View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by ScholarlyCommons@Penn

Penn Libraries
University of Pennsylvania

University of Pennsylvania
ScholarlyCommons

Departmental Papers (CIS)

Department of Computer & Information Science

7-2017

Process Algebraic Approach to the Schedulability Analysis and Workload Abstraction of Hierarchical Real-Time Systems

Junkil Park
University of Pennsylvania, park11@cis.upenn.edu

Insup Lee
University of Pennsylvania, lee@cis.upenn.edu

Oleg Sokolsky
University of Pennsylvania, sokolsky@cis.upenn.edu

Dae Yon Hwang

Sojin Ahn

See next page for additional authors

Follow this and additional works at: http://repository.upenn.edu/cis_papers

Part of the Computer Engineering Commons, and the Computer Sciences Commons

Recommended Citation

Junkil Park, Insup Lee, Oleg Sokolsky, Dae Yon Hwang, Sojin Ahn, Jin-Young Choi, and Inhye Kang, "Process Algebraic Approach to the Schedulability Analysis and Workload Abstraction of Hierarchical Real-Time Systems", *Journal of Logical and Algebraic Methods in Programming (JLAMP)* 92, 1-18. July 2017.

Journal of Logical and Algebraic Methods in Programming (JLAMP), Volume 92, November 2017, Pages 1-18

This paper is posted at ScholarlyCommons. http://repository.upenn.edu/cis_papers/832
For more information, please contact repository@pobox.upenn.edu.

Process Algebraic Approach to the Schedulability Analysis and Workload Abstraction of Hierarchical Real-Time Systems

Abstract

Real-time embedded systems have increased in complexity. As microprocessors become more powerful, the software complexity of real-time embedded systems has increased steadily. The requirements for increased functionality and adaptability make the development of real-time embedded software complex and error-prone. Component-based design has been widely accepted as a compositional approach to facilitate the design of complex systems. It provides a means for decomposing a complex system into simpler subsystems and composing the subsystems in a hierarchical manner. A system composed of real-time subsystems with hierarchy is called a *hierarchical real-time system*

This paper describes a process algebraic approach to schedulability analysis of hierarchical real-time systems. To facilitate modeling and analyzing hierarchical real-time systems, we conservatively extend an existing process algebraic theory based on ACSR-VP (Algebra of Communicating Shared Resources with Value-Passing) for the schedulability of real-time systems. We explain a method to model a resource model in ACSR-VP which may be partitioned for a subsystem. We also introduce *schedulability relation* to define the schedulability of hierarchical real-time systems and show that satisfaction checking of the relation is reducible to deadlock checking in ACSR-VP and can be done automatically by the tool support of ERSA (Verification, Execution and Rewrite System for ACSR). With the schedulability relation, we present algorithms for abstracting real-time system workloads.

Keywords

ACSR, Process Algebra, Real-time embedded systems

Disciplines

Computer Engineering | Computer Sciences

Comments

Journal of Logical and Algebraic Methods in Programming (JLAMP), Volume 92, November 2017, Pages 1-18

Author(s)

Junkil Park, Insup Lee, Oleg Sokolsky, Dae Yon Hwang, Sojin Ahn, Jin-Young Choi, and Inhye Kang

This journal article is available at ScholarlyCommons: http://repository.upenn.edu/cis_papers/832

A Process Algebraic Approach to the Schedulability Analysis and Workload Abstraction of Hierarchical Real-Time Systems

Junkil Park[a,1,*], Insup Lee[b], Oleg Sokolsky[b], Dae Yon Hwang[a], Sojin Ahn[a], Jin-Young Choi[a], Inhye Kang[c]

[a] *Korea University, Seoul 136-713, Korea*
[b] *University of Pennsylvania, Philadelphia, PA 19104 USA*
[c] *University of Seoul, Seoul 130-743, Korea*

Abstract

Real-time embedded systems have increased in complexity. As microprocessors become more powerful, the software complexity of real-time embedded systems has increased steadily. The requirements for increased functionality and adaptability make the development of real-time embedded software complex and error-prone. Component-based design has been widely accepted as a compositional approach to facilitate the design of complex systems. It provides a means for decomposing a complex system into simpler subsystems and composing the subsystems in a hierarchical manner. A system composed of real-time subsystems with hierarchy is called a *hierarchical real-time system*.

This paper describes a process algebraic approach to schedulability analysis of hierarchical real-time systems. To facilitate modeling and analyzing hierarchical real-time systems, we conservatively extend an existing process algebraic theory based on ACSR-VP (Algebra of Communicating Shared Resources with Value-Passing) for the schedulability of real-time systems. We explain a method to model a resource model in ACSR-VP which may be partitioned for a subsystem. We also introduce *schedulability relation* to define the schedulability of hierarchical real-time systems and show that satisfaction checking of the relation is reducible to deadlock checking in ACSR-VP and can be done automatically by the tool support of VERSA (Verification, Execution and Rewrite System for ACSR). With the schedulability relation, we present algorithms for abstracting real-time system workloads.

Keywords: ACSR, Process Algebra, Real-time embedded systems,

[*] Corresponding author

Email addresses: park11@cis.upenn.edu (Junkil Park), lee@cis.upenn.edu (Insup Lee), sokolsky@cis.upenn.edu (Oleg Sokolsky), dyhwang@formal.korea.ac.kr (Dae Yon Hwang), sjspirit@formal.korea.ac.kr (Sojin Ahn), choi@formal.korea.ac.kr (Jin-Young Choi), inhye@uos.ac.kr (Inhye Kang)

[1] Current address: University of Pennsylvania, Philadelphia, PA 19104 USA.

1. Introduction

Real-time embedded systems have increased in complexity. As microprocessors become more powerful, the software complexity of real-time embedded systems has been increasing steadily. The requirements for increased functionality and adaptability make the development of real-time embedded software complex and error-prone. Component-based design has been widely accepted as a compositional approach to facilitate the design of complex systems. It provides a means for decomposing a complex system into simpler subsystems and composing the subsystems in a hierarchical manner. A system composed of real-time subsystems with hierarchy is called *hierarchical real-time system*. For example, ARINC-653 [1] standards by the Airlines Electronic Engineering Committee specify partition-based design of avionics applications. Also, hypervisors for real-time virtual machines provide temporal partitions to guarantee real-time performance [2][3][4]. To share system resources (e.g., CPU) among real-time subsystems in a reliable and efficient manner, the hierarchical scheduling theory has been developed in [5][6][7][8][9][10]. A challenge of analyzing hierarchical real-time systems is dealing with complicated system aspects including various resource sharing/scheduling policies, complicated task behaviors (e.g., inter-dependency) and supporting multi-processors.

There has been a process algebraic approach to the schedulability analysis of real-time systems [11]. This approach is based on the timed process algebra ACSR-VP (Algebra of Communicating Shared Resources with Value-Passing) which is an extension of ACSR [12] (Algebra of Communicating Shared Resources) with the value-passing capability. ACSR-VP has the notion of resources such as the first class entities and the notion of priorities to capture the resource contention among concurrent processes. In this approach, real-time tasks are modeled as ACSR-VP processes, and scheduling algorithms can be specified by assigning proper priorities to the timed actions in the ACSR-VP processes. Due to the capability of value-passing, ACSR-VP can model the systems where the processes interchange the values of priorities (e.g., priority inheritance protocol [11]). Once a set of tasks and a scheduling algorithm of a real-time system are modeled in ACSR-VP, the schedulability of the task set can be analyzed by means of bisimulation and deadlock checking. ACSR-VP is expressive enough to model various kinds of tasks and scheduling algorithms. The schedulability analysis with this approach offers the exact schedulability condition. VERSA [13] (Verification, Execution and Rewrite System for ACSR) is a tool for ACSR-VP, and can automatically perform analysis of ACSR-VP processes such as bisimulation checking, deadlock checking and rewriting.

Although the approach [11] is well-developed and suitable for the analysis of real-time systems, but it does not consider hierarchical real-time systems, thus not supporting schedulability analysis of hierarchical real-time systems. In the hierarchical scheduling setting, only partitioned resources may be available to real-time subsystems because subsystems share resources with other subsystems. However, this approach [11] assumes that resources are dedicated and fully available to workloads in real-time systems, and has not been considered for specifying resource models which may be partitioned. Therefore, the existing approach [11] cannot provide an analysis method of the schedulability hierarchical real-time systems.

This paper proposes a process algebraic method to the schedulability analysis of hierarchical real-time systems. We provide a method to specify not only workload models and scheduling algorithms but also resource models which may not be fully available to workloads. Our method conservatively extends the previous work [11] in that the dedicated resource model that the previous work [11] only considers is a special case of our resource models. In our approach, not only workload models but also resource models are specified as ACSR-VP processes in order to express general resource models. Processes specifying workload models are called *demand processes*, and processes specifying resource models are referred to as *supply processes*. We formally define *schedulability relation* between supply processes and demand processes to provide a necessary and sufficient condition for schedulability for hierarchical real-time systems. We also show that satisfaction checking of the schedulability relation is reducible to deadlock checking in the ACSR-VP framework. Schedulability analysis for hierarchical real-time systems can be done automatically by tool-support of VERSA. Although this work focuses on the systems with a single processor, the topic of multi-processors is an avenue of future work.

In this paper, our contributions are as follows: we provide a conservative extension of the previous work [11] for hierarchical real-time systems, which provides a framework to conduct schedulability analysis and workload abstraction of hierarchical real-time systems with various scheduling algorithms. In other words, the schedulability analysis of the previous work [11] is only a special case of our work's. We demonstrate the use of VERSA to automatically perform schedulability analysis and workload abstraction. We validate the accuracy of the workload abstraction, comparing the result with CARTS (Compositional Analysis of Real-Time Systems) [14].

The rest of this paper is organized as follows: Section 2 introduces the syntax and the semantics of ACSR-VP. Section 3 describes hierarchical real-time system models in ACSR-VP. Section 4 defines the schedulability relation and the schedulability of hierarchical real-time systems. Section 5 describes the real-time workload abstraction in our framework. Finally, we conclude the paper in Section 6 with discussion of future work.

1.1. Related Work

As related works, there has been much work on analytical approaches to compositional hierarchical scheduling based on real-time scheduling theory [5][6][7][8][9][10]. There is a tool called CARTS [14] which implements the theory of [8][9]. Typically, such approaches to schedulability analysis use the schedulability condition defined as a condition in which the minimum possible resource supply can satisfy the maximum resource demand. The minimum resource supply is represented by supply bound function which under-approximates a resource supply. The maximum resource demand is represented by demand bound function which over-approximates the demand of tasks. For the exactly same type of operations, such analytical methods are generally more efficient compared to formal method-based approaches like ours which usually require more computations (e.g., state space exploration). However, the advantage of our approach is that it is flexible to adapt to the analysis of different variations of system settings (e.g., resource model, task model, scheduling algorithms). However, these analytical approaches can only handle independent tasks without inter-task dependencies and assume scheduling algorithms to be Earliest Deadline First (EDF), Rate Monotonic (RM), and Fixed Priority (FP). In practical applications, a more general class of the task model is desirable.

In addition to analytical approaches based on scheduling theory, several computational approaches based on formal methods have been developed. To perform the schedulability analysis of real-time systems, computational approaches involve modeling real-time tasks with scheduling algorithms and analyzing the state space of the models. For computational methods for the schedulability analysis of real-time systems, there have been process algebraic approaches [15][16]; automata theories based on stopwatch automata [17], task automata [18][19] and timed automata [20][21]; net-based models such as preemptive Time Petri Nets (pTPNs) [22], Petri Nets with hyper-arcs [23] and Scheduling-TPNs [24]. However, none of these approaches consider the modeling of resource model explicitly and can analyze hierarchical real-time systems.

The timed automata-based approach for compositional hierarchical scheduling [25][26] provide a framework which takes as input systems settings such as the topology of the schedulers and tasks with their attributes. It uses the formal method tool UPPAAL to verify the system's schedulability. Moreover, there has been an approach based on linear hybrid automata [27]. Compared to other automatic theoretic formal method-based approaches to hierarchical scheduling, we take a distinctive approach based on a resource-bound real-time process algebra which has the notion of resources such as the first class entities and the notion of priorities to capture the resource contention among concurrent processes. Specifically, while we consider both periodic and EDP resource models and various scheduling algorithms, the work [26] considers only the periodic resource model and less types of scheduling algorithms. Unlike ours, the work [27] only considers the cases where the global scheduler performs EDF among periodic servers and the tasks that have fixed priorities.

In [28], the authors extend the work of [22] for the schedulability analysis of two-level hierarchical real-time systems. A two-level hierarchical real-time system is composed of several subsystems and has a global scheduler at the top level that allocates resources to subsystems. Each subsystem has its own local scheduler. The work of [28][29] provides an analysis method for two-level hierarchical real-time systems with a time division multiplexing global scheduler and preemptive fixed priority local schedulers. This approach can handle tasks with inter-task dependencies. However, this approach does not provide a framework for hierarchical real-time systems with more than a two level hierarchy and restricts the resource model only to the time division multiplexing method.

There have been other process algebraic approaches to the analysis for hierarchical scheduling. The work of [30][31] is inspired by the timed process algebra ACSR. These approaches introduce a new language and formalism called PADS (Process Algebra for Demand and Supply) which are essentially based on ACSR. PADS extends the notion of resource in ACSR. While ACSR has the concept of resource request, PADS has the concept of resource request, resource grant and resource consumption. In [30][31], the authors define the schedulability in terms of PADS processes and show the schedulability analysis with independent and periodic tasks under the EDF scheduling algorithm and the resource supply by a supply process. The languages of these approaches are limited because tasks with inter-task dependencies cannot be modeled, while our approach is based on ACSR-VP, which has the feature of instantaneous actions with value-passing, so that the task model is not limited to independent tasks. Moreover, PADS is currently hard to apply to larger scale examples due to the lack of automatic tool support.

2. ACSR-VP

The Algebra of Communicating Shared Resources, ACSR [12], is a real-time process algebra. ACSR incorporates the notions of concurrency, communication, resources, priorities and timed behavior. ACSR with Value Passing [11] [32] extends ACSR [12] with dynamic priorities and value-passing capacity. In ACSR-VP, priorities can be value expressions that contain variables (e.g., $\{(cpu, x+1)\}$). In ACSR-VP, instantaneous events are augmented with value expressions to represent value-passing. For instance, $(l?x, 3)$ is an input event that receives a value in x from the channel l with the priority of 3, and $(l!7, 3)$ is an output event that sends 7 to the channel l with the priority of 3. If multiple events are available on a channel at the same time, the one with the highest priority (i.e., largest number) is triggered first. In the defining form of ACSR-VP, processes can have parameters.

2.1. Syntax

ACSR-VP distinguishes two types of actions: timed actions and instantaneous actions. While timed actions consume time to execute, instantaneous actions do not consume time. Timed actions may require access to system resources, e.g., cpu, bus, memory, etc. In contrast, instantaneous actions provide a synchronization mechanism between two concurrent processes.

A system has a finite set of serially-reusable resources, $\mathcal{R}$. A timed action $A \in \mathcal{D}_R$ takes one time unit to execute and is represented as a list of resources and associated fixed priorities, e.g., $\{(data, 2), (cpu, 1)\}$. For example, a timed action $\{(r, p)\}$ denotes the use of some resource $r \in \mathcal{R}$ running at priority level p. A distinguished timed action $\{\}$, or $\emptyset$, stands for one time unit of idling. For an action A, we use $\rho(A)$ to denote the set of resources it uses, and $\pi_r(A)$ to denote the priority of A on resource r; if A does not use resource r, $\pi_r(A)$ is zero.

Instantaneous actions, or events, provide process synchronization in ACSR. An event $a \in \mathcal{D}_E$ takes no time to execute, and is denoted by a pair (l, p), where l is the label of the event, and p is its priority, e.g., $(chan, 3)$. Labels represent input and output actions on channels. For an event a, we use $\gamma(a)$ for its label and $\pi(a)$ for its priority. As in CCS (Calculus of Communicating Systems), the special identity label, τ, arises when two events with input and output on the same channel synchronize. We define $\mathcal{L}$ as the set of all event labels.

We use $\mathcal{D}_R$ to denote the domain of timed actions, $\mathcal{D}_E$ to denote the domain of events, and $\mathcal{D} = \mathcal{D}_R \cup \mathcal{D}_E$ to denote the entire domain of actions. Let $DExp$ be a set of data expressions ranged over by e, e', e_1 and e_2. Let Val be a set of values ranged over by v, v', and v_1. Let Var be a set of data variables ranged over by x, y. Let $BExp$ be a set of boolean expressions ranged over by b, b' and b_1. We denote a vector of value variables and value expressions as $\vec{x}$ and $\vec{e}$ respectively. Also, we use l, l' and l_1 to denote event labels and r, r' and r_1 to denote resources. F and I represent sets of event labels and resources respectively. Let $\mathcal{K}$ be a set of process constants ranged over by P, Q. Let $\mathcal{T}$ be the set of all process terms ranged over by t, u. We assume that for every process constant P there is a defining equation of the form $P(\vec{x}) \overset{def}{=} t$. ACSR-VP terms are described by the following grammar:

$$\begin{array}{rcl}
t & ::= & \mathrm{NIL} \mid a.t \mid A{:}t \mid b \rightarrow t \mid t+u \mid t \| u \mid t\backslash F \mid [t]_I \mid t\backslash\backslash I \mid \mathsf{P}(\vec{e}) \\
a & ::= & (\tau, e) \mid (l?, e) \mid (l!, e) \mid (l?x, e) \mid (l!e_1, e_2) \\
A & ::= & \emptyset \mid \{S\} \\
S & ::= & (r, e) \mid (r, e), S
\end{array}$$

In the input-prefixed term $(l?x, e).t$, variable x is a bound variable with scope t, leading to the usual definitions of *free* and *bound* occurrences of value variables in process terms. In $(l?x, e).t$, the occurrence of variable x is bound. For instance, consider a term $(l_1?x, 1).(l_2!x, 1).(l_3?x, 1).(l_4!x, 1).\mathrm{NIL}$. This term can be rewritten as $(l_1?x_1, 1).(l_2!x_1, 1).(l_3?x_2, 1).(l_4!x_2, 1).\mathrm{NIL}$. We write $fv(t)$ for the set of free variables of t. We note that in an input prefix $(l?x, e).t$, e should not contain the bound variable x. We refer to process terms that contain no free variables as *processes*. Let $\mathcal{P}$ be the set of all processes ranged over by p, q, P, Q, R, S.

The syntax of process terms NIL, $a.t$, $A\!:t$, $t + u$, $t\|u$, $[t]_I$, $t\backslash F$ and $t\backslash\backslash F$ is the same as that of ACSR [12]. The syntax for actions a and A are modified so that values are added to represent dynamic priority and value passing. Note that we also write (l, e) for $(l?, e)$, and $(\bar{l}, e)$ for $(l!, e)$. NIL represents a deadlock process which performs nothing. Process terms $a.t$ and $A\!:t$ are instantaneous actions and timed action prefixes, respectively. Instantaneous actions are referred to as events and timed actions are referred to as actions. $t + u$ defines a nondeterministic choice between t and u. The choice is affected by the priority of the first actions of t and u. $t\|u$ represents the parallel composition of t and u in which the events from t and u synchronize or interleave while the timed actions from t and u, if they do not share any resource, reflect the synchronous passage of time. The Close operator, $[t]_I$, produces a process that uses the resources in I exclusively. The Restriction operator, $t \backslash F$, restricts events with labels in F from executing. $t \backslash\backslash I$ is used to represent the behavior of t in which the identities of resources in I are concealed. Besides the above operators inherited from ACSR, ACSR-VP introduces the conditional process term $b \rightarrow t$. The process term $b \rightarrow t$ behaves like t if the boolean expression b evaluates to true, or NIL if b is false. Moreover, P is a process constant with a certain arity. Each process constant P with arity n is associated with a process definition of the form $\mathsf{P}(\vec{x}) \overset{def}{=} t$, where $\vec{x}$ is a vector of n variables. In a process definition $\mathsf{P}(\vec{x}) \overset{def}{=} t$, we require that $fv(\mathsf{P}) \subseteq \vec{x}$, i.e, the definition body t does not contain free variables other than those in $\vec{x}$.

2.2. Semantics

The operational semantics of ACSR-VP is given in the usual way using a labeled transition system. A labeled transition system is a triple $(\mathcal{T}, \mathcal{D}, \rightarrow)$ where $\mathcal{T}$ is a set of process terms, $\mathcal{D}$ is a set of actions, and $\rightarrow \subseteq \mathcal{T} \times \mathcal{D} \times \mathcal{T}$ is a transition relation. The operational semantics of ACSR-VP is defined in two steps. First, we develop an *unprioritized* transition system $\rightarrow$ in which no priority information is used. Subsequently we refine $\rightarrow$ to create a prioritized transition system, $\rightarrow_\pi$, in which preempted executions are pruned.

The unprioritized transition rules in Table 1 represent the unconstrained operational semantics of ACSR-VP processes. We call it *concrete operational semantics* in order to distinguish it from *symbolic operational semantics* which will be introduced in a later chapter. We assume $\mathbb{Z}$, the set of integers, to be the domain of value variables. In Table 1, we use $m, n, k, \in \mathbb{Z}$ for constants. In addition, we only consider processes; that is, all value variables are bound and instantiated before their value is referenced. We use $[\![e]\!]$ to denote the value of e in $\mathbb{Z}$ and $[\![b]\!]$ to denote the boolean value of b. We use $t[e/x]$ to denote the process term resulting from substituting expression e for the free variable x in the process term t. Note that rules Input1, Input2, Output1, Output2, REC and ParC2 are derived from their counterparts in ACSR to create value passing semantics. Rule Cond is new to ACSR-VP. For the meaning of the aforementioned rules, we refer to [11]. The remaining rules are identical to the transition rules described for ACSR [12]

To define an operational semantics of ACSR-VP which accounts for preemption, we define the preemption relation between actions, based on priorities, as follows:

Definition 1 (Preemption Relation)**.** For two actions, α and β, we say that α is preempted by β ($\alpha \prec \beta$), if one of the following cases holds:

1. Both α and β are timed actions in $\mathcal{D}_R$, where $(\rho(\beta) \subseteq \rho(\alpha)) \wedge (\forall r \in \rho(\alpha).\pi_r(\alpha) \leq \pi_r(\beta)) \wedge (\exists r \in \rho(\beta).\pi_r(\alpha) < \pi_r(\beta))$
2. Both α and β are events in $\mathcal{D}_E$, where $\pi(\alpha) < \pi(\beta) \wedge \gamma(\alpha) = \gamma(\beta)$
3. $\alpha \in \mathcal{D}_R$ and $\beta \in \mathcal{D}_E$, with $\gamma(\beta) = \tau$ and $\pi(\beta) > 0$.

Input1	$\dfrac{}{(l?,e).t \xrightarrow{(l?,[\![e]\!])} t}$	Input2	$\dfrac{}{(l?x,e).t \xrightarrow{(l?v,[\![e]\!])} t[v/x]} \quad v \in Val$
Output1	$\dfrac{}{(l!,e).t \xrightarrow{(l!,[\![e]\!])} t}$	Output2	$\dfrac{}{(l!e_1,e_2).t \xrightarrow{(l![\![e_1]\!],[\![e_2]\!])} t}$
ActionTau	$\dfrac{}{(\tau,e).t \xrightarrow{(\tau,[\![e]\!])} t}$	ActionT	$\dfrac{}{A : t \xrightarrow{A} t}$
ChoiceL	$\dfrac{t \xrightarrow{\alpha} t'}{t + u \xrightarrow{\alpha} t'}$	ChoiceR	$\dfrac{u \xrightarrow{\alpha} u'}{t + u \xrightarrow{\alpha} u'}$
Cond	$\dfrac{t \xrightarrow{\alpha} t'}{(b \to t) \xrightarrow{\alpha} t'} \quad [\![b]\!] = true$	Rec	$\dfrac{t[[\![\vec{e}]\!]/\vec{x}] \xrightarrow{\alpha} t'}{\mathsf{P}(\vec{e}) \xrightarrow{\alpha} t'} \quad \mathsf{P}(\vec{x}) \overset{def}{=} p$
ParIL	$\dfrac{t \xrightarrow{\alpha} t'}{t \vert\vert u \xrightarrow{\alpha} t' \vert\vert u}$	ParT	$\dfrac{p \xrightarrow{A_1} p' \quad q \xrightarrow{A_2} q'}{p \vert\vert q \xrightarrow{A_1 \cup A_2} p' \vert\vert q'} \quad \rho(A_1) \cap \rho(A_1) = \emptyset$
ParIR	$\dfrac{u \xrightarrow{\alpha} u'}{t \vert\vert u \xrightarrow{\alpha} t' \vert\vert u}$	ParC2	$\dfrac{p \xrightarrow{(l!v,m)} p' \quad q \xrightarrow{(l?v,n)} q'}{p \vert\vert q \xrightarrow{(\tau,m+n)} p' \vert\vert q'}$
CloseI	$\dfrac{p \xrightarrow{a} p'}{[p]_I \xrightarrow{a} [p']_I}$	CloseT	$\dfrac{p \xrightarrow{A_1} p'}{[p]_I \xrightarrow{A_2} [p']_I} \quad A_2 = \{(r,0) \mid r \in I - \rho(A_1)\}$
ResT	$\dfrac{p \xrightarrow{A} p'}{p \backslash F \xrightarrow{A} p' \backslash F}$	ResI	$\dfrac{p \xrightarrow{a} p'}{p \backslash F \xrightarrow{a} p' \backslash F} \quad \gamma(e) \notin F$
HideI	$\dfrac{p \xrightarrow{a} p'}{p \backslash\backslash I \xrightarrow{a} p' \backslash\backslash I}$	HideT	$\dfrac{p \xrightarrow{A} p'}{p \backslash\backslash I \xrightarrow{A'} p' \backslash\backslash I} \quad A' = \{(r,p) \in A \mid r \notin I\}$

Table 1: Concrete Operational Semantics of ACSR-VP

We define the prioritized transition system $\rightarrow_\pi$ which simply refines $\rightarrow$ to account for preemption. The labeled transition system $\rightarrow_\pi$ is defined as follows: $p \xrightarrow{\alpha}_\pi p'$ if and only if

- $p \xrightarrow{\alpha} p'$ is an unprioritized transition, and
- There is no unprioritized transition $p \xrightarrow{\beta} p''$; such that $\alpha \prec \beta$.

The prioritized transition relation $\rightarrow_\pi$ defines the operational semantics of ACSR-VP.

3. Modeling Hierarchical Real-Time Systems in ACSR-VP

A *real-time system* can be modeled by a triple (W, R, A) where W is a workload model which consists of a set of tasks running in parallel in the system, R is a resource model that describes the resource allocations available to the system, and A is a scheduling algorithm that defines how the tasks use the resource at all times. We use W_A to denote a workload model W under a scheduling algorithm A. For resource models, we assume that system resource in a real-time system consists of a single processor on which a task is executed. We call the processor *cpu*. A resource model R is said to be *dedicated* if it is exclusively available to a single real-time system, or *shared* otherwise. A resource model can be represented by a task model, from the viewpoint that a resource model is the task of jobs providing the resource to the system with timing constraints [9]. Consider a resource model R of a real-time system which is represented by a single task T. R guarantees that *cpu* is allocated to the system such that the timing requirement of T is satisfied. Given a task T, we use $R(T)$ to denote a resource model R represented by T. Given a resource model R, we use $T(R)$ to denote a task T which represents R. For example, consider a resource model $R(T)$ where T is a standard periodic task which executes e time units every p time units. Then, R means a partitioned resource that guarantees *cpu* allocations of e time units every p time units to the system.

Definition 2. A *hierarchical real-time system* is recursively defined by either:

- a real-time system RS or
- a triple (W_H, R, A) where
  - W is a hierarchical workload in that W_H consists of a set of hierarchical real-time systems and tasks $\{HS_1, \ldots, HS_n\, T_1, \ldots, T_m\}$,
  - R is a resource model that describes the resource allocation available to the system HS,
  - A is a scheduling algorithm that defines how the set of systems in W shares the resource at all times.

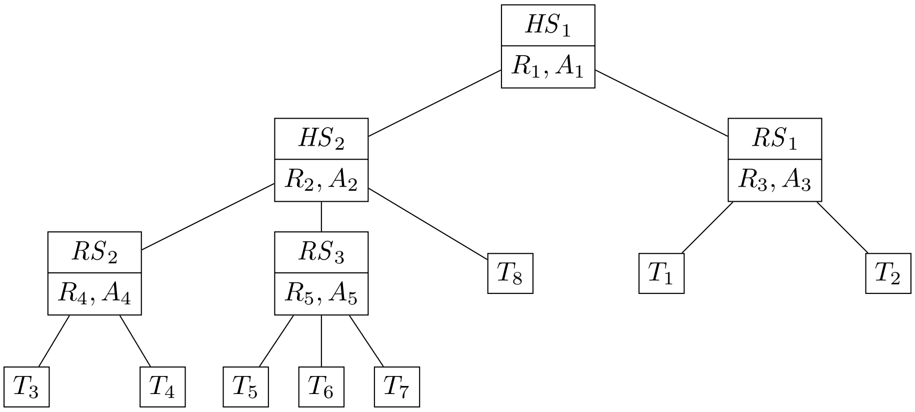

Figure 1: Hierarchical Real-time system example

Consider $HS = (\{HS_1, \ldots, HS_n, T_1, \ldots, T_m\}, R, A)$ where $HS_i = (W_{Hi}, R_i, A_i)$. For hierarchical scheduling, a parent system HS considers the task $T(R_i)$ as an abstraction of the timing requirement of the child system HS_i for each i. As long as the parent system HS satisfies the resource requirements imposed by the task $T(R_i)$, HS is able to satisfy the resource demand of the child system HS_i. This scheme makes it possible for a parent system to supply resources to its child systems without knowing how the child systems schedule resources for their own tasks.

Example 3. Consider a hierarchical real-time system HS_1 as shown in Fig. 1, whose subsystems are defined as follows:

- $HS_1 = (\{HS_2, RS_1\}, R_1, A_1)$
- $HS_2 = (\{RS_2, RS_3, T_8\}, R_2, A_2)$
- $RS_1 = (\{T_1, T_2\}, R_3, A_3)$
- $RS_2 = (\{T_3, T_4\}, R_4, A_4)$
- $RS_3 = (\{T_5, T_6, T_7\}, R_5, A_5)$

where $R_1, \ldots, R_5$ are resource models, $A_1, \ldots, A_5$ are scheduling algorithms, and $T_1, \ldots, T_8$ are tasks.

In a hierarchy, systems at leaf-level are real-time systems such as RS_1, RS_2, RS_3 that contain a set of tasks as the workload. We regard a real-time system as a hierarchical real-time system with hierarchical depth of one. Systems at non-leaf-level are hierarchical real-time systems such as HS_1 and HS_2, each of which has a set of hierarchical real-time systems as a hierarchical workload model.

In our framework, workloads consisting of a set of tasks, scheduling algorithms and resource models are specified in ACSR-VP. However, neither hierarchical workloads nor the hierarchy of hierarchical real-time systems is specified in ACSR-VP. In the remainder of this section, we describe the task and workload model, scheduling algorithms and the resource model. We also describe how they are specified in ACSR-VP.

3.1. Task and Workload Model

A *task* is regarded as a sequential process composed of a sequence of jobs which are executed serially. In this paper, we assume that each task is a preemptable, independent and periodic hard real-time task. A task T is characterized by temporal parameters (φ, p, e, d) where

- φ is a phase (or start time), which is the release time of the first job in the task,
- p is a period, which is the length of all time intervals between release times of consecutive jobs in the task,
- e is a maximum execution time of all jobs in the task, and
- d is a relative deadline, which is the maximum allowable response time of all jobs in the task.

We use $T(p, e)$ and $T(p, e, d)$ as shorthand for $T(0, p, e, p)$ and $T(0, p, e, d)$ respectively. If it is under a fixed priority scheduling algorithm, a task can be given a functional parameter fp, which is a positive number that represents the fixed priority of the task. Otherwise, priorities are assigned to jobs in a task based on a scheduling algorithm. We will discuss this priority assignment in the next subsection.

A task T with phase φ, period p, execution time e and relative deadline d can be modeled as an ACSR-VP process T, specified as follows:

$$\begin{array}{|rcl|}\hline \mathsf{T} & \overset{def}{=} & \emptyset^{\varphi} : \mathsf{J}(0,0) \\ \hline \mathsf{J}(s,t) & \overset{def}{=} & (s < e \wedge t < d) \rightarrow (\{(cpu, \pi)\} : \mathsf{J}(s+1, t+1) \\ & & \qquad\qquad\qquad\qquad + \emptyset : \mathsf{J}(s, t+1)) \\ & + & (s = e) \rightarrow \emptyset^{p-t} : \mathsf{J}(0,0) \\ \hline \end{array}$$

The process T waits until the phase of the task and then proceeds to $\mathsf{J}(0, 0)$. $\mathsf{J}(s, t)$ specifies jobs that repeatedly execute for e time units within d time units where the period of repetition is p. For $\mathsf{J}(s, t)$, the variables s and t are used to accumulate the execution time and elapsed time, respectively, of the current job. $\mathsf{J}(0, 0)$ represents a job released at the moment. The process $\mathsf{J}(s, t)$ models a preemptable job since whenever action $\{(cpu, \pi)\}$ is offered, the idling action $\emptyset$ is also offered as an alternate. If a job completes execution, i.e. $s = e$, it idles for $(p - t)$ time units waiting until the next job is released.

A workload model is a set of tasks running in parallel. A workload model $W = \{T_1, \ldots, T_n\}$ can be specified as an ACSR-VP process W specified as follows:

$$\mathsf{W} \overset{def}{=} \mathsf{T}_1 \parallel \cdots \parallel \mathsf{T}_n$$

where T_i specifies the task T_i.

W is not a complete specification in that priorities of timed actions in W are not specified yet. In the next subsection, we describe how priorities can be specified according to a given scheduling algorithm.

3.2. Workload Model under Scheduling Algorithm

In this subsection, we explain how to specify scheduling algorithms for workload models. A scheduling algorithm can be specified by assigning proper priorities to timed actions in an ACSR-VP process. When we have specified the task models using ACSR-VP, we need to assign proper priorities to the timed actions to reflect the scheduling algorithm used for the tasks. As an example, we present a complete specification of a workload model scheduled by the earliest deadline first (EDF) scheduling algorithm. EDF is a dynamic priority scheduling algorithm in which an uncompleted job with an earlier deadline has a higher priority. Consider a workload W which consists of a set of tasks $T_1, \ldots, T_n$ with phases $\varphi_1, \ldots, \varphi_n$, periods $p_1, \ldots, p_n$, execution times $e_1, \ldots, e_n$ and deadlines $d_1, \ldots, d_n$ respectively. We use d_{max} to denote $(1 + \max(d_1, \ldots, d_n))$. Basically, we use the following value expression as a priority for each task T_i specified by ACSR-VP process T_i:

$$\pi_i = d_{max} - (d_i - t)$$

where $i = 1, \ldots, n$ and t is the variable for the elapsed time. The priority π_i defined above has the property that the earlier the absolute deadline of the current job, the larger its value.

A workload model W under EDF, denoted by W_{EDF}, can be specified as a ACSR-VP process $\mathsf{W}_{\mathsf{EDF}}$ as follows:

$$\begin{array}{|rcl|}
\hline
\mathsf{W}_{\mathsf{EDF}} & \overset{def}{=} & \mathsf{T}_1 \parallel \cdots \parallel \mathsf{T}_n \\
\hline
\mathsf{T}_i & \overset{def}{=} & \emptyset^{\varphi i} : \mathsf{J}_i(0,0) \\
\hline
\mathsf{J}_i(s,t) & \overset{def}{=} & (s < e_i \wedge t < d_i) \rightarrow \\
 & & (\{(cpu, d_{max} - (d_i - t))\} : \mathsf{J}_i(s+1, t+1) \\
 & & +\emptyset : \mathsf{J}_i(s, t+1)) \\
 & + & (s = e_i) \rightarrow \emptyset^{p-t} : \mathsf{J}_i(0,0) \\
\hline
\end{array}$$

where $i = 1, \ldots, n$ and $\varphi i = \varphi_i$.

Workload models under other scheduling algorithms can be modeled similarly. The following table lists the priority assignments for other scheduling algorithms such as deadline monotonic (DM), least remaining time first (LRTF) and least laxity first (LLF):

$$\begin{array}{ll}
DM & \pi_i = d_{max} - d_i \\
LRTF & \pi_i = e_{max} - (c_i - s) \\
LLF & \pi_i = d_{max} - ((d_i - t) - (c_i - s))
\end{array}$$

where e_{max} denotes $(1 + \max\{e_1, \ldots, e_n\})$. For fixed priority scheduling (FPS) algorithm, fixed priorities $fp_1, \ldots, fp_n$ are given to tasks $T_1, \ldots, T_n$ respectively. Then, the algorithm can be simply modeled by the priority assignment $\pi_i = fp_i$.

3.3. Resource Model

In a real-time system, a resource model describes the resource allocations available to the system. Given a task T, $R(T)$ is a resource model that allocates *cpu* such that the timing requirement of T is satisfied. Although a resource model is represented by a task T, the resource model is specified in ACSR-VP differently from tasks in a workload. For a task T in a workload, the timing requirement of T may not be satisfied because of other tasks running in parallel. For a resource model $R(T)$, R guarantees the *cpu* allocations such that the timing requirement of T is satisfied. $R(T)$ can be specified as an ACSR-VP process similarly to the specification of T in Section 3.1, but R is specified to always complete the execution by the deadline. A resource model $R(T)$ where T is a task with phase φ^2, period p, execution time e and relative deadline d can be modeled as an ACSR-VP process R specified as follows:

$$
\begin{array}{|rcl|}
\hline
\mathsf{R} & \overset{def}{=} & \emptyset^{\varphi} : \mathsf{J}'(0,0) \\
\hline
\mathsf{J}'(c,t) & \overset{def}{=} & (c < e \wedge d - t > e - c) \rightarrow (\emptyset : \mathsf{J}'(c, t+1) \\
 & & +\{(cpu, 1)\} : \mathsf{J}'(c+1, t+1)) \\
 & + & (c < e \wedge d - t = e - c) \rightarrow \{(cpu, 1)\} : \mathsf{J}'(c+1, t+1) \\
 & + & (c = e \wedge t < p) \rightarrow \emptyset : \mathsf{J}'(c, t+1) \\
 & + & (c = e \wedge t = p) \rightarrow \mathsf{J}'(0,0) \\
\hline
\end{array}
$$

We assume that no multiple resource models compete in supplying resources. So, we simply use the priority assignment $\pi = 1$ to the timed actions in the specification of R. The process R waits until the phase of the task and then proceeds to $\mathsf{J}'(0,0)$. $\mathsf{J}'(0,0)$ specifies jobs that repeatedly execute for *exactly* e time units within d time units where the period of repetition is p. For $\mathsf{J}'(t,c)$, the variables c and t are used to accumulate the execution time and elapsed time, respectively, of the current job. $\mathsf{J}'(0,0)$ represents a job released at the moment. If a job completes execution, i.e. $c = e$, it waits until the next job is released, when $t = p$. The laxity of a job is equal to $(d - t) - (e - c)$. If the laxity is equal to zero, $\mathsf{J}'(t,c)$ is required to execute during the time unit, i.e. $\mathsf{J}'(c,t)$ has to perform the action $\{(cpu, \pi)\}$ and cannot choose to idle in order to complete the deadline. If the laxity is larger than zero, then, together with $\{(cpu, \pi)\}$, the idling action $\emptyset$ is also offered as an alternate.

4. Schedulability of Hierarchical Real-Time Systems

In this section, we explain how schedulability analysis of hierarchical real-time systems can be performed in an ACSR-VP framework. In the framework, the schedulability relation is a central notion. The schedulability relation '$\models$' is a binary relation on ACSR-VP processes. In terms of the schedulability relation

[2] the phase φ is used to model the extended blocking time period [9].

to be defined later, we first define the schedulability of hierarchical real-time systems as follows:

Definition 4. A real-time system $RS(W, R, A)$ is said to be *schedulable* iff $R \models W_A$.

Definition 5. A hierarchical real-time system HS is said to be *schedulable* iff,

1. if HS is a real-time system RS, then RS is schedulable, or
2. if $HS = (\{HS_1, \ldots, HS_n, T_1, \ldots, T_m\}, R, A)$ where $HS_i = (W_{Hi}, R_i, A_i)$, then
   - HS_i is schedulable for all i and
   - $R \models \{T(R_1), \ldots, T(R_n), T_1, \ldots, T_m\}_A$.

In the remainder of this section, first of all, we define the schedulability relation $\models$. Later, we describe a method to check the satisfaction of the schedulability relation. By this method, satisfaction checking of the schedulability relation can be done automatically by a tool-support. Lastly, we give an example of a hierarchical real-time system and show how its schedulability can be analyzed in our framework.

4.1. Schedulability Relation

In the previous section, we described how a real time system $RS(W, R, A)$ can be modeled in ACSR-VP. We use a term *demand process* to denote an ACSR-VP process specifying a workload model under a scheduling algorithm W_A as described in Section 3.1 and 3.2. We also use a term *supply process* to denote an ACSR-VP process specifying a resource model R as described in Section 3.3. We say $R \models W_A$ iff $\mathsf{R} \models \mathsf{W_A}$ where R is the supply process for R, and $\mathsf{W_A}$ is the demand process for W_A.

We now introduce the schedulability relation $\models$ between supply processes and demand processes. For $S \models P$, we restrict S to be a supply process and P to be a demand process. Let S be a supply process which specifies a resource model. In order for S to proceed, it may require *cpu*. Once S acquires *cpu* to proceed, S provides the resource to a demand process rather than consume *cpu* by itself. In that sense, we interpret S as a supply process providing *cpu* to a demand process. We interpret a supply process $\{(cpu, 1)\} : S$ as one that provides *cpu* during the first time unit and proceeds to S. Likewise, $\emptyset : S$ provides no resource during the first time unit and proceeds to S. A supply process $S_1 + S_2$ nondeterministically behaves as either S_1 or S_2 regardless of the resource demand of a demand process.

Let S be a supply process and $P = P_1 \parallel \cdots \parallel P_n$ be a demand process. Then, by expansion law [33], there exists a process in normal form which is equivalent to P, i.e.

$$P = \sum_{i \in I} \emptyset : Q_i + \sum_{j \in J} \{(cpu, \pi)\} : R_j$$

such that $\forall i \in I, P \xrightarrow{\emptyset}_\pi Q_i$ and $\forall j \in J, P \xrightarrow{\{(cpu,\pi)\}}_\pi R_j$.[3] In the definition of schedulability relation, we use the normal form of a demand process P. In the case that S provides *cpu* during a time unit, i.e. $S = \{(cpu, 1)\} : S'$, then P may consume *cpu* and proceed. If P doesn't demand *cpu*, i.e. $P = \emptyset : P'$, then P idles without using *cpu* in the time unit and proceeds. In another case where S doesn't provide *cpu* during a time unit, i.e., $S = \emptyset : S'$, then P may idle and proceed. While $S = \emptyset : S'$, if P_i only demands *cpu*, i.e. $P_i = \{(cpu, \pi)\} : P'_i$, then P cannot proceed. If $S = S_1 + S_2$, S may behave as either S_1 or S_2 nondeterministically. Thus, in order for P to be schedulable under S, P should be schedulable both under S_1 and S_2. Moreover, if P itself contains an inherent deadlock thus being unable to proceed, P is not schedulable regardless of S. There is a deadlock in $P = P_1 \parallel \cdots \parallel P_n$, when a conflict of demanding *cpu* between some processes P_i and P_j is unavoidable. Given a supply process S and a demand process $P = P_1 \parallel \cdots \parallel P_n$, if it is possible for P to proceed at all times with any resource supply behavior of S, each task P_i can meet its deadline during each period. Then, we say S and P satisfy the *schedulability* relation, written $S \models P$.

Definition 6. A binary relation $\mathcal{R} \subseteq \mathcal{P} \times \mathcal{P}$ is a schedulability relation if it satisfies: $(S, P) \in \mathcal{R}$ implies that

(i) $P \neq \mathrm{NIL}$

(ii) if $S \equiv \emptyset : S_1$ then, $I \neq \emptyset$ and $\forall i \in I, (S_1, Q_i) \in \mathcal{R}$

(iii) if $S \equiv \{(cpu, 1)\} : S_1$ then, if $J \neq \emptyset$ then $\forall j \in J, (S_1, R_j) \in \mathcal{R}$
else $\forall i \in I, (S_1, Q_i) \in \mathcal{R}$

(iv) if $S \equiv S_1 + S_2$, then $(S_1, P) \in \mathcal{R}$ and $(S_2, P) \in \mathcal{R}$.

(v) if $S \equiv C$, then Let $C \overset{def}{=} S1$. Then, $(S_1, P) \in \mathcal{R}$

where $P = \sum_{i \in I} \emptyset : Q_i + \sum_{j \in J} \{(cpu, \pi)\} : R_j$.

We write $S \models P$ if $(S, P) \in \mathcal{R}$ for some schedulability relation $\mathcal{R}$. Now, we will provide several examples to illustrate how to show that real-time systems are schedulable according to the schedulability relation.

Example 7. Consider a real-time system $RS_1(\{T(2,1)\}, R(T(2,1)), EDF)$. Let S_1 be a supply process to specify a resource model $R(T(2,1))$. Let P_1 be a demand process to model a single workload $\{T(2,1)\}$ under EDF. S_1 and

[3] The index sets I and J are used to represent the subsequent processes of P followed by the idle action $\emptyset$ and the resource consuming action $\{(cpu, \pi)\}$ respectively.

P_1 are as follows:

$$\begin{array}{rcl} S_1 & = & \{(cpu,1)\}\!:\!S_2 + \emptyset\!:\!S_3 \\ S_2 & = & \emptyset\!:\!S_1 \\ S_3 & = & \{(cpu,1)\}\!:\!S_1 \\ P_1 & = & \{(cpu,2)\}\!:\!P_2 + \emptyset\!:\!P_3 \\ P_2 & = & \emptyset : P_1 \\ P_3 & = & \{(cpu,3)\}\!:\!P_1 + \emptyset\!:\!\mathrm{NIL} \end{array}$$

$S_1 \models P_1$ because $\{(S_1, P_1), (\{(cpu,1)\}\!:\!S_2, P_1), (\emptyset\!:\!S_3, P_1), (S_2, P_2), (S_3, P_3)\}$ is a schedulability relation. Since $S_1 \models P_1$, $R(T(2,1)) \models \{T(2,1)\}_{\mathrm{EDF}}$ and RS_1 is schedulable.

Example 8. Consider a real-time system $RS_2(\{T(3,1)\}, R(T(2,1)), EDF)$. Let S_1 be a supply process to specify a resource model $R(T(2,1))$ as described in the previous example. Let Q_1 be a demand process to model a single workload $\{T(3,1)\}$ under EDF as follows:

$$\begin{array}{rcl} Q_1 & = & \{(cpu,1)\}\!:\!Q_2 + \emptyset\!:\!Q_3 \\ Q_2 & = & \emptyset : Q_4 \\ Q_3 & = & \{(cpu,2)\}\!:\!Q_4 + \emptyset\!:\!Q_5 \\ Q_4 & = & \emptyset : Q_1 \\ Q_5 & = & \{(cpu,3)\}\!:\!Q_1 + \emptyset\!:\!\mathrm{NIL} \end{array}$$

$S_1 \models Q_1$ because

$$\left\{ \begin{array}{ccc} (S_1, Q_1), & (\{(cpu,1)\}\!:\!S_2, Q_1), & (\emptyset\!:\!S_3, Q_1), \\ (S_3, Q_3), & (\{(cpu,1)\}\!:\!S_2, Q_4), & (S_1, Q_4), \\ (S_2, Q_1), & (\{(cpu,1)\}\!:\!S_2, Q_3), & (S_1, Q_3), \\ (S_3, Q_1), & (\{(cpu,1)\}\!:\!S_2, Q_2), & (\emptyset\!:\!S_3, Q_3), \\ (S_2, Q_4), & (S_3, Q_5), & (S_2, Q_4), \\ (S_2, Q_2), & (\emptyset\!:\!S_3, Q4), & (S_1, Q_2), \\ (\emptyset\!:\!S_3, Q_2), & (S_3, Q_4) & \end{array} \right\}$$

is a schedulability relation. Since $S_1 \models Q_1$, $R(T(2,1)) \models \{T(3,1)\}_{\mathrm{EDF}}$ and RS_2 is schedulable.

Example 9. Consider a real-time system $RS_3(\{T(2,1), T(3,1)\}, R_D, \mathrm{EDF})$. Let P_1 and Q_1 be the demand processes for $T(2,1)$ and $T(3,1)$ respectively as defined in the previous examples. Let S_D be a supply process to specify a dedicated resource model R_D (i.e., the resource is exclusively available) as follows:

$$S_D \quad = \quad \{(cpu,1)\}\!:\!S_D$$

$S_D \models (P_1 \parallel Q_1)$ because $\{(S_D, P_1 \parallel Q_1), (S_D, P_2 \parallel Q_3), (S_D, P_1 \parallel Q_4), (S_D, P_2 \parallel Q_1), (S_D, P_1 \parallel Q_2), (S_D, P_2 \parallel Q_4)\}$ is a schedulability relation. Since $S_1 \models (P_1 \parallel Q_1)$, $R_D \models \{T(2,1), T(3,1)\}_{\mathrm{EDF}}$ and RS_3 is schedulable.

Example 10. Consider a real-time system $RS_4(\{T(2,1)\}, R(T(3,1)), EDF)$. Let P_1 be the demand process for $\{T(2,1)\}$ as defined in an earlier example.

Let U_1 be the supply process for $R(T(3,1))$, defined as follows:

$$\begin{array}{lcl} U_1 & = & \{(cpu,1)\}\!:U_2 + \emptyset\!:U_3 \\ U_2 & = & \emptyset : U_4 \\ U_3 & = & \{(cpu,1)\}\!:U_4 + \emptyset\!:U_5 \\ U_4 & = & \emptyset : U_1 \\ U_5 & = & \{(cpu,1)\}\!:U_1 \end{array}$$

RS_4 is not schedulable because $U_1 \not\models P_1$ (i.e., there is no schedulability relation that contains (U_1, P_1)). This can be shown by a contradiction. Suppose $\mathcal{R}$ is a schedulability relation such that $(U_1, P_1) \in \mathcal{R}$. Then, $\mathcal{R}$ should also contain (U_3, P_3) and (U_5, NIL) by the definition of the schedulability relation. However, it is a contradiction that if $(U_5, \mathrm{NIL}) \in \mathcal{R}$, $\mathcal{R}$ is not a schedulability relation. Intuitively speaking, RS_4 is not schedulable because when $R(T(3,1))$ does not provide the resource for the first two time slots but the third time slot, then $T(2,1)$ misses the deadline.

In order to analyze the schedulability of larger systems, we need an automatic method to check satisfaction of the schedulability relation. In the next subsection, we explain how satisfaction checking can be done automatically.

4.2. Reduction to Deadlock Checking

In this section, we show that satisfaction checking of the schedulability relation $\models$ is reducible to deadlock checking in ACSR-VP. Given S and P, we can check $S \models P$ by translating S to an ACSR-VP process, composing the process with P in parallel, closing the composite process on the resource *cpu* and checking deadlock-freeness. Deadlock-freeness of a given ACSR-VP process can be checked automatically by VERSA [13] which is a tool for ACSR-VP.

Definition 11. We define a translation function $\mathcal{T} : \mathcal{P} \to \mathcal{P}$ as follows:

$$\begin{array}{lcl} \mathcal{T}(\{(cpu,1)\} : S) & = & \emptyset : \mathcal{T}(S) \\ \mathcal{T}(\emptyset : S) & = & \{(cpu,1)\} : \mathcal{T}(S) \\ \mathcal{T}(S + T) & = & (\tau,1).\mathcal{T}(S) + (\tau,1).\mathcal{T}(T) \end{array}$$

Furthermore, each definition $C \overset{def}{=} S$ is translated into $\mathcal{T}(C) \overset{def}{=} \mathcal{T}(S)$.

The translation function $\mathcal{T}$ transforms the action $\{(cpu,1)\}$ into $\emptyset$ and conversely $\emptyset$ into $\{(cpu,1)\}$. For a supply process $S_1 \overset{def}{=} \emptyset\!: S_1$, if $\mathcal{T}(S_1)$ is running with a demand process P, $\mathcal{T}(S_1)$ prevents P from consuming the resource *cpu* because $\mathcal{T}(S_1) = \{(cpu,1)\} : \mathcal{T}(S_1)$. On the other hand, if $\mathcal{T}(S_2) \overset{def}{=} \{(cpu,1)\}\!: S_2$ is running with a demand process P, $\mathcal{T}(S_2)$ allows P to consume *cpu* because $\mathcal{T}(S_2) = \emptyset : \mathcal{T}(S_2)$. Moreover, the function $\mathcal{T}$ transforms $S + T$ into $(\tau,1).\mathcal{T}(S) + (\tau,1).\mathcal{T}(T)$ where an action $(\tau,1)$ blocks the first actions of $\mathcal{T}(S)$ and $\mathcal{T}(T)$ respectively. The prefix action $(\tau,1)$ enables the process $(\tau,1).\mathcal{T}(S) + (\tau,1).\mathcal{T}(T)$ to non-deterministically evolve into either $\mathcal{T}(S)$ or $\mathcal{T}(T)$ regardless of the processes.

Example 12. Let S_1 be a supply process to specify a resource model $R(T(2,1))$ as described in the example of the previous subsection. The translated process $\mathcal{T}(S_1)$ can be described as follows:

$$\begin{aligned}&\mathcal{T}(S_1) = (\tau,1).\emptyset\colon\mathcal{T}(S_2) + (\tau,1).\{(cpu,1)\}\colon\mathcal{T}(S_3)\\&\mathcal{T}(S_2) = \{(cpu,1)\}\colon\mathcal{T}(S_1)\\&\mathcal{T}(S_3) = \emptyset\colon\mathcal{T}(S_1)\end{aligned}$$

Definition 13. We say a process P is *deadlocked* or P is a *deadlock process* if P has no action to perform at the first execution step. We say P' is a *derivative* of P, written $P \Longrightarrow P'$ if there is a sequence of transitions of the form $P \xrightarrow{\alpha_1} P_1 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_n} P_n = P'$. We consider a process is a derivative of itself, that is $P \Longrightarrow P$. We say P *contains a deadlock* or P is a *deadlock-containing* process if $P \Longrightarrow_\pi P'$ and P' is a deadlock process. If P doesn't contain a deadlock, we say P is *deadlock-free.*

Definition 14. P is n-step-deadlock-free if P cannot evolve into a deadlock process within n steps. P is 0-step-deadlock-free if P is not a deadlock process.

Lemma 15. *If P is deadlock-free and $P \Longrightarrow_\pi P'$, then P' is deadlock-free.*

Proof. If P' is deadlock-containing such that $P' \Longrightarrow_\pi P''$ where P'' is a deadlock process, then P is also deadlock-containing because $P \Longrightarrow_\pi P' \Longrightarrow_\pi P''$. □

Now, we present Lemma 16 and Lemma 18 to prove that satisfaction checking can be seen as deadlock checking, i.e., $S \models P$ iff $[\mathcal{T}(S) \parallel P]_{\{cpu\}}$ is deadlock-free.

Lemma 16. *If $[\mathcal{T}(S) \parallel P]_{\{cpu\}}$ is deadlock-free, then $S \models P$.*

Proof. We will show $\mathcal{R} = \{(S,P) \mid [\mathcal{T}(S) \parallel P]_{\{cpu\}}$ is deadlock-free$\}$ is a schedulability relation. Let there be S, P such that $(S,P) \in \mathcal{R}$. Then, $P \neq \mathrm{NIL}$ because otherwise $[\mathcal{T}(S) \parallel P]_{\{cpu\}}$ is deadlocked. Let $P = \sum_{i\in I} \emptyset\colon Q_i + \sum_{j\in J}\{(cpu,\pi)\}\colon R_j$. We argue by cases on the form of S.

Case 1. $S = \emptyset : S_1$.

Since $\mathcal{T}(S) = \{(cpu,1)\} : \mathcal{T}(S_1)$, we have $I \neq \emptyset$, otherwise $[\mathcal{T}(S) \parallel P]_{\{cpu\}}$ is deadlocked. $\forall i \in I, (S_1, Q_i) \in \mathcal{R}$ because $\forall i \in I, [\mathcal{T}(S) \parallel P]_{\{cpu\}} \xrightarrow{\{(cpu,1)\}}_\pi [\mathcal{T}(S_1) \parallel Q_i]_{\{cpu\}}$ and $[\mathcal{T}(S_1) \parallel Q_i]_{\{cpu\}}$ is deadlock-free by Lemma 15.

Case 2. $S = \{(cpu,1)\} : S_1$.

If $J \neq \emptyset$, then $\forall j \in J, (S_1, R_j) \in \mathcal{R}$ because $\forall j \in J, [\mathcal{T}(S) \parallel P]_{\{cpu\}} \xrightarrow{\{(cpu,\pi)\}}_\pi [\mathcal{T}(S_1) \parallel R_j]_{\{cpu\}}$ and $[\mathcal{T}(S_1) \parallel R_j]_{\{cpu\}}$ is deadlock-free by 15. If $J = \emptyset$, then $\forall i \in I, (S_1, Q_i) \in \mathcal{R}$ because $\forall i \in I, [\mathcal{T}(S) \parallel P]_{\{cpu\}} \xrightarrow{\emptyset}_\pi [\mathcal{T}(S_1) \parallel Q_i]_{\{cpu\}}$ and $[\mathcal{T}(S_1) \parallel Q_i]_{\{cpu\}}$ is deadlock-free by 15.

Case 3. $S = S_1 + S_2$.

$\mathcal{T}(S) = (\tau,1).\mathcal{T}(S_1) + (\tau,1).\mathcal{T}(S_2)$. $[\mathcal{T}(S) \parallel P]_{\{cpu\}}$ non-deterministically evolves into either $[\mathcal{T}(S_1) \parallel P]_{\{cpu\}}$ or $[\mathcal{T}(S_2) \parallel P]_{\{cpu\}}$ performing the action $(\tau,1)$. Since both $[\mathcal{T}(S_1) \parallel P]_{\{cpu\}}$ and $[\mathcal{T}(S_2) \parallel P]_{\{cpu\}}$ are deadlock-free by 15, we have $(S_1,P) \in \mathcal{R}$ and $(S_2,P) \in \mathcal{R}$.

Case 4. $S = C$.

Let $C \overset{def}{=} S_1$. $[\mathcal{T}(S_1) \parallel P]_{\{cpu\}}$ is deadlock-free because $\mathcal{T}(S) = \mathcal{T}(S_1)$ and $[\mathcal{T}(S) \parallel P]_{\{cpu\}}$ is deadlock-free.

□

Lemma 17. *If $S \models P$, then $[\mathcal{T}(S) \parallel P]_{\{cpu\}}$ is not a deadlock process.*

Proof. Since $S \models P$, $P \neq \text{NIL}$. Consider the cases for S letting $P = \sum_{i \in I} \emptyset : Q_i + \sum_{j \in J} \{(cpu, \pi)\} : R_j$.

Case 1. $S = \emptyset : S_1$.

$\mathcal{T}(S) = \{(cpu, 1)\} : \mathcal{T}(S_1)$, and $[\mathcal{T}(S) \parallel P]_{\{cpu\}} \xrightarrow{(cpu,1)}_{\pi}$ because $I \neq \emptyset$.

Case 2. $S = \{(cpu, 1)\} : S_1$.

$\mathcal{T}(S) = \emptyset : \mathcal{T}(S_1)$, and $[\mathcal{T}(S) \parallel P]_{\{cpu\}} \xrightarrow{A}_{\pi}$ because $P \neq \text{NIL}$.

Case 3. $S = S_1 + S_2$.

$\mathcal{T}(S) = (\tau, 1).\mathcal{T}(S_1) + (\tau, 1).\mathcal{T}(S_2)$, and $[\mathcal{T}(S) \parallel P]_{\{cpu\}} \xrightarrow{(\tau,1)}_{\pi}$.

Case 4. $S = C$

Let $C = S_1$ such that $S_1 \neq C_1$. Otherwise, $C = \text{NIL}$ because $C \overset{def}{=} C_1 \overset{def}{=} \cdots \overset{def}{=} C_n \overset{def}{=} C$. Since one of the cases above can apply to S_1, $[\mathcal{T}(S_1) \parallel P]_{\{cpu\}} \xrightarrow{\alpha}_{\pi}$. Hence, $[\mathcal{T}(S) \parallel P]_{\{cpu\}} \xrightarrow{\alpha}_{\pi}$.

□

Lemma 18. *If $S \models P$, then $[\mathcal{T}(S) \parallel P]_{\{cpu\}}$ is deadlock-free.*

Proof. We will show that if $S \models P$, then $\forall n \geq 0, [\mathcal{T}(S) \parallel P]_{\{cpu\}}$ is n-step-deadlock-free. As the base case, by Lemma 17, we prove $S \models P \implies [\mathcal{T}(S) \parallel P]_{\{cpu\}}$ is 0-step-deadlock-free.

Let's assume for some k, if $S \models P$, then $[\mathcal{T}(S) \parallel P]_{\{cpu\}}$ is n-step-deadlock-free. Let there be S, P such that $S \models P$. We will prove that $E = [\mathcal{T}(S) \parallel P]_{\{cpu\}}$ is $(k+1)$-step-deadlock-free by showing that E is not deadlocked and whenever $E \xrightarrow{\alpha}_{\pi} E'$ then E' is n-step-deadlock-free. E is not deadlocked because of Lemma 17. Let $P = \sum_{i \in I} \emptyset : Q_i + \sum_{j \in J} \{(cpu, \pi)\} : R_j$. Now, we consider the cases for S.

Case 1. $S = \emptyset : S_1$.

Since $\mathcal{T}(S) = \{(cpu, 1)\} : \mathcal{T}(S_1)$, only idling actions $P \xrightarrow{\emptyset} Q_i$ are possible for P in $[\mathcal{T}(S) \parallel P]_{\{cpu\}}$. Whenever $[\mathcal{T}(S) \parallel P]_{\{cpu\}} \xrightarrow{\{cpu,1\}}_{\pi} [\mathcal{T}(S_1) \parallel Q_i]_{\{cpu\}}$, then $S_1 \models Q_i$ and $[\mathcal{T}(S_1) \parallel Q_i]_{\{cpu\}}$ is n-step-deadlock-free by the induction hypothesis.

Case 2. $S = \{(cpu, 1)\} : S_1$

$\mathcal{T}(S) = \emptyset : \mathcal{T}(S_1)$. If $J \neq \emptyset$, then only $[\mathcal{T}(S) \parallel P]_{\{cpu\}} \xrightarrow{\{cpu,\pi\}}_{\pi} [\mathcal{T}(S_1) \parallel R_j]_{\{cpu\}}$ are possible transitions. $[\mathcal{T}(S_1) \parallel R_j]_{\{cpu\}}$ is n-step-deadlock-free because $S_1 \models R_j$.

If $m = 0$, then only $[\mathcal{T}(S) \parallel P]_{\{cpu\}} \xrightarrow{\emptyset}_{\pi} [\mathcal{T}(S_1) \parallel Q_i]_{\{cpu\}}$ are possible transitions. $[\mathcal{T}(S_1) \parallel Q_i]_{\{cpu\}}$ is n-step-deadlock-free because $S_1 \models Q_i$.

Case 3. $S = S_1 + S_2$

$\mathcal{T}(S) = (\tau, 1).\mathcal{T}(S_1) + (\tau, 1).\mathcal{T}(S_2)$. $[\mathcal{T}(S) \parallel P]_{\{cpu\}}$ can evolve into only two processes $[\mathcal{T}(S_1) \parallel P]_{\{cpu\}}$ and $[\mathcal{T}(S_2) \parallel P]_{\{cpu\}}$ which are n-step-deadlock-free because $S_1 \models P$ and $S_2 \models P$.

Case 4. $S = C$

Let $C = S_1$ such that $S_1 \neq C_1$. Otherwise, $C = \mathrm{NIL}$ because $C \overset{def}{=} C_1 \overset{def}{=} \cdots \overset{def}{=} C_n \overset{def}{=} C$. Since one of the cases above can apply to S_1, whenever $[\mathcal{T}(S_1) \parallel P]_{\{cpu\}} \xrightarrow{\alpha}_{\pi} E$ then E is n-step-deadlock-free. Hence, whenever $[\mathcal{T}(S) \parallel P]_{\{cpu\}} \xrightarrow{\alpha}_{\pi} E$ then E is n-step-deadlock-free.

□

Combining the two results above, we obtain the following theorem and derive some corollaries from the theorem.

Theorem 19. *$S \models P$ iff $[\mathcal{T}(S) \parallel P]_{\{cpu\}}$ is deadlock-free.*

Proof. By Lemma 16 and Lemma 18. □

Corollary 20. *A real-time system $RS(W, R, A)$ is schedulable iff $[\mathcal{T}(\mathsf{R}) \parallel \mathsf{W_A}]_{\{cpu\}}$ is deadlock-free.*

Proof. $RS(W, R, A)$ is schedulable iff $R \models W_A$ iff $\mathsf{R} \models \mathsf{W_A}$ iff $[\mathcal{T}(\mathsf{R}) \parallel \mathsf{W_A}]_{\{cpu\}}$ is deadlock-free. □

Corollary 21. *A real-time system $RS(W, R_D, A)$ where R_D is dedicated is schedulable iff $[\mathsf{W_A}]_{\{cpu\}}$ is deadlock-free.*

Proof. By Corollary 20, $RS(W, R_D, A)$ is schedulable iff $[\mathcal{T}(\mathsf{R_D}) \parallel \mathsf{W_A}]_{\{cpu\}}$ is deadlock-free. Since $\mathcal{T}(\mathsf{R_D}) = \emptyset^{\infty}$, $[\mathcal{T}(\mathsf{R_D}) \parallel \mathsf{W_A}]_{\{cpu\}} = [\mathsf{W_A}]_{\{cpu\}}$. □

Corollary 21 is the main result of the existing theory of schedulability in ACSR-VP [11], which only assumes that a resource model is dedicated and doesn't consider a resource model explicitly. Our framework considers an explicit resource model which may not be dedicated like Corollary 20. In that sense, our framework in this paper is a conservative extension of [11]. In the next subsection, we present a hierarchical real-time system as an example and its schedulability analysis.

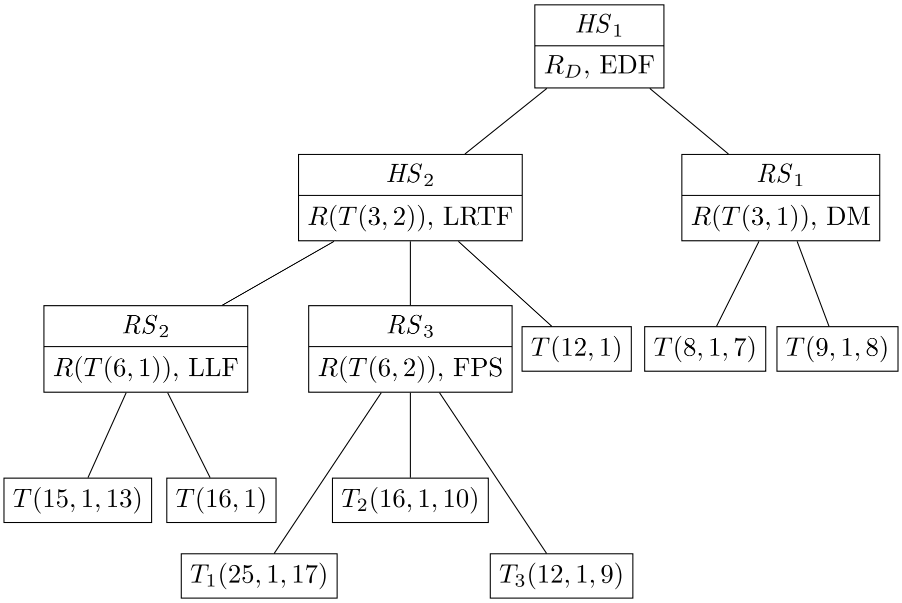

Figure 2: A Hierarchical Real-Time System of Various Real-Time Systems with Different Scheduling Algorithms

4.3. Case Study

Consider given hierarchical real-time systems HS_1, HS_2, RS_1, RS_2 and RS_3 with a hierarchy in Fig. 2. Each system can have a different scheduling algorithm. Let the systems be defined as follows:

- $HS_1 = (\{HS_2, RS_1\}, R_D, \text{EDF})$
- $HS_2 = (\{RS_2, RS_3, T(12,1)\}, R(T(3,2)), \text{LRTF})$
- $RS_1 = (\{T(8,1,7), T(9,1,8)\}, R(T(3,1)), \text{DM})$
- $RS_2 = (\{T(15,1,13), T(16,1)\}, R(T(6,1)), \text{LLF})$
- $RS_3 = (\{T_1(25,1,17), T_2(16,1,10), T_3(12,1,9)\}, R(T(6,2)), \text{FPS})$ with fixed priorities $fp_1 = 1$, $fp_2 = 3$ and $fp_3 = 2$ for T_1,T_2 and T_3 respectively.

Schedulability of HS_1 can be analyzed in a bottom-up manner following the steps below:

1. RS_1 is schedulable because $R(T(3,1)) \models \{T(8,1,7), T(9,1,8)\}_{\text{DM}}$
2. RS_2 is schedulable because $R(T(6,1)) \models \{T(15,1,13), T(16,1)\}_{\text{LLF}}$
3. RS_3 is schedulable because $R(T(6,2)) \models \{T_1(25,1,17), T_2(16,1,10),$ $T_3(12,1,9)\}_{\text{FPS}}$

4. HS_2 is schedulable because RS_2 and RS_3 are schedulable and $R(T(3,2)) \models \{T(6,1), T(6,2), T(12,1)\}_{\text{LRTF}}$
5. HS_1 is schedulable because HS_2 and RS_1 are schedulable and $R_D \models \{T(3,2), T(3,1)\}_{\text{EDF}}$

5. Abstraction of Timing Requirement of Real-Time System Workload

A real-time system is defined by a triple $RS(W, R, A)$. In practice, timing information of R may not be given while that of W and A are given. Abstracting W_A is finding a single task T so that $R(T)$ satisfies W_A. Now, we define *abstractions* as follows:

Definition 22. Given a workload W and scheduling algorithm A such that $R_D \models W_A$, a single task T is called an abstraction if $R(T) \models W_A$.

An abstraction can be a periodic task $T(p, e)$ [9] called *periodic abstraction*, or an explicit deadline periodic (EDP) task $T(p, e, d)$ [8] called *EDP abstraction.* As for the case of periodic abstraction, $T(p, e)$ demands less than $T(p, e')$ for any $e' > e$. Thus, we say a periodic abstraction $T(p, e)$ of W_A is optimal if there is no periodic abstraction $T(p, e')$ of W_A such that $e' < e$. Likewise, $T(p, e, d)$ demands no more than $T(p, e', d')$ for any $e' \geq e$ and $d' \leq d$. Therefore, we say an EDP abstraction $T(p, e, d)$ is optimal if there is no EDP abstraction $T(p, e', d')$ such that $e' < e$ or $e' = e \wedge d' > d$.

Definition 23. Given a workload W and scheduling algorithm A such that $R(T) \models W_A$, the workload abstraction problem is to find an optimal abstraction of W_A.

In the next subsection, we explain how to find periodic abstraction and EDP abstraction in our ACSR-VP framework, and compare our results to the existing methods [9][8][14].

5.1. Periodic and EDP Abstraction

We consider the problem of finding a periodic abstraction $T(p, e)$. Given a workload W, a scheduling algorithm A and a period p of the periodic abstraction T, we will find the minimal execution time e of T such that $R(T(p, e)) \models W_A$. Given W_A and p, Algorithm 1 computes the execution time e.

ALGORITHM 1: Finding Periodic Abstraction

```
for $e = 1$ to $p$ do
  if $R(T(p, e)) \models W_A$ then
    return $e$
  end if
end for
```

Algorithm 1 is straightforward. Increasing e from 1 to the given period p, the loop in the algorithm finds the minimum e such that $R(T(p, e)) \models W_A$. The algorithm guarantees that the execution time e found is optimal in the discrete domain of execution time.

We now consider the problem of finding an EDP abstraction $T(p, e, d)$. Given a workload W, a scheduling algorithm A and a period p of the EDP abstraction T, we will find the minimal execution time e of T such that $R(T(p, e, e)) \models W_A$. Then, we will find the maximal relative deadline d of T such that $R(T(p, e, d)) \models W_A$. Given W_A and p, Algorithm 2 computes the execution time e and the relative deadline d.

ALGORITHM 2: Given W, A and p, find e and d

```
for $e = 1$ to $p$ do
    if $R(T(p, e, e)) \models W_A$ then
        exit for
    end if
end for
for $d = p$ downto $e$ do
    if $R(T(p, e, d)) \models W_A$ then
        return $e, d$
    end if
end for
```

Algorithm 2 is also straightforward. Increasing e from 1 to the given period p, the first loop in the algorithm finds the minimum e such that $R(T(p, e, e)) \models W_A$ is schedulable. The second loop, decreasing d from p to e, yields the maximum d such that $R(T(p, e, d)) \models W_A$. The algorithm guarantees that an optimal abstraction in the discrete domain of execution time and deadline can be found.

As for the complexities of Algorithm 1 and Algorithm 2, both algorithms depend on the satisfaction queries for the schedulability relation $\models$, whose complexity is determined by the size of the labeled transition system of the supply/demand processes. The maximum number of queries are pseudo-polynomial in p for both algorithms.

Although Algorithms 1 and 2 find the optimal solution in a linear search manner with respect to the granularity of the budget, note that the algorithms can be improved in complexity by using a traditional binary search technique, which thus more efficiently finds the optimal solution in the candidate budget space. In the next subsection, we will show how much the performance improves by applying the binary search technique to the workload abstraction. Lastly, as each query in the algorithms are processed by the VERSA tool, we plan to enhance the efficiency of VERSA, applying symbolic and/or statistical model checking techniques, thus improving the overall efficiency of our proposed framework.

p	e	
	CARTS	ACSR-VP/VERSA
10	8	8
20	16.6667	17
30	26.6667	27
40	35	35
50	45	45
60	55	55
70	65	65
80	75	75
90	85	85
100	95	95

Table 2: Outcome Comparison of CARTS and ACSR-VP/VERSA in Periodic Abstraction

5.2. Experimental Results

In this subsection, we perform an experiment on finding an optimal abstraction to a real-time workload. In order to make the procedure to be automatic, we implement the algorithms described in the previous subsection, use an existing tool for ACSR-VP, called VERSA, to detect the existence of deadlocks. We now validate the correctness of our approach (referred to as ACSR-VP/VERSA) by comparing its result to CARTS' [14].

Experiment 24. Consider a workload model $W = \{T(0, 20, 10, 20), T(0, 40, 10, 40)\}$ under EDF scheduling algorithm. Table 2 shows the minimum execution time e given period p that CARTS and ACSR-VP/VERSA calculate respectively. As can be seen, the result of ACSR-VP/VERSA is identical to that of CARTS except the cases of $p = 20$ and $p = 30$. The difference of the cases is caused by the domains that they use. While CARTS finds the minimum execution time in the real domain, ACSR-VP/VERSA finds it in the integer domain. For example, for the case of $p = 20$, CARTS calculates 16.6667 as the minimum execution time which is different to 17 of ACSR-VP/VERSA's. In order for the result to be used in the practical real-time system where a basic time unit is 1, the value 16.6667 has no practical meaning, and needs to be rounded up to 17. Therefore, the result of CARTS is consistent with that of ACSR-VP/VERSA for the case. The case of $p = 30$ can be argued in the same way. In that sense, Table 2 shows that the result of ACSR-VP/VERSA is practically identical to that of CARTS up to rounding up (or ceiling).

Experiment 25. Consider a workload model $W = \{T(50, 10), T(70, 10)\}$ under DM scheduling algorithm. Table 3 shows e and d of the EDP abstraction that CARTS and ACSR-VP/VERSA calculate respectively for a given period p and W_{DM}. We observe that the result of CARTS is consistent with that of ACSR-VP/VERSA.

p	e, d	
	CARTS	ACSR-VP/VERSA
10	4,4	4,4
20	10,20	10,20
30	15,25	15,25
40	20,30	20,30
50	20,20	20,20
60	30,40	30,40
70	30,30	30,30
80	40,40	40,40
90	50,50	50,50
100	60,60	60,60

Table 3: Outcome Comparison of CARTS and ACSR-VP/VERSA in EDP Abstraction

Experiment 26. Consider a workload model $W = \{T(50, 10), T(70, 10)\}$. Table 4 shows the minimum execution time e and the deadline d that ACSR-VP/VERSA calculates for a given period p, W and a scheduling algorithm A such as LRFT, LLF. CARTS is currently unable to find EDP abstractions for scheduling algorithm other than EDF and DM. However, our approach is capable of finding the optimal EDP abstractions for the scheduling algorithms such as LRFT, LLF shown in Table 4 and is capable for any other algorithm that can be modeled in ACSR-VP.

In the following experiments, we evaluate the performance (i.e., running time) of our approach. To do this, we used a Ubuntu linux machine with Intel Xeon CPU E5-2667 v2 with 128 GB of RAM. Although the following experiments focus on the periodic workload abstraction and the LLF scheduling algorithm, we believe that the experiment result would also show a similar trend with the EDP workload abstraction and the other scheduling algorithms.

Experiment 27. Consider a workload model $W = \{T(10, 1), T(20, 2), T(30, 2)\}$ and a scheduling algorithm A = LLF. In this experiment, we test if W_A is schedulable under the periodic resource model $R(T(30, C))$ varying C from 1 to 30. Note that W_A is schedulable under $R(T(30, C))$ if C is greater than or equal to 26, and is not schedulable if C is less than 26. Figure 3 shows the running time required to test whether $R(T(30, C)) \models W_A$ varying C from 1 to 30. As can be observed from the figure, the running time is neither increasing nor decreasing monotonously. As our approach reduces the schedulability checking problem into the deadlock checking problem, the running time of the schedulability test depends on the size of the state space of the translated ACSR-VP process which is constructed for deadlock checking.

Experiment 28. In this experiment, we evaluate the performance (i.e., running time) of the periodic workload abstraction algorithm (Algorithm 1), and

p	e, d	
	LRFT	LLF
10	4,4	4,4
20	10,20	10,20
30	15,20	15,25
40	20,30	20,30
50	20,20	20,20
60	30,40	30,40
70	30,30	30,30
80	40,40	40,40
90	50,50	50,50
100	60,60	60,60

Table 4: EDP Abstraction of Real-Time Systems with LRFT and LLF Scheduling Algorithms

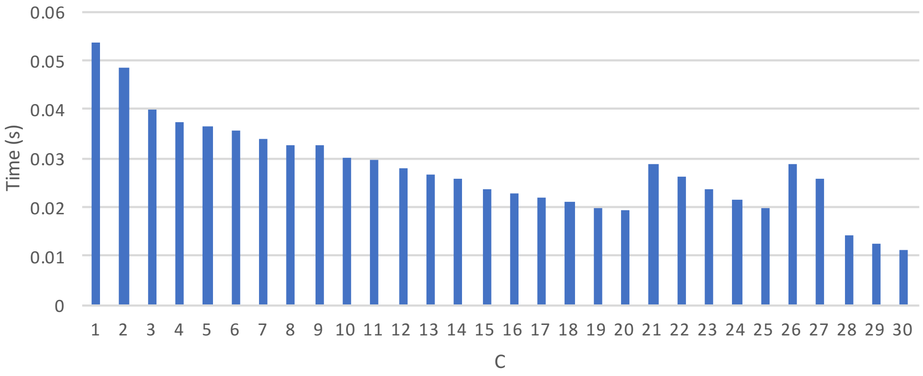

Figure 3: Running Time to Test $R(T(30,C)) \models \{T(10,1), T(20,2), T(30,2)\}_{\text{LLF}}$

show that the binary search technique significantly improves the performance of Algorithm 1 (i.e., linear search-based approach). To evaluate, we used the workload model $W = \{T(10,1), T(20,2), T(30,2)\}$ and the scheduling algorithm $A = \text{LLF}$. We performed the periodic workload abstraction for the periods ranged from 5 to 100 using both the linear search-based approach and the binary search-based approach. Although the two approaches produce the same outcomes (i.e., the minimum execution time), they greatly differ in efficiency, as shown in Figure 4. This figure shows that the binary search-based approach scales well, while the linear search-based approach's running time rises quite steeply as P increases.

Experiment 29. This experiment evaluates the correlation between the running time of the workload abstraction and the number of tasks in the workload model. We consider finding the optimal periodic abstraction with the period of

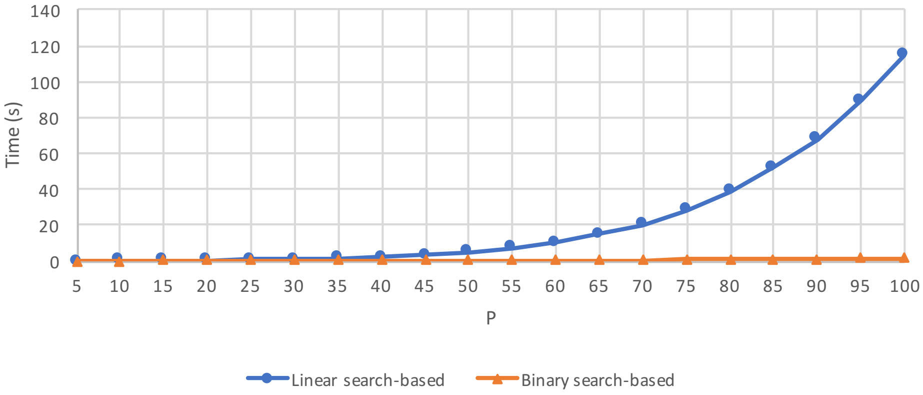

Figure 4: The Running Time Comparison for Workload Abstraction Algorithms

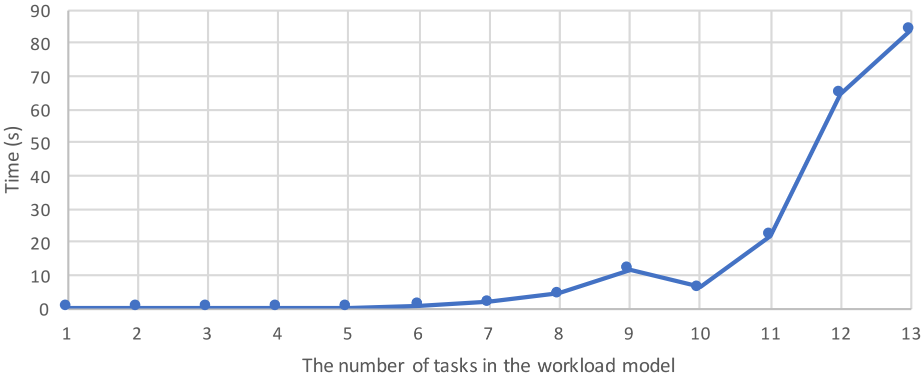

Figure 5: The Running Time of Workload Abstraction for the Different Number of Tasks in the Workload Model

50 for the workload which consists of multiple task instances of $T(15, 1)$ under the LLF scheduling algorithm (e.g., the workload model is $\{T(15, 1), T(15, 1)\}$ when the number of tasks is 2). Figure 5 shows the running time of the binary search-based workload abstraction approach for the different number of tasks in the workload model. We observe that when the number of tasks reaches a certain number (e.g., 12 in Figure 5), the running time rapidly increases. The reason is that our approach is a computational method which performs the state space exploration of the system to analyze its schedulability. Typically, increasing the number of tasks eventually leads to the exponential growth of the state space.

6. Conclusions and Future Work

In this paper, we conservatively extended the scheduling theory of ACSR to model and analyze hierarchical real-time systems. We presented a method to model not only workload and scheduling algorithms but also resource models which may not be fully available to workload. We defined schedulability relation to provide necessary and sufficient conditions for the schedulability of hierarchical real-time systems. We showed that satisfaction checking of the schedulability relation is reducible to deadlock checking in ACSR-VP, thus being able to be done automatically using VERSA's tool support.

We also tackled another scheduling problem in hierarchical real-time systems. The problem is abstracting the timing requirement of real-time system workload. A workload of a real-time system consists of a set of tasks which are running under a scheduling algorithm. Abstracting a set of tasks is finding a single task representing the task set such that whenever the timing requirement of the single task is satisfied, the timing requirement of a set of tasks in the workload model is also satisfied. The problem is practically important because it is related to determining a resource model of a real-time subsystem in a hierarchical real-time system. We presented experimental results to prove the correctness and the performance of our approach.

There are several directions for future work. First of all, our future goals are to extend our work to handle inter-task dependencies in workload models, thus enabling the analysis of more realistic systems. Moreover, since this work focused on single processor systems, we hope to develop a framework for hierarchical real-time systems with multiprocessors. Since timed actions in ACSR-VP already allow having multiple resource requirements expressed, we plan to extend the schedulability relation with the notion of multiple resource supply/demand, and provide a method to automatically check the satisfaction of the schedulability relation.

Lastly, since our approach is a computational method, as a natural consequence, the larger the system is, the longer the time required to analyze the system is (e.g., the workload model with a large number of tasks). Thus, we plan to enhance the efficiency of the analysis tool/technique for ACSR-VP so as to handle/reduce the large state space occurring in the analysis.

Acknowledgments

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology(2012R1A1A2009354). This research was supported by the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the ITRC(Information Technology Research Center) support program (IITP-2016-H8501-16-1012) supervised by the IITP(Institute for Information & communications Technology Promotion). This research was supported in part by ONR N00014-16-1-2195 and NSF CNS-1329984, and by Global Research Laboratory Program (2013K1A1A2A02078326) through NRF and the DGIST Research and Development Program (CPS Global Center) funded by the Ministry of Science, ICT & Future Planning.

References

[1] A. E. E. Committee, Arinc specification 653: Avionics application software standard interface, Aeronautical radio, Inc., Annapolis, Maryland (Jan. 1997).

[2] RT-XEN, Real-time virtualization based on hierarchical scheduling (2011).
URL `https://sites.google.com/site/realtimexen/`

[3] Linuxworks, Lynxsecure embedded hypervisor and separation kernel (2011).
URL `http://www.lynuxworks.com/virtualization/hypervisor.php`

[4] R.-T. S. Gmbh, Real-time hypervisor (2011).
URL `http://www.real-time-systems.com`

[5] Z. Deng, J. W.-S. Liu, Scheduling real-time applications in an open environment, in: Proceedings of the 18th IEEE Real-Time Systems Symposium, RTSS '97, IEEE Computer Society, Washington, DC, USA, 1997, pp. 308–.

[6] X. Feng, A. K. Mok, A model of hierarchical real-time virtual resources, in: Real-Time Systems Symposium, 2002. RTSS 2002. 23rd IEEE, IEEE, 2002, pp. 26–35.

[7] G. Lipari, E. Bini, Resource partitioning among real-time applications, in: Real-Time Systems, 2003. Proceedings. 15th Euromicro Conference on, IEEE, 2003, pp. 151–158.

[8] A. Easwaran, M. Anand, I. Lee, Compositional analysis framework using edp resource models, in: Proceedings of the 28th IEEE International Real-Time Systems Symposium, RTSS '07, IEEE Computer Society, Washington, DC, USA, 2007, pp. 129–138.

[9] I. Shin, I. Lee, Compositional real-time scheduling framework with periodic model, ACM Trans. Embed. Comput. Syst. 7 (2008) 30:1–30:39.

[10] F. Dewan, N. Fisher, Bandwidth allocation for fixed-priority-scheduled compositional real-time systems, ACM Transactions on Embedded Computing Systems (TECS) 13 (4) (2014) 91.

[11] H. Ben-Abdallah, J.-Y. Choi, D. Clarke, Y. S. Kim, I. Lee, H.-L. Xie, A process algebraic approach to the schedulability analysis of real-time systems, Real-Time Syst. 15 (1998) 189–219.

[12] I. Lee, P. Bremond-Gregoire, R. Gerber, A process algebraic approach to the specification and analysis of resource-bound real-time systems, Proceedings of the IEEE 82 (1) (1994) 158–171.

[13] D. Clarke, I. Lee, H. liang Xie, Versa: A tool for the specification and analysis of resource-bound real-time systems, Journal of Computer and Software Engineering 3.

[14] L. T. X. Phan, J. Lee, A. Easwaran, V. Ramaswamy, S. Chen, I. Lee, O. Sokolsky, Carts: a tool for compositional analysis of real-time systems, SIGBED Rev. 8 (2011) 62–63.

[15] M. N. nez, I. Rodríguez, Pamr: A process algebra for the management of resources in concurrent systems, in: Proceedings of the IFIP TC6/WG6.1 - 21st International Conference on Formal Techniques for Networked and Distributed Systems, FORTE '01, Kluwer, B.V., Deventer, The Netherlands, The Netherlands, 2001, pp. 169–184.

[16] M. Mousavi, M. Reniers, T. Basten, M. Chaudron, Pars: A process algebra with resources and schedulers, in: K. Larsen, P. Niebert (Eds.), Formal Modeling and Analysis of Timed Systems, Vol. 2791 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2004, pp. 134–150.

[17] F. Cassez, K. G. Larsen, The impressive power of stopwatches, in: Proceedings of the 11th International Conference on Concurrency Theory, CONCUR '00, Springer-Verlag, London, UK, UK, 2000, p. 138 152.

[18] E. Fersman, P. Pettersson, W. Yi, Timed automata with asynchronous processes: Schedulability and decidability, in: J.-P. Katoen, P. Stevens (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, Vol. 2280 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2002, pp. 125–149.

[19] E. Fersman, P. Krcal, P. Pettersson, W. Yi, Task automata: Schedulability, decidability and undecidability, Inf. Comput. 205 (2007) 1149–1172.

[20] A. David, J. Illum, K. G. Larsen, A. Skou, Model-based framework for schedulability analysis using uppaal 4.1, Model-based design for embedded systems 1 (1) (2009) 93–119.

[21] M. Mikučionis, K. G. Larsen, J. I. Rasmussen, B. Nielsen, A. Skou, S. U. Palm, J. S. Pedersen, P. Hougaard, Schedulability analysis using uppaal: Herschel-planck case study, in: Proceedings of the 4th international conference on Leveraging applications of formal methods, verification, and validation - Volume Part II, ISoLA'10, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 175–190.

[22] G. Bucci, A. Fedeli, L. Sassoli, E. Vicario, Timed state space analysis of real-time preemptive systems, IEEE Trans. Softw. Eng. 30 (2) (2004) 97–111.

[23] O. Roux, D. Lime, Time petri nets with inhibitor hyperarcs. formal semantics and state space computation, in: J. Cortadella, W. Reisig (Eds.), Applications and Theory of Petri Nets 2004, Vol. 3099 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2004, pp. 371–390.

[24] D. Lime, O. H. Roux, Formal verification of real-time systems with preemptive scheduling, Real-Time Syst. 41 (2) (2009) 118–151.

[25] A. Boudjadar, A. David, J. H. Kim, K. G. Larsen, M. Mikučionis, U. Nyman, A. Skou, Hierarchical scheduling framework based on compositional analysis using uppaal, in: International Workshop on Formal Aspects of Component Software, Springer, 2013, pp. 61–78.

[26] A. Boudjadar, A. David, J. H. Kim, K. G. Larsen, M. Mikučionis, U. Nyman, A. Skou, Widening the schedulability of hierarchical scheduling systems, in: International Workshop on Formal Aspects of Component Software, Springer, 2014, pp. 209–227.

[27] Y. Sun, G. Lipari, R. Soulat, L. Fribourg, N. Markey, Component-based analysis of hierarchical scheduling using linear hybrid automata, in: Embedded and Real-Time Computing Systems and Applications (RTCSA), 2014 IEEE 20th International Conference on, IEEE, 2014, pp. 1–10.

[28] L. Carnevali, L. Ridi, E. Vicario, Putting preemptive time petri nets to work in a v-model sw life cycle, IEEE Trans. Softw. Eng. 37 (6) (2011) 826–844.

[29] L. Carnevali, A. Pinzuti, E. Vicario, Compositional verification for hierarchical scheduling of real-time systems, Software Engineering, IEEE Transactions on 39 (5) (2013) 638–657.

[30] A. Philippou, I. Lee, O. Sokolsky, J.-Y. Choi, A process algebraic framework for modeling resource demand and supply, in: Proceedings of the 8th international conference on Formal modeling and analysis of timed systems, FORMATS'10, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 183–197.

[31] A. Philippou, I. Lee, O. Sokolsky, Pads: An approach to modeling resource demand and supply for the formal analysis of hierarchical scheduling, Theoretical Computer Science 413 (1) (2012) 2–20, quantitative Aspects of Programming Languages (QAPL 2010).

[32] H.-H. Kwak, Process algebraic approach to the parametric analysis of real-time scheduling problems, Ph.D. thesis, University of Pennsylvania (2000).

[33] P. Brémond-Grégoire, J.-Y. Choi, I. Lee, A complete axiomatization of finite-state ACSR processes, Inf. Comput. 138 (2) (1997) 124–159.