VALIDATION TEMPORELLE D'APPLICATIONS TEMPS-RÉEL STRICTES AU MOYEN D'AUTOMATES FINIS ET DE SÉRIES GÉNÉRATRICES

JEAN-PHILIPPE DUBERNARD AND DOMINIQUE GENIET

RÉSUMÉ. Nous avons mis en place une technique de validation temporelle d'application temps-réel qui s'exprime en termes de langages rationnels. Nous enrichissons ici cette technique en associant aux langages rationnels des fonctions génératrices, pour donner au modèle une expressivité suffisante pour valider temporellement des systèmes temps réel généraux, c'est à dire comportant des composantes périodiques et de type *alarme*, et destinés à fonctionner sur des cibles multi-processeurs. La contribution principale de cet article est de montrer que l'on peut valider un système temps-réel avec des tâches périodiques et apériodiques à partir de sa seule composante périodique.

ABSTRACT. We set a technique, based on regular languages, to validate hard real-time software. Here, we upgrade this technique by associating the regular languages with generating functions, to give the model enough expressivity to validate general hard real-time systems, i.e. systems composed of both periodic and sporadic (or *alarm*) tasks, and designed for multi-processor target architectures. The main contribution of this article is to show that we can validate a real-time system, which contains periodic and aperiodic tasks, only from its periodic component.

1. INTRODUCTION

Un système informatique est temps-réel lorsque, devant assurer le pilotage d'un procédé physique, il se doit de fonctionner à une vitesse compatible avec l'évolution du procédé dans son environnement physique.

1.1. **Les systèmes temps-réel.** Par essence même, un tel système est donc d'une part **réactif** (car devant prendre en compte, à la volée, les informations entrantes qui lui indiquent l'évolution du procédé dans son environnement physique), et d'autre part **concurrent** (car l'ensemble des opérations relatives au pilotage –observations de l'environnement et actes de conduite– doivent être réalisées simultanément). Il est donc naturellement composé d'un ensemble de **tâches** élémentaires, chacune d'elles implémentant une réaction (ou une partie de réaction) que le système devra fournir à une sollicitation extérieure (qui s'exprime sous la forme d'un **événement**).

Certaines informations entrant dans le système suivent des flots réguliers : ce sont, généralement, des données transmises par des capteurs périodiques. Les tâches de lecture et de traitement de ces informations doivent donc être synchronisées avec les capteurs en question. Ce sont des tâches **périodiques**. Si le système comporte n tâches périodiques, nous les appelons ici $(\tau_i)_{i\in[1,n]}$. Dans le modèle classique de tâches temps-réel [1], la spécification temporelle de τ_i est constituée par la donnée de quatre caractéristiques :

- La **date de première activation** (notée $\mathbf{r_i}$) est l'instant de création de τ_i (l'origine des temps est $\min_{i\in[1,n]}(r_i)$)
- La **période** (notée $\mathbf{T_i}$) est le délai séparant deux activations successives de τ_i

- Le **délai critique** (noté $\mathbf{D_i}$) est le délai séparant la date d'activation d'une occurrence de τ_i de la date à laquelle l'exécution de cette occurrence doit être terminée (cette date est nommée **échéance** de l'occurrence de τ_i)
- La **durée d'exécution** (notée $\mathbf{C_i}$) est la durée pendant laquelle chaque occurrence de τ_i devra disposer d'un processeur pour pouvoir terminer son exécution

Notons que *date de première activation*, *période* et *délai critique* sont imposés par le procédé à piloter. À ce titre, ce sont des paramètres hérités. À l'opposé, la *durée d'exécution* est un paramètre synthétisé à partir des caractéristiques du processeur cible et du corps de τ_i.

Les signaux d'alarmes, ou encore les interventions d'un éventuel superviseur, constituent des flots d'informations entrantes non-périodiques. Les tâches de prise en compte de ces informations, activées uniquement lorsque c'est nécessaire, sont appelées tâches **sporadiques** ou **apériodiques.** Une tâche sporadique est caractérisée par la connaissance d'un délai minimum séparant deux de ses occurrences successives. Dans le cas où un tel délai ne peut être garanti, la tâche est dite apériodique. Si le système comporte p tâches sporadiques [1], nous les appelons ici $(\alpha_i)_{i\in[1,p]}$.

1.2. **Techniques de validation.** Avant sa mise en exploitation, tout système doit être validé, de façon à garantir que son comportement correspond aux besoins affichés. Ceci est particulièrement vrai pour les applications de contrôle-commande. Pour les applications temps-réel, la validation comporte deux aspects. D'une part la classique validation fonctionnelle, mais également la validation temporelle [2] : il s'agit de garantir que, quelle que soit l'évolution du procédé, le système temps-réel est apte à traiter dans les temps (c'est à dire *en respectant ses contraintes temporelles*) l'ensemble des signaux susceptibles de se présenter. Le pilotage d'un système temps-réel est classiquement abordé suivant deux approches différentes (voir une synthèse fournie dans [2]) : l'ordonnancement **en ligne** et l'ordonnancement **hors-ligne.**

1.2.1. *Politiques en ligne.* Le pilotage en ligne consiste à implanter dans la machine cible un ordonnanceur qui s'appuie sur une politique spécifique (basée sur des priorités, où sur les paramètres temporels des tâches, par exemple) : chaque événement (signal ou commutation) voit l'ordonnanceur élire **à la volée** la prochaine tâche en fonction de sa politique. Cette approche a motivé un grand nombre de travaux ([3], par exemple, pour les systèmes périodiques, [4] propose des politiques adaptées à la gestion des ressources critiques, on trouvera dans [5] une synthèse sur l'intégration des sporadiques dans cette approche). Si l'approche en ligne est souple, elle est toutefois limitée par sa non-optimalité [3] dès que les tâches sont interdépendantes (c'est à dire à peu près tout le temps, dans les cas réels !).

1.2.2. *Politiques hors-ligne.* En toute généralité, le problème de décision de l'ordonnançabilité d'un système temps-réel est NP-complet [6]. Il est donc illusoire de rechercher une politique en ligne optimale dans le cas général. Cet état de fait a motivé la mise en place de stratégies basées sur l'énumération exhaustive des ordonnancements possibles d'une configuration donnée : les stratégies hors ligne. Ici, l'énumération permet, dès lors que la configuration est ordonnançable, d'exhiber une séquence valide qui, implantée dans un séquenceur, pilotera l'application.

---

[1] Nous nous focalisons ici sur les systèmes constitués de tâches périodiques et sporadiques.

[2] Un *bon* résultat rendu hors délai est faux.

[3] Un algorithme d'ordonnancement est **optimal** si il n'échoue que sur les configurations de tâches qui ne peuvent être ordonnancées (voir [2] pour une définition formelle de l'optimalité).

Pour les systèmes périodiques [4], la production d'une telle séquence suppose la connaissance de la durée minimale de simulation. [7] en fournit une borne, et [2] généralise ce résultat.

Les méthodes hors-ligne s'appuient sur des modèles de tâches à base de systèmes à états (automates et réseaux de Petri). Certains sont à temps explicite [8] [9] [10] (on parle de modèles **temporisés**), d'autres à temps implicite [11]. Les modèles à temps explicite sont exploités via des techniques de simulation, ceux à temps implicite via des techniques d'analyse. Dans les deux cas, un premier axe d'étude consiste à produire des techniques de modélisation exhaustives (i.e. qui prennent en compte l'ensemble des configurations réelles possibles). Par ailleurs, les complexités de décision étant exponentielles, un axe majeur des travaux est la mise en œuvre de techniques d'amélioration des temps de calcul (essentiellement par la détection précoce de branches correspondant à des cas d'échec). Les systèmes périodiques interdépendants pour lesquels les tâches ont une durée d'exécution fixée sont généralement pris en compte. Ce cas de figure est tout à fait irréaliste, puisque dès que le corps d'exécution d'une tâche intègre un test du type *If... Then... Else...*, on sort du modèle. Dans [12], nous montrons que notre approche permet d'étendre l'analyse aux systèmes de tâches comportant des instructions de choix.

1.2.3. *Validation.* Valider une application temps-réel (i.e. un systême de tâches) consiste à prouver que, quelle que soit l'évolution du procédé qu'elle pilote, elle ne se trouvera jamais en situation de violation de ses contraintes temporelles. Dans le cas où l'objectif est un pilotage en ligne, la validation s'obtient à l'aide de techniques de simulation. Dans le cas hors-ligne, on utilise des techniques de model-checking pour produire une séquence d'ordonnancement en accord avec les contraintes spécifiées.

1.3. **Cadre du travail.** Nous avons introduit dans [13] une technique de modélisation de tâches temps-réel à base d'automates finis, et nous avons montré qu'elle prend en compte les configurations de tâches périodiques interdépendantes à durée d'exécution fixe. Nous étendons la validité de la méthode au cas des tâches à durées variables dans [12], et une première approche pour les sporadiques est proposée dans [14].

Le temps y étant implicite, notre modèle présente, par rapport aux autres approches orientées modèles (automates temporisés [15], réseaux de Petri [2]) un certain nombre d'avantages. D'une part, il permet de disposer de processus de décision basés sur les techniques d'analyse des automates finis, dont la puissance, l'efficacité et la modularité sont établies depuis longtemps. Par exemple, le résultat central de [2], qui établit la cyclicité des ordonnancements en environnement mono processeurs de systèmes de tâches interdépendants, se retrouve, dans notre approche, comme un corollaire de la rationnalité de l'ensemble des comportements valides d'une application temps-réel. La rationnalité de cet ensemble persistant en multi-processeur, nous pouvons affirmer que la cyclicité des ordonnancements persiste en multi-processeurs. D'autre part, l'aspect comportemental de notre approche permet l'expression de propriétés assez fines, plus difficilement exprimables par d'autres approches, soit parce qu'elles nécessitent des manipulations équationnelles élaborées (approches *temporisées*), soit parce qu'elles imposent des définitions structurelles assez lourdes (approches *réseaux de Petri*, par exemple).

Ici, après une synthèse de la modélisation et une réflexion sur les méthodes de calcul les plus adaptées, nous finalisons l'enrichissement de notre modèle proposé dans [14], dans l'optique de valider, à partir de la seule composante périodique de notre modèle, des systèmes constitués de tâches périodiques et sporadiques. Cet enrichissement s'appuie sur des séries

[4] i.e. uniquement constitués de tâches périodiques.

formelles non commutatives, auxquelles nous associons une sémantique temporelle. Nous établissons ensuite la modularité de ce modèle étendu.

Ce modèle possède une expressivité suffisante pour l'étude de systèmes temps-réel quelconques **et** réels. Par rapport aux modèles temporisés [10][15], il présente l'intérêt d'expliciter le temps, et donc d'utiliser les techniques de model checking les plus performantes qui soient : celles du modèle des automtes finis déterministes.

1.4. **Plan de l'article.** En section 2, nous présentons le modèle temporel utilisé pour la validation de systèmes de tâches. En section 3, nous montrons comment prendre en compte les phénomènes d'interdépendance, et la synchronisation. En section 4, nous montrons comment, à l'aide de séries commutatives et non-commutatives, nous validons des systèmes constitués de tâches périodiques et sporadiques à partir de la seule modélisation de la composante périodique du système.

2. Modélisation de tâches à contraintes strictes

Nous nous intéressons ici à la validation temporelle d'applications. Notre modèle doit donc être à même de répondre à des questions centrées sur l'aspect opérationnel (respect des échéances, donc), et non sur l'aspect fonctionnel (ce que fait le programme).

2.1. **Tâches périodiques.** Dans un premier temps, nous considérons les tâches à durée d'exécution fixes. Intéressons nous ici à la tâche τ_i [5], dont les paramètres temporels sont r_i, C_i, D_i et T_i. À partir de sa date d'activation (un entier de $r_i + T_i\mathbb{N}$), chaque occurrence de τ_i doit impérativement se voir allouer un processeur pendant exactement C_i unités de temps sur sa période. En représentant par le symbole a_i l'attribution d'un processeur à τ_i pendant une unité de temps, et par le symbole $\bullet$ la suspension de τ_i pendant une unité de temps, tout mot [6] de $a_i^{C_i}\text{Ш}\bullet^{D_i-C_i}$ correspond, sur tout intervalle temporel du type $[r_i + k.T_i, r_i + k.T_i + D_i[$, à une configuration d'allocation d'un processeur à τ_i compatible avec ses contraintes temporelles. Cet ensemble est rationnel. Prendre en compte le fait que, sur l'intervalle $[r_i + k.T_i + D_i, r_i + (k+1).T_i[$, τ_i est toujours inactive, revient à suffixer tout mot de $a_i^{C_i}\text{Ш}\bullet^{D_i-C_i}$ par $\bullet^{T_i-D_i}$. La concaténation de ces deux langages est le langage $a_i^{C_i}\text{Ш}\bullet^{D_i-C_i}\bullet^{T_i-D_i}$, également rationnel. Sur le plan opérationnel, τ_i est la suite $(\tau_{ij})_{j\in\mathbb{N}}$ de ses occurrences. Donc, allouer globalement à τ_i un processeur de manière compatible avec ses contraintes temporelles revient à allouer à chacune des occurrences τ_{ij} un processeur de manière compatible avec ses contraintes temporelles. Chaque mot ω de $a_i^{C_i}\text{Ш}\bullet^{D_i-C_i}\bullet^{T_i-D_i}$ est de longueur T_i : il décrit une configuration d'allocation **valide** [7] du processeur à une occurrence de τ_i. Donc, si $(\omega_j)_{j\in[1,n]} \in \left(\left(a_i^{C_i}\text{Ш}\bullet^{D_i-C_i}\right)\bullet^{T_i-D_i}\right)^n$, le mot $\omega_1.\omega_1\ldots\omega_n$ modélise une configuration d'allocation du processeur valide pour n occurrences de τ_i. D'une façon générale, tout mot ω de $\left(\left(a_i^{C_i}\text{Ш}\bullet^{D_i-C_i}\right)\bullet^{T_i-D_i}\right)^*$ modélise une configuration d'allocation de processeur valide pour τ_i sur n'importe quel intervalle temporel du type $[r_i + k.T_i, r_i + k.T_i + |\omega|[$, et donc, notamment, sur l'intervalle $[r_i, r_i + |\omega|[$. Étant donné que τ_i est inactive sur l'intervalle $[0, r_i[$, tout mot $\bullet^{r_i}\omega$, où $\omega \in \left(\left(a_i^{C_i}\text{Ш}\bullet^{D_i-C_i}\right)\bullet^{T_i-D_i}\right)^*$,

[5] Nous ne considérons que le cas des tâches non réentrantes, c'est à dire des tâches ne pouvant avoir deux occurences fonctionnant simultanément.

[6] L'opération de **mélange** de mots (notée Ш) se définit de la façon suivante

$$\begin{cases} \forall a \in \Sigma, a\text{Ш}\epsilon = \epsilon\text{Ш}a = \{a\} \\ \forall (a,b,\omega_1,\omega_2) \in \Sigma^2 \times (\Sigma^*)^2, a.\omega_1\text{Ш}b.\omega_2 = a.(\omega_1\text{Ш}b.\omega_2) \cup b.(a.\omega_1\text{Ш}\omega_2) \end{cases}$$

[7] c'est à dire compatible avec les contraintes temporelles de la tâche.

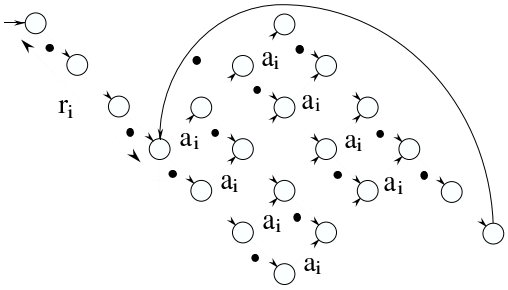

FIG. 1. Comportements de la tâche

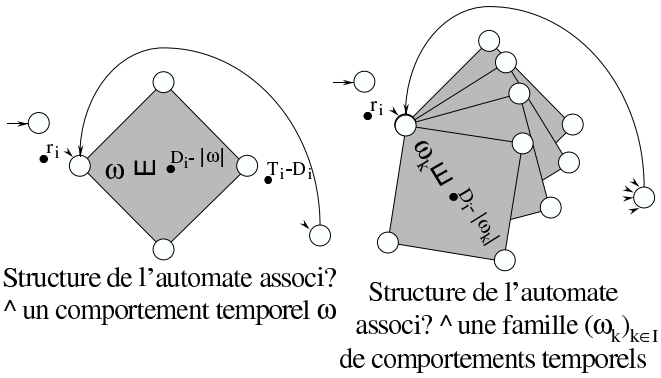

FIG. 2. Allure de l'automate associé â une tâche à durée variable

modélise une configuration d'allocation de processeur valide pour τ_i sur l'intervalle $[0, r_i + |\omega|[$. Le mot ω pouvant être de longueur arbitrairement grande (le langage est une étoile de langage rationnel), il modélise les configurations d'allocation valides pour n'importe quelle durée de vie de la tâche. Le langage $\bullet^{r_i}\left(\left(a_i^{C_i}\text{Ш}\bullet^{D_i-C_i}\right)\bullet^{T_i-D_i}\right)^*$ est rationnel, et reconnu par l'automate présenté en Figure 1. Le problème de l'ordonnancement temps-réel consiste, à un instant t donné de la vie de la tâche, à pouvoir prédire ses possibilités d'évolution dans un environnement donné. Bien entendu, le passé de la tâche est connu. Pour ce qui nous intéresse ici, ce passé est simplement l'historique des allocations CPU de τ_i un mot ω de $\{\bullet, a_i\}^*$. Par construction, ω est de la forme $\omega_1.\mu$, où $\omega_1 \in \bullet^{r_i}\left(\left(a_i^{C_i}\text{Ш}\bullet^{D_i-C_i}\right)\bullet^{T_i-D_i}\right)^*$ et

$$\exists\nu \in \{\bullet, a_i\}^* \text{ tel que } \mu\nu \in \left(a_i^{C_i}\text{Ш}\bullet^{D_i-C_i}\right)\bullet^{T_i-D_i}$$

Donc, ω est un préfixe d'un mot de $\bullet^{r_i}\left(\left(a_i^{C_i}\text{Ш}\bullet^{D_i-C_i}\right)\bullet^{T_i-D_i}\right)^*$. Par ailleurs, par construction également, quel que soit l'instant f appartenant au futur]t,+∞[, il existe un mot ω_2 dans $\left(\left(a_i^{C_i}\text{Ш}\bullet^{D_i-C_i}\right)\bullet^{T_i-D_i}\right)^*$, et tel que $\omega\mu\nu\omega_2$ soit de longueur supérieure à f. À tout instant t, le passé ω de τ_i est donc un mot du centre [8] de $\bullet^{r_i}\left(\left(a_i^{C_i}\text{Ш}\bullet^{D_i-C_i}\right)\bullet^{T_i-D_i}\right)^*$. Réciproquement, par définition, tout mot du centre est le passé d'une configuration valide d'allocation de processeur à la tâche.

Considérons maitenant le cas où la tâche τ_i comporte des instructions de choix. Soient $(\omega_k)_{k\in I}$ les mots correspondant à des comportements possibles de τ_i (au sens de la trace du programme associé). Nous notons L_{C_i} l'ensemble de ces mots. Cet ensemble est fini, car la

[8] Le centre de L est l'ensemble des préfixes de L indéfiniment prolongeables dans L.

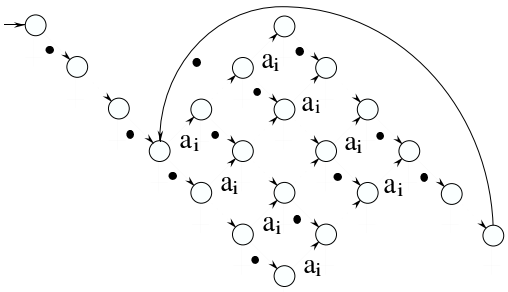

FIG. 3. Comportements temporels valides de la tâche

durée d'exécution de τ_i est majorée par C_i. I est donc fini également. Les comportements des différentes occurrences de τ_i correspondent toutefois à des ω_k non nécessairement égaux. Un comportement temporel valide de τ_i est donc, globalement, élement de $Centre\left(\bullet^{r_i}\left(\bigcup_{k\in I}\left(\omega_k \text{Ш} \bullet^{D_i-|\omega_k|}\right)\bullet^{T_i-D_i}\right)^*\right)$. I étant un ensemble fini, ce langage est rationnel : il est accepté par l'automate dont l'allure est présenté en Figure 2.Droite.

Définition 1. *On appelle* **comportement temporel valide** *de la tâche τ_i tout mot ω_k appartenant au langage $Centre\left(\bullet^{r_i}\left(\bigcup_{k\in I}\left(\omega_k \text{Ш} \bullet^{D_i-|\omega_k|}\right)\bullet^{T_i-D_i}\right)^*\right)$.*

Un comportement temporel valide de τ_i modélise donc une configuration d'allocation de processeur à τ_i pour laquelle on est en mesure de garantir que τ_i dispose d'au moins une possibilité de continuer son exécution en respectant ses contraintes temporelles. Dans la suite, nous notons $L(\tau_i)$ ou, plus succinctement, L_i, l'ensemble des comportements temporels valides de τ_i. Ce langage est rationnel, et reconnu par l'automate présenté en Figure 3.

Remarque

Dans [12], nous montrons que ce raisonnement reste valide pour les systèmes constitués de tâches périodiques et sporadiques.

3. MODÉLISATION DE SYSTÈMES DE TÂCHES

Une application temps-réel est constituée d'un ensemble de tâches, indépendantes ou non. Nous nous intéressons ici aux applications temps-réel à contraintes strictes, c'est à dire uniquement constituées de tâches soumises à des contraintes temporelles strictes. Une telle application est donc définie par la donnée de deux familles de tâches $(\tau_i)_{i\in[1,n]}$ et $(\alpha_i)_{i\in[1,p]}$: les α_i sont des sporadiques, et les τ_i des périodiques.

3.1. Fonctionnement simultané de l'ensemble des tâches.

Un système temps-réel étant réactif, l'ensemble des tâches qui le composent fonctionnent simultanément. Pour représenter cette simultanéité, nous utilisons les **produits de Hadamard** de langages : si $(\Sigma_i)_{i\in[1,n]}$ sont des alphabets, le produit de Hadamard est le morphisme de concaténation

$$(\Sigma_i)_{i\in[1,n]} \longrightarrow \prod_{i=1}^{n}\Sigma_i : (a_i)_{i\in[1,n]} \longmapsto \begin{pmatrix} a_1 \\ a_2 \\ : \\ a_n \end{pmatrix}$$

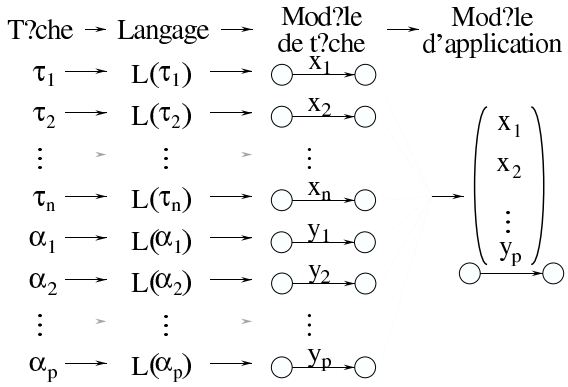

FIG. 4. Des produits de Hadamard pour modéliser les systèmes de tâches

La sémantique associée au produit de Hadamard d'une famille de lettres (chaque lettre représentant l'état d'activité d'une tâche) est la simultanéité [9]. Dans la suite, nous notons $\underset{i=1}{\overset{i=n}{\Omega}} L_i$ le langage produit de Hadamard des langages $(L_i)_{i\in[1,n]}$. Notons que le produit de Hadamard d'une famille de centres de langages rationnels reste un centre de langages rationnels [17]. La notion de comportement temporel valide, définie plus haut pour les tâches, s'étend naturellement aux systèmes de tâches.

Définition 2. *On appelle comportement temporel valide d'une application temps-réel à contraintes strictes constituée de deux familles de tâches* $(\tau_i)_{i\in[1,n]}$ *et* $(\alpha_i)_{i\in[1,p]}$ *indépendantes tout mot de* $\left(\underset{i=1}{\overset{i=n}{\Omega}} L(\tau_i)\right) \Omega \left(\underset{i=1}{\overset{i=p}{\Omega}} L(\alpha_i)\right)$

Dans la suite de ce travail, nous notons $L((\tau_i)_{i\in[1,n]}, (\alpha_i)_{i\in[1,p]})$ cet ensemble.

3.2. **Interdépendance des tâches.** Lorsque les tâches communiquent ou partagent des ressources critiques, l'ensemble des comportements temporels valides de l'application est un sous-ensemble de $L\left((\tau_i)_{i\in[1,n]}, (\alpha_i)_{i\in[1,p]}\right)$: certaines périodes d'activité de certaines tâches se trouvent retardées, du fait de l'exclusion mutuelle sur l'accès aux ressources, ou du fait de l'attente de messages. Certains comportements temporels valides dans le cas de l'indépendance des tâches ne sont plus valides.

Pour collecter l'ensemble des comportements valides, nous utilisons la technique proposée par Arnold et Nivat [18]. L'idée consiste à

(1) Définir une tâche virtuelle ρ_R pour chaque élément critique R (communication par message ou par rendez-vous, partage de ressource), dont l'objet est de tracer les états de l'élément en question (trace des états de la ressource, par exemple)

(2) Calculer le produit de Hadamard $H = L\left((\tau_i)_{i\in[1,n]}, (\alpha_i)_{i\in[1,p]}, (\rho_R)_{\text{R élément critique}}\right)$

(3) Construire l'ensemble S des vecteurs de lettres correspondant aux configurations instantannées valides (une seule attribution d'une ressource critique donnée à un instant donné, par exemple) : ce sont les seuls vecteurs dont l'apparition dans tout mot de H est autorisée

(4) Calculer $H \cap S^*$, qui collecte l'ensemble des comportements du système de tâches compatibles avec les contraintes de synchronisation

[9] Dans le cadre de la modélisation de systèmes concurrents, cette approche a été introduite par [16].

3.3. **Application au calcul d'ordonnançabilité.** L'intersection de langages n'est pas interne dans la classe des centres de langages rationnels. Dans notre étude, cette propriété traduit le fait que le partage de ressources (par exemple) entre n tâches soumises à des contraintes strictes peut amener certaines d'entre elles en situation de faute temporelle. Ce résultat est bien connu, et a motivé dans le passé un grand nombre de travaux [4]. Notre but est ici de décider de l'ordonnançabilité du système de tâches dans ce contexte d'interdépendance. Nous avons montré dans [13][19][14] que cette approche permet une résolution extrêmement efficace de ce problème.

La stratégie de décision repose sur des propriétés caractéristiques du langage associé au système de tâches. Dans le cas où les tâches sont à durées fixes [13], l'ordonnançabilité est acquise dès lors que le langage est non vide. Dans le cas où certaines (ou toutes) tâches sont à durées d'exécution variables [12], la décision d'ordonnançabilité est une caractéristique du langage :

Théorème 3. [12] *Un ensemble de tâches* $(\tau_i)_{i\in[1,n]}$ *est globalement ordonnançable si et seulement si* [10]

$$\forall (x_i)_{i\in[1,n]} \in \prod_{i=1}^{n} (X(\tau_i)), \exists \omega \in Centre\left(L\left((\tau_i)_{i\in[1,n]}\right)\right) \text{ tel que}$$

$$\forall i \in [1,n], \pi_{\neg\bullet}(\pi_i(\omega)) = x_i$$

où $(X(\tau_i))$ *désigne l'alphabet asocié à la tâche* τ_i.

4. Utilisation de séries génératrices pour améliorer le calcul de l'ordonnançabilité d'un système intégrant des sporadiques

Cette méthode de calcul d'ordonnançabilité est utilisable pour des systèmes quelconques de tâches : présence simultanée de périodiques et de sporadiques, interdépendance de tâches, ordonnancements toujours préemptif, toujours non-préemptif, ou bien globalement préemptif avec des sections localement non-préemptives (sections critiques d'accès aux ressources critiques, par exemple), etc.

Toutefois, l'analyse d'études de cas réels fait apparaître des configurations qui, bien que prises en compte par notre méthodologie, méritent une étude spécifique. C'est, par exemple, le cas des systèmes de tâches dans lesquels les sporadiques sont indépendantes des autres tâches [11]. Modulo un enrichissement du modèle, nous montrons ci-dessous que, dans ce cas, il est possible de décider de l'ordonnançabilité du système complet à partir de la seule connaissance de $Centre\left(L\left((\tau_i)_{i\in[1,n]}\right)\right)$.

Cet enrichissement est basé sur l'utilisation de certaines fonctions génératrices afin de prédire s'il est possible de traiter l'activation d'une tâche sporadique à un instant donné. Dans un premier temps, nous décrivons comment calculer ces fonctions génératrices. Puis, nous mettons en évidence une de leur propriété qui va nous permettre d'étendre la modularité du modèle originel à celui enrichi par les séries génératrices.

[10] Pour tout $i \in [1,n]$, on note π_i le morphisme $(x_j)_{i\in[1,n]} \to x_i$. Pour tout alphabet Σ et tout $A \subset \Sigma$, on note π_A le morphisme $\begin{cases} x \in A \Leftrightarrow \pi_A(x) = x \\ x \notin A \Leftrightarrow \pi_A(x) = \varepsilon \end{cases}$ et par $\pi_{\neg A}$ le morphisme $\begin{cases} x \in A \Leftrightarrow \pi_{\neg A}(x) = \varepsilon \\ x \notin A \Leftrightarrow \pi_{\neg A}(x) = x \end{cases}$

[11] Donc indépendantes entre elles, et indépendantes des périodiques.

4.1. **Des séries génératrices pour prévoir l'avenir.** Considérons un système de tâches [12] partitionné en n tâches périodiques $(\tau_i)_{i\in[1,n]}$ et p tâches sporadiques $(\alpha_i)_{i\in[1,p]}$. Pour éviter toute confusion, notons X_i^p les caractéristiques temporelles concernant les périodiques et X_i^s celles concernant les sporadiques. Chaque τ_i et chaque α_j sont à durée d'exécution fixe. Chaque α_j est indépendante de l'ensemble des autres tâches du système. Nous supposons ici que le système réduit à sa composante périodique est ordonnançable, c'est à dire que $Centre\left(L\left((\tau_i)_{i\in[1,n]}\right)\right) \neq \emptyset$, et qu'il vérifie le thórème 3. Le système global (c'est à dire composé des périodiques **et** des sporadiques) est ordonnançable si, pour tout $j \in [1,p]$ et tout $t \in \left[\min_{i\in[1,n]}(r_i^p), +\infty\right[\cap \left\{ \begin{array}{c}\text{Dates d'activation}\\ \text{possibles de } \alpha_j\end{array} \right\}$, l'activation de α_j à l'instant t n'amème pas le système en situation de faute temporelle. Désignons par $A\left((\tau_i)_{i\in[1,n]}\right)$ l'automate déterministe minimal d'acceptation du langage $Centre\left(L\left((\tau_i)_{i\in[1,n]}\right)\right)$.

(1) Pour chaque état i de $A\left((\tau_i)_{i\in[1,n]}\right)$, nous devons pouvoir déterminer à l'avance quels sont les chemins issus de i, et permettant l'ordonnancement de chacune des α_j susceptibles d'être activées lorsque la composante périodique du système se trouve dans l'état i. Il nous faut donc associer à chaque transition t issue de i l'information $\left(D_j^s, K_{t,D_j^s}\right)_{j\in[1,p]}$, où K_{t,D_j^s} est le nombre maximal d'instants d'oisiveté du processeur dans les D_j^s prochaines unités de temps, si la transition t est choisie (voir Figure 5).

(2) Dans le cas où l'ordonnancement s'effectue sur multi-processeurs, cette seule information n'est cependant pas suffisante : k tâches peuvent être amenées à fonctionner simultanénement, il peut donc, à chaque unité de temps, y avoir jusqu'à k processeurs oisifs de front. Comme les tâches sont non parallélisables, on doit différencier *instants simultanés d'oisiveté* et *instants consécutifs d'oisiveté*. Dans le premier cas, deux instants d'oisiveté ne peuvent être alloués à la même tâche, alors qu'ils le peuvent dans le second cas. L'information à associer à t est donc $\left(D_j^s, \left(K_{t,j,D_j^s}\right)_{j\in[0,k]}\right)$, où K_{t,j,D_j^s} est le nombre d'unités de temps de l'intervalle [13] $\left[0, D_j^s\right]$ pendant lesquelles exactement j processeurs sont simultanément libres. On remarque que, ramené au cas mono-processeur, cette information correspond exactement à la durée d'oisiveté recherchée.

(3) Mais la seule donnée de cette information rend le modèle non modulaire, dans le sens où lors de la construction du produit de Hadamard d'une famille $(A_i)_{i\in[1,n]}$ d'automates finis, le calcul de l'information associée à une transition T du produit $\underset{i=1}{\overset{i=n}{\Omega}} A_i$ n'est pas réalisable à partir de la seule donnée des informations associées aux transitions $(t_i)_{i\in[1,n]}$ des automates A_i. Considérons, par exemple, les transitions t_1 et t_2, respectivement associées aux informations suivantes :

t_1**:** On dispose, dans les trois unités de temps à venir, d'un processeur oisif pendant trois unités de temps –on a deux processeurs en tout– et de deux processeurs simultanément libres pendant deux unités de temps. La configuration

[12] On rappelle que les tâches constituant les systèmes étudiés ne sont pas parallélisables.

[13] L'origine des temps est ici, évidemment, l'instant auquel le système périodique se trouve dans l'état i.

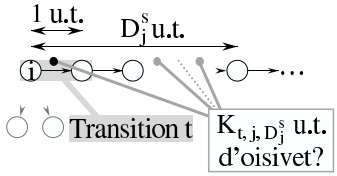

FIG. 5. Informations nécessaires à la décision d'ordonnançabilité d'une sporadique

d'oisiveté dans les trois unités de temps à venir est donc élément de l'ensemble

$$\left\{ \begin{pmatrix}\bullet\\\bullet\end{pmatrix}\begin{pmatrix}\bullet\\\bullet\end{pmatrix}\begin{pmatrix}\\\bullet\end{pmatrix}, \begin{pmatrix}\bullet\\\bullet\end{pmatrix}\begin{pmatrix}\\\bullet\end{pmatrix}\begin{pmatrix}\bullet\\\bullet\end{pmatrix}, \begin{pmatrix}\\\bullet\end{pmatrix}\begin{pmatrix}\bullet\\\bullet\end{pmatrix}\begin{pmatrix}\bullet\\\bullet\end{pmatrix} \right\}$$

t_2**:** On sait que, dans les trois unités de temps à venir, le processeur –il n'y en a qu'un– est oisif pendant une seule unité de temps. La configuration d'oisiveté dans les trois unités de temps à venir est donc élément de l'ensemble

$$\{(\bullet)\,(\,)\,(\,),(\,)\,(\bullet)\,(\,),(\,)\,(\,)\,(\bullet)\}$$

Appelons T la transition de l'automate produit correspondant à la simultanéité de t_1 et t_2. En fonction des configurations respectives d'oisiveté de t_1 et de t_2, la configuration d'oisiveté de T est l'un des éléments de l'ensemble

$$\left\{ \begin{pmatrix}\bullet\\\bullet\\\\\bullet\end{pmatrix}\begin{pmatrix}\bullet\\\bullet\\\bullet\\\bullet\end{pmatrix}\begin{pmatrix}\\\bullet\\\\\bullet\end{pmatrix}, \begin{pmatrix}\bullet\\\bullet\\\\\bullet\end{pmatrix}\begin{pmatrix}\\\bullet\\\bullet\\\bullet\end{pmatrix}\begin{pmatrix}\bullet\\\bullet\\\bullet\\\bullet\end{pmatrix}, \begin{pmatrix}\bullet\\\bullet\\\\\bullet\end{pmatrix}\begin{pmatrix}\bullet\\\bullet\\\bullet\\\bullet\end{pmatrix}\begin{pmatrix}\bullet\\\bullet\\\bullet\\\bullet\end{pmatrix}, \begin{pmatrix}\bullet\\\bullet\\\\\bullet\end{pmatrix}\begin{pmatrix}\bullet\\\bullet\\\bullet\\\bullet\end{pmatrix}\begin{pmatrix}\bullet\\\bullet\\\bullet\\\bullet\end{pmatrix}, \begin{pmatrix}\\\bullet\\\bullet\\\bullet\end{pmatrix}\begin{pmatrix}\bullet\\\bullet\\\bullet\\\bullet\end{pmatrix}\begin{pmatrix}\bullet\\\bullet\\\bullet\\\bullet\end{pmatrix} \right\}$$

Or, les différents éléments de cet ensemble correspondent à des configurations d'oisiveté qui n'offrent pas les mêmes capacités d'ordonnançabilité pour d'éventuelles tâches sporadiques. L'information associée à t_1 et t_2 n'est donc pas suffisante pour calculer l'information à associer à T : nous enrichissons donc cette information en associant à chaque transition l'information $\Big(D, (Oisiv_i)_{i\in[1,D]}\Big)$, où $D = \underset{\alpha\in\{(\alpha_i)_{i\in[1,p]}\}}{Max}(D_\alpha)$, et $Oisiv_i \in \mathbb{N}$ est le nombre de processeurs oisifs lors de la $i^{\text{ème}}$ unité de temps à venir. Cette information répond à ce problème : si t_1 est associée à l'information $\Big(D, (Oisiv_{1i})_{i\in[1,D]}\Big)$, et t_2 à l'information $\Big(D, (Oisiv_{2i})_{i\in[1,D]}\Big)$, alors, T est associée à l'information $\Big(D, (Oisiv_{1i} + Oisiv_{2i})_{i\in[1,D]}\Big)$ (notons qu'il s'agit d'un morphisme, que nous appelons // dans la suite).

Nous associons donc à chaque état (s) de l'automate fini une fonction $F_s(y) \in \mathbb{N} \ll (p_i)_{i\in[0,\pi]} \gg [y]$, dont la sémantique est la suivante :

- π est le nombre de processeurs disponibles
- L'indéterminée y trace la longueur des mots, c'est à dire le temps : la présence de y^a dans un monôme indique que l'on s'intéresse aux mots de longueur a. Comme, dans notre modèle, *longueur=durée*, cela signifie que l'on s'intéresse aux comportements de durée a.

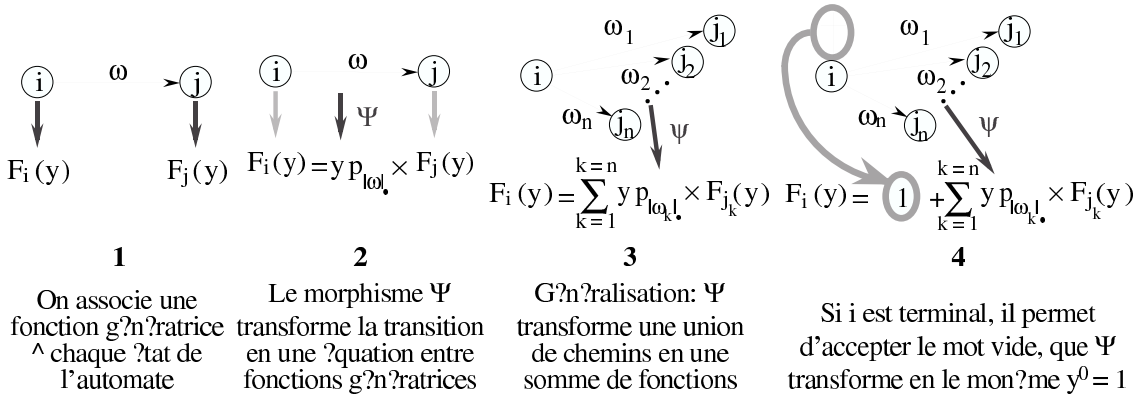

FIG. 6. Transformation de l'automate fini en un système linéaire

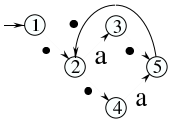

FIG. 7. Un exemple

– La suite des indéterminées p_i modélise le vecteur $(Oisiv_i)_{i\in[1,D]}$: la présence de p_i en $j^{\text{ème}}$ position dans la suite indique qu'à la $j^{\text{ème}}$ unité de temps à venir, on disposera d'exactement i processeurs oisifs. Par exemple, le facteur $p_1p_0p_2p_1$ modélise la configuration

Instant	1	2	3	4
Nombre de processeurs oisifs	1	0	2	1

L'automate est transformé en un système linéaire $M \times F = T$, où M est une matrice à coefficients polynomiaux, F un vecteur d'inconnues, et T un vecteur de polynômes de degrés nuls (voir Figure 6). Le vecteur de fractions rationnelles $(F_i(y))_{i\in[1,N]}$ (N est le nombre d'états de l'automate fini) est obtenu par la résolution du système linéaire.

Considérons par exemple l'automate représenté sur la Figure 7. On obtient le système :

$$\begin{cases} F_1(y) &= p_1yF_2(y)+1 \\ F_2(y) &= p_0yF_3(y)+p_1yF_4(y)+1 \\ F_3(y) &= p_1yF_5(y)+1 \\ F_4(y) &= p_0yF_5(y)+1 \\ F_5(y) &= p_1yF_2(y)+1 \end{cases}$$

La résolution de ce système fournit :

$$\begin{cases} F_1(y) &= 1+\dfrac{p_1y+(p_1p_0+p_1^2)y^2}{1-y^3(p_1p_0p_1+p_1^2p_0)} \\ F_2(y) &= 1+y(p_0+p_1)+y^2(p_0p_1+p_1p_1)\dfrac{p_1y+(p_1p_0+p_1^2)y^2}{1-y^3(p_1p_0p_1+p_1^2p_0)} \\ F_3(y) &= 1+yp_1+yp_1\dfrac{p_1y+(p_1p_0+p_1^2)y^2}{1-y^3(p_1p_0p_1+p_1^2p_0)} \\ F_4(y) &= 1+yp_0+yp_0\dfrac{p_1y+(p_1p_0+p_1^2)y^2}{1-y^3(p_1p_0p_1+p_1^2p_0)} \\ F_5(y) &= 1+\dfrac{p_1y+(p_1p_0+p_1^2)y^2}{1-y^3(p_1p_0p_1+p_1^2p_0)} \end{cases}$$

Une fois les fonctions génératrices déterminées, lorsque l'on veut savoir si une tâche sporadique de caractéristiques (C_s, D_s, P_s), se déclenchant à l'instant t, peut être traitée par le système, il suffit alors d'observer la fonction génératrice F associée à l'état de l'automate dans lequel se trouve le système à l'instant t :

- on développe F à l'ordre D_s+1 ;
- on recherche, dans le coefficient de y^{D_s}, s'il existe un monôme indiquant au moins C_s unités de temps pendant lesquelles un processeur est oisif ;
- s'il en existe un, la tâche sporadique peut être prise en compte, sinon non.

4.2. **Caractérisation des fractions rationnelles.** En raison de leur mode de calcul, la caractérisation des fonctions associées aux transitions repose intégralement sur une caractérisation des F_i. Ce sont donc ces fonctions que nous allons étudier. Un premier résultat nous fournit une caractérisation de ces fonctions sur laquelle nous nous appuierons ensuite pour établir le résultat central de notre travail, dans l'optique de valider des systèmes temps-réel.

Théorème 4. *La fonction génératrice F_i, associée à l'état ⓘ, et obtenue par transformation de l'automate fini d'acceptation de $L\left((\tau_i)_{i\in[1,n]}\right)$ suivant le morphisme Ψ, est de la forme*

$$N_1(y)+\frac{N_2(y)}{1-y^PD}.$$

N_1, N_2 et D sont des polynômes à coefficients dans $\mathbb{N}\ll(p_j)_{j\in[0,\pi]}\gg$ et $P=\underset{i\in[1,n]}{ppcm}(T_i^p)$.

Preuve

L'automate fini vérifie les deux propriétés suivantes :

- Les chemins joignant l'état ⓓ à l'état ⓐ partagent la même longueur : [2] établit que, en configuration mono-processeur, et lorsque les tâches sont à durées fixes, cette longueur est inférieure à $\underset{i\in[1,n]}{Max}(r_i)+\underset{i\in[1,n]}{ppcm}(T_i)$.
- Les chemins joignant l'état ⓐ à ⓔ partagent également la même longueur. Par construction de l'automate (que les tâches soient ou non à durées variables), cette longueur est $\underset{i\in[1,n]}{ppcm}(T_i)$.

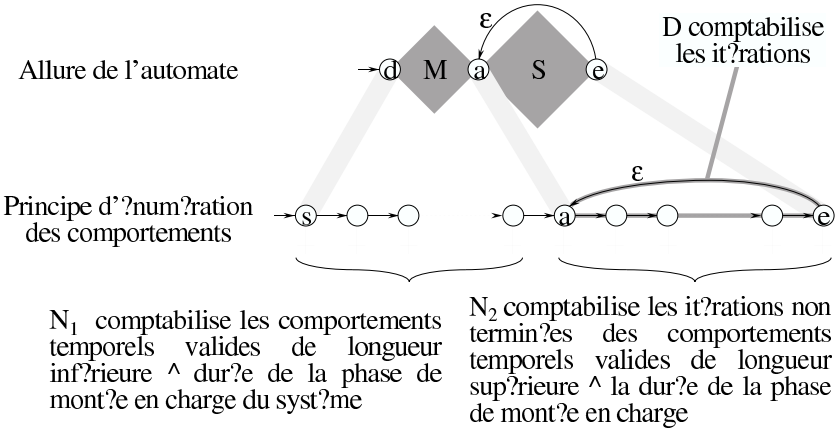

FIG. 8. Allure de l'automate

Soit ⓢ un état de l'automate. La structure de l'automate (voir Figure 8) entraîne que ◯ s se trouve soit sur (au moins) un chemin reliant ⓓ à ⓐ, soit sur un chemin reliant ⓐ à ⓔ. Donc, tout chemin issu de ⓢ est étiqueté par un mot ω, de la forme $\mu\nu\rho$, où

- μ est la trace d'un chemin reliant ⓢ à ⓔ
- ν est la trace d'un chemin reliant ⓐ à ⓐ. À ce titre, il est donc élément d'un langage étoilé
- ρ est la trace d'un chemin, de longueur inférieure à $\underset{i\in[1,n]}{ppcm}(T_i)$, joignant ⓐ à l'un (quelconque) des états atteignables depuis ⓐ

Notons P l'entier $\underset{i\in[1,n]}{ppcm}(T_i)$, l la longueur $|\mu|$, et $m_s(i)$ le polynôme associé aux chemins de longueur i, issus de ⓢ. Si $(\Sigma, Q, d, Q, \delta)$ est l'automate d'acceptation de $Centre\left((\tau_i)_{i\in[1,n]}\right)$, appelons L_s le langage accepté par l'automate $(\Sigma, Q, s, Q, \delta)$. On a

$m_s(i) = \sum_{\omega\in L_s\cap\Sigma^i} m_{s,(\beta_h(\omega))_{h\in[1,k]}} \prod_{h=1}^{k} x_h^{\beta_h(\omega)}$, où $m_{s,(\beta_h(\omega))_{h\in[1,k]}}$ est le nombre de mots ω de L_s, de longueur i, correspondant à des ordonnancements pour lesquels on dispose, pour tout h, de $\beta_h(\omega)$ unités de temps pendant lesquelles h processeurs sont simultanément oisifs. L'état ⓐ est atteint depuis ⓢ en $l-1$ étapes. Calculons alors $F_s(y)$.

$$\begin{aligned} F_s(y) &= \sum_{i\geq 0} m_s(i)y^i \\ &= \sum_{i=0}^{l-1} m_s(i)y^i + \sum_{i\geq l} m_s(i)y^i. \end{aligned}$$

Or, tout chemin de longueur supérieure à l issu de ⓢ se décompose en

- Un chemin de ⓢ à ⓐ, de longueur l
- Un chemin issu de ⓐ

Les premiers chemins sont énumérés par $m_s(l)y^l$ et les seconds par $F_a(y)$. Donc

$$F_s = \sum_{i=0}^{l-1} m_s(i)y^i + m_s(l)y^l F_a(y)$$

Calculons alors $F_a(y)$.

Tous les chemins issus de ⓐ sont contenus dans une boucle de longueur P contenant ⓐ.

Donc, tout chemin issu de ⓐ de longueur supérieure à P débute par un chemin de longueur P de ⓐ à ⓐ. En utilisant ces remarques pour exprimer $F_a(y)$, on obtient :

$$\begin{aligned} F_a(y) &= \sum_{i\geq 0} m_a(i)y^i \\ &= \sum_{i=0}^{P-1} m_a(i)y^i + \sum_{i\geq P} m_a(i)y^i \\ &= \sum_{i=0}^{P-1} m_a(i)y^i + \sum_{i\geq P} \left(m_a(P)y^P m_a(i-P)y^{i-P}\right) \\ &= \sum_{i=0}^{P-1} m_a(i)y^i + m_a(P)y^P \left(\sum_{i\geq 0} m_a(i)y^i\right) \\ &= \sum_{i=0}^{P-1} m_a(i)y^i + m_a(P)y^P F_a(y) \end{aligned}$$

Donc

$$F_a(y) = \frac{\sum_{i=0}^{P-1} m_a(i)y^i}{1 - m_a(p)y^P}$$

et

$$F_s(y) = \sum_{i=0}^{l-1} m_s(i)y^i + \frac{m_s(l)y^l \sum_{i=0}^{p-1} m_a(i)y^i}{1 - m_a(P)y^P}$$

En posant $N_1(y) = \sum_{i=0}^{l-1} m_s(i)y^i$, $N_2(y) = m_s(l)y^l \sum_{i=0}^{P-1} m_a(i)y^P$, et $D = m_a(P)$, on obtient :

$$F_s(y) = N_1(y) + \frac{N_2(y)}{1 - Dy^P}$$

□

Remarques

- La construction des différents polynômes entraîne les inégalités suivantes :
$$deg(N_1, y) < l,$$
$$deg(N_2, y) < P + l - 1.$$
- La sémantique naturellement associée aux polynômes s'exprime, en termes de chemins, de la façon suivante :
  - $N_1(y)$ correspond à la partie acyclique des chemins issus de ⓢ
  - $N_2(y)$ correspond à la fonction génératrice des chemins issus de ⓢ, passant par ⓐ , mais ne décrivant pas totalement un cycle de L_a
  - D correspond à la fonction génératrice de chemins de longueur P issus de ⓐ

4.3. **Modularité du modèle temporel enrichi.** Notre objectif est ici d'établir (voir Figure 9) que le calcul de la fonction génératrice associée à un état donné $\left\{(s_i)_{i\in[1,n]}\right\}$ du produit de Hadamard ne repose que sur la donnée des fonctions associées aux états $(s_i)_{i\in[1,n]}$ des automates *composantes*.

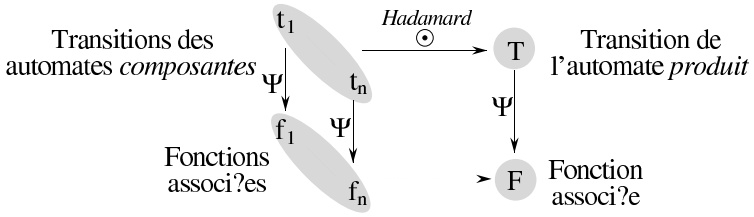

FIG. 9. Compatibilité des fonctions génératrices avec le produit de Hadamard

À partir de la famille $(A_i)_{i\in[1,n]}$ des automates associés aux tâches $(\tau_i)_{i\in[1,n]}$ (chaque A_i est un automate d'acceptation de $L(\tau_i)$), on calcule l'automate d'acceptation $A\left((\tau_i)_{i\in[1,n]}\right)$ de l'ensemble des comportements temporels valides du système $(\tau_i)_{i\in[1,n]}$ sur la base d'un produit de Hadamard des $(A_i)_{i\in[1,n]}$: chaque transition t de $A\left((\tau_i)_{i\in[1,n]}\right)$ est obtenue à partir du seul n-uplet $(t_i)_{i\in[1,n]}$ de transitions de $(A_i)_{i\in[1,n]}$ dont elle représente l'exécution simultanée. Pour le modèle enrichi, nous montrons que cette propriété est conservée : la fonction génératrice associée à t est calculable à partir de la seule donnée du n-uplet des fonctions génératrices associées aux $(t_i)_{i\in[1,n]}$.

Notre technique de calcul s'appuie sur la théorie des automates à multiplicités : l'idée consiste à associer à la fonction génératrice de chaque t_i un automate à multiplicité B_i puis, à partir de manipulations algébriques sur les B_i, à produire l'automate B associé à t. À partir de B, on est alors capable d'exprimer la fonction résultat [14].

4.3.1. *Automates à multiplicités.* Un K-sous-ensemble S de E est un morphisme de E dans K (où K est un semi-anneau) [17]. On appelle automate à multiplicité dans K tout 5-uplet $\text{B}=(A,Q,I,T,E)$ où A est un alphabet, Q un ensemble fini (les états de B), I (les états initiaux) et T (les états terminaux) des K-sous-ensembles de Q et E un K-sous-ensemble de $Q\times A\times Q$ (les transitions et leur multiplicité).

4.3.2. *Représentation linéaire d'automates à multiplicités.* L'école de Schützenberger [17] a introduit la notion de représentation linéaire (λ,μ,γ), $\lambda\in K^{1\times n}$, $\mu\in K^{n\times n}$ et $\gamma\in K^{n\times 1}$ [20]. L'équivalence se fait de la façon suivante . Pour toute lettre $a\in A$, on construit une matrice $Q\times Q$, notée $\mu(a)$, telle que $\mu_{i,j}(a)=E(i,a,j)$ pout tout $(i,j)\in Q^2$. On définit ensuite $\lambda=(\lambda_1,\ldots\lambda_n)$ et $\gamma=(\gamma_1,\ldots,\gamma_n)$ de telle sorte que :

$$\lambda_i = \begin{cases} 0 \text{ si l'état } i \text{ de l'automate n'est pas initial} \\ \text{multiplicité associée à l'entrée dans l'état } i \text{ sinon} \end{cases}$$

.

$$\gamma_i = \begin{cases} 0 \text{ si l'état } i \text{ de l'automate n'est pas terminal} \\ \text{multiplicité associée à la sortie de l'état } i \text{ sinon} \end{cases}$$

Si l'on considère I comme une matrice ligne $K^{1\times Q}$ et T comme une matrice colonne $K^{Q\times 1}$, le comportement de B est donné par

$$\text{C}(\mathbf{B})=\sum_{w\in A^*} IE(w)Tw,$$

qui est la série formelle reconnue par B. Si l'on définit un étiquetage $Q=\{q_1,\ldots,q_n\}$, on obtient, via cette correspondance, un triplet (λ,μ,γ) ($\lambda\in K^{1\times n}$, $\mu:A\to K^{n\times n}$, $\gamma\in K^{n\times 1}$) qui est appelé représentation linéaire de $\text{C}(\mathbf{B})$.

[14] Les auteurs remercient Gérard Duchamp pour ses conseils sur ce sujet.

Soit $K \ll A \gg$ l'ensemble des séries formelles non commutatives sur un alphabet fini A et à coefficients dans K. Une série $S = \sum_{w \in A^*} (S, w)w$ est reconnaissable si et seulement si il existe une représentation linéaire (λ, μ, γ), de telle sorte que, pour tout $w \in A^*$, on a $(S, w) = \lambda\mu(w)\gamma$. Dans la suite, on notera cette propriété $S : (\lambda, \mu, \gamma)$ [21].
Ce travail est réalisé dans $K = \mathbb{N} \ll (p_j)_{j\in[0,\pi]} \gg\ll y \gg$. Si l'on ne considère que les indices des lettres p (par exemple $(1, 0, 2)$ au lieu de $p_1p_0p_2$), on se ramène au semi-anneau $K = (\mathbb{N} \cup \{\infty\})^{\mathbb{N}}$ muni des lois $(max, +)$, où $+$ correspond au morphisme // défini dans la section 4.1 et où max donne la structure de semi-anneau mais n'est pas utilisé dans notre étude.

4.3.3. *Automate à multiplicités associé à une fonction génératrice.* Considérons la fonction génératrice $F(y) = N(y)\,(yQ(y))^* = \frac{N(y)}{1-yQ(y)}$ associée à l'un des états (disons l'état ⓘ) de l'automate. N et Q sont des éléments de $\mathbb{N} \ll (p_i)_{i\in[0,\pi]} \gg [y]$. Notons $N(y) = \sum_{i=0}^{m} n_i y^i$ et $Q(y) = \sum_{i=0}^{P} q_i y^i$. Étant donné que N est obtenu à partir d'une réduction au même dénominateur (fusion des polynômes N_1 et N_2 –voir Théorème 4–), on a $P < m$.

Le résultat suivant fournit la représentation linéaire pour les fonctions génératrices.

Théorème 5. *Soient* $F(y) = \dfrac{N(y)}{1 - yQ(y)} \in \mathbb{N} \ll (p_i)_{i\in[0,\pi]} \gg [y]$, *où*
$N(y) = \sum_{i=0}^{P} n_i y^i$, $Q(y) = \sum_{i=0}^{m} q_i y^i$, *et* $r = max(deg(N, y), deg(Q, y)) + 1$
Alors la représentation linéaire de F(y) est le triplet (λ, μ, γ) *défini par :*

$$\lambda = \left(n_0, \ldots, n_P, 0_{1\times(r-P-1)}\right),\ \gamma = \begin{pmatrix} 1 \\ 0_{(r-1)\times 1} \end{pmatrix},\ \mu \in \mathbb{N} \ll (p_i)_{i\in[0,\pi]} \gg^{r\times r}$$

et

$$\mu(y) = \left(\begin{array}{ccccc|c} q_0 & \cdots & \cdots & \cdots & q_m & \\ 1 & 0 & \cdots & \cdots & 0 & \\ 0 & \ddots & \ddots & & \vdots & 0_{(r-m-1)\times(m+1)} \\ \vdots & \ddots & \ddots & \ddots & \vdots & \\ 0 & \cdots & 0 & 1 & 0 & \\ \hline \multicolumn{5}{c|}{0_{(m+1)\times(m+1)}} & 0_{(r-m-1)\times(r-m-1)} \end{array}\right)$$

Preuve

Dans un premier temps, construisons l'automate reconnaissant $yQ(y)$, c'est à dire $y\sum_{k=0}^{m} q_k y^k$. Chacun des monômes de cette expression est de la forme $q_k y^{k+1}$. Considéré comme un mot, ce monôme étiquette le chemin (unique) dont l'origine est l'état initial $\left(\frac{y^{m-k+1}}{D}\right)$ de l'automate présenté en Figure 10, et l'arrivée le seul état terminal de cet automate. Ici, la multiplicité q_k, qui étiquette la flèche d'entrée dans l'automate, intervient au même titre qu'une lettre. Chaque état initial permet l'acceptation d'un monôme. Le langage accepté par cet automate est donc $\bigcup_{k=0}^{m} \left\{q_k y^{k+1}\right\}$. Le polynôme canoniquement associé à ce langage

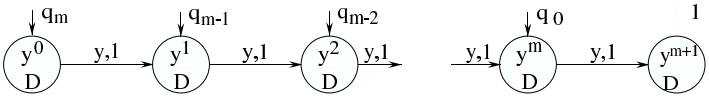

FIG. 10. Automate reconnaissant $yQ(y)$

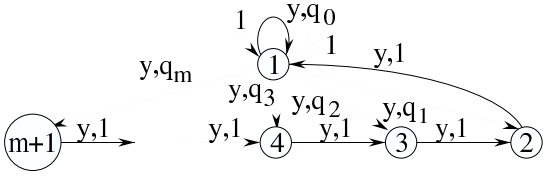

FIG. 11. Automate reconnaissant $(yQ(y))^*$

est donc $yQ(y)$. Sa representation linéaire est

$$\lambda^1 = (0, q_0, \ldots, q_m),\ \gamma^1 = \begin{pmatrix} 1 \\ 0_{(m+1)\times 1} \end{pmatrix}$$

$$\text{et } \mu^1(y) = \begin{pmatrix} 0 & \cdots & \cdots & \cdots & 0 \\ 1 & \ddots & & & \vdots \\ 0 & \ddots & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & 0 \end{pmatrix}.$$

Pour obtenir la représentation linéaire de $(yQ(y))^* = \dfrac{1}{1-yQ(y)}$, on utilise la forme linéaire de l'automate associé à un langage, présentée en section 4.3.2. Dans [22], il est établi que, si $S : (\lambda, \mu, \gamma)$ (de dimension m) est la représentation linéaire d'un automate, la représentation linéaire de S^* est [15] :

$$\left((0_{1\times m}, 1), \left(\begin{array}{c|c} \mu(y) + \gamma\lambda\mu(y) & 0_{m\times 1} \\ \hline \lambda\mu(y) & 0 \end{array} \right), \begin{pmatrix} \gamma \\ 1 \end{pmatrix} \right)$$

En appliquant cette forme au calcul de la forme linéaire de $(yQ(y))^*$, on obtient :

$$\lambda^2 = (0_{1\times(m+2)}, 1),\ \gamma^2 = \begin{pmatrix} 1 \\ 0_{(m+1)\times 1} \\ 1 \end{pmatrix},\ \mu^2 \in \mathbb{N} \ll (p_j)_{j\in[0,\pi]} \gg^{(m+3)\times(m+3)}$$

$$\text{et } \mu^2(y) = \left(\begin{array}{ccccc|c|c} q_0 & \cdots & \cdots & \cdots & q_m & 0 & 0 \\ 1 & 0 & \cdots & \cdots & 0 & \vdots & \vdots \\ 0 & \ddots & \ddots & & \vdots & \vdots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & 1 & 0 & 0 & 0 \\ \hline q_0 & \cdots & \cdots & \cdots & q_m & 0 & 0 \end{array} \right)$$

Cette forme correspond à un automate qui, après minimisation, se présente sous la forme proposée en Figure 11. En termes de langages, $(yQ(y))^* = \frac{1}{1-yQ(y)}$.

Modifions cet automate de façon que l'état initial soit ⓘ, de multiplicité [16] m. Chaque

[15] Ce résultat n'est valide que lorsque $\lambda\gamma = 0$, ce qui est le cas dans notre étude.

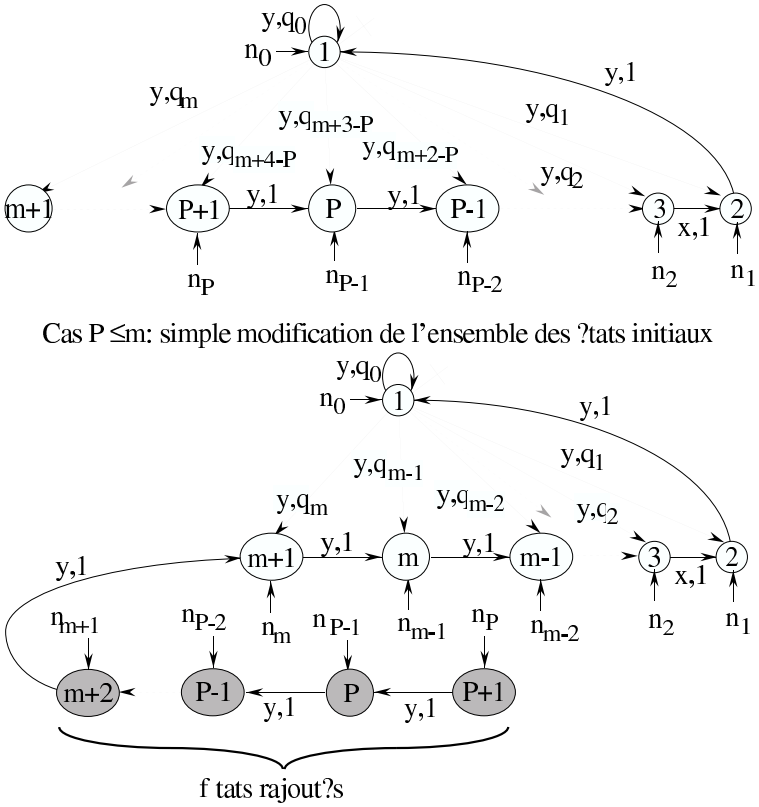

FIG. 12. Automate reconnaissant F(y)

monôme m reconnu par l'automate d'acceptation de $y(Q(y))^*$ est maintenant reconnu multiplié par my^i (coefficient étiquetant le seul chemin de ⓘ à ①, l'état terminal). L'automate modifié accepte donc $\frac{my^i}{1-yQ(y)}$.

Nous effectuons cette opération pour tout $i \in [0, P]$: cela modifie l'automate reconnaissant $(yQ(y))^*$ de telle sorte que l'état *i+1* soit initial et de multiplicité n_i. Notons que l'entrée dans l'état *1* est alors valuée par n_0 et non plus par 1. L'automate obtenu reconnait alors $\frac{1}{1-yQ(y)}\sum_{i=0}^{P} n_i y^i$, c'est à dire $\frac{N(y)}{1-yQ(y)}$. Remarquons que, dans certains cas, cette opération impose de rajouter des états manquants à la structure de l'automate initial. Ces nouveaux états sont alors liés par des transitions $(j, (y, 1), j-1))$. Le nombre d'états de cet automate est finalement $r = Max\,(Deg(N, y), Deg(Q, y)) + 1$. L'automate correspondant est présenté en Figure 12. Sa représentation linéaire (λ, μ, γ) est :

$$\lambda = (n_0, \ldots, n_P, 0_{1\times(r-P-1)})\,,\ \gamma = \begin{pmatrix} 1 \\ 0_{(r-1)\times 1} \end{pmatrix},\ \mu \in \mathbb{N} \ll (p_j)_{j\in[0,\pi]} \gg^{r\times r}$$

[16] L'état ① n'est donc plus initial.

et

$$\mu(y) = \left(\begin{array}{ccccc|c} q_0 & \cdots & \cdots & \cdots & q_m & \\ 1 & 0 & \cdots & \cdots & 0 & \\ 0 & \ddots & \ddots & & \vdots & 0_{(r-m-1)\times(m+1)} \\ \vdots & \ddots & \ddots & \ddots & \vdots & \\ 0 & \cdots & 0 & 1 & 0 & \\ \hline \multicolumn{5}{c|}{0_{(m+1)\times(m+1)}} & 0_{(r-m-1)\times(r-m-1)} \end{array}\right)$$

□

Une fois déterminée la forme linéaire des fonctions génératrices que nous étudions, nous pouvons réaliser le produit de Hadamard de ces fonctions à l'aide de la propriété suivante :

Proposition 6. [22] *Soit* $R : (\lambda^r, \mu^r, \gamma^r)$ *(resp.* $S : (\lambda^s, \mu^s, \gamma^s)$ *). Une représentation du produit de Hadamard est :*

$$R \odot S : (\lambda^r \otimes \lambda^s, \mu^r \otimes \mu^s, \gamma^r \otimes \gamma^s)$$

où $\otimes$ *désigne le produit tensoriel.*

Nous rappelons que, dans ce résultat, le produit tensoriel est l'opération de composition élémentaire des fonctions. Ici, il s'agit du morphisme // défini dans la section 4.1

Grâce à la proposition 6[22], lorsque nous réalisons un produit de Hadamard de langages (ou des automates associés), il n'est plus obligatoire de recalculer totalement le système des fonctions génératrices asociées, mais uniquement d'effectuer les produits de Hadamard de ces fonctions nécessaires. De ce fait, la modularité du modèle à automates se trouve étendue au modèle enrichi.

Le résultat des produits de Hadamard des fonctions génératrices étant exprimé sous forme de représentation linéaire, il nous faut maintenant déterminer l'expression algébrique de la série génératrice associée.

Soit $S(y)$ une série génératrice de représentation linéaire (λ, μ, γ).
Par définition, $S(y) = \sum_{i\geq 0}(S, y^i)y^i$. Comme $(S, y^i) = \lambda\mu(y^i)y^i\gamma$, on obtient :

$$S(y) = \sum_{i\geq 0} \lambda\mu(y^i)y^i\gamma = \lambda\left(\sum_{i\geq 0}\mu(y^i)y^i\right)\gamma = \lambda\left(\sum_{i\geq 0}(\mu(y)y)^i\right)\gamma = \lambda\left(\mu(y)y\right)^*\gamma.$$

Puisque $y\mu(y)$ est une matrice carrée, on calcule $(y\mu(y))^*$ en appliquant récursivement le résultat suivant [20] : soit $M = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ une matrice $n \times n$ structurée en blocs carrés a, b, c et d ; alors,

$$M^* = \left(\begin{array}{c|c} (a + bd^*c)^* & a^*b\,(ca^*b + d)^* \\ \hline d^*c\,(a + bd^*c)^* & (ca^*b + d)^* \end{array}\right)$$

5. Conclusion

L'utilisation des langages rationnels pour l'analyse d'ordonnançabilité s'est donc révélée très fructueuse. D'une part, le modèle est parfaitement adapté aux tâches à durées variables, pour lesquels les résultats établis dans [13] restent valides, et nous obtenons un critère de décision d'ordonnançabilité pour les systèmes de tâches à durée variables, interdépendants, et en environnement multi-processeurs. À notre connaissance, à ce jour, un tel critère n'a été obtenu par aucune autre approche. Par ailleurs, cette approche étend le résultat de la cyclicité des ordonnancements d'un système temps-réel (établi par [2] pour les systèmes périodiques en environnement mono-processeur) d'une part aux systèmes de tâches à durées

variables, et d'autre part aux environnement multi-processeur, sous l'hypothèse de migration totale.

Pour les systèmes intégrant des tâches sporadiques, notre méthodologie fournit des résultats similaires. Ces tâches pouvant se modéliser à l'aide de langages rationnels, les résultats de décision d'ordonnançabilité et de cyclicité obtenus pour les systèmes périodiques persistent pour les systèmes intégrant des sporadiques. Par ailleurs, lorsque les tâches sporadiques sont indépendantes des tâches périodiques, nous utilisons un modèle enrichi (mis en place dans [14]) qui permet, à partir de la seule composante périodique du système, de décider des capacités du système à supporter la surcharge occasionnée par l'activation de tâches sporadiques. Nous fournissons ici une méthode de calcul des fonctions génératrices associées aux transitions compatible avec le produit de Hadamard de langages rationnels.

Actuellement, nous étudions les capacités d'analyse de notre modèle pour les systèmes distribués. Par ailleurs, les études comportementales du procédé de calcul semblent montrer qu'une explosion combinatoire reflète souvent une sous-charge du système analysé. Nous recherchons des raffinements du modèle, dans le but de limiter cette explosion, tout en conservant la puissance des outils d'analyse de langages rationnels. Dans le même ordre d'idée, nous étudions des modes de calcul des fonctions génératrices permettant de limiter la lourdeur du calcul. Le fait que les dénominateurs, par exemple, soient communs à l'ensemble des fonctions est un premier résultat ; la modularité facilitera grandement les calculs de produits. À moyen terme, nous désirons utiliser ces fonctions d'une part dans un but d'analyse de résistance à la surcharge, ou de résistance aux pannes, et d'autre part dans une optique d'analyse statistique des fautes temporelles contrôlées : l'idée sous-jacente est la mise en place d'une **mesure** de validité temporelle d'un système temps-réel, qui permette d'évaluer continûment sa validité, plutôt que de manière binaire, comme c'est le cas actuellement. À plus long terme, notre objectif est de mettre en place une méthodologie d'aide à la spécification temporelle d'applications temps-réel qui intègre l'aspect *gestion des risques* : les risques correspondront à des faiblesses connues et tolérées, qui pourront faire partie de la spécification temporelle.

Références

[1] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1) :46–61, 1973.

[2] E. Grolleau. *Ordonnancement Temps-Réel Hors-Ligne Optimal à l'Aide de Réseaux de Petri en Environnement Monoprocesseur et Multiprocesseur*. PhD thesis, Univ. Poitiers, 1999.

[3] A. Choquet-Geniet. Ordonnancement des applications temps-réel. In *Actes de l'école d'éte École Temps Réel'99*, pages 53–68. ENSMA, 1999.

[4] T.P. Baker. Stack-based scheduling of real-time processes. *the Journal of Real-Time Systems*, 3 :67–99, 1991.

[5] J. Delacroix. Ordonnancement de tâches apériodiques. In *Actes de l'école d'éte E.T.R'99*, pages 69–82, 1999.

[6] S.K. Baruah, L.E. Rosier, and R.R. Howell. Algorithms and complexity concernig the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems*, pages 301–324, 1990.

[7] J.Y.T. Leung and M.L. Merill. A note on preemptive scheduling of periodic real-time tasks. *Information Processing Letters*, 11(3) :115–118, 1980.

[8] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126 :183–235, 1994.

[9] G. Florin and S. Natkin. Les réseaux de petri stochastiques. *Techniques et Sciences Informatiques*, 4(1) :143–160, February 1985.

[10] C. Ramchandani. Analysis of asynchronous concurrent systems by petri nets. Technical report, Massachuchet Institute of Technology, project MAC, 1974.

[11] A. Choquet-Geniet, D. Geniet, and F. Cottet. Exhaustive computation of the scheduled task execution sequences of a real-time application. In *Proc. of Symposium on Formal Techniques in Real Time and Fault Tolerent Systems'96*, number 1135 in L.N.C.S., pages 246–262. Springer-Verlag, October 1996.

[12] D. Geniet and G. Largeteau. Validation temporelle de systèmes de tâches temps-réel strictes à durées variables à l'aide de langages. In *Proc. of Mod'elisation des Systèmes Réactifs' 2001*, pages 243–258. Hermes, October 2001.

[13] D. Geniet. Validation d'applications temps-réel à contraintes strictes à l'aide de langages rationnels. In *RTS'2000*, pages 91–106. Teknea, 2000.

[14] D. Geniet and J.P. Dubernard. Ordonnancement de tâches sporadiques à contraintes strictes à l'aide de séries génératrices. In *Proc. of Real Time Systems'01*, pages 149–166. Teknea, 2001.

[15] L. Aceto, P. Bouyer, A. Burgue, and K. G. Larsen. The power of reachability testing for timed automata. In *Proc. of* 18^{th} *Conf. Found. of Software Technology and Theor. Comp. Sci.*, LNCS 1530, pages 245–256. Springer-Verlag, December 1998.

[16] A. Arnold and M. Nivat. Comportements de processus. Technical Report 82-12, Univ. Paris 7, 1982.

[17] S. Eilenberg. *Automata Languages and machines*, volume A. Academic Press, 1976.

[18] A. Arnold. *Finite transition systems*. Prentice Hall, 1994.

[19] G. Largeteau, D. Geniet, and J.P. Dubernard. Validation of distributed periodic real-time systems using can protocol with finite automata. In *Proc. of* 5^{th} *S.C.I.* I.I.I.S., 2001.

[20] J. Berstel and C. Reutenauer. Rational series ans their languages. In *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1988.

[21] M. Fliess. Sur divers produits de séries formelles. *Bull. Sc. Math.*, 104 :181–191, 1974.

[22] G. Duchamp, M. Flouret, É. Laugerotte, and J.G. Luque. Direct and multiple laws for automata with multiplicities. *Theoretical Computer Science*, 267, 2001.

Laboratoire d'Informatique Fondamentale et Appliquée de Rouen, Université de Rouen, Place Émile Blondel, F-76821 Mont-Saint-Aignan Cédex
E-mail address: Jean-Philippe.Dubernard@univ-rouen.fr

Laboratoire d'Informatique Scientifique et Industrielle, Université de Poitiers & E.N.S.M.A., Téléport 2, 1 av. C. Ader, BP 40109, F-86961 Futuroscope Chasseneuil Cédex
E-mail address: dgeniet@ensma.fr