Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®
Theoretical Computer Science 313 (2004) 119–132

Theoretical Computer Science

www.elsevier.com/locate/tcs

Scheduling hard sporadic tasks with regular languages and generating functions

Dominique Geniet[a,*], Jean-Philippe Dubernard[b]

[a] *Laboratoire d'Informatique Scientifique et Industrielle, Téléport 2, Site du Futuroscope, Futuroscope Chasseneuil Cédex F-86961, France*

[b] *Laboratoire d'Informatique Fondamentale et Appliquée de Rouen, Place Émile Blondel, Mont-Saint-Aignan Cédex F-76821, France*

Abstract

In this paper, we consider offline validation of hard real-time systems composed of both periodic and sporadic tasks, embedded on centralized multi-processor architectures. To model hard real-time systems, we use untimed finite automata: each accepted word is a valid operational behavior of the periodic component of the system. Then, by associating generating functions with edges of the automaton, we give a modular decisional technique to decide the feasibility of sporadic tasks.

Keywords: Hard real-time systems; Finite automata; Generating functions; Operational validation

1. Introduction

1.1. Real-time systems

A real-time system is both reactive (it must react to ingoing events) and concurrent (all scan and control operations must progress simultaneously). Then, it is composed of a set of elementary tasks. Each task implements the reaction (or a part of the reaction) corresponding to a given ingoing event.

* Corresponding author.
E-mail addresses: dominique.geniet@laposte.net (D. Geniet), jeanphi.dubernard@wanadoo.fr (J.-P. Dubernard).

0304-3975/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2003.10.009

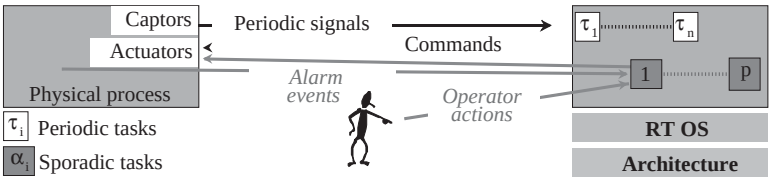


Fig. 1. Functional structure of a hard real-time system.

Some ingoing events follow regular flows: they are often transmitted from periodic captors. Tasks designed both to read and to treat these events must be synchronized with the corresponding captors. They are periodic tasks (see Fig. 1) we denote by $(\tau_i)_{i\in[1,n]}$ these tasks. Real-time systems contain also sporadic tasks, which are designed to react to alarm signals. These tasks are denoted by $(\alpha_i)_{i\in[1,p]}$.

We consider systems composed of both periodic and sporadic tasks. Periodic tasks can share resources and communicate. Sporadic tasks are independent from others (excepting for processor sharing, of course).

Time specifications of real-time tasks have been defined in [16]. The operational specification of hard task ξ is composed of four characteristics:

- The first activation date (r_ξ) is the date of creation of ξ.
- The critical delay (D_ξ) is the delay between the activation date of an instance of ξ and its deadline (later possible completion date).
- The CPU time (C_ξ) of ξ is the total CPU owning time needed by each instance of ξ to end its execution.
- When ξ is periodic, the period (T_ξ) is the delay between the activation dates of two successive instances of ξ. If ξ is not periodic, it is the minimal time delay between the occurrences of two successive activation events of ξ.

1.2. Validation techniques

A software must be validated before being used, in order to guarantee that its behavior correspond to its specifications. For real-time systems, validation deals with two aspects: functional validation and time validation. [1] One must guarantee that, whatever be the process behavior and the incoming event flow, the real-time system is able to react according to its time specifications.

As far as a real-time system is composed of tasks which have to progress simultaneously, both drive and validation processes depend on the used scheduling policies. Two different approaches are used: online and offline. An online scheduling algorithm selects the task to activate from the knowledge of an instantaneous view of the system. On the contrary, while using an offline approach, all valid sequences are produced. Then, at time t, whatever be the future of the task system, we can decide the feasibility of the system.

[1] A valid result is wrong if it is obtained too late.

Of course, the feasibility of the system depends on both the physical and software configurations. Graham et al. [12] proposed a formal description for the scheduling contexts and they have been extended and adapted in [4,13]. Here, we do not detail these descriptions. The scheduling context concerned by our study is:
Hardware: One node, one clock, multi-processors, all processors follow the clock, share memory.
Software: Periodic and sporadic tasks, there is resource sharing between periodic tasks, message communication through memory sharing, processor sharing, fixed CPU load for each task.

1.2.1. Optimal scheduling algorithms

Let us consider a scheduling context C and a task system S, to be scheduled under context C. The set $V_C(S)$ collects all the valid scheduling sequences for configuration S in context C: they are sequences which can guarantee that whatever the life duration of the application, no task misses its deadline. The predicate $V_C(S) = \emptyset$ means that configuration S cannot be scheduled in context C. On the opposite, $V_C(S) \neq \emptyset$ means that S can be scheduled. A scheduling algorithm A is a function that associates a set $V_{C,A}(S)$ of scheduling sequences with a configuration S, under context C. We get $V_{C,A}(S) \subset V_C(S)$.

Definition 1. A is *optimal* in context $C \Leftrightarrow \{V_C(S) \neq \emptyset \Rightarrow V_{C,A}(S) \neq \emptyset\}$.

1.2.2. Online policies

Usually, real-time scheduling is based on preemptive priority-based policies. Scheduling decisions are taken online by the scheduler: the higher the priority, the earlier the task chosen. Priorities can be arbitrary integer constants. In this case, they are associated with tasks during the software specification process, and we deal with "static online scheduling". Priorities can also be defined from time characteristics of tasks (e.g. priority is the relative deadline of the task). In this case, we deal with "dynamic online scheduling".

This approach has generated lot of work. It is often used, because of its flexibility, but it is limited because it is not optimal as soon as there are interdependent tasks (i.e. in all real cases!).

1.2.3. Offline policies

The hard real-time scheduling decision problem is NP-hard [3]. Then, there is no optimal online strategy for general systems. This is why strategies based on the exhaustive enumeration of valid scheduling sequences were found. They are offline strategies. Grolleau [13] and Leung and Merill [15] give a minimal simulation duration value, useful for the exhaustive enumeration process.

Offline strategies are based on state models (automata, Petri nets, etc.). Some models are explicit in time [1,10]. They are timed models. They support simulation techniques. Some other models are implicit in time [13]. They support analysis techniques. In both the two cases, a central class of studies addresses the problem of production of all valid scheduling sequences. Moreover, as far as the complexity of feasibility decision

problem is exponential, a major aim of the work is the research of improvement techniques (often by early detection of branches corresponding to invalid sequences).

1.2.4. Validation

Validating a real-time system consists in proving that in the whole life of the software, no deadline will be missed. If we plan to drive the process with an online algorithm, the validation can be obtained both by simulation of the algorithm or by model checking techniques, i.e. an offline strategy validates an online scheduling. We can also produce a valid sequence, thanks to an offline strategy, and embed it in a sequencer. But this approach is very inflexible.

1.3. Aim of this work

All these techniques are used to validate periodic systems where there are resource sharing, task precedence and message communication, in mono-processor environment. As far as we know, no work deals with sporadic tasks, and few work deal with multi-processor scheduling validation.

Here, we use a model based on regular languages [11] to collect the valid behavior of a real-time system. Each task (sporadic and periodic are concerned) is associated with a regular language, which collects its timely correct behavior. Concurrency is modeled by the homogeneous product of languages, and task interdependence (resource sharing, for instance) is integrated, thanks to Arnold–Nivat's technique [2].

First, we show how untimed finite automata can be used to model operational behavior. Then, we give a result useful to decide the validity of a task system in a specific scheduling context. Next, we extend the model to give the feasibility decision of the whole system by analyzing its periodic component.

2. Time valid behaviors

Our model is useful to address problems concerning operational behaviors (deadlines, etc.), but not functional behaviors (what the program does, etc.). First, we present the model for periodic tasks. Then, we show that sporadic tasks can be integrated in a very easy way.

2.1. A regular language to model behaviors

Let us consider task τ_i, of parameters $r_i \in \mathbb{N}$, $C_i \in \mathbb{N}^*$, $D_i \in \mathbb{N} \cap [C_i, +\infty[$ and $T_i \in \mathbb{N} \cap [D_i, +\infty[$. From its activation date $r_i + k \times T_i$, the kth instance of τ_i must own a CPU resource for C_i time units on the time interval $[r_i + k \times T_i, r_i + k \times T_i + T_i[$. Let us denote by a_i the state τ_i owns a CPU, and by $\bullet$ the state τ_i does not own a CPU.

Definition 2. Let $L_1 \subset \Sigma_1^*$ and $L_2 \subset \Sigma_2^*$. The *Shuffle* operator, denoted ш, is defined on the following way: $\forall a \in \Sigma_1 \cup \Sigma_2, a \text{ ш} \varepsilon = \{a\}$ $\forall (a, b, \omega, \xi) \in \Sigma_1 \times \Sigma_2 \times \Sigma_1^* \times \Sigma_2^*$ such that $(a\omega, b\xi) \in L_1 \times L_2$, $a\omega \text{ ш} b\xi = a(\omega \text{ ш} b\xi) \cup b(a\omega \text{ ш} \xi)$.

Every word of $a_i^{C_i}\text{ш}\bullet^{D_i-C_i}$ corresponds, on any time interval of the form $[r_i + k \times T_i, r_i + k \times T_i + D_i[$, to a processor allocation configuration compatible with the time constraints of τ_i. This set is regular. If the scheduling configuration is valid, τ_i is inactive on every time interval of the form $[r_i + k \times T_i + D_i, r_i + (k+1) \times T_i[$. This inactivity is modeled by the word $\bullet^{T_i-D_i}$. Then, every word of $(a_i^{C_i}\text{ш}\bullet^{D_i-C_i})\bullet^{T_i-D_i}$ is a correct CPU allocation configuration for τ_i on any time interval of the form

$$[r_i + k \times T_i, r_i + (k+1) \times T_i[.$$

On the operational plan, τ_i is defined as the sequence $(\tau_{ij})_{j\in\mathbb{N}}$ of its instances. Then, a processor allocation compatible with τ_i's time constraints is a sequence of processor allocations compatible with τ_i's successive instances time contraints. Let $(\omega_j)_{j\in\mathbb{N}} \in ((a_i^{C_i}\text{ш}\bullet^{D_i-C_i})\bullet^{T_i-D_i})^{\mathbb{N}}$. For each $n \in \mathbb{N}$, the word $\omega_0\omega_1 \ldots \omega_n$ models a valid[2] processor allocation configuration for any sequence of $n+1$ successive instances of τ_i. In a general way, any word ω of $((a_i^{C_i}\text{ш}\bullet^{D_i-C_i})\bullet^{T_i-D_i})^*$ models a valid processor allocation configuration for τ_i on any time interval of the form $[r_i + k \times T_i, r_i + k \times T_i + |\omega|[$, and then on the time interval $[r_i, r_i + |\omega|[$. As far as τ_i is inactive on interval $[0, r_i[$, the word $\bullet^{r_i}\omega$ models a valid processor allocation on the time interval $[0, r_i + |\omega|[$. ω can be as long as we want. Then, the language $\bullet^{r_i}((a_i^{C_i}\text{ш}\bullet^{D_i-C_i})\bullet^{T_i-D_i})^*$ collects all valid processor allocations for τ_i.

The scheduling validation problem consists, at time t, in deciding the evolution possibilities of τ_i in both the given hardware and software context. Of course, the past of τ_i is known. Here, this past is the history of τ_i's CPU allocations, that is a finite word ω of $\{a_i, \bullet\}^*$. During this past, some instances of τ_i were completed, and the current instance is on (we can consider it even if it is just beginning). Then, by construction, ω is of the form $\omega_1\mu$, where $\omega_1 \in \bullet^{r_i}((a_i^{C_i}\text{ш}\bullet^{D_i-C_i})\bullet^{T_i-D_i})^*$ (past instances were valid), and $\exists v \in \{a_i, \bullet\}^*$ such that $\mu v \in ((a_i^{C_i}\text{ш}\bullet^{D_i-C_i})\bullet^{T_i-D_i})$ (the current instance can be completed according to the time constraints). Then, ω is a prefix of a word of $\bullet^{r_i}((a_i^{C_i}\text{ш}\bullet^{D_i-C_i})\bullet^{T_i-D_i})^*$. For any instant f belonging to $]t, +\infty[$ (the future), there exists a word η in $\bullet^{r_i}((a_i^{C_i}\text{ш}\bullet^{D_i-C_i})\bullet^{T_i-D_i})^*$ such that $|\omega v\eta| > f$. Recall that

Definition 3. The center of language L is the set of prefixes of L that can be indefinitely extended in L.

Remark. Algebraically, $Center(L) = L^*.LeftFactors(L)$.

Then, the past ω of τ_i belongs to the center of $\bullet^{r_i}((a_i^{C_i}\text{ш}\bullet^{D_i-C_i})\bullet^{T_i-D_i})^*$. Reciprocally, by definition, every word of this center language is the past of a valid processor allocation configuration.

Definition 4. We call *time valid behavior* of task τ_i every word of the language $Center(\bullet^{r_i}((a_i^{C_i}\text{ш}\bullet^{D_i-C_i})\bullet^{T_i-D_i})^*)$.

[2] I.E. compatible with τ_i's time constraints.

A time valid behavior of τ_i models a processor allocation configuration such that, in the future, the past of τ_i can effectively be extended according to τ_i time constraints. In the following, we denote by $L(\tau_i)$ this language.

2.2. Integrating sporadic tasks

We have seen in Section 1.1 that sporadic tasks are connected with alarm signals: each occurrence of an alarm releases an instance of the task. This behavior leads firstly to the first occurrence date to be unknown, and secondly, the beginning of the task to follow the occurrence of the alarm event: this occurrence is modeled by a *Send* statement (the physical process is the sender), the code of the task begins with the corresponding *Receive* statement, and the synchronization takes the delay into account to decide the feasibility of the task.

The prefix $\bullet^{r_i}$ that corresponds, for periodic tasks, to the inactivity delay before the creation of the task is replaced, for sporadic tasks, by $\bullet^*$: the delay can be any positive duration. A similar analysis for the time interval that separates two successive instances of the task leads us to replace the suffix $\bullet^{T_i-D_i}$ by the suffix $\bullet^{T_i-D_i}\bullet^*$: for sporadic tasks, T_i is a floor value.

Then, the time model for α_i, of characteristics $r_i=\perp$, $D_i>0$, $C_i\in[0,D_i]$, and $T_i\geqslant D_i$, is $L(\alpha_i)=Center(\bullet^*(((R_{Alarm_i}(a_i))^{C_i-1})\text{ш}\bullet^{D_i-C_i})\bullet^{T_i-D_i}\bullet^*)^*)$.

2.3. Time valid behaviors for task systems

To model concurrency, we use the homogeneous product of regular languages, which is defined in the following way:

Definition 5. Let Σ_1 and Σ_2 be finite alphabets, and $L_1\subset\Sigma_1^*$ and $L_2\subset\Sigma_2^*$.

- Let $\alpha=(\alpha_i)_{i\in[1,|\alpha|]}\in L_1$ and $\beta=(\beta_i)_{i\in[1,|\beta|]}\in L_2$. The homogeneous product of α and β is the word $\alpha\Omega\beta\in(\Sigma_1\times\Sigma_2)^*$, defined in the following way: $|\alpha|\neq|\beta|\Rightarrow\alpha\Omega\beta$ is not defined and

$$|\alpha|=|\beta| \;\Rightarrow\; \alpha\Omega\beta=\begin{pmatrix}\alpha_1\\ \beta_1\end{pmatrix}\begin{pmatrix}\alpha_2\\ \beta_2\end{pmatrix}\ldots\begin{pmatrix}\alpha_{|\alpha|}\\ \beta_{|\beta|}\end{pmatrix}.$$

- The homogeneous product of L_1 and L_2 is $L_1\Omega L_2=\{\alpha\Omega\beta,\alpha\in L_1,\beta\in L_2\}$.

Ω is a binary operator. Then, on an algebraic plan, $((a)\Omega(b))\Omega(c)$ is equal to $\binom{\binom{a}{b}}{c}$, and $(a)\Omega((b)\Omega(c))$ is equal to $\binom{a}{\binom{b}{c}}$. However, by using the equivalence

$$\begin{pmatrix}a\\ b\\ c\end{pmatrix}\equiv\begin{pmatrix}a\\ \begin{pmatrix}b\\ c\end{pmatrix}\end{pmatrix}\equiv\begin{pmatrix}a\\ b\\ c\end{pmatrix},$$

we consider that $((a)\Omega(b))\Omega(c)=(a)\Omega((b)\Omega(c))=(a)\Omega(b)\Omega(c)$. This is why, in the following, we consider that Ω is associative, and we use the notation $\Omega_{i=1}^{i=n}L_i$.

Since $L(\tau_i)$ is a center of regular language, there is no task whose set of time valid behaviors is reduced by the homogeneous product. This comes from the property [3] $\forall j \in [1,n], \pi_j(\Omega_{i=1}^{i=n}(L(\tau_i))) = L(\tau_j)$, which is valid for centers of regular languages. Note that, since periodic and sporadic tasks are modeled by regular languages, this property stands for the two classes of tasks.

To integrate task interdependence (communication, resource sharing) between $(\tau_i)_{i\in[1,n]}$ (since we only use properties of regular languages, τ_i can be periodic or sporadic), we use Arnold–Nivat's technique [2]: each resource (shared resource or communication message) is modeled by a virtual task ξ_R, used to trace its states (busy/idle, for instance). $(\Omega_{i=1}^{i=n} L(\tau_i))\Omega L(\xi_R)$ collects the behaviors of the system composed of both the tasks $(\tau_i)_{i\in[1,n]}$ and the resource R. Consider now the set S of the instantaneous configurations compatible with the resource protocol. $((\Omega_{i=1}^{i=n} L(\tau_i))\Omega L(\xi_R)) \cap S^*$ collects the sole behaviors of $(\tau_i)_{i\in[1,n]}$ compatible with the resource protocol management.

The class of regular languages centers is not closed by intersection. Here, this property means that sharing a critical resource can lead a real-time task system to miss at least one deadline. Since we are interested in the set of time valid behaviors, we only consider the subset of $((\Omega_{i=1}^{i=n} L(\tau_i))\Omega L(\xi_R)) \cap S^*$ that collects the time valid behaviors (i.e. the scheduling sequences that can indefinitely be extended according to the system time contraints) for the whole task system. Following the same reasoning than for a single task, we define the set of time valid behaviors of a task system in the following way:

Definition 6. Let $(\tau_i)_{i\in[1,n]}$ be a task system and $(R_j)_{j\in[1,r]}$ a set of synchronization constraints. A *time valid behavior* of $(\tau_i)_{i\in[1,n]}$ according to resources $(R_j)_{j\in[1,r]}$ is an element of the language

$$L((\tau_i)_{i\in[1,n]}) = Center\left(\pi_{[1,n]}\left(\left(\left(\underset{i=1}{\overset{i=n}{\Omega}}(L(\tau_i))\right)\Omega\left(\underset{i=1}{\overset{i=r}{\Omega}}(L(R_i))\right)\right)\cap S^*\right)\right).$$

3. Time validation

By using our model we give here a predicate useful to decide the feasibility of a task system. In this section, τ_i can be sporadic or periodic, since we only use properties of regular languages. We denote by $\Pi_{1,n}^{1,r}$ the synchronized product $\pi_{[1,n]}(((\Omega_{i=1}^{i=n}(L(\tau_i)))\Omega(\Omega_{i=1}^{i=r}(L(R_i))))\cap S^*)$. This language is partioned into $Center(\Pi_{1,n}^{1,r}) \cup (\Pi_{1,n}^{1,r} \setminus Center(\Pi_{1,n}^{1,r}))$. $Center(\Pi_{1,n}^{1,r})$ collects the set of time valid behaviors, and $\Pi_{1,n}^{1,r} \setminus Center(\Pi_{1,n}^{1,r})$ the set of time unvalid behaviors: since they do not belong to the center of the language, they model the CPU allocation sequences such that it is sure that, in the future, at least one of the tasks will miss a deadline. Then, the center of $\Pi_{1,n}^{1,r}$ exactly gives the set of time valid behaviors. The decision predicate for the feasibility of a given task system is then $Center(\Pi_{1,n}^{1,r}) \neq \emptyset$.

[3] π_j is the projection $(x_i)_{i\in[1,n]} \to x_j$. This notation is naturally extended to intervals: $\pi_{[a,b]}$ is the projection $(x_i)_{i\in[1,n]} \to (x_i)_{i\in[Max(1,a),Min(b,n)]}$.

4. Using generating functions to decide feasibility of sporadic tasks

Our goal is to decide the feasibility of sporadic tasks from the sole knowledge of the behaviors of the periodic component of the system. Then, when an alarm signal occurs, we must decide from the knowledge of the current state of the periodic component if the corresponding sporadic task α can be scheduled. Then, one must find a processor idle for C_α time units in the next D_α time units. To answer to this question, we associate the periodic component model (the finite automaton) with additional informations, whose role is to know a *good* approximation of the early future.

This additive information will be implemented in the model, thanks to generating functions. In Section 4.1, we show that useful functions are multivariate generating series. Then, in Section 4.2, we show how to use these functions to decide the feasibility of sporadic tasks. Finally, in Section 4.3, we show that this technique is modular.

4.1. The used multivariate series and their semantics

Let us consider a task system[4] composed of n periodic tasks $(\tau_i)_{i\in[1,n]}$ and p sporadic tasks $(\alpha_i)_{i\in[n+1,n+p]}$. Periodic tasks share r critical resources. Each α_j is independant of any other task. We suppose that the periodic component of the system is feasible, i.e. $Center(\Pi_{1,n}^{1,r}) \neq \emptyset$. The whole system is feasible if

$$\forall j \in [1,p], \quad \forall t \in \left[\min_{i\in[1,n]}(r_i^p), +\infty\right[\cap \left\{ \begin{array}{c} \text{Possible activation} \\ \text{dates of } \alpha_j \end{array} \right\},$$

the activation of α_j at time t does not induce a temporal fault.

We denote by $\Delta_{1,n}^{1,r}$ the minimal deterministic automaton such that $L(\Delta_{1,n}^{1,r}) = Center(\Pi_{1,n}^{1,r})$. A state i of $\Delta_{1,n}^{1,r}$ models a state of the system $(\tau_i)_{i\in[1,n]}$. If an alarm event occurs when $\Delta_{1,n}^{1,r}$ is in state i, the corresponding sporadic task α_i must be scheduled (according to its time specifications) in the next D_i time units. There are two possibilities: firstly, there is no edge issued from i compatible with the time specifications of α_i; secondly, there exists edges issued from i that are compatible with the time specifications of α_i. We must be able to decide if such outgoing edges exist and, if there exists at least one, we must be able to select the *good* edges. To solve this problem, we must associate to each edge e issued from i the information $(D_i, K_{D_i})_{i\in[n+1,n+p]}$, where K_{D_i} is the maximal number of idleness instants of the processor in the D_i next time units. With these informations, for each e issued from i, we can decide if choosing e is compatible with α_i time specifications.

When the scheduling context concerns multi-processor architectures, other informations are needed: at time t, many tasks can be simultaneously active, and processors can be idle. We must separate the *simultaneous idle time units*, that concern different processors, from *consecutive idle time units*, that concern any processor: since tasks are sequential, two consecutive idle time units can be allocated to the same task, but two simultaneous idle time units cannot be. To decide if an edge e is compatible with

[4] Let us recall that tasks of the application cannot be parallelized.

time specifications of α_i, one must know if choosing e as first step is compatible with the allocation to α_i of C_i consecutive CPU time units in the next D_i time units. Then, if π is the number of processors, we associate with e the information $(D_i,(K_{k,D_i})_{k\in[0,\pi]})$, where K_{k,D_i} is the number of time units, in the next D_i time units, where exactly k processors are simultaneously idle. If k sporadic tasks are active, the feasibility of the system comes from the predicate $K_{k,D_i}\geqslant C_i$. In the mono-processor case, this information coincides with K_{D_i}. In both cases, if there are more sporadic tasks than processors, they must be scheduled on the processors: this operation can be solved thanks to any scheduling technique.

For a finite automata family $(A_i)_{i\in I}$ the homogeneous product construction process consists in computing an edge e of $\Omega_{i\in I}(A_i)$ from a family $(e_i)_{i\in I}$ of edges of the A_i's. Let $\phi(e)$ be the information $(D_i,(K_{k,D_i})_{k\in[0,\pi]})$ that we associate with e, and let us consider, for instance, on a 2-processor configuration, edges e_1 and e_2, respectively associated with the following informations:

e_1 We dispose of in the three next time units, one processor idle for three time units, and two processors simultaneously idle for two time units. The idleness configuration belongs to $\{(\bullet)(\bullet)(\_),(\bullet)(\_)(\bullet),(\_)(\bullet)(\bullet)\}$.

e_2 In the next three time units, the processor—there is only one—is idle during one time unit. The idleness configuration belongs to $\{(\bullet)()(),()(\bullet)(),()()(\bullet)\}$.

The product automaton contains an edge e, which models the simultaneity $e_1//e_2$. The respective idleness configurations of e_1 and e_2 lead e idleness configuration to belong to

$$\left\{\begin{array}{c}(\vdots)(:)(\_),(\vdots)(\_)(:),(:)(:)(:),(:)(\vdots)(\_),\\ (\_)(\vdots)(:),(:)(\_)(\vdots),(\_)(:)(\vdots)\end{array}\right\}.$$

One must remark that there are elements of this set that correspond to different idleness configurations (in terms of feasibility capacities for sporadic tasks). Then, knowing $\phi(e_1)$ and $\phi(e_2)$ is useless to build $\phi(e)$: the number of time units where there are idle processors must be completed by the exhaustive description of each configuration, because $\phi(e)$ is permutation dependent.

To answer to this problem, we enrich $\phi(e_1)$ by associating with each transition the information $(D,(Idle_i)_{i\in[1,D]})$, where $D=\mathrm{Max}_{\alpha\in\{(\alpha_i)_{i\in[1,p]}\}}(D_\alpha)$ is the maximal deadline of the sporadic tasks, and $Idle_i\in\mathbb{N}$ is the number of idle processors in the next ith time unit. This strategy leads $\phi(e)$ to be directly computable from $\phi(e_1)$ and $\phi(e_2)$ and, more generally, to be computable from $(\phi(e_i))_{i\in I}$.

To implement $\phi(e)$ into generating series, we associate with each state s of $\Delta_{1,n}^{1,r}$ the function $F_s(y)\in\mathbb{N}\ll(p_i)_{i\in[0,\pi]}\gg[y]$, where

- π is the number of available processors.
- y's power collects deadlines (i.e., here, lengths of words). For instance, in $y^{12}p_1^4p_2^7$, y^{12} means that we are looking forward the next 12 time units.
- In the same way, the p_i's are designed to implement the vector $(Idle_i)_{i\in[1,D]}$. For instance, in $p_1p_7p_5p_1p_4$, the predicate p_5 *appears in* 3rd *position* means that dur-

ing the next 3rd time unit, there will be exactly 5 processors simultaneously idle. More generally, p_i appearing in jth position in the sequence means that in the next jth time unit, there will be exactly i processors simultaneously idle. Note that this implementation of $\phi(e)$ imposes to consider non-commutative series. This is why $F_s(y) \in \mathbb{N} \ll (p_i)_{i\in[0,\pi]} \gg [y]$.

$\Delta_{1,n}^{1,r}$ is converted into a linear system $M \times F = T$ by the use of Chomsky–Schützenberger's technique [6]. M and T are, respectively, a matrix and a vector whose coefficients belong to $\mathbb{N} \ll (p_i)_{i\in[0,\pi]} \gg [y]$, and F the unknown vector. M and T are obtained by the classical constructive technique.

4.2. Feasibility decision for sporadic tasks

In this section, we show how to use these generating functions, in order to decide the feasibility of a given sporadic task.

Let α be a sporadic task of time characteristics $(C_\alpha, D_\alpha, T_\alpha)$, whose activator alarm event occurs when the periodic component of the system is in state i. We denote by E_i the set of edges outgoing from i, and F_e the generating function associated with edge e. α is feasible if there exists an edge $e \in E_i$ compatible with α's time characteristics.

$[y^{D_\alpha}]F_e(y, (p_k)_{k\in[1,\pi]})$ is a polynomial of $\mathbb{N} \ll (p_k)_{k\in[1,\pi]} \gg$, that collects the words which model the possible scheduling sequences for the next D_α time units, when selecting e as first step. α is feasible if one of its monomials corresponds to a sequence with at least C_α consecutive idle time units. The operational semantics associated with the p_i's leads this property to stand as soon as there exists a monomial m in $[y^{D_\alpha}]F_e(y, (p_k)_{k\in[1,\pi]})$ such that $|\lambda(m)| \geqslant C_\alpha$, where λ is the morphism defined on the p_i's on the following way: $\lambda(p_0) = \varepsilon$ and $i > 0 \Rightarrow \lambda(p_i) = a$. On the opposite—no $e \in E_i$ satisfies this property—α is not feasible, i.e. the system cannot accept the alarm signal according to its time constraints: operational specifications and functional conception of the system must be reviewed!

If there are many sporadic tasks concerned, the criterion is the conjunction, for all tasks, of the existence of such sequences (because we assume sporadic tasks to be independent from others). In this case, we must extend the criterion to words that model valid scheduling sequences for all concerned sporadic tasks.

4.3. Modularity of the enriched model

In this section, we show that the computing of the generating series associated with the edges of the automaton is modular.

The structure of $\Delta_{1,n}^{1,r}$ [11] leads F_e's to be of the form [7] $N_1(y) + N_2(y)/(1 - y^P D)$, where N_1, N_2 and D are polynomials of $\mathbb{N} \ll (p_k)_{k\in[0,\pi]} \gg [y]$ and $P = \mathrm{lcm}_{i\in[1,n]}(T_i)$ is the lcm of the periods of the periodic tasks. We saw in Section 2.3 that $\Pi_{1,n}^{1,r}$ is computed by the homogeneous product operator. On an algorithmic plan, we built the automaton $\Delta_{1,n}^{1,r}$, which accepts $Center(\Pi_{1,n}^{1,r})$.

Each edge e of $\Delta_{1,n}^{1,r}$ is obtained from the sole n-uple $(e_i)_{i\in[1,n]}$ of edges of the automata $(A_i)_{i\in[1,n]}$ which accept the $L(\tau_i)$'s. In this part, we show that this property

stands for the extended model: the generating function associated with e can be computed from the sole generating functions associated with the $(e_i)_{i\in[1,n]}$. We map the generating functions associated with edges $(e_i)_{i\in[1,n]}$ to weighted automata, and we use algebraic handlings on these automata to compute the generating function associated with e.

4.3.1. Linear representation of weighted automata

For weighted automata [14], we follow the definition:

Definition 7. Let $\mathbb{K}$ be a semiring. A $\mathbb{K}$-automata is a 5-uple $(\Sigma, Q, I, T, \delta)$, where Σ is an alphabet, Q a finite set, I (the initial states) and T (the terminal states) $\mathbb{K}$-subsets [5] of Q, and δ a $\mathbb{K}$-subset of $Q\times A\times Q$.

Such automata are usually presented under a linear form (λ,μ,γ), where $\lambda\in\mathbb{K}^{1\times n}$, $\mu\in\mathbb{K}^{n\times n}$ and $\gamma\in\mathbb{K}^{n\times 1}$ [5]. For each $a\in\Sigma$, we build a matrix $\mu(a)\in\mathfrak{M}_{Q\times Q}(\mathbb{K})$ in the following way: $\forall(i,j)\in Q^2, \mu_{i,j}(a)=E(i,a,j)$. In the $\mathbb{K}$ subset I (resp. T), noninitial (resp. terminal) states are characterized by a zero weight. Then, I and T can be viewed as vectors $\lambda=(\lambda_1,\ldots,\lambda_n)$ and $\gamma=(\gamma_1,\ldots,\gamma_n)$ such that $\forall s\in Q,\ \lambda_s=I(s)$ and $\gamma_s=T(s)$. Let Б be a weighted automaton. For $\omega\in A^*$, $IE(\omega)T\omega$ gives the acceptation value of ω. The behaviors of Бis the series $C(\text{Б})=\Sigma_{\omega\in A^*} IE(\omega)T\omega$. The 3-ple (λ,μ,γ) is called linear representation of $C(\text{Б})$. This notion coincides with the notion of language when weights belong to $\{0,1\}$.

A series S is recognizable if there exists a weighted automaton Б(and then a linear representation (λ,μ,γ)) such that $C(\text{Б})=S$ [9]. In this paper, a recognizable series S is denoted by (λ,μ,γ).

4.3.2. Series operators to model task composition

In this work, we deal with polynomials of $\mathbb{N}\ll(p_k)_{k\in[0,\pi]}\gg\ll y\gg$, that are associated with states and edges of the automaton $\Delta_{1,n}^{1,r}$. Then, this set of polynomials is designed to be our semi-ring $\mathbb{K}$. To compose polynomials of this set, we use the two following different operators.

Let us define the first operator. Let $s_1=p_1p_3p_0p_2y^4$ and $s_2=p_3p_1p_2y^3$ be two polynomials of $\mathbb{K}$. Each of them models a specific CPU load configuration, i.e. a specific forward path in the automaton.

Outgoing from state i, we must select the best edge, in terms of CPU load, i.e. define a max operator, obviously based on an order. Many orders can be considered, dependent on the criterion we search to optimize. We do not detail this point, and we consider the order of maximal width. For instance, in the following example, we get

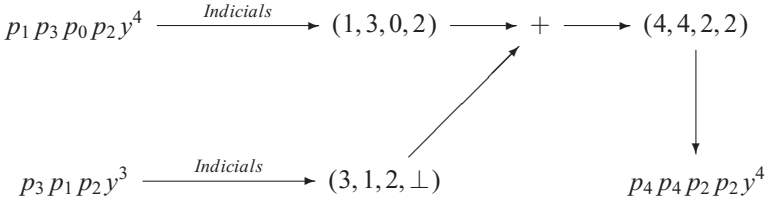
$$s_1 = p_1p_3p_0p_2y^4 \xrightarrow{indicals} (1,3,0,2) \xrightarrow{min} 0,$$

$$s_2 = p_3p_1p_2y^3 \xrightarrow{indicals} (3,1,2) \xrightarrow{min} 1.$$

Then, $max(s_1,s_2)=s_2$. $(\mathbb{K},max)$ has a structure of commutative monoid.

[5] A $\mathbb{K}$-subset S of E is a mapping $E\to\mathbb{K}$, where $\mathbb{K}$ is a semiring.

The second-operator models simultaneity of sequences. Since in p_k, the number of processors simultaneously idle is modeled by k, this simultaneity operator comes from the addition of indicials. Let us denote by $\otimes$ this operator. For instance, we get $p_1p_3p_0p_2y^4 \otimes p_3p_1p_2y^3 = p_4p_4p_2p_2y^4$. This operation is implemented in the following way:



$\otimes$ is internal and associative in $\mathbb{K}$, and its zero element is 1: $(\mathbb{K}, \otimes)$ is a monoid, and then $(\mathbb{K}, max, \otimes)$ is a semiring.

4.3.3. Weighted automata associated with generating functions

We consider now the state i of $\Delta^{1,r}_{1,n}$. We saw in Section 4.3 that the generating function associated with i is $F(y) = N(y)(yQ(y))^* = N(y)/(1 - yQ(y))$, where N and Q belong to $\mathbb{N} \ll (p_k)_{k\in[0,\pi]} \gg [y]$. Let us denote by n_i and q_i the respective general terms of the polynomials N and Q. We get $N(y) = \sum_{i=0}^{i=m} n_i y^i$ and $Q(y) = \sum_{i=0}^{i=P} q_i y^i$. Note that P is the lcm of the periods of the periodic tasks: this is a property of Q that comes from the cyclicity of scheduling sequences [13]. As N is obtained by a reduction to the same denominator, we get $P < m$.

The following result makes the model modular (we denote $Max(x, y)$ by $x \bowtie y$):

Theorem 1. *Let*

$$F(y) = \frac{\sum_{i=0}^{i=P} n_i y^i}{1 - y \sum_{i=0}^{i=m} q_i y^i} F(y)$$

is the 3*-ple* (λ, μ, γ)*, with*

$$\lambda = (\, n_0 - n_P, \quad 0_{1\times((P\bowtie m)-P)}\,) \qquad \gamma = \begin{pmatrix} 1 \\ 0_{(P\bowtie m)\times 1} \end{pmatrix}$$

$$\mu = \left(\begin{array}{c|c} \begin{array}{ccccc} q_0 & & & & q_m \\ 1 & 0 & & & 0 \\ 0 & & \ddots & & \\ 0 & 0 & & 1 & 0 \end{array} & 0_{((P\bowtie m)-m)\times(m+1)} \\ \hline 0_{(m+1)\times(m+1)} & 0_{((P\bowtie m)-m)\times((P\bowtie m)-m)} \end{array}\right)$$

Proof. We build the automaton accepting $yQ(y) = y\sum_{k=0}^{k=m} q_k y^k$. Note that each of the monomials $q_k y^{k+1}$ labels a unique path which starts at state (y^{m-k+1}/D). The linear

representation of this automaton is

$$\lambda^1 = (0, q_0, \ldots, q_m), \quad \gamma^1 = \begin{pmatrix} 1 \\ 0_{(m+1)\times 1} \end{pmatrix} \quad \text{and} \quad \mu^1(y) = \begin{pmatrix} 0 & & & & 0 \\ 1 & \diagdown & & \diagup & \\ 0 & & \diagdown & & \vdots \\ 0 & \cdots & 0 & 1 & 0 \end{pmatrix} .$$

The linear representation of $(yQ(y))^*$ is obtained from $(\lambda^1, \mu^1, \gamma^1)$ by using the results of [8] (classical operators on series are expressed in term of weighted automata 'linear form'). From the automaton associated with S, the automaton associated with S^* can be computed if $\lambda\gamma = 0$, which is the case in our study. We obtain the automaton $(\lambda^2, \mu^2, \gamma^2)$, where

$$\lambda^2 = (0_{1\times(m+2)}, 1), \quad \gamma^2 = \begin{pmatrix} 1 \\ 0_{(m+1)\times 1} \\ 1 \end{pmatrix} \quad \text{and} \quad \mu^2 = \begin{pmatrix} q_0 & \cdots & q_m & 0 & 0 \\ 1 & 0 & & & 0 \\ 0 & \diagdown & & \diagup & \vdots \\ 0 & \cdots & 0 & 1 & \\ q_0 & \cdots & q_m & 0 & 0 \end{pmatrix} .$$

Now, we make i the unique initial state of this automaton, weighted by m. Each monomial accepted by the automaton $y(Q(y))^*$ is now accepted multiplied by my^i: the modified automaton accepts $my^i/(1 - yQ(y))$.

This operation is performed for each $i \in [0, P]$. The entry in state 1 is now valuated by n_0 and no more by 1. So, the obtained automaton accepts $1/(1 - yQ(y)) \sum_{i=0}^{i=P} n_i y^i$, i.e. $N(y)/(1 - yQ(y))$. From the expression of the behavior of this automaton, we obtain its linear form. □

Duchamp et al. [8] shows that the Hadamard product $r \odot s$ of functions r and s can be computed through the linear form of the associated automata $(\lambda^r, \mu^r, \gamma^r)$ and $(\lambda^s, \mu^s, \gamma^s)$. The series $r \odot s$ is the behavior of $(\lambda^r \otimes \lambda^s, \mu^r \otimes \mu^s, \gamma^r \otimes \gamma^s)$, where $\otimes$ is the tensorial product, that is here the product operation of $\mathbb{K}$.

In this way, we compute the generating function associated with a product edge (e_1, e_2) from the sole knowledge of the generating functions associated with e_1 and e_2. The enriched model is then modular.

5. Conclusion

The main contribution of this work is to give a constructive technique to decide the feasibility of systems that contain sporadic tasks. If sporadic tasks have to be synchronized with periodic, decision is given by using the basic model. If not, the extended model gives a technique to avoid the construction of very expensive structures. We give in Section 4.3 a constructive result that makes this extended model modular. Many works are ongoing. Firstly, we intend to extend this result to the real case of

sporadic tasks in which CPU load is not fixed (this case concerns the very frequent case of tasks whose body contains $if \ldots then \ldots else$ statements). Secondly, we are improving the model by searching to avoid the effective construction of some parts of the automata. Finally, when a task configuration is not feasible, we search a strategy to give diagnostic informations to the user, to help him to review the time specifications of the system, in order to make it feasible.

References

[1] R. Alur, D. Dill, A theory of timed automata, Theoret. Comput. Sci. 126 (1994) 183–235.
[2] A. Arnold, Finite Transition Systems, Prentice-Hall, Englewood Cliffs, NJ, 1994.
[3] S.K. Baruah, L.E. Rosier, R.R. Howell, Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor, Real-Time Systems, Kluwer Academic Press, 1990, pp. 301–324.
[4] J.P. Beauvais. Étude d'Algorithmes de Placement de Tâches Temps Réel Périodiques Complexes dans un Système Réparti, Ph.D. Thesis, École Centrale de Nantes, France, 1996.
[5] J. Berstel, C. Reutenauer, Rational series and their languages, in: EATCS Monographs on Theoretical Computer Science. Springer, Berlin, 1988.
[6] N. Chomsky, M.P. Schützenberger, The algebraic theory of context-free languages, Computer Programming and Formal Systems (1963) 118–161.
[7] J.P. Dubernard, D. Geniet, Validation temporelle d'applications temps-réel strictes au moyen d'automates finis et de séries génératrices, in: Proc. of Formal Power Series and Algebraic Combinatorics, to appear.
[8] G. Duchamp, M. Flouret, É. Laugerotte, J.G. Luque, Direct and multiple laws for automata with multiplicities, Theoret. Comput. Sci. 267 (2001) 105–120.
[9] M. Fliess, Sur divers produits de séries formelles, Bull. Sci. Math. 104 (1974) 181–191.
[10] G. Florin, S. Natkin, Les réseaux de petri stochastiques, Tech. Sci. Informat. 4 (1) (1985) 143–160.
[11] D. Geniet, Validation d'applications temps-réel à contraintes strictes à l'aide de langages rationnels, in: RTS'2000, Teknea, 2000, pp. 91–106.
[12] R.L. Graham, E.W. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, Ann. Discrete Math. 5 (1979) 287–326.
[13] E. Grolleau, Ordonnancement Temps-Réel Hors-Ligne Optimal à l'Aide de Réseaux de Petri en Environnement Monoprocesseur et Multiprocesseur, Ph.D. Thesis, University Poitiers, 1999.
[14] W. Kuich, A. Salomaa, Semirings, Automata, Languages, Springer, Berlin, 1985.
[15] J.Y.T. Leung, M.L. Merill, A note on preemptive scheduling of periodic real-time tasks, Inform. Process. Lett. 11 (3) (1980) 115–118.
[16] C.L. Liu, J.W. Layland, Scheduling algorithms for multiprogramming in a hard real-time environment, J. ACM 20 (1) (1973) 46–61.