).Constraint_n'(t, \vec{v}) \\
&\quad + \emptyset : Constraint_n(t + 1, \vec{v}) \\
Constraint_n'(t, \vec{v}) &\stackrel{\text{def}}{=} (Finished?, 1).Constraint_n''(t, \vec{v}) \\
&\quad + \emptyset : Constraint_n'(t + 1, \vec{v}) \\
Constraint_n''(t, \vec{v}) &\stackrel{\text{def}}{=} b_n \rightarrow \emptyset : IDLE
\end{aligned}
$$

To perform the analysis of the problem, we compose the job processes together with the constraint process:

$$
\begin{aligned}
&System(Start_1, \cdots, Start_2) \stackrel{\text{def}}{=} \\
&\quad (\, Constraint(0) \parallel Job_1(0, Start_1) \parallel \cdots \\
&\quad \parallel Job_n(0, Start_n)\,) \backslash \{Start, Finished\}
\end{aligned}
$$

The resulting system will deadlock if the constraints are not satisfied. Again, we can apply the symbolic algorithm to

this ACSR-VP process to obtain the set of predicate equations. The solution to these equations will give us the range of admissible start times for each of the jobs.

The proposed technique gives, to our knowledge, the first static algorithm for the problem. The method proposed in [7] contains a static component that analyzes constraints, and a dynamic component. The static component produces a calendar, a set of functions that is used by the dynamic component to compute start times.

Moreover, ACSR-VP specification allows us to remove the requirement of a fixed total order on job execution. A slightly more complex specification of the constraints can be constructed that will use the partial order induced by the constraints instead. The resulting specification still yields to analysis in many practical cases.

## 5. Examples

In this section, we present results of analysis of the two scheduling problems outlined in Section 4. To make examples manageable, we consider small instances of the problems.

### 5.1. Shortest Job First Scheduling

Consider the system containing two jobs. The first job has execution time 1 and period 2. The execution time of the second job is 2 and the admissible range of its period has to be determined by the algorithm. Job 1, therefore, has the higher priority of the two. The ACSR-VP specifications of the jobs are as follows:

$$
\begin{aligned}
Job_1 &\stackrel{\text{def}}{=} \{(cpu,2)\} : \emptyset : Job_1 \\
Job_2(t,p) &\stackrel{\text{def}}{=} (t<p) \rightarrow (\{(cpu,1)\} : Job_2'(t+1,p) \\
&\qquad + \emptyset : Job_2(t+1,p)) \\
Job_2'(t,p) &\stackrel{\text{def}}{=} (t<p) \rightarrow (\{(cpu,1)\} : Job_2''(t+1,p) \\
&\qquad + \emptyset : Job_2'(t+1,p)) \\
Job_2''(t,p) &\stackrel{\text{def}}{=} (t=p) \rightarrow Job_2(0,p) \\
&\quad + \ (t<p) \rightarrow \emptyset : Job_2''(t+1,p)
\end{aligned}
$$

We simplified the specification for $Job_1$, since it has the highest priority in the system and will never need idling. $Job_2(t,p)$ represent the job that has not have access to the processor in the current period, $Job_2'(t,p)$ is the same job after it has used the processor for one time units, and $Job_2''(t,p)$ represent its idling state.

The whole system is specified as $SJF(prd) \stackrel{\text{def}}{=} [Job_1 \| Job_2(0,prd)]_{\{cpu\}}$. When we analyze this specification, the symbolic algorithm produces the following set of equations:

$$
\begin{aligned}
X_{00}(prd) &= 0 < prd \wedge X_{12}(1,prd) \\
X_{12}(t,p) &= t < p \wedge X_{01}(t+1,p) \\
X_{01}(t,p) &= t < p \wedge X_{11}(t+1,p) \\
X_{11}(t,p) &= t < p \wedge X_{03}(t+1,p) \\
X_{03}(t,p) &= ((t<p \wedge X_{13}(t+1,p)) \\
&\qquad \vee (t=p \wedge 0<p \wedge X_{12}(1,p))) \\
&\qquad \wedge (t<p \rightarrow X_{13}(t+1,p)) \\
&\qquad \wedge ((t=p \wedge 0<p) \rightarrow X_{12}(1,p)) \\
X_{13}(t,p) &= ((t=p \wedge 0<p \wedge X_{01}(1,p)) \\
&\qquad \vee (t<p \wedge X_{03}(t+1,p))) \\
&\qquad \wedge ((t=p \wedge 0<p) \rightarrow X_{01}(1,p)) \\
&\qquad \wedge (t<p \rightarrow X_{03}(t+1,p))
\end{aligned}
$$

When the predicate variable $X_{00}$ is true, we can conclude that *SJF(prd)* is schedulable. Since all the boolean expression in the predicate equations are linear, we can use a constraint solver or a constraint logic programming tool to solve the predicate equations. We used SICStus tool to conclude that the system is schedulable when $prd \geq 4$.

### 5.2. Scheduling with Constraints

Consider a system with two jobs, $Job_1$ and $Job_2$, shown in Figure 5. Let $e_1$ and $e_2$ be respective execution times of $Job_1$ and $Job_2$, and $s_1$ and $s_2$, their start times. The bounds on execution times of $Job_1$ and $Job_2$ are $5 \leq e_1 \leq 7$ and $3 \leq e_2 \leq 4$. There are four timing constraints:

1. $Job_1$ should be finished before or at 12 ($s_1 + e_1 \leq 12$).
2. $Job_2$ should be finished within 10 time units after $Job_1$ finishes ($s_2 + e_2 \leq s_1 + e_1 + 10$).
3. $Job_2$ should be finished before or at 25 ($s_2 + e_2 \leq 25$).
4. $Job_2$ should start after or at 14 ($s_2 \geq 14$).

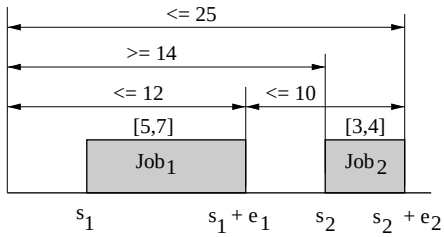



**Figure 5. An Instance of the Start-time Assignment Problem**

Since the order of the tasks is clear from the constraints and the constraints themselves are simple enough, we chose to incorporate them directly into the ACSR-VP specifications of the jobs instead. The resulting specification is as follows:

$$
\begin{aligned}
&System(t,s_1,s_2) \stackrel{\text{def}}{=} \\
&\quad (t \leq 12-7) \rightarrow
\end{aligned}
$$

$$
\begin{aligned}
&\quad (\,(t < s1) \rightarrow \emptyset : System(t+1, s_1, s_2) \\
&\quad +(t = s_1) \rightarrow \{(cpu, 1)\} : Job_1(0, t+5, s_2)) \\
&Job_1(e, t, s_2) \stackrel{\text{def}}{=} \\
&\quad (e < 7-5) \rightarrow \{(cpu, 1)\} : Job_1(e+1, t+1, s_2) \\
&\quad + (e \leq 7-5) \rightarrow System'(t, 14, s_2) \\
&System'(f_1, t, s_2) \stackrel{\text{def}}{=} \\
&\quad (f_1 \leq 12 \wedge t \leq 25-4) \rightarrow \\
&\qquad (\,(t < s_2) \rightarrow \emptyset : System'(f_1, t+1, s_2) \\
&\qquad +(t = s_2) \rightarrow \{(cpu, 1)\} : Job_2(f_1, 0, t+3, s_2)) \\
&Job_2(f_1, e, t, s_2) \stackrel{\text{def}}{=} \\
&\quad (e < 4-3) \rightarrow \{(cpu, 1)\} : Job_2(f_1, e+1, t+1, s_2) \\
&\quad + (e \leq 4-3) \rightarrow End(f_1, t, s_2) \\
&End(f_1, t, s_2) \stackrel{\text{def}}{=} \\
&\quad (f_1 \leq 12 \wedge t - f_1 \leq 10 \wedge s_2 \geq 14) \rightarrow IDLE
\end{aligned}
$$

Process $System$ represents the system, in which the jobs are scheduled to start at times $s_1$ and $s_2$ but has not been started yet. The process evolves into $Job_1$ at time $s_1$. After $Job_1$ completes at time $f_1$, process $System'$ is idle until it's time to start $Job_2$. Finally, the process $End$ checks that all end-to-end constraints are satisfied. The system is started at time 0 and will be deadlock-free for the admissible values of $s_1$ and $s_2$. The symbolic weak bisimulation algorithm generates the predicate equations system shown below. The equations were solved with the help of SICStus tool. The generated solutions are listed in Table 1.

$$
\begin{aligned}
&X_0(t, s_1, s_2) \quad = \\
&\quad (t \leq 5 \wedge t < s_1) \rightarrow X_1(t+1, s_1, s_2) \\
&\quad \wedge \ (t < 5 \wedge t = s_1) \rightarrow X_2(0, t+5, s_2) \\
&\quad \wedge \ ((t \leq 5 \wedge t < s_1 \wedge X_1(t+1, s_1, s_2)) \\
&\qquad \vee (t < 5 \wedge t = s_1 \wedge X_2(0, t+5, s_2))) \\
&X_1(t, s_1, s_2) \quad = \\
&\quad (t \leq 5 \wedge t < s_1) \rightarrow X_1(t+1, s_1, s_2) \\
&\quad \wedge \ (t \leq 5 \wedge t = s_1) \rightarrow X_2(0, t+5, s_2) \\
&\quad \wedge \ ((t \leq 5 \wedge t < s_1 \wedge X_1(t+1, s_1, s_2)) \\
&\qquad \vee (t \leq 5 \wedge t = s_1 \wedge X_2(0, t+5, s_2))) \\
&X_2(e, s_1, s_2) \quad = \\
&\quad (e < 2) \rightarrow X_2(e+1, t+1, s_2) \\
&\quad \wedge \ (e \leq 2 \wedge t \leq 12 \wedge 14 < s_2) \rightarrow X_3(t, 15, s_2) \\
&\quad \wedge \ (e \leq 2 \wedge t \leq 12 \wedge 14 = s_2) \rightarrow X_4(t, 0, 17, s_2) \\
&\quad \wedge \ (\,(e < 2 \wedge X_2(e+1, t+1, s_2)) \\
&\qquad \vee (e \leq 2 \wedge t \leq 12 \wedge 14 < s_2 \wedge X_3(t, 15, s_2)) \\
&\qquad \vee (e \leq 2 \wedge t \leq 12 \wedge 14 = s_2 \wedge X_4(t, 0, 17, s_2))) \\
&X_3(f_1, t, s_2) \quad = \\
&\quad (f_1 \leq 12 \wedge t \leq 21 \wedge t < s_2) \rightarrow X_3(f_1, t+1, s_2) \\
&\quad \wedge \ (f_1 \leq 12 \wedge t \leq 21 \wedge t = s_2) \rightarrow X_4(f_1, 0, t+3, s_2) \\
&\quad \wedge \ (\,(f_1 \leq 12 \wedge t \leq 21 \wedge t < s_2 \wedge X_3(f_1, t+1, s_2)) \\
&\qquad \vee (f_1 \leq 12 \wedge t \leq 21 \wedge t = s_2 \wedge X_4(f_1, 0, t+3, s_2))) \\
&X_4(f_1, e, t, s_2) = \\
&\quad (e < 1 \rightarrow X_4(f_1, e+1, t+1, s_2) \\
&\quad \wedge \ (e \leq 1 \wedge f_1 \leq 12 \wedge t - f_1 \leq 10 \wedge s_2 \geq 14) \rightarrow X_5 \\
&\quad \wedge \ ((e < 1 \wedge X_4(f_1, e+1, t+1, s_2)) \\
&\qquad \vee (e \leq 1 \wedge f_1 \leq 12 \wedge t - f_1 \leq 10 \wedge s_2 \geq 14 \wedge X_5)) \\
&X_5 = true
\end{aligned}
$$

| $s_1$ | 3 | 4 | 4 | 5 | 5 | 5 |
|---|---|---|---|---|---|---|
| $s_2$ | 14 | 14 | 15 | 14 | 15 | 16 |

**Table 1. Solutions of Start-time Assignment Problem**

## 6. Conclusions

We have described a formal framework for the specification and analysis of real-time scheduling problems. Our framework is based on ACSR-VP and symbolic bisimulation. The major advantage of our approach is that the same framework can be used for scheduling problems with different assumptions and parameters. In other scheduling-theory based approaches, new analysis algorithms need to be devised for problems with different assumptions since applicability of a particular algorithm is limited to specific system characteristics.

We believe that ACSR-VP is expressive enough to model any real-time system. In particular, our method is appropriate to model many complex real-time systems and can be used to solve the *priority assignment problem*, *execution synchronization problem*, and *schedulability analysis problem*. It is, in most cases, efficient in the sense that resulting predicate equation systems can be solved with existing techniques such as linear programming or constraint programming, which can solving linear equation constraints efficiently in practice [21].

The novel aspect of our approach is that schedulability of real-time systems can be described formally and analyzed automatically, all within a process-algebraic framework. It has often been noted that scheduling work is not adequately integrated with other aspects of real-time system development [3]. Our work is a step toward such an integration, which helps to meet our goal of making the timed process algebra ACSR a useful formalism for supporting the development of reliable real-time systems. Our approach allows the same specification to be subjected to the analysis of both schedulability and functional correctness.

There are several issues that we need to address to make our approach practical. We showed that resulted predicate equation systems can be solved with constraint logic programming or linear programming. We plan to investigate when resulting equation systems are simple or difficult to solve. In general, we may have to use a more powerful technique such as theorem prover; however, it is not clear whether any reasonable real-time system scheduling problem can result in such a complex equation system. We are currently augmenting PARAGON, the toolset for ACSR, to support the full syntax of ACSR-VP directly and imple-

menting a symbolic bisimulation algorithm. This toolset will allow us to experimentally evaluate the effectiveness of our approach with a number of large scale real-time systems.

## References


[1] R. Bettati. *End-to-end Scheduling to Meet Deadlines in Distributed Systems*. PhD thesis, University of Illinois at Urbana-Champaign, 1994.

[2] P. Brémond-Grégoire, I. Lee, and R. Gerber. ACSR: An Algebra of Communicating Shared Resources with Dense Time and Priorities. In *Proc. of CONCUR '93*, 1993.

[3] A. Burns. Preemptive priority-based scheduling: An appropriate engineering approach. In S. H. Song, editor, *Advances in Real-Time Systems*, chapter 10, pages 225–248. Prentice Hall, 1995.

[4] M. Chen and K. Lin. Dynamic Priority Ceilings: A Concurrency Control Protocol for Real-Time Systems. *Real-Time Systems*, 2(4):325–346, 1990.

[5] J.-Y. Choi, I. Lee, and H.-L. Xie. The Specification and Schedulability Analysis of Real-Time Systems using ACSR. In *Proc. of IEEE Real-Time Systems Symposium*, December 1995.

[6] R. Gerber, D. Kang, S. Hong, and M. Saksena. End-to-End Design of Real-Time Systems. In D. Mandrioli and C. Heitmeyer, editors, *Formal Methods in Real-Time Computing*. John Wiley & Sons, 1996.

[7] R. Gerber, W. Pugh, and M. Saksena. Parametric Dispatching of Hard Real-Time Tasks. *IEEE Transactions on Computers*, 44(3), March 1995.

[8] C. Heitmeyer and D. Mandrioli. *Formal Methods for Real-Time Computing*. Jonh Wiley and Sons, 1996.

[9] M. Hennessy and H. Lin. Symbolic bisimulations. *Theoretical Computer Science*, 138:353–389, 1995.

[10] M. Joseph. *Real-Time Systems: Specification, Verification and Analysis*. Prentice Hall Intl., 1996.

[11] M. Joseph and P. Pandya. Finding Response Times in a Real-Time System. *Computer Journal*, 29(5):390–395, 1986.

[12] H. Kwak, J. Choi, I. Lee, and A. Philippou. Symbolic weak bisimulation for value-passing calculi. *Technical Report, MS-CIS-98-22, Department of Computer and Information Science, University of Pennsylvania*, 1998.

[13] I. Lee, P. Brémond-Grégoire, and R. Gerber. A Process Algebraic Approach to the Specification and Analysis of Resource-Bound Real-Time Systems. *Proceedings of the IEEE*, pages 158–171, Jan 1994.

[14] J. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, pages 2:237–250, 1982.

[15] H. Lin. Symbolic graphs with assignment. In U.Montanari and V.Sassone, editors, *Proceedings CONCUR 96*, volume 1119 of *Lecture Notes in Computer Science*, pages 50–65. Springer-Verlag, 1996.

[16] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multi-programming in A Hard-Real-Time Environment. *Journal of the Association for Computing Machinery*, 20(1):46 – 61, January 1973.

[17] J. W. S. Liu and R. Ha. Efficient methods of validating timing constraints. In S. H. Song, editor, *Advances in Real-Time Systems*, chapter 9, pages 199–233. Prentice Hall, 1995.

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

[19] P. Paczkowski. Characterizing bisimilarity of value-passing parameterised processes. In *Proceedings of the Infinity Workshop on Verification of Infinite State Systems*, pages 47–55, 1996.

[20] R. Rajikumar, L. Sha, and J. Lehoczky. Real-Time Synchronization Protocols for Multiprocessors. In *Proc. of IEEE Real-Time Systems Symposium*, pages 259–272, 1989.

[21] R. Saigal. *Linear Programming : A Modern Integrated Analysis*. Kluwer Academic Publishers, 1995.

[22] L. Sha, R. Rajkumar, and J. Lehoczky. Priority Inheritance Protocols: An Approach to Real-time Synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, September 1990.

[23] L. Sha, R. Rajkumar, J. Lehoczky, and K. Ramamritham. Mode change Protocols for Priority Driven Preemptive Scheduling. *Real-Time Systems: The International Journal of Time Critical Computing Systems*, 1(3), December 1989.

[24] B. Sprunt, L. Sha, and J. Lehoczky. Aperiodic Task Scheduling for Hard-Real-Time Systems. *Real-Time Systems: The International Journal of Time Critical Computing Systems*, 1(1):27–60, 1989.

[25] J. Sun. *Fixed-priority End-to-end Scheduling in Distributed Real-time Systems*. PhD thesis, University of Illinois at Urbana-Champaign, 1997.

[26] J. Sun and J. W. Liu. Bounding Completion Times of Jobs with Arbitrary Release Times and Variable Execution Times. In *Proceedings of 17$^{th}$ IEEE Real-Time Systems Symposium*, December 1996.

[27] K. Tindell and J. Clark. Holistic Schedulability Analysis for Distributed Hard Real-time Systems. *Microprogramming*, 50(2):117–134, April 1994.