N° D'ORDRE



# THÈSE DE DOCTORAT

SPECIALITE : PHYSIQUE

*Ecole Doctorale « Sciences et Technologies de l'Information des Télécommunications et des Systèmes »*

Présentée par : *Patrick MEUMEU YOMSI*

Sujet :

## PRISE EN COMPTE DU COÛT EXACT DE LA PRÉEMPTION DANS L'ORDONNANCEMENT TEMPS RÉEL MONOPROCESSEUR AVEC CONTRAINTES MULTIPLES

……

Soutenue le 02 Avril 2009 devant les membres du jury :

| | |
|---|---|
| M. Alain Mérigot | Président |
| Mme Maryline Silly-Chetto | Rapporteurs |
| M. Ye-Qiong Song | |
| M. Laurent George | Examinateurs |
| M. Joël Goossens | |
| M. Yves Sorel | |

Directeur de thèse : Yves Sorel
Directeur de Recherche, INRIA

Qu'est ce qu'une idée ?
C'est une image qui se peint dans mon cerveau

(Voltaire)

Les travaux présentés dans cette thèse ont été financés par une bourse INRIA, Institut National de Recherche en Informatique et Automatique.

# Remerciements

*Soyons reconnaissants aux personnes qui nous donnent*
*du bonheur ; elles sont les charmants jardiniers*
*par qui nos âmes sont fleuries.*
**Marcel Proust**

*Le temps met tout*
*en lumière.*
**Thalès**

Le seul moyen de se délivrer d'une tentation, c'est d'y céder paraît-il ! Alors j'y cède en disant en grand Merci aux personnes qui ont cru en moi et qui m'ont permis d'arriver au bout de cette thèse.

Je tiens à exprimer mes plus vifs remerciements à Yves Sorel qui fut pour moi un directeur de thèse attentif et disponible malgré ses nombreuses charges. Sa compétence, sa rigueur scientifique et sa clairvoyance m'ont beaucoup appris. Ils ont été et resteront des moteurs de mon travail de chercheur.

J'exprime tous mes remerciements à l'ensemble des membres de mon jury : Madame Maryline Silly-Chetto et Messieurs Ye-Qiong Song, Alain Mérigot, Laurent George, Joël Goossens et Yves Sorel.

J'adresse toute ma gratitude à tous mes ami(e)s et à toutes les personnes qui m'ont aidé dans la réalisation de ce travail. Je remercie Madame Sophie Cluet et Monsieur Antoine Petit pour m'avoir accueilli dans l'unité de recherche INRIA-Paris Rocquencourt et de m'avoir permis de travailler dans d'aussi bonnes conditions.

Je remercie toutes les personnes formidables que j'ai rencontrées par le biais de l'INRIA. Merci pour votre support et vos encouragements. Je pense particulièrement à toutes les assistantes des équipe-projets, à toutes les personnes du service de communication et en particulier à ma petite Gaëlle Foret qui a pris le temps de lire les premières pages de cette thèse. Je ne saurais terminer sans remercier toutes ces personnes dans l'ombre dont la contribution à mon travail est non négligeable, les conducteurs des navettes, les personnels de la cantine et les techniciens de surface.

Je remercie tous les membres et ex-membres de l'équipe-projet AOSTE (Modèles et Méthodes pour l'Analyse et l'Optimisation des Systèmes Temps-Réel Embarqués) pour

le climat sympathique dans lequel ils m'ont permis de travailler. Les nombreuses discussions que j'ai pu avoir avec chacun m'ont beaucoup apporté. Merci donc à Yves Sorel, Christophe Gensoul, Liliana Cucu, Nicolas Pernet, Cyril Faure, Djamel Louar, Fadoi Lakhal, Arnaud Rouanet, Dimitru Potop, Omar Kermia, Cecile Stentzel, Mohamed Marouf et Daniel de Rauglaudre. Je voudrais exprimer particulièrement toute mon amitié à Nelly Maloisel, Christine Anocq et Laurence Bourcier pour leur gentillesse, leur compétence et leur humour.

Mention spéciale à Daniel de Rauglaudre qui m'a supporté et m'a permis de me lever motivé, le coeur léger et l'esprit tranquille depuis le début de ma thèse. Très humblement, je voudrais te dire merci pour ton soutien pendant mes périodes de doutes et pour tes multiples encouragements répétés. Tu as su mettre en musique les paroles de ma composition scientifique.

Enfin, les mots les plus simples étant les plus forts, j'adresse toute mon affection à ma famille, et en particulier à ma maman qui m'a fait comprendre que la vie n'est pas faite que de problèmes qu'on pourrait résoudre grâce à des formules mathématiques et des algorithmes. Malgré mon éloignement depuis de (trop) nombreuses années, leur intelligence, leur confiance, leur tendresse, leur amour me portent et me guident tous les jours. Merci pour avoir fait de moi ce que je suis aujourd'hui. Est-ce un bon endroit pour dire ce genre de choses ? Je n'en connais en tous cas pas de mauvais. Je vous aime.

Une pensée pour terminer ces remerciements pour toi qui n'a pas vu l'aboutissement de mon travail mais je sais que tu en aurais été très fier de ton fils ! ! !

*à mes parents,*
*et à tous ceux que je ne nomme pas, mais qui se reconnaîtront.*

# Résumé

*La promesse de la chenille*
*n'engage pas le papillon.*
**André Gide**

Nous nous intéressons aux problèmes d'ordonnancement de tâches périodiques dans les systèmes temps réel critiques (durs). De tels systèmes nécessitent un respect absolu de toutes les contraintes qui les caractérisent sinon des conséquences catastrophiques peuvent survenir.

Dans la littérature, il existe des algorithmes d'ordonnancement à priorités fixes permettant une analyse d'ordonnançabilité de ces systèmes. Cependant, l'approximation du coût de la préemption, qui est la partie variable du coût du système d'exploitation (OS), dans le WCET des tâches conduit à faire un compromis entre gaspillage et sûreté de l'ordonnançabilité, ce qui n'est pas satisfaisant. Quelques travaux ont été proposés pour résoudre ce problème mais les résultats conduisent à la prise en compte soit d'un nombre minimal soit d'un nombre maximal de préemptions. Par conséquent, il n'est pas possible de garantir de façon sûre l'ordonnançabilité d'un système temps réel critique.

Dans cette thèse, les modèles classiques n'étant pas adaptés pour prendre en compte le coût exact de la préemption, nous introduisons un nouveau modèle pour résoudre le problème général de l'ordonnancement de systèmes temps réel durs avec des contraintes multiples telles que la précédence, la périodicité stricte, la latence et la gigue, tout en tenant compte du coût exact de la préemption pour tous les scénarii de premières activations de toutes les tâches (simultané ou non simultané). Fondé sur ce nouveau modèle, nous proposons une analyse d'ordonnançabilité qui utilise la définition d'une opération binaire d'ordonnancement $\oplus$ dont les opérandes sont appelés otâches. Nous montrons la correspondance entre une otâche et une tâche périodique. Nous examinons deux approches. Premièrement, nous considérons le cas où la priorité de chaque otâche est connue (Rate Monotonic, Deadline Monotonic, Audsley, etc.), conduisant ainsi à un ordre décroissant des priorités des otâches. Deuxièmement, nous considérons le cas où la priorité de chaque otâche n'est pas connue. Nous étudions de près l'impact du scénario de premières activations de toutes les otâches sur l'analyse d'ordonnançabilité. Nous prouvons d'une part que le scénario, où les premières activations de toutes les otâches sont simultanées, ne correspond pas au pire cas, et d'autre part qu'il n'existe pas un tel scénario lorsque le coût de la préemption est pris en compte. Ensuite, nous prouvons que le choix des priorités des otâches fondé sur la politique d'Audsley n'est plus applicable,

ni optimal. Pour résoudre ce problème, nous proposons un algorithme optimal de choix de priorités des otâches. Ici optimal signifie que s'il existe une politique de choix des priorités qui conduit à un ordonnancement valide alors le choix des priorités proposé permettra également d'obtenir un ordonnancement valide.

Nous avons développé un logiciel appelé SAS (Simulation Analysis of Scheduling) pour mettre à la disposition d'utilisateurs les résultats que nous avons obtenus. SAS est un logiciel qui effectue l'analyse d'ordonnançabilité de systèmes d'otâches périodiques dans le cas monoprocesseur. Sa principale contribution, par rapport à d'autres outils commerciaux et académiques du même type, est qu'il prend en compte le coût exact de préemption au cours de l'analyse d'ordonnançabilité avec des contraintes multiples.

# Table des matières

# Table des figures

# Introduction générale

*La science consiste à passer*
*d'un étonnement à un autre.*
**Aristote**

La théorie de l'ordonnancement vise à déterminer l'ordre dans lequel certaines tâches doivent être exécutées avant certaines contraintes d'échéance. En tant qu'êtres humains, nous traitons des problèmes d'ordonnancement tous les jours. Par exemple, un étudiant doit faire ses devoirs avant une échéance appropriée, un chercheur doit avoir terminé l'ébauche de son article avant la date de soumission, etc. Lorsque nous avons une seule tâche à exécuter, respecter son échéance est relativement simple. Cependant notre vie quotidienne est faite de plusieurs tâches ayant chacune des échéances pouvant être différentes et plus ou moins fortes (feuilles d'impôt, révisions de la voiture, réunions, etc). Nous devons donc employer des techniques d'ordonnancement pour traiter nos diverses tâches, de sorte qu'elles soient toutes exécutées avant leurs échéances. Il y a certaines tâches dont les échéances ne sont pas strictes (on peut demander une prolongation pour soumettre sa fiche d'impôts, ou le report de la date de révision de sa voiture, etc). Par contre il y a d'autres tâches qui, elles, ont des échéances beaucoup plus strictes (une convocation judiciaire, une proposition de financement, etc). Dans de telles situations, si l'échéance est manquée, cela peut avoir de graves conséquences (par exemple, être envoyé en prison, ne pas recevoir de financement, etc). Ces exemples montrent que nous vivons au quotidien en temps réel. En plus des échéances à respecter, il existe aussi d'autres contraintes. Une situation typique est celle d'une demande de financement : sa fiche de demande doit être correctement remplie, mais également, elle doit être soumise avant l'échéance.

Dans cette thèse nous ne considérerons que des systèmes *temps réel durs*. Un système *temps réel* est un système dont la correction ne dépend pas seulement des résultats logiques de ses traitements, mais dépend aussi de la date à laquelle ils sont produits [Sta88]. Les systèmes temps réel *durs* sont des systèmes temps réel pour lesquels le coût associé à l'echec d'une échéance est extrêmement élevé. Un exemple de système temps réel dur est un ordinateur qui commande le freinage lors de l'atterrissage d'un avion : les gouvernails de direction et les ABS doivent répondre aux entrées des capteurs avant une échéance précise. Si les réponses sont incorrectes ou sont trop tardives, l'avion peut s'écraser. En raison de la nature des systèmes temps réel durs pour lesquels il est généralement difficile d'estimer le coût du système d'exploitation / *OS*, nous nous attache-

rons principalement à la garantie à priori du bon fonctionnement de tels systèmes sans faire aucune approximation sur les temps d'exécution des tâches à exécuter. En effet si une politique d'ordonnancement produit des ordonnancements valides en moyenne, mais est connue pour échouer dans quelques situations, alors personne ne souhaiterait faire confiance à une telle politique pour la commande du système d'atterrissage d'un avion.

Dans la littérature, les systèmes temps réel durs sont habituellement composés d'un ensemble de tâches ayant chacune une infinité d'activations dans le temps. Nous nous concentrerons principalement sur l'étude des tâches périodiques, et nous considérerons des tâches sporadiques, le cas échéant, avant de faire une généralisation de ce modèle. Dans tous les cas, chaque activation d'une tâche aura une échéance correspondante à la date avant laquelle la tâche devra avoir absolument terminé son exécution. De plus, les tâches seront éventuellement soumises à d'autres multiples contraintes de précédence, de périodicité stricte, de latence ou encore de gigue d'activation sur leurs périodes qu'il faudra aussi garantir avant de déclarer le système ordonnançable.

En général, un *algorithme d'ordonnancement* est nécessaire pour pouvoir garantir l'ordonnançabilité d'un système donné. Un algorithme d'ordonnancement est un algorithme qui utilise les informations sur l'ensemble des tâches, et détermine quand ordonnancer quelle tâche. Tous les algorithmes d'ordonnancement peuvent être considérés comme étant à priorités (la tâche ayant la priorité la plus élevée et dont l'exécution n'est pas encore achevée doit être ordonnancée). Dans ce contexte, on distingue deux grandes classes d'algorithmes d'ordonnancement :

1. la classe des algorithmes pour lesquelles les priorités des tâches sont fixes et ne changent pas pendant leur exécution (dans ce cas, on parle de *priorités statiques*) [LL73, Law73, LW82, LSD89],
2. la classe des algorithmes pour lesquelles les priorités des tâches peuvent changer pendant leur exécution (dans ce cas, on parle de *priorités dynamiques*) [LL73, LM80, Mok83, BMR90, SRL93].

Dans cette thèse, nous nous intéresserons essentiellement à la classe des algorithmes d'ordonnancement à priorités fixes dans le cas monoprocesseur car les algorithmes à priorités dynamiques présentent des désavantages génants pour les applications visées. Ils sont :

1. difficilement prédictibles – les temps de réponse des tâches sont beaucoup plus difficiles à calculer dans la classe des algorithmes à priorités dynamiques que dans celle à priorités statiques. Dans la classe des algorithmes à priorités dynamiques, le temps de réponse d'une tâche dépend de toutes les tâches du système alors que dans celle des algorithmes à priorités statiques, il ne dépend que de l'ensemble des tâches plus prioritaires –,

2. moins contrôlables – lorsque nous voulons réduire le temps de réponse d'une tâche, il peut suffir d'augmenter sa priorité dans la classe des algorithmes à priorités fixes tandis qu'on ne peut absolument rien faire dans la classe des algorithmes à priorités dynamiques –,

3. plus sensibles aux "effets domino" – dans les cas de surcharge du système, nous pouvons avoir des effets domino, c'est-à-dire quasiment toutes les tâches violent leur échéance dans une plage de temps très réduite dans la classe des algorithmes à priorités dynamiques tandis que seules les tâches ayant les priorités les plus faibles dans la classe des algorithmes à priorités fixes violeraient leur échéance –.

Les modèles classiques n'étant pas adaptés pour prendre en compte le coût exact de la préemption, nous introduirons un nouveau modèle pour résoudre le problème général de l'ordonnancement de systèmes temps réel durs avec des contraintes multiples telles que la précédence, la périodicité stricte, la latence et la gigue, **tout en tenant compte du coût exact du système d'exploitation (*OS*)**. Nous tiendrons compte du coût exact du système d'exploitation afin d'éviter d'une part le gaspillage de la ressource (CPU), et d'autre part garantir le bon fonctionnement du système lors de l'exécution. En effet le coût de l'OS est constitué de deux parties : une partie fixe, facile à déterminer, liée à l'ordonnanceur et une partie variable, difficile à déterminer, liée à l'occurrence de chaque préemption lors de l'exécution des tâches. Le nouveau modèle proposé sera utilisable pour tout scénario de première activation de toutes les tâches (simultané ou non simultané) et permettra de résoudre des problèmes rarement proposés dans les autres modèles existants dans la littérature. Fondé sur ce nouveau modèle, nous proposerons une analyse d'ordonnançabilité qui utilisera la définition d'une opération binaire d'ordonnancement $\oplus$ dont les opérandes seront appelés otâches. Nous montrerons la correspondance entre une otâche et une tâche. Nous examinerons deux approches : le cas où la priorité de chaque otâche est connue et le cas où la priorité de chaque otâche n'est pas connue. Dans le second cas, nous proposerons un algorithme de choix de priorités *optimal* des otâches. Ici optimal signifie que s'il existe une politique de choix des priorités qui conduit à un ordonnancement valide alors le choix des priorités proposé permettra également d'obtenir un ordonnancement valide. Nous présenterons enfin un logiciel appelé *SAS* pour *Scheduling, Analysis and Simulation of Real-Time Systems* pour montrer l'applicabilité de notre approche. Ce logiciel permettra à l'utilisateur d'effectuer l'analyse d'ordonnançabilité de systèmes d'otâches périodiques dans le cas monoprocesseur. La principale contribution de *SAS*, par rapport à d'autres outils commerciaux et académiques du même type, est qu'il prend en compte le coût exact de préemption au cours de l'analyse d'ordonnançabilité.

## Structure de la thèse

**Le chapitre 1** présente les notions de base nécessaires à compréhension de la suite de cette thèse.

**Le chapitre 2** présente la théorie classique de l'ordonnancement. Il introduit le lecteur à l'ordonnancement temps réel, rappelle les concepts de base liés à cette problématique et fait un bref rappel de l'état de l'art. Il présente également quelques outils d'analyse communément utilisés. Comme nous étudions les algorithmes préemptifs dans un cadre où chaque tâche a une priorité fixe, il présente les algorithmes classiques et les différentes approches d'analyse connues. Il présente enfin le modèle que nous avons choisi pour prendre en compte le coût exact de la préemption et le choix de notre approche d'analyse.

**Le chapitre 3** présente de la première partie de notre contribution : il concerne la théorie de l'ordonnancement temps réel sans coût de la préemption. Il présente le nouveau modèle qui nous permettra de résoudre le problème général de l'ordonnancement de systèmes temps réel durs avec des contraintes multiples telles que la précédence, la périodicité stricte, la latence et la gigue, tout en tenant compte du coût exact du système d'exploitation (*OS*). Il fait la correspondance entre les notions introduites et l'ordonnancement temps réel classique et présente également des résultats concernant la réduction du domaine d'analyse d'ordonnançabilité d'un système lorsque le coût de la préemption n'est pas pris en compte. Enfin il présente l'opération d'ordonnancement $\oplus$ dans ce cadre lorsque les priorités sont imposées.

**Le chapitre 4** présente la seconde partie de notre contribution : il concerne la théorie de l'ordonnancement temps réel avec prise en compte du coût exact de la préemption et utilise le modèle introduit dans le chapitre 3. Il présente dans un premier temps l'impact de la prise en compte du coût de la préemption sur l'ordonnancement d'un système temps réel à travers quelques théorèmes. Dans un second temps, il présente l'opération d'ordonnancement $\oplus$ dans ce cadre lorsque les priorités sont imposées. Enfin, il présente l'impact du choix des priorités sur l'ordonnancement et propose un choix optimal des priorités. Le choix ici étant optimal au sens que s'il existe un ordonnancement valide pour le système considéré, alors le choix de priorités proposé conduira aussi à un ordonnancement valide.

**Le chapitre 5** présente la troisième partie de notre contribution : il concerne l'ordonnancement temps réel avec contraintes multiples. Dans ce chapitre, en plus de prendre en compte le coût exact de la préemption dans les conditions d'ordonnançabilités, des contraintes temps réel sont prises en compte. Les contraintes temps réel prises en compte

sont la précédence, la périodicité stricte, la latence et la gigue. Ces contraintes sont prises en compte individuellement puis conjointement avec les autres contraintes. Ce chapitre présente l'impact de chaque contrainte sur les conditions d'ordonnançabilité obtenues, puis adapte l'opération d'ordonnancement dans ce cadre.

**Le chapitre 6** présente le logiciel **SAS** – *Scheduling, Analysis and Simulation of Real-Time Systems* – que nous avons développé et utilisé pour réaliser la majorité des figures présentées dans cette thèse. **SAS** est un logiciel de simulation et d'analyse d'ordonnancements temps réel qui a été utilisé pour illustrer la majorité des résultats théoriques présentés dans cette thèse. Il diffère des autres outils académiques ou commerciaux du même type par le fait qu'il intègre la prise en compte du coût exact de la préemption dans les conditions d'ordonnançabilité, prend en compte des contraintes temps réel multiples telles que la précédence, la périodicité stricte ou encore la latence. Il illustre une simulation de l'ordonnancement obtenu. Ce logiciel est programmé en *OCaml* et s'exécute pour le moment sous le système d'exploitation *Linux* et sous le système d'exploitation *Windows*. *OCaml* est un langage de programmation fonctionnel fortement typé developpé à l'*INRIA*.

Nous terminons notre présentation par une conclusion et donnerons perspectives à ce travail de thèse. Nous décrivons dans le dernier chapitre des applications de notre travail et présentons quelques axes d'amélioration futures.

# Chapitre 1
# Notions de base

*L'imagination est plus importante*
*que la connaissance.*
**Albert Einstein**

## 1.1 Introduction

Un système temps réel est un système de traitement de l'information dont le but est la commande d'un processus en respectant les contraintes auxquelles il est soumis à travers son interface avec ce dernier. Il diffère d'un système de traitement d'informations classique par le fait que la valeur d'une donnée qu'il produit ne dépend pas seulement de la correction de son calcul mais aussi de la date à laquelle la donnée est disponible. Le respect des contraintes temporelles est donc un facteur prédominant dans l'évaluation de la qualité d'un système temps réel au moment de sa conception. On retrouve les contraintes temporelles dans de nombreux secteurs d'activités tels que l'aéronautique au travers des systèmes de pilotage embarqués (avions, satellites), l'automatisation d'ensembles industriels au travers des systèmes de contrôle de procédé (usines, centrales nucléaires), le contrôle du traffic aérien, les télécommunications, l'automobile, la robotique, le suivi des valeurs boursières dans les salles de marché, etc.

## 1.2 Systèmes temps réel

Nous nous intéressons à des applications pour lesquelles un ordinateur assure la commande (ou la supervision) d'un processus en temps réel. Par conséquent, il est raisonnable de diviser de telles applications en deux grandes parties : le *système de contrôle* ou *contrôleur* et *le processus contrôlé* ou *procédé*. Cette seconde partie correspond à l'entitée physique à laquelle le système de contrôle est connecté afin de contrôler le comportement. Ainsi, le contrôleur doit fournir une politique "simple" de gestion des tâches à exécuter au cours du temps. Concrètement, les applications déclenchent des événements périodiques, sporadiques ou aléatoires (stimuli) à travers des *capteurs*, et exigent que le

système informatique associé réagisse avant une échéance donnée ou un temps fixe à travers des *actuateurs*. La latitude temporelle de réaction du système est limitée puisque les réponses ou les commandes du système doivent être envoyées avant une échéance précise et prédéfinie. La figure 1.1 illustre un aperçu typique des interactions qui existent entre contrôleur et procédé d'un système temps réel.

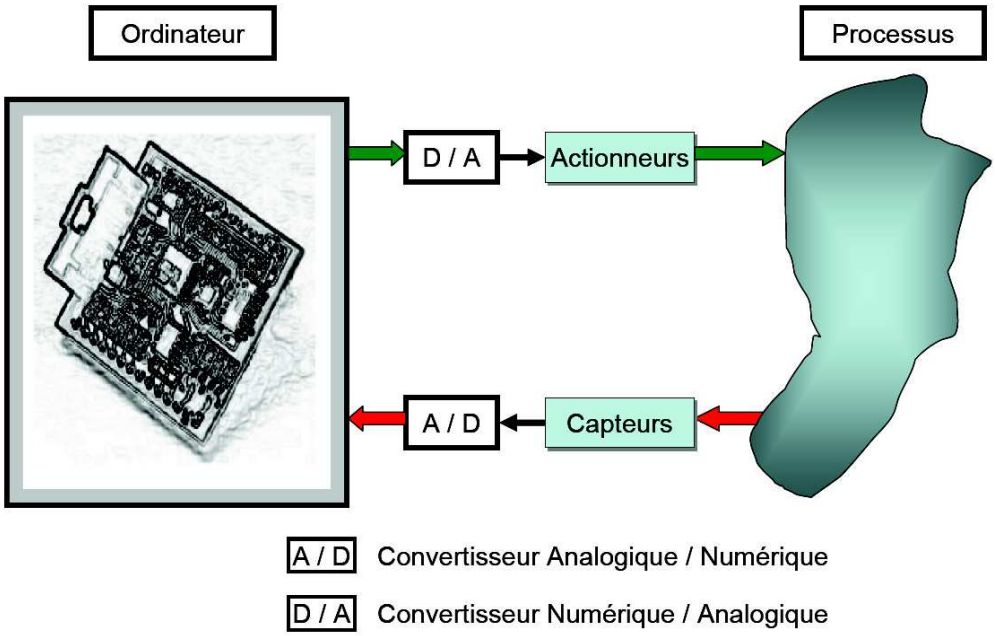


FIG. 1.1 – Interaction entre contrôleur et procédé d'un système temps réel.

Notons que l'échelle de temps peut fortement varier d'une application donnée à une autre, elle est par exemple de l'ordre d'une micro-seconde pour un radar, une seconde pour un humain, une minute pour une chaîne de montage, ou une heure pour réaction chimique.

Le type de contraintes temporelles mène à classifier les applications temps réel suivant deux grandes catégories : *strictes / dures / hard* ou *molles / souples / soft*. Les applications ont des contraintes temps réel *dures* lorsque la moindre faute temporelle peut avoir comme conséquence un désastre économique, humain ou écologique : pilote automatique d'avion, système de surveillance de centrale nucléaire, etc. A l'inverse, elles ont des contraintes temps réel *souples* dans le cas où les fautes temporelles causent des dommages dont le coût est considéré tolérable sous certaines conditions sur la fréquence des fautes ou le retard des services. Dans ce cas, la principale contrainte du système d'exploitation (OS) est de garantir statistiquement la qualité de service (*QoS*) des paramètres tels que les échéances des tâches. Le dépassement de quelques échéances n'est

pas fatal du moment que la majorité des échéances sont respectées : téléphonie, télévision HD, visioconférence, jeux en réseau, etc. Dans cette thèse, nous ne considèrerons que des applications temps réel dures.

## 1.3 Modèles de tâches

Un système temps réel est principalement constitué d'un certain nombre de *ressources* (un ou plusieurs processeurs), d'un certain nombre de *tâches* devant respecter des contraintes, et d'un *ordonnanceur* qui assigne les tâches au(x) processeur(s) suivant une politique d'ordonnancement. Nous ferons l'hypothèse que nous ne disposons que d'une seule ressource, c'est-à-dire d'un seul processeur : le CPU. Par conséquent, une tâche est un programme devant s'exécuter de façon séquentielle, éventuellement plusieurs fois, sur le CPU sans s'auto-interrompre mais pouvant être interrompue par une autre tâche ou par l'ordonnanceur. Il est important de noter que l'ordonnanceur est aussi un programme devant s'exécuter de façon séquentielle, et de plus qui ne peut pas être interrompu par une tâche. Une tâche est généralement constituée d'une infinité d'activations ou instances d'exécution. Elle peut donc être soit *périodique*, dans ce cas il y a un nombre d'unités de temps constant entre deux activations successives ; soit *sporadique*, chaque activation doit arriver au moins un temps constant après l'activation précédente ; ou enfin *apériodique*, il n'y a pas de corrélation entre deux activations successives. A chaque instant, la tâche dans l'état *"exécution"/"running"* est celle qui est considérée comme la plus prioritaire parmi les tâches candidates à l'attribution du CPU (tâches dans l'état *"prêt"/"ready"*, la tâche dans l'état *"bloqué"/"blocked"* est celle dont l'exécution est interrompue et qui est en attente d'un événement. La tâche dans l'état *"inactif"/"inactive"* est celle qui soit n'existe pas encore car elle attend son activation, soit qui a terminé son exécution et attend une prochaine activation. La figure 1.2 illustre les différents états possibles d'une tâche ainsi que les transitions d'un état à un autre. Ce fonctionnement est assimilable à la gestion d'un automate pour chaque tâche par l'ordonnanceur, les événements d'entrées de l'automate de chaque tâche correspondant aux transitions dans la figure 1.2. Dans ce cas, l'événement "sélectionner" est produit par l'ordonnanceur tandis que l'événement "terminer" est produit par la tâche elle-même. A l'arrivée de l'événement *"activer"* d'une des tâches, l'ordonnanceur compare sa priorité avec la priorité de la tâche qui est en cours (état *"exécution"*) et celles de toutes les tâches qui sont dans l'état *"prêt"*. Si la priorité de cette tâche est supérieure, il produit un événement *"préempter"* pour l'automate de la tâche en cours, sinon il ne fait rien. De même, à l'arrivée de l'événement *"terminer"* de la tâche en cours, l'ordonnanceur compare les priorités des tâches dans l'état *"prêt"* et produit l'événement *"sélectionner"* pour l'automate de la tâche la plus prioritaire.

Par la suite, nous nous concentrerons principalement sur l'étude d'un ensemble $\Gamma$

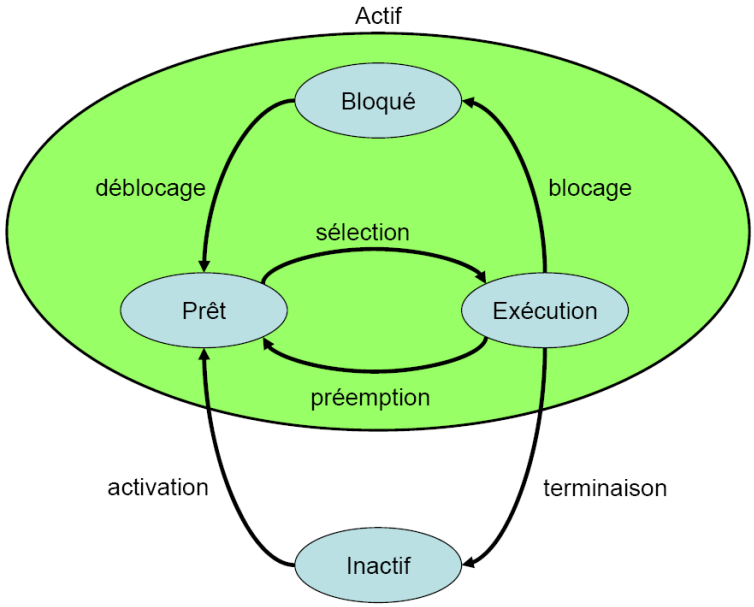


FIG. 1.2 – Illustration des états et transitions possibles d'une tâche.

de tâches périodiques. Nous utiliserons deux modèles différents de tâches : le *modèle classique* et le *modèle sans date de réveil* détaillés ci-dessous. Ces deux modèles tirent leur essence de la variété des domaines d'application dans lesquels on les retrouve. Dans ces deux modèles, toutes les caractérisques temporelles sont des entiers positifs ou nuls, c'est-à-dire elles sont multiples d'un certain intervalle atomique (par exemple le "tick du CPU", la plus petite unité de temps indivisible du processeur). Ainsi, un *ordonnancement* de $\Gamma$ est une fonction $Ordo : \mathbb{N}^+ \mapsto \Gamma \cup \{\emptyset\}$ telle que $Ordo(t) = \tau_i$ si la tâche $\tau_i$ s'exécute à l'instant $t$ et $Ordo(t) = \emptyset$ si aucune tâche ne s'exécute à l'instant $t$.

### 1.3.1 Modèle de tâches classique

Ce modèle de tâches, introduit par Liu & Layland [LL73] pour l'étude des tâches périodiques, est le plus utilisé lors de la conception des systèmes temps réel. Dans ce modèle, une tâche $\tau_i = (r_i^1, C_i, D_i, T_i)$ est caractérisée par une date de première activation ou first release $r_i^1$, une période $T_i$, une échéance relative $D_i$ (durée maximale pour l'exécution de $\tau_i$ depuis sa dernière activation), et une durée d'exécution $C_i$ supposée constante à chaque activation de $\tau_i$. Les tâches étant périodiques, la date de la $k^{ieme}$ activation de la tâche $\tau_i$ est donnée par $r_i^k = r_i^1 + (k-1)T_i$, et elle doit terminer son exécution avant la date $d_i^k = r_i^1 + (k-1)T_i + D_i$. Le temps qui s'écoule entre la date d'activation et la date de fin d'exécution de la $k^{ieme}$ activation désigne le *temps de ré-*

*ponse* $R_i^k$ et le maximum de tous les temps de réponse désigne le *pire temps de réponse* $R_i$ de la tâche $\tau_i$. On suppose que l'inégalité $C_i \leq D_i \leq T_i$ est toujours satisfaite et que la valeur du paramètre $C_i$ correspond à la pire durée d'exécution possible de $\tau_i$, en incluant toutes les approximations telles que le coût des préemptions lorsqu'elles sont autorisées, et donc le coût du système d'exploitation (*OS*). Cette dernière assertion est une hypothèse importante dans tous les travaux qui ont suivi le papier fondateur de Liu & Layland [LL73]. La figure 1.3 illustre ce modèle de tâches.

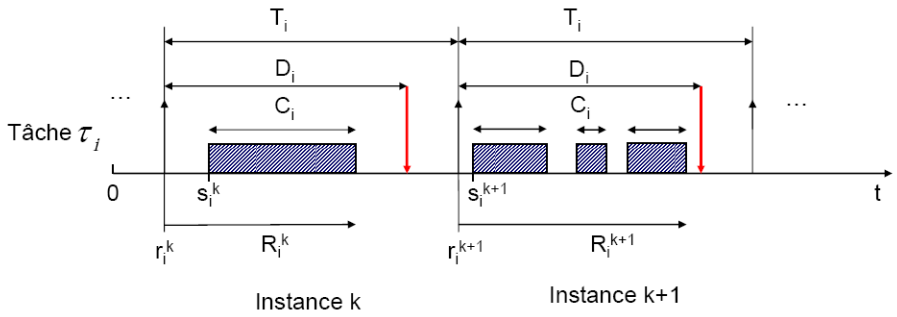


FIG. 1.3 – Illustration du modèle de tâches classique.

Etant donné un système de $n$ tâches périodiques : $\Gamma_n = \{\tau_1, \tau_2, \cdots, \tau_n\}$, lorsque toutes les tâches ont la même date de première activation, c'est-à-dire $r_i^1 = r_j^1 \quad \forall i, j \in \{1, 2, \cdots, n\}$, le système est dit *synchrone* ou encore *à démarrage simultané*, sinon il est dit *asynchrone* ou encore *à démarrage non simultané*.

### 1.3.2 Modèle de tâches sans date de réveil

Ce modèle de tâches est très utilisé pour la modélisation de circuits intégrés [Cuc04]. S'il semble être un cas particulier du modèle classique (voir figure 1.3), il présente cependant une différence majeure par rapport à ce dernier. En effet si on connaît la date de début d'exécution effective $s_i^1$ de la première activation d'une tâche $\tau_i$ donnée, alors la date de début d'exécution effective $s_i^k$ de la $k^{ieme}$ activation de $\tau_i$ est aussi connue et elle est donnée par $s_i^k = s_i^1 + (k-1)T_i$, où $T_i$ est la période de la tâche $\tau_i$. Cette particularité relativement "forte" du modèle permet une meilleure prédictibilité du comportement des systèmes temps réel relativement à leurs entrées/sorties, par exemple lors de la modélisation de problèmes provenant de l'automatique et du traitement de signal [KAL96, Cuc04]. Ce modèle peut ou pas autoriser de préemption, c'est-à-dire une tâche qui a démarré son exécution peut être interrompue au profit d'une autre tâche jugée plus importante. Il a été utilisé dans [Cuc04] dans le cadre *non préemptif* afin d'éviter des

approximations dans les durées d'exécution des tâches. La figure 1.4 illustre ce modèle de tâches.

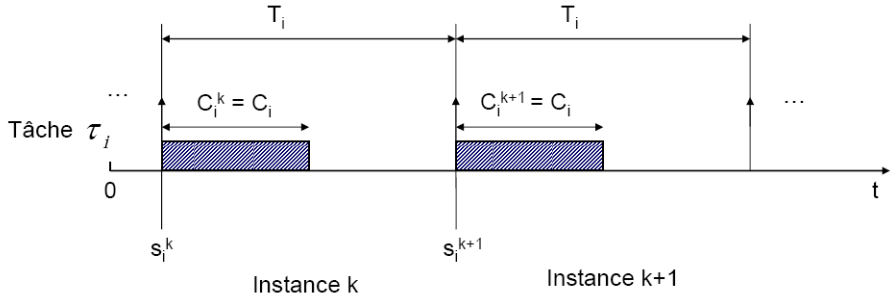


FIG. 1.4 – Illustration du modèle de tâches sans date de réveil.

Dans cette thèse, nous utiliserons ce modèle en autorisant les préemptions entre les tâches et en prennant en compte leurs coûts exacts.

## 1.4 Contraintes temps réel

Dans la section 1.2, nous avons introduit la notion de contraintes des systèmes temps réel qui en fait concernent essentiellement les tâches. Parmi les multiples contraintes sur les tâches dans la litérature, nous traiterons principalement les contraintes suivantes :

### 1.4.1 Echéances

Nous avons déjà mentionné que les contraintes d'échéance sont privilégiées lors de la conception et l'analyse d'ordonnançabilité de systèmes temps réel car elles permettent d'exprimer une condition sur la date de fin au plus tard de chaque tâche du système. Suivant la relation entre la période $T_i$ et l'échéance relative $D_i$ de chaque tâche $\tau_i$ ($i = 1, \cdots, n$), nous distinguons trois types de tâches périodiques :

1. $D_i = T_i$ : dans ce cas, chaque activation d'une tâche donnée doit être executée avant l'activation suivante de la même tâche. On parle aussi de tâches à échéances sur activations.
2. $D_i \leq T_i$ : dans ce cas, chaque activation d'une tâche donnée doit être executée au plus tard à une date éventuellement strictement inférieure à la date de l'activation suivante de la même tâche.
3. $D_i \perp T_i$ : dans ce cas, il n'y a pas de corrélation entre la date de fin d'exécution au plus tard de chaque activation d'une tâche donnée et la période de la même tâche. Ainsi, on peut avoir $D_i \leq T_i$ ou $D_i > T_i$.

### 1.4.2 Précédence

Afin de prendre en compte les interactions éventuelles entre les tâches d'un système temps réel, il est nécessaire de définir la notion de précédence entre les tâches. En effet, lorsque les tâches communiquent en s'échangeant des données, des *contraintes de précédence* sont introduites par les interactions entre elles et celles-ci définissent un ordre partiel d'exécution des tâches [MBFSV06, Ram95, Law71, PK89, Law78, Law73, CMT90, SS94, OCSF97, Cuc04]. Dans ce cas, il est primordial de prendre en compte le fait qu'une tâche réceptrice d'une donnée soit en attente tant que la tâche émettrice associée n'a pas produit une donnée. Par conséquent, on dit qu'il existe une contrainte de précédence entre une tâche émettrice $\tau_i$ et une tâche réceptrice $\tau_j$ si et seulement si $\tau_j$ ne peut pas commencer son exécution avant la fin de l'exécution de $\tau_i$. En celà, les contraintes de précédence sont indirectement des contraintes temps réel et on dit alors que la tâche $\tau_i$ est un *prédécesseur* de la tâche $\tau_j$ ou encore que $\tau_j$ est un *successeur* de la tâche $\tau_i$. Lorsque les tâches ne sont définies que par leurs paramètres temporels, on dit qu'elles sont *indépendantes*. La figure 1.5 illustre les interactions éventuelles entre les tâches où les contraintes de précédence sont représentées par des arcs orientés sans cycles. Dans cette figure, la tâche $A$ est par exemple un prédécesseur de la tâche $F$, la tâche $D$ est un prédécesseur de la tâche $C$ mais les tâches $A$ et $G$ sont indépendantes car il n'existe pas d'arcs orientés reliant ces deux tâches.

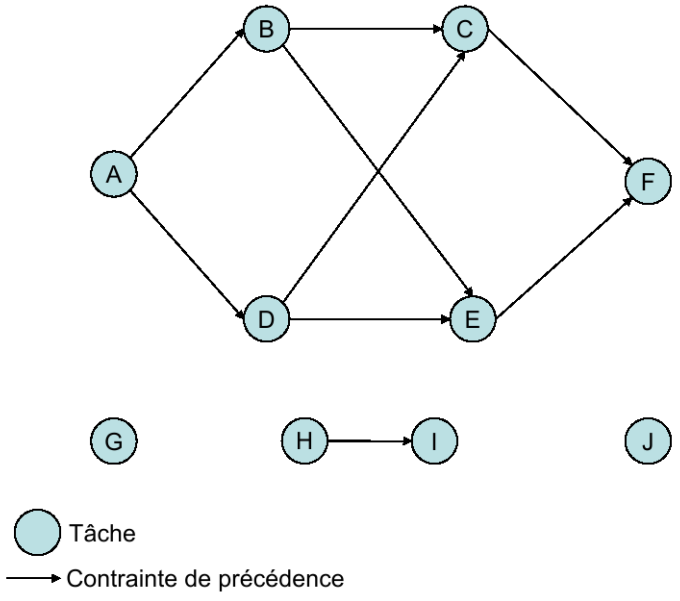


FIG. 1.5 – Illustration des interactions éventuelles entre les tâches.

### 1.4.3 Périodicité stricte

Etant donnée une tâche périodique $\tau_i$ appartenant à un système temps réel, la périodicité stricte impose que le temps qui s'écoule entre deux dates de début d'exécution consécutives corresponde exactement à la période de la tâche considérée. L'intérêt de cette contrainte est que la connaissance de la date de début effective de la première instance $s_i^1$ de la tâche $\tau_i$ implique la connaissance de la date de début effective de toutes les instances suivantes de la même tâche [Cuc04], celà s'exprime par la relation $s_i^{k+1} = s_i^1 + kT_i \quad \forall k \geq 1$. C'est une contrainte d'échéance dans le modèle de tâches sans date de réveil, présenté précédemment, dans le cas où l'échéance est égale à la période pour chaque tâche (voir figure 1.4). En général les tâches d'entrées/sorties qui correspondent aux capteurs et aux actionneurs des systèmes critiques nécessitent d'être à périodicité stricte.

### 1.4.4 Latence

Dans certaines applications temps réel telles que le freinage, il est important de borner le temps total qui s'écoule entre la date de début effective d'une tâche et la date de fin effective d'une autre tâche afin de garantir la pertinence des résultats finaux obtenus [GPM97, DRM97, SW98, CKS02, Cuc04]. Lorsqu'une telle contrainte est imposée entre deux tâches, on parle de *contrainte de latence* entre les deux tâches impliquées. Dans la littérature, l'un des travaux fondateurs concernant l'ordonnançabilité de systèmes temps réel dont certaines tâches sont soumises à cette contrainte a été réalisé par L. Cucu [Cuc04]. Dans sa thèse, elle a proposé une étude détaillée des contraintes de latence pouvant exister entre des tâches ayant différents types de relation d'un système temps réel. Parmi les types de relation entre tâches étudiés, nous retrouvons :

1. la relation $||$ : il n'y a aucun chemin à partir d'une paire de tâches ayant une contrainte de latence vers une autre paire de tâches ayant elles aussi une contrainte de latence.
2. la relation $Z$ : il y a au moins un chemin d'une première paire de tâches ayant une contrainte de latence vers une seconde paire de tâches ayant elles aussi une contrainte de latence mais il n'y a aucun chemin de la seconde paire vers la première paire.
3. la relation $X$ : il y a au moins un chemin d'une première paire de tâches ayant une contrainte de latence vers une seconde paire de tâches ayant elles aussi une contrainte de latence et inversement il y a un chemin de la seconde paire vers la première paire.

Pour chaque type de latence, L. Cucu a proposé une condition d'ordonnançabilité dans le cas où les tâches sont ordonnancées dans un cadre non préemptif. La figure 1.6

illustre les contraintes de latence éventuelles entre les tâches. Dans cette figure, le temps qui s'écoule entre la date de début effective de la tâche $A$ et la date de fin effective de la tâche $F$ ne doit pas excéder $L_{AF}$ unités de temps, le temps qui s'écoule entre la date de début effective de la tâche $A$ et la date de fin effective de la tâche $E$ ne doit pas excéder $L_{AE}$ unités de temps et le temps qui s'écoule entre la date de début effective de la tâche $D$ et la date de fin effective de la tâche $F$ ne doit pas excéder $L_{DF}$ unités de temps.

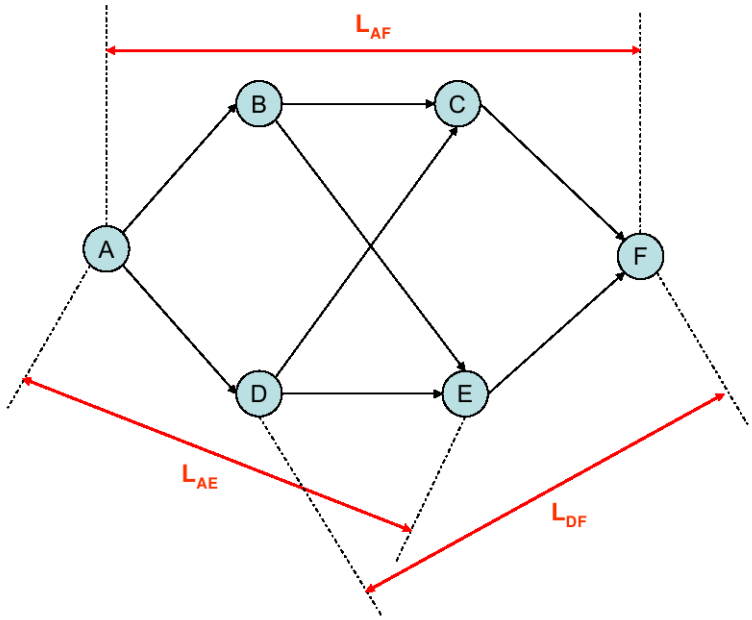


FIG. 1.6 – Illustration des contraintes de latence entre les tâches.

### 1.4.5 Gigue

La *gigue* ou *jitter* $J_i$ d'une tâche $\tau_i$ appartenant à un système temps réel est un facteur permettant de caractériser la variation pendant l'exécution de toutes les instances de la tâche $\tau_i$. Elle se caractérise par une incertitude sur la date d'occurrence de chaque activation de $\tau_i$. Par conséquent, sa prise en compte lors de l'analyse d'ordonnançabilité du système est étroitement liée aux temps de réponse de toutes les instances successives. Il existe plusieurs méthodes de traitement de la gigue dans la littérature [Fer90, VZF91, TBW94, BBGL99, BCM97, BBC$^+$01, RT02]. Parmi ces méthodes, on retrouve par exemple celle qui s'appuie sur la prise en compte de la variation des dates de début d'exécution effectives des instances de chaque tâche, ou encore celle qui s'appuit sur la prise en compte de la variation des durées d'exécution effectives des instances suc-

cessives de chaque tâche, etc. Dans cette thèse, nous nous intéresserons principalement à la prise en compte de la *gigue d'activation*. Lorsque les tâches sont indépendantes, chaque tâche peut démarrer son exécution dès sa date d'activation. Cette assertion n'est plus vraie dès que les tâches sont dépendantes. Dans les modèles de tâches indépendantes, les tâches ne comportent pas de synchronisation ou de communication. Les communications et synchronisations ont lieu au début et en fin d'exécution des tâches. Ainsi, une tâche recevant des données pourra démarrer son exécution après sa date d'activation et lorsque toutes les données seront arrivées. Le décalage entre la date d'activation de l'instance et l'instant où elle peut démarrer son exécution est appelé *gigue d'activation*.

## 1.5 Problématique et motivations de la thèse

Le problème d'ordonnancement temps réel monoprocesseur consiste à exécuter des tâches sur un processeur en prenant en compte toutes les contraintes du système d'une part et l'urgence relative de chaque tâche d'autre part. Cette exécution des tâches est réalisée suivant une politique d'ordonnancement précise appelée *algorithme d'ordonnancement* et est assurée par l'*ordonnanceur* / *scheduler*. Dans la pratique, on classe les algorithmes d'ordonnancement suivant les caractéristiques du système sur lequel ils sont implantés. Ainsi, un algorithme d'ordonnancement peut appartenir aux différentes classes suivantes :

- **Préemptif vs Non préemptif**
  ***Préemptif*** – Une tâche en cours d'exécution peut, à tout moment, être interrompue pour céder le CPU à une autre tâche de priorité supérieure sans avoir à recommencer l'exécution au début. En général, malgré la difficulté d'évaluer le coût lié à la préemption, l'avantage dans cette classe d'algorithmes est l'amélioration du temps de réponse des tâches les plus prioritaires du système ainsi que le nombre plus important de systèmes ordonnançables.
  ***Non préemptif*** – Une fois qu'elle a démarrée son exécution une tâche ne peut pas être interrompue pour céder le CPU à une autre tâche quelque soit la priorité de celle-ci. En général, malgré le nombre moins important de systèmes ordonnançables, l'avantage de cette classe d'algorithmes est la limitation du surcoût lié au système d'exploitation (coût de la préemption et coût de l'ordonnanceur).

- **Non oisif vs Oisif**
  ***Non oisif*** – Dès qu'au moins une tâche est prête à être exécutée, alors l'ordonnanceur en élit forcément une parmi elles. Dans ce cas, on dit aussi que l'ordonnanceur fonctionne sans insertion de temps creux (non-idling ou work-conservative). En général, bien qu'il conduise dans le cas non-préemptif à un nombre plus im-

portant de systèmes non ordonnançables, l'avantage de cette classe d'algorithmes est qu'on peut atteindre des charges processeur plus élevées. En revanche dans le cas préemptif le nombre de systèmes ordonnançables est plus important.
***Oisif*** – L'ordonnanceur peut ne sélectionner aucune tâche pendant un certain temps, alors qu'il en existe au moins une prête à être exécutée. Dans ce cas, on dit aussi que l'ordonnanceur fonctionne par insertion de temps creux (with inserted idle times). En général, bien qu'on soit limité en terme de charge processeur à cause des temps creux, l'avantage de cette classe d'algorithmes est plus important en non-préemptif. En effet certains systèmes de tâches non ordonnançables en non-préemptif non oisif le sont en non-préemptif oisif.

– **Hors-ligne vs En-ligne**
***Hors-ligne*** – L'ordonnancement des tâches est planifié avant l'exécution du système. La séquence obtenue est stockée dans une table et est reproduite telle quelle par un *ordonnanceur simplifié* ou *séquenceur* lors de l'exécution du système. En général, malgré la nécessité d'une connaissance détaillée de l'algorithme d'ordonnancement, de l'architecture et des caractéristiques temporelles des tâches d'une part et le fait que cette classe d'algorithmes ne peut pas prendre en compte des tâches non prévues lors de la spécification d'autre part, l'avantage dans cette classe est la simplicité de sa mise en oeuvre et son faible surcoût dû à la simplicité du séquenceur.
***En-ligne*** – L'ordonnancement des tâches se fait pendant l'exécution du système. A chaque instant (éventuellement uniquement à chaque activation ou fin de tâche), l'algorithme décide de l'exécution d'une tâche sur le processeur. En général, malgré les importants surcoûts liés à l'exécution du système, l'avantage dans cette classe d'algorithmes est la possibilité de prendre en compte des événements susceptibles de se produire de manière apériodique. Les approches en-ligne permettent de prendre en compte des tâches non prévues lors de la spécification.

– **Priorité Statique vs Dynamique**
***Statique*** – Les priorités des tâches sont calculées une seule fois avant l'exécution du système et sont basées sur des paramètres qui ne changent pas pendant l'exécution du système, où la tâche active de plus haute priorité est sélectionnée et exécutée. En général, bien qu'on ne puisse pas changer la priorité pendant l'exécution, l'avantage dans cette classe d'algorithmes est la possibilité d'identifier, avant l'exécution, les tâches qui dépasseront leurs échéances.
***Dynamique*** – Les priorités des tâches sont calculées plusieurs fois pendant l'exécution du système et sont basées sur des paramètres qui peuvent changer pendant l'exécution du système, où la tâche active de plus haute priorité est sélectionnée et exécutée. En général, malgré le surcoût élevé lié au calcul permanent des priori-

tés des tâches, l'avantage dans cette classe d'algorithmes est lié à la possibilité de modifier les priorités pendant l'exécution afin de respecter les échéances même si les tâches ne sont pas forcément périodiques.

Bien qu'il existe de bonnes raisons, notamment concernant *la prédictibilité, la contrôlabilité, l'insensibilité aux effets domino, etc.* [SKB91], d'utiliser des algorithmes non oisifs préemptifs à priorités fixes, comme cela sera le cas dans cette thèse, dans la plupart des systèmes temps réel, la préemption crée également un certain nombre de problèmes pour les développeurs et en particulier à ceux de logiciels embarqués. Parfois, la complexité de sa maîtrise se traduit par le gaspillage des ressources, des défaillances du système et presque toujours, elle allonge les cycles de développement et de débogage.

En général, les systèmes temps réel critiques que nous considérons sont avant tout des systèmes réactifs. Un tel système reçoit l'état du procédé à travers des capteurs, effectue des traitements qui dépendent de cet état, et produit en réaction des commandes pour le procédé à travers des actionneurs. Il est soumis à des contraintes qu'il faut absolument respecter pour son bon fonctionnement. La préemption améliore la plupart du temps le temps de réponse de chaque tâche. Cependant, il y a plusieurs incidences négatives lorsque la préemption n'est pas ou est mal maîtrisée. Parmi les incidences négatives, deux sont lourdes de conséquences. La première est le gaspillage des ressources dû à la nécessité d'effectuer un *changement de contexte* / *context switch* chaque fois que se produit une préemption. La seconde, qui elle est plus dangereuse, est une condition d'ordonnançabilité éronée. En effet, une mauvaise estimation du coût temporel dû aux préemptions peut nous conduire à conclure qu'un système est ordonnançable alors qu'elle n'a aucune chance de l'être. En substance, puisque le temps de réponse d'une tâche préemptée dépend directement du nombre et de la durée des interruptions qu'elle subit, et de tous les coûts liés à la sauvegarde et la restauration de l'état du processeur, moins de cycles processeur sont disponibles pour l'exécution effective de la tâche à chacune de ses activations. Concrètement, lorsqu'une tâche de plus grande priorité préempte une autre de plus faible priorité, les étapes suivies par l'ordonnanceur sont similaires à celles que le processeur effectue à la réception d'une interruption. Tout d'abord, le contexte de la tâche préemptée doit être sauvé en mémoire, puis la nouvelle tâche s'exécute. Si la nouvelle tâche a démarré auparavant, alors son contexte a aussi été sauvé, et donc celui-ci doit être restauré avant de pouvoir continuer son exécution. Lorsque la tâche la plus prioritaire est terminé, l'ordonnanceur sauve son dernier contexte et libère le processeur afin qu'il puisse exécuter la tâche qui avait été préemptée. La tâche la moins prioritaire reprend alors son exécution comme si rien ne s'était passé (autre que le décalage temporel). Le challence que nous allons résoudre dans cette thèse est de prendre en compte tous ces aspects dans les conditions d'ordonnançabilité que nous fournirons en considérant le coût correspondant au nombre exact de préemptions survenues. En effet le coût de l'OS est constitué de deux parties : une partie fixe, facile à déterminer, liée à l'ordonnanceur

et une partie variable, difficile à déterminer, liée à l'occurrence de chaque préemption lors de l'exécution des tâches.

## 1.6 Conclusion

Dans ce chapitre, nous avons commencé par présenter les concepts généraux liés au fonctionnement des systèmes temps réel. Ensuite, nous avons présenté les modèles de tâches à partir desquels nous démarrerons l'analyse d'ordonnançabilité de tels systèmes dans le cas monoprocesseur. Nous avons présenté les contraintes – échéance, précédence, périodicité stricte, latence, gigue – que nous prendrons en compte lors de notre analyse. Enfin, dans la problématique et les motivations de cette thèse, nous avons souligné d'une part, l'intérêt de choisir des algorithmes préemptifs à priorités fixes pour la résolution des problèmes d'ordonnancement dans les systèmes temps réel durs et d'autre part, la difficulté liée à la prise en compte du coût de l'OS dans les conditions d'ordonnançabilité existantes dans la littérature.

# Chapitre 2

# Théorie classique de l'ordonnancement temps réel

*On ne connaît pas complètement une science*
*tant qu'on n'en sait pas l'histoire.*
**Auguste Comte**

## 2.1 Introduction

Le chapitre 1 a souligné, dans le cas monoprocesseur, l'intérêt de choisir des algorithmes préemptifs à priorités fixes pour la résolution des problèmes d'ordonnancement dans les systèmes temps réel durs [ABR+93, ABW93, BB02, HT97, Han98, DG00, CMK99, JP86b, LMM98, Leh90, LSD89, MA98, PNKK95]. Il a suscintement présenté les contraintes temps réel que nous considérerons par la suite et a également présenté la problématique et les motivations de cette thèse. Le but de ce chapitre, quant à lui, est de présenter les techniques ou approches classiques utilisées dans la littérature pour l'analyse et l'ordonnancement des systèmes temps réel durs, puis les avantages et inconvénients de chacune d'elles. L'idée principale est de donner un aperçu de l'état de l'art afin de justifier le choix de notre approche d'analyse et les hypothèses que nous choisirons par la suite.

## 2.2 Modèle des tâches

Ici, nous considérerons principalement les systèmes constitués de $n$ tâches périodiques $\Gamma_n = \{\tau_1, \tau_2, \cdots, \tau_n\}$ basés sur le modèle de Liu & Layland (c.f. chapitre 1 : section 1.3.1). Dans ce modèle, une tâche $\tau_i = (r_i^1, C_i, D_i, T_i)$ est caractérisée par une date de première activation $r_i^1$, une période $T_i$, une échéance relative $D_i$ (durée maximale pour l'exécution de $\tau_i$ depuis sa dernière activation), et une durée d'exécution $C_i$ supposée constante à chaque activation de la tâche $\tau_i$. Dans la majorité des analyses d'ordonnançabilité de tels systèmes proposées dans la littérature, quatre hypothèses fon-

damentales sont habituellement faites concernant les tâches [LL73] :

- **Hypothèse 1** *: toutes les tâches sont périodiques,*
- **Hypothèse 2** *: toutes les tâches ont une échéance relative correspondante à la durée qui s'écoule entre deux activations successives, c'est-à-dire l'échéance relative de chaque tâche est égale à la période de la tâche,*
- **Hypothèse 3** *: toutes les tâches sont indépendantes, c'est-à-dire il n'y a aucune contrainte de ressource (autre que le CPU), ni de précédence entre les tâches,*
- **Hypothèse 4** *: toutes les tâches ont une durée d'exécution constante, égale à la "pire durée d'exécution". Dans cette quantité, on approxime le temps nécessaire pour démarrer et terminer chaque requête d'exécution et le coût de toutes les préemptions de la tâche.*

## 2.3 Analyse d'ordonnançabilité de tâches périodiques

A partir de maintenant, $\Gamma_n = \{\tau_1, \tau_2, \cdots, \tau_n\}$ désigne un système de $n$ tâches périodiques tels que chaque tâche $\tau_i = (r_i^1, C_i, D_i, T_i),\ i = 1, 2, \cdots, n$ [1].

**Définition 1** *Soient $\tau_i$ et $\tau_j$ deux tâches appartenant au système $\Gamma_n$, le* déphasage *entre les dates de premières activations de $\tau_i$ et $\tau_j$ est définit par $\gamma_{ij} = r_j^1 - r_i^1$.*

**Définition 2** *Le système $\Gamma_n$ sera dit* ordonnançable *si toutes les échéances et toutes les contraintes sont satisfaites pour toutes les activations des tâches.*

### 2.3.1 Intervalle d'analyse

Dans la théorie classique de l'ordonnancement, les algorithmes préemptifs à priorités fixes présentés sont généralement basés sur la considération du *pire cas*, c'est-à-dire ils évaluent l'ordonnançabilité d'un système de tâches périodiques à un *instant critique*, un instant critique étant un instant où le temps de réponse de l'ordonnanceur à l'activation de toutes les tâches est le plus long. Cette approche d'analyse d'ordonnançabilité se justifie par le fait qu'on ne connaît pas toujours les dates de premières activations de toutes les tâches. Dans ce cas, on parle de *tâches non-concrètes*. Lorsqu'on connaît les dates de premières activations de toutes les tâches, on parle plûtot de *tâches concrètes*. Les premiers travaux significatifs sur les algorithmes préemptifs à priorités fixes ont été publiés par Liu & Layland [LL73]. Dans leur papier les auteurs proposent la définition suivante :

[1] Les indices feront toujours référence à la tâche considérée tandis que les exposants feront toujours référence à une instance de la tâche considérée.

**Définition 3 (LL73)** *Un* instant critique *est un instant où la date d'activation de chaque tâche arrive simultanément avec la date d'activation de toutes les tâches de priorité supérieure.*

Par conséquent, le "pire cas" est obtenu dès que les déphasages entre les dates de premières activations des tâches sont tous nuls, c'est-à-dire $\forall i, j \in \{1, 2, \cdots, n\},\ r_i^1 = r_j^1$ : on parle aussi de *démarrage simultané* de toutes les tâches. Dans ce cas, il suffit de vérifier que la première demande d'exécution de chaque tâche est exécutée avant son échéance.

### 2.3.2 Tâches non concrètes

Les tâches d'un système $\Gamma_n$ sont dites non-concrètes lorsqu'on ne connaît pas toujours les dates de premières activations de toutes les tâches. Dans ce cas, pour conclure sur l'ordonnançabilité du système, on le considère à "*l'instant critique*" et on vérifie que la première demande d'exécution de chaque tâche est exécutée avant son échéance. Suivant les caractéristiques des tâches, on distingue les différents cas suivants.

#### Tâches à échéances sur activations

Dans ce cadre, *Liu et Layland* ont présenté un algorithme préemptif à priorité fixe dans lequel à tout moment la tâche qui s'exécute est celle de plus haute priorité parmi les tâches dans l'état *prêt*. Dans cet algorithme, les priorités sont attribuées aux tâches de telle sorte que plus la période d'une tâche est petite, plus sa priorité est forte. Lorsque deux tâches ont la même période, alors la plus prioritaire des deux est choisie de façon arbitraire. Cette politique d'attribution des priorités aux tâches est connue sous le nom *Rate Monotonic (RM)*. *Liu et Layland* ont montré que *RM* est *optimal* pour les systèmes de tâches périodiques à démarrage simultanée et échéances sur activations. Ici, *optimal* signifie que si pour une autre politique de choix des priorités des tâches le système est ordonnançable, alors il le sera aussi par *RM*.

S'appuyant sur les hypothèses faites dans leur papier, les auteurs proposent la définition suivante du *facteur d'utilisation du processeur*.

**Définition 4** *Etant donné le système $\Gamma_n$, le facteur d'utilisation $U_n$ du processeur est donné par*

$$U_n = \sum_{i=1}^{n} \frac{C_i}{T_i} \tag{2.1}$$

Par la suite, nous ferons référence à cette quantité comme étant le facteur d'utilisation *classique* du processeur puisque nous proposerons d'autres définitions. De la définition

ci-dessus, Liu et Layland [LL73], ainsi que Servin [Ser72] proposent une condition d'ordonnançabilité assez simple pour un système constitué de $n$ tâches. Cette condition, qui n'est cependant que suffisante [LL73], est donnée par l'inégalité 2.2.

$$U_n \leq n \cdot \left(2^{1/n} - 1\right) \tag{2.2}$$

Elle exprime le fait que si le facteur d'utilisation du processeur est en dessous d'une certaine valeur (dépendant seulement du nombre de tâches dans le système), alors le système est ordonnançable. Cependant, notons que

$$\lim_{n \to +\infty} n \cdot \left(2^{1/n} - 1\right) = ln2 \simeq 69.31\%$$

Puisque la condition 2.2 n'est que suffisante, on ne peut pas écarter des systèmes de tâches qui ne la satisfont pas. Par conséquent, elle est pessimiste. Pour illustrer le pessimisme de cette condition, considérons l'exemple suivant de l'ordonnancement d'un système constitué de deux tâches $\Gamma_2 = \{\tau_1, \tau_2\}$ à échéances sur activations (c'est-à-dire $D_i = T_i,\ 1 \leq i \leq 2$) par l'algorithme ***RM***, on suppose que la date de première activation de toutes les tâches est nulle, c'est-à-dire $r_i^1 = 0,\ 1 \leq i \leq 2$. Les caractéristiques des tâches sont résumées dans le tableau 2.1.

| Tâche | $r_i^1$ | $C_i$ | $D_i$ | $T_i$ |
|---|---|---|---|---|
| $\tau_1$ | 0 | 4 | 6 | 6 |
| $\tau_2$ | 0 | 2 | 9 | 9 |

TAB. 2.1 – Caractéristiques des tâches

Pour ce système, le facteur d'utilisation du processeur vaut $U_2 = 4/6 + 2/9 \simeq 88.89\%$ tandis que $2(2^{1/2} - 1) \simeq 82.84\%$. Clairement, on ne peut pas conclure sur l'ordonnançabilité de ce système en utilisant la condition 2.2. Cependant, ce système de tâches est ordonnançable par ***RM*** comme le montre la figure 2.1.

Pour contourner la difficulté illustrée par la figure 2.1, il existe dans la littérature des approches consistant à transformer un système de tâches donné en un autre système de tâches pour lequel il est plus facile de conclure sur l'ordonnançabilité. Une des approches souvent utilisée est basée sur la transformation du système de tâches en système de tâches *harmoniques* [2]. En effet un système de tâches harmoniques $\Gamma_n$ est ordonnançable si et seulement si $U_n \leq 1$ [LL73]. Par conséquent, si un système de tâches donné peut être transformé en un système de tâches harmoniques ordonnançable, alors le système initial est lui-même ordonnançable [HT97, Han98]. Bien que cette approche soit intéressante,

[2] Le système $\Gamma_n = \{\tau_i = (r_i^1, C_i, D_i, T_i) \mid i = 1, \cdots, n\}$ est dit harmonique si et seulement si on a : $T_{i+1} \equiv 0 \pmod{T_i}, \forall\, i \in \{1, \cdots, n-1\}$.

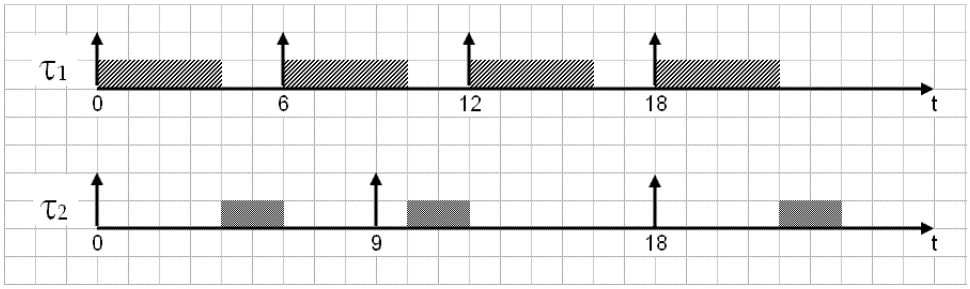


FIG. 2.1 – Illustration de l'algorithme RM.

son efficacité est limitée au moins sous deux aspects. Premièrement, une transformation n'est pas toujours possible. Deuxièmement, la transformation d'un système peut changer la conclusion de l'ordonnançabilité du système initial : le système transformé peut ne pas être ordonnançable alors que le système original l'est suivant *RM*.

Puisqu'à chaque tâche d'un système de tâches donné $\Gamma_n$ est attribuée une priorité fixe, alors la conclusion de l'ordonnançabilité de $\Gamma_n$ peut être obtenue par étapes successives de la tâche la plus prioritaire à la tâche la moins prioritaire. En effet à chaque instant, seuls les tâches plus prioritaires peuvent rallonger le temps d'exécution d'une tâche. Concrètement, si on suppose sans nuire à la généralité que les tâches sont rangées suivant les priorités décroissantes, c'est-à-dire $\tau_1$ est plus prioritaire que $\tau_2$, $\tau_2$ est plus prioritaire que $\tau_3$, ainsi de suite et $\tau_{n-1}$ est plus prioritaire que $\tau_n$, alors l'ordonnançabilité de la tâche $\tau_i,\ i = 1, \cdots, n$, ne dépendra que des tâches $\tau_1$ jusqu'à $\tau_{i-1}$ [LL73, LM80, LW82].

**Définition 5** *Soit $r_i^k$ (resp. $f_i^k$) la date d'activation (resp. la date de fin d'exécution) de la $k^{ieme}$ instance de la tâche $\tau_i$ , alors le* temps de réponse *de cette instance est donné par*

$$R_i^k = f_i^k - r_i^k$$

**Définition 6** *Le* pire temps de réponse $R_i$ *de la tâche $\tau_i$ correspond au maximum des temps de réponse de toutes ses instances.*

$$R_i = \max_{k \geq 1}(R_i^k)$$

Joseph et Pandya [Jos85, JP86a], Lehoczky et al [LSD89], ainsi que Audsley et al [AARW91] ont présenté une condition d'ordonnançabilité basée sur le pire temps de réponse de chaque tâche permettant de conclure sur l'ordonnançabilité de systèmes dont le facteur d'utilisation du processeur peut atteindre $90\%$ et pour lesquels on ne peut rien conclure en utilisant la condition 2.2. Concrètement, leur approche consiste à dire que

pour un système de $n$ tâches périodiques $\Gamma_n = \{\tau_1, \tau_2, \cdots, \tau_n\}$ ordonnancé par ***RM*** ; le temps de réponse de la tâche $\tau_i$ est borné supérieurement par

$$R_i = C_i + \sum_{j \in hp(\tau_i)} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j \tag{2.3}$$

où $hp(\tau_i)$ désigne l'ensemble des tâches de $\Gamma_n$ de priorité supérieure ou égale à celle de $\tau_i$. Pour la tâche $\tau_i$, déterminer la valeur de $R_i$ revient à calculer le plus petit point fixe dans l'expression 2.3. La résolution de cette équation s'obtient en utilisant la suite ci-dessous :

$$\begin{cases} R^{(0)} = C_i \\ \\ R_i^{(k+1)} = C_i + \sum_{j \in hp(\tau_i)} \left\lceil \frac{R_i^{(k)}}{T_j} \right\rceil \cdot C_j \end{cases}$$

La valeur de $R_i$ est définie par le plus petit entier $k$ tel que soit $R_i = R_i^{(k+1)} = R_i^{(k)}$, soit $R_i^{(k+1)} > T_i$. Dans le second cas, la tâche $\tau_i$ est dite non ordonnançable. Ainsi, $\Gamma_n$ est ordonnançable si et seulement si :

$$\forall i \in \{1, 2, \cdots, n\}, \quad R_i \leq T_i \tag{2.4}$$

### Tâches à échéances avant activations

Un des inconvénients de l'algorithme ***RM*** est de considérer les tâches à échéances sur activations, c'est-à-dire $D_i = T_i,\ 1 \leq i \leq n$. cette considération implique que plus la période d'une tâche est petite, plus sa priorité est haute. Par conséquent, il n'existe pas de mécanisme pour faire face à la criticité des tâches et cela peut causer des problèmes lorsque certaines tâches avec de longues périodes ont besoin d'une priorité haute. Il s'avère alors intéressant de considérer aussi le cas des tâches avec des échéances inférieures à la période : $D_i < T_i$. Alan Burns et al [BWBF93] ont fourni un exemple réel d'application pour un système de télémétrie par satellite, où le logiciel de transfert de données vers ou depuis la station au sol n'est nécessaire qu'une fois par jour. Dans leur application, l'exécution du logiciel dans une fenêtre temporelle étroite est essentielle pour garantir l'efficacité et la pertinence des résultats du système. Avec l'algorithme ***RM***, la tâche de télémétrie aurait la priorité la plus basse possible et, par conséquent, pourrait manquer son échéance. Résoudre ce problème en utilisant ***RM*** exige que cette tâche ait une période beaucoup plus courte pour ainsi avoir une plus grande priorité, afin de satisfaire son échéance. Clairement, cette approche peut causer un échec d'échéance d'autres tâches du système. Strosnider, Katcher et Arakawa [KAS93] ont fait deux remarques importantes dans leur article concernant ***RM***. Premièrement, ils ont montré que

la condition d'ordonnançabilité d'un système basée sur le facteur d'utilisation du processeur est pessimiste. Deuxièmement, il ont montré que la politique de choix des priorités des tâches suivant *RM* est sous-optimal – c'est-à-dire le système de tâches considéré peut être ordonnançable suivant une politique de choix des priorités différente de celle basée sur *RM* et pas suivant *RM* – lorsque les échéances des tâches sont différentes de leurs périodes ou lorsque les déphasages entre les dates de premières activations des tâches ne sont pas tous nuls.

la solution au problème de criticité est fournie par l'algorithme *Deadline Monotonic (DM)*. C'est un algorithme où les priorités sont attribuées aux tâches par rapport à leurs échéances relatives. La plus haute priorité est attribuée à la tâche ayant l'échéance relative la plus courte. Lorsque deux tâches ont la même échéance relative, alors la plus prioritaire est choisie de façon arbitraire. Les premiers travaux originaux concernant l'algorithme *DM* ont été publiés par *Leung et Withehead* [LW82]. Cet algorithme est équivalent à l'algorithme *RM* dans le cas où les tâches sont à échéances sur activations. *Leung et Withehead* montrent que *DM* est optimal pour des systèmes de tâches préemptifs où, pour toutes les tâches, les déphasages entre les dates de premières activations sont tous nuls et les échéances sont inférieures ou égales aux périodes. En utilisant l'algorithme *DM*, un système de $n$ tâches périodiques $\Gamma_n = \{\tau_1, \tau_2, \cdots, \tau_n\}$ est ordonnançable si

$$\forall i,\ 1 \leq i \leq n : \quad C_i + \sum_{j \in hp(\tau_i)} \left\lceil \frac{D_i}{T_j} \right\rceil \cdot C_j \leq D_i \tag{2.5}$$

Dans cette condition d'ordonnançabilité, $\left( \sum_{j \in hp(\tau_i)} \left\lceil \frac{D_i}{T_j} \right\rceil \cdot C_j \right)$ est une mesure de l'interférence des tâches de plus haute priorité dans l'exécution de $\tau_i$. Cette condition énonce que pour une tâche $\tau_i$ , la somme de sa durée d'exécution, plus l'interférence dans son exécution qui lui est imposée par les tâches de priorités supérieures, ne doit jamais dépasser son échéance.

Pour illustrer de cette condition, considérons l'exemple suivant de l'ordonnancement d'un système de deux tâches $\Gamma_2 = \{\tau_1, \tau_2\}$ par l'algorithme *DM*, on suppose que la date de première activation de toutes les tâches est nulle, c'est-à-dire $r_i^1 = 0,\ 1 \leq i \leq 2$. Les caractéristiques des tâches sont résumées dans le tableau 2.2.

| Tâche | $r_i^1$ | $C_i$ | $D_i$ | $T_i$ |
|---|---|---|---|---|
| $\tau_1$ | 0 | 4 | 6 | 6 |
| $\tau_2$ | 0 | 2 | 4 | 9 |

TAB. 2.2 – Caractéristiques des tâches

Contrairement à ce qu'on aurait avec *RM*, la tâche $\tau_2$ est la plus prioritaire en suivant l'algorithme *DM*. Puisque $C_2 = 2 \leq 4$, alors $\tau_2$ est ordonnançable. De même la tâche $\tau_1$ est ordonnançable puisque $C_1 + \left\lceil \frac{D_1}{T_2} \right\rceil \cdot C_2 = 4 + \left\lceil \frac{6}{9} \right\rceil \cdot 2 = 6 \leq 6$. La figure 2.2 illustre l'ordonnancement obtenu.

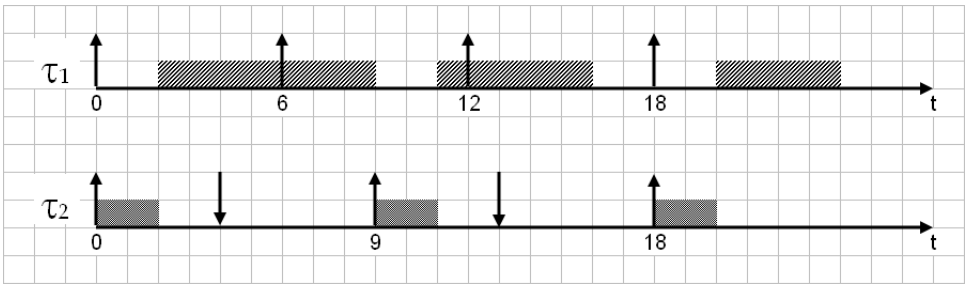


FIG. 2.2 – Illustration de l'algorithme *DM*.

La condition 2.5 est suffisante mais pas nécessaire, puisque la mesure de l'interférence effectuée est pessimiste. En effet, elle ne prend pas en compte les dates de premières activations des différentes tâches. Finalement, en utilisant l'algorithme *DM* le problème d'ordonnançabilité de la tâche de télémétrie du satellite Olympus est résolu par l'attribution d'une échéance appropriée. Cela donne à la tâche une priorité suffisamment élevée pour satisfaire son échéance sans avoir à gaspiller des ressources par le biais de variations multiples et inutiles de la période de la tâche. Néanmoins, il reste deux restrictions dans l'algorithme *DM* qui l'empêchent d'être applicable dans certains systèmes. Le plus important est le manque de prise en compte des déphasages entre les dates de premières activations des tâches et, dans une moindre mesure le fait que les échéances doivent être inférieures ou égales à la période. Pour résoudre ce problème, plusieurs travaux ont mis l'accent sur les *périodes d'activités du processeur* car il est inutile de rechercher une faute temporelle lorsque celui-ci est inutilisé.

**Définition 7** *Une* période d'activité du processeur *est un intervalle de temps dans lequel le processeur est pleinement utilisé.*

La plus longue période d'activité du processeur survient lorsque toutes les tâches sont activées simultanément, c'est-à-dire elle est initiée par l'instant critique [LL73]. Pour déterminer la longueur de cette période d'activité il faut calculer la durée cumulée des exécutions des tâches sur l'intervalle de temps $[0, t[$. La fonction de travail $W(t)$ définit la durée cumulée des tâches activées dans l'intervalle $[0, t[$ :

$$W(t) = \sum_{j=1}^{n} \left\lceil \frac{t}{T_j} \right\rceil C_j \tag{2.6}$$

La fonction $W(t)$ est une fonction en escalier et la droite affine $f(t) = t$ définit la capacité maximum de traitement du processeur. Lorque $f(t) = W(t)$, alors le processeur a terminé toutes les instances activées avant $t$. Cette égalité correspond à un point fixe de l'équation $W(t) = t$. Ainsi la plus longue période d'activité $L$ est obtenue en résolvant l'équation $W(L) = L$ où on procède de même façon que pour la résolution de l'équation 2.3 précédente. La résolution de cette équation s'obtient en calculant la suite suivante :

$$\begin{cases} t^{(0)} = \sum_{i=1}^{n} C_i \\ \\ t^{(k+1)} = W(t^{(k)}) \end{cases}$$

La longueur $L$ est alors définie par le plus petit entier $k$ tel que $L = t^{(k+1)} = t^{(k)}$. Si le facteur d'utilisation du processeur est inférieur ou égal à 1 : $U_n \leq 1$, alors la convergence de la suite est assurée puisque pour tout $t, 0 \leq t < L$, on a $W(t) > t$. Par contre la suite diverge dès lors que $U_n > 1$. Ceci correspond au cas où le processeur ne peut pas traiter entièrement la charge de travail.

**Remarque 1** *La première période d'activité du processeur, correspondant au plus petit point fixe de la suite précédente, ne précède pas forcément un temps creux (instant d'inactivité du processeur / idle time). La valeur $L$ indique juste que toutes les tâches activées dans l'intervalle $[0, L[$ sont terminées. De nouvelles instances peuvent être activées à la date $L$.*

### 2.3.3 Tâches concrètes

Dans la pratique, il est souvent nécessaire d'introduire un déphase non nul entre les dates de premières activations des tâches afin de garantir leurs exécutions ou respecter certaines contraintes. Quelques raisons pour lesquelles il est nécessaire d'utiliser des déphasages dans un système sont :

1. assurer qu'une tâche a produit une action avant la date de première activation d'une autre tâche,
2. assurer l'exécution des tâches à des dates précises, de sorte que la gigue soit réduite,
3. assurer que les contraintes de précédence sont satisfaites en garantissant que certaines tâches ont terminé leurs exécutions avant le début de l'exécution d'autres tâches.

Les tâches d'un système $\Gamma_n$ sont dites concrètes lorsqu'on connaît les dates de premières activations de toutes les tâches. Dans ce cas, pour conclure sur l'ordonnançabilité

du système, on se base sur un certain intervalle où on vérifie que toutes les requêtes d'exécution de toutes les tâches sont exécutées avant leurs échéances. Celà correspond à vérifier que le pire temps de réponse de chaque tâche est inférieur ou égal à son échéance relative dans cet intervalle. En effet, bien que l'ordonnancement lui même soit infini, la périodicité des tâches permet de limiter la recherche des fautes temporelles (non respect d'échéances ou de contraintes) dans un intervalle de temps borné, appelé *intervalle de faisabilité* (*feasibility interval*). Plusieurs travaux ont été menés par rapport à l'analyse des systèmes de tâches asynchrones, c'est-à-dire ayant des déphasages non tous nuls entre les dates de premières activations [LM80, LW82, Tin93, Aud93, Goo98, Gro99]. Ces travaux sont motivés par le fait que l'algorithme *DM* n'est plus optimal dans ce cas. La preuve de cette assertion est directe : elle utilise le système de tâches introduit par *Leung et Withehead* dont les caractéristiques sont résumées dans le tableau 2.3. Pour

| Tâche | $r_i^1$ | $C_i$ | $D_i$ | $T_i$ |
|---|---|---|---|---|
| $\tau_1$ | 10 | 1 | 12 | 12 |
| $\tau_2$ | 0 | 6 | 12 | 12 |
| $\tau_3$ | 0 | 3 | 8 | 8 |

TAB. 2.3 – Carectéristiques des tâches

ce système, l'attribution des priorités aux tâches suivant l'algorithme *DM* où la priorité la plus élevée et la priorité la moins élévée sont respectivement attribuées aux tâches $\tau_3$ et $\tau_2$ rend le système non ordonnançable (voir figure 2.3). Cependant, ce système est ordonnançable lorsque la priorité la plus élevée et la priorité la moins élévée sont respectivement attribuées aux tâches $\tau_3$ et $\tau_1$ (voir figure 2.4).

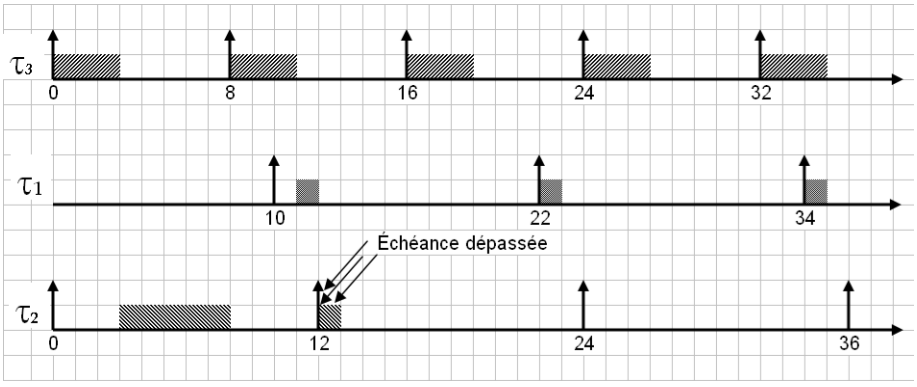


FIG. 2.3 – La priorité la plus élevée et la priorité la moins élévée sont respectivement attribuées aux tâches $\tau_3$ et $\tau_2$

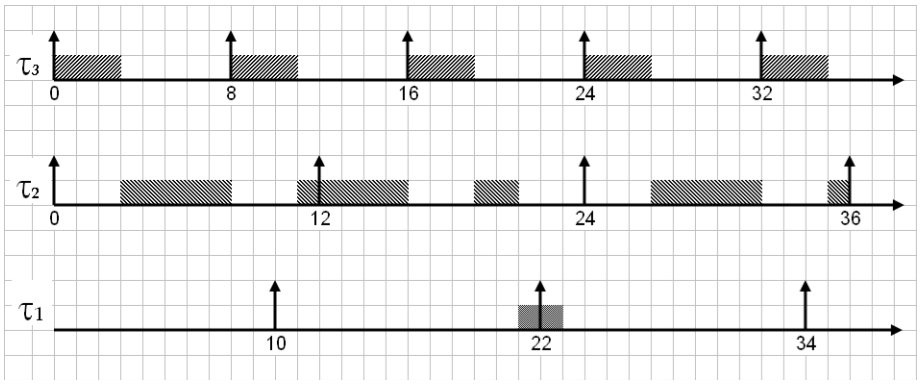


FIG. 2.4 – La priorité la plus élevée et la priorité la moins élévée sont respectivement attribuées aux tâches $\tau_3$ et $\tau_1$.

En fait les deux choix d'attribution des priorités dans l'exemple correspondent à la fois à *DM* et *RM*. La non optimalité de l'algorithme est dûe au choix arbitraire pour attribuer les priorités des tâches $\tau_1$ et $\tau_2$. Cette ambiguïté pourrait être complètement levée en considérant un autre système dont les tâches ont des caractéristiques différentes deux à deux : plus de détails sont donnés dans la thèse de Joël Goossens, chapitre $5$, section $5.4.3$. Grâce aux remarques précédentes, nous considérerons par la suite des systèmes pour lesquelles des priorités fixes sont attribuées aux tâches mais pas forcément suivant les algorithmes *DM* et *RM* pour faire l'étude d'ordonnançabilité d'un système donné. Notons que de toutes façons les algorithmes *DM* et *RM* imposent des contraintes sur la politique d'attribution des priorités aux tâches (suivant *RM*, les priorités attribuées doivent forcément être inversement proportionnelles aux périodes des tâches et suivant *DM*, elles doivent forcément être inversement proportionnelles aux échéances relatives des tâches).

**Définition 8** *Soit $\Gamma_n$ un système de $n$ tâches périodiques, la séquence d'ordonnancement des tâches de $\Gamma_n$ produite par un algorithme donné est* valide *si toutes les contraintes des différentes tâches sont respectées.*

Puisque nous sommes libres d'attribuer les priorités aux tâches selon nos besoins, pour un système de tâches donné, il existe une méthode évidente permettant de trouver la bonne attribution permettant de garantir l'ordonnançabilité du système : tester tous les attributions de priorités possibles et choisir une attribution parmi toutes celles qui produisent un ordonnancement valide (s'il en existe au moins une) mais cet algorithme "naïf" d'attribution requiert un nombre *exponentiel* de tests en fonction du nombre de tâches. Un algorithme d'attribution qui requiert un nombre *polynomial* de tests a été proposé par *Audsley* [Aud91, ATB93, ABR+93] pour un système de tâches indépendantes,

périodiques, concrètes non simultanées et à échéances arbitraires. Cet algorithme d'attribution de priorités est basé sur l'hypothèse selon laquelle il existe une condition d'ordonnançabilité pour une attribution fixée et qu'elle satisfait les hypothèses 1 et 2 ci-dessous.

**Hypothèses nécessaires à l'application de l'algorithme d'Audsley**

1. Pour une attribution de priorités fixée, il est possible de vérifier si une tâche $\tau$ satisfait ou pas toutes ses contraintes, et le résultat de l'analyse d'ordonnançabilité pour la tâche $\tau$ (ordonnançable ou pas) ne change pas si deux tâches qui ont une priorité plus élevée que celle de $\tau$ échangent leurs priorités. En d'autres termes les résultats de l'analyse d'ordonnançabilité d'une tâche $\tau$ ne devraient dépendre que de l'ensemble des tâches ayant une priorité plus élevée que $\tau$, et non de leurs priorités exactes.
2. Si une tâche $\tau$ n'est pas ordonnançable pour une certaine attribution de priorités dans laquelle la tâche $\tau^{'}$ est moins prioritaire que $\tau$, alors la tâche $\tau$ n'est pas aussi ordonnançable pour l'attribution dans laquelle $\tau$ et $\tau^{'}$ échangent leurs priorités. En d'autres termes, si la tâche $\tau$ n'est pas ordonnançable, nous ne pouvons pas la rendre ordonnançable en abaissant sa priorité.

L'algorithme d'*Audsley* divise un système de tâches donné en deux groupes : d'une part les tâches auxquelles ont déjà été attribuées une priorité, et d'autre part les tâches qui n'ont pas de priorités. De même, les priorités sont divisés en deux groupes : d'une part celles qui sont déjà attribuées et d'autre part celles qui sont encore disponibles. L'algorithme attribue toujours la priorité la plus basse disponible à toute tâche qui satisfait ses contraintes lorsque cette priorité lui est attribuée. L'algorithme peut terminer de deux façons. Soit il attribue des priorités à toutes les tâches, dans ce cas, une attribution valide a été trouvée, ou à un certain point, la priorité la plus basse disponible (par exemple $p_{min}$) ne peut pas être attribuée à l'une des tâches restantes. Dans toute attribution de priorités, au moins une de ces tâches doit avoir une priorité inférieure ou égale à $p_{min}$, mais :

- attribuer la priorité $p_{min}$ à l'une de ces tâches conduit à une échéance non respectée, comme le montre les conditions d'ordonnançabilité,
- attribuer une priorité inférieure à $p_{min}$ à l'une de ces tâches conduit également à une échéance non respectée, parce que (par hypothèse) les resultats d'ordonnançabilité ne peuvent pas être améliorés par abaissement de la priorité.

Par conséquent, nous pourrions conclure qu'il n'existe pas d'attribution de priorités conduisant à un ordonnancement valide (c'est-à-dire satisfaisant toutes les contraintes). L'algorithme est donc *optimal*, en ce sens qu'elle trouve toujours une attribution de priorités valide, si une telle attribution existe. Le nombre de tests requis augmente quadratiquement avec le nombre de tâches du système (c'est-à-dire en $\mathcal{O}(n^2)$ où $n$ est le nombre

de tâches), ce qui est une grande amélioration par rapport à l'agorithme "naïf" qui vérifie toutes les attributions de priorités possibles et requiert un nombre exponentiel de tests. Maintenant, le plus grand challenge est la détermination du scénario de premier démarrage pire cas [Aud93], c'est-à-dire l'instant critique. En effet, lorsque la date de première activation d'une tâche a un déphasage par rapport à celle d'autres tâches, déterminer la date qui implique son instant critique nécessite la connaissance de toutes les interférences qu'elle peut subir. Cependant, un ordonnancement valide produit par un algorithme à priorités fixes est périodique à partir d'une certaine date. La difficulté théorique qui se pose est de trouver le domaine de faisabilité minimal nécessaire à la connaissance d'une séquence d'ordonnancement ? Ce domaine de faisabilité doit être déterminé de façon indépendante de tout algorithme d'ordonnancement.

**Théorème 1** *[Goo98] Soit $\Gamma_n$ un système de $n$ tâches périodiques rangées par priorités décroissantes, c'est-à-dire $i < j$ implique que la priorité de $\tau_i$ est supérieure à la priorité de $\tau_j$. Soit $(s_i)_{i\in\mathbb{N}^*}$ la suite définie par*

$$\begin{cases} s_1 = r_1^1 \\ \\ s_i = r_i^1 + \left\lceil \dfrac{(s_{i-1} - r_i^1)^+}{T_i} \right\rceil \cdot T_i, \quad 2 \leq i \leq n \end{cases}$$

*S'il existe un ordonnancement valide de $\Gamma_n$ jusqu'à la date $s_n + H_n$ où $H_n = ppcm\{T_i \mid i = 1, \cdots, n\}$ et $x^+ = max(x, 0)$, alors cet ordonnancement est valide et périodique de période $H_n$ à partir de $s_n$.*

La preuve du théorème 1 donné ci-dessous est intéressante car elle ne dépend que des dates de premières activations et des périodes des tâches. Celà implique que ce théorème reste valide même lorsqu'on prend en compte des contraintes additionnelles telles que le coût du système d'exploitation (OS).

**Preuve** : *– Par induction sur $n$ –*
On a $\Gamma_n = \{\tau_1, \tau_2, \cdots, \tau_n\}$ où $\tau_i = (r_i^1, C_i, D_i, T_i)$, avec $C_i \leq D_i \leq T_i, \ 1 \leq i \leq n$.

Si $n = 1$, le théorème est vérifié : l'ordonnancement de la tâche $\tau_1$ est valide et périodique de période $T_1$ dès la première date d'activation $s_1 = r_1^1$ puisque $C_1 \leq D_1$ et $\tau_1$ est ordonnançable par hypothèse.

Supposons maintenant que $n > 1$. Par hypothèse d'induction, supposons que le théorème est vérifié jusqu'au rang $i - 1 < n$ et il existe un ordonnancement du sous système $\Gamma_i = \{\tau_1, \tau_2, \cdots, \tau_i\}$ valide jusqu'à la date $s_i + H_i$ où $H_i = ppcm\{T_j \mid j = 1, \cdots, i\}$. Notons que $s_i$ est la date de première activation de la tâche $\tau_i$ supérieure ou égale à

$s_{i-1}$ : $s_i \geq s_{i-1}$ et $H_i \geq H_{i-1}$ impliquent $s_i + H_i \geq H_{i-1} + s_{i-1}$. Par hypothèse d'induction, l'ordonnancement du sous système $\Gamma_{i-1} = \{\tau_1, \tau_2, \cdots, \tau_{i-1}\}$ est valide et périodique de période $H_{i-1}$ à partir de $s_{i-1}$. Puisque les tâches sont ordonnées par priorités décroissantes, la périodicité de l'ordonnancement des $i-1$ premières tâches n'est pas modifiée par les activations de la tâche $\tau_i$ et l'ordonnancement se répète à la date $s_i + ppcm(H_{i-1}, T_i)$. Ainsi, l'ordonnancement du sous système $\Gamma_i = \{\tau_1, \tau_2, \cdots, \tau_i\}$ est valide et périodique de période $H_i$ à partir de $s_i$. ■

**Corollaire 1** *Un ordonnancement valide d'un système $\Gamma_n$ de $n$ tâches périodiques synchrones (c'est-à-dire $\forall 1 \leq i \leq n, r_i^1 = 0$) est périodique de période $H_n = ppcm\{T_i | i = 1, \cdots, n\}$ à partir de la date $t = 0$.*

**Preuve** : La preuve découle directement du théorème 1 et du fait que le système est synchrône, c'est-à-dire $\forall\ 1 \leq i < j \leq n,\ \ r_i^1 = r_j^1 = 0$, et donc $\forall\ 1 \leq i < j \leq n,\ \ s_i^1 = s_j^1 = 0$. ■

Il est important de souligner ici que même si $H_n$ représente la période effective de l'ordonnancement qui se répète indéfiniment – *la phase permanente* –, souvent désignée sous le nom d'*hyper-période*, $s_n$ n'est pas forcément la date de début au plus tôt de la périodicité du système relativement à la phase qui précède la phase permanente – *la phase transitoire* –. Cette date a été améliorée par Emmanuel Grolleau. Dans sa thèse, il a rammener cette date à celle du dernier temps creux acyclique d'une séquence d'ordonnancement valide. Un temps creux acyclique est une unité de temps non exécutée et non périodique intervenant qu'une et une seule fois dans toute la séquence d'ordonnancement [Gro99]. A ce point, il existe quelques outils commerciaux et/ou académiques permettant de garantir l'ordonnançabilité des systèmes temps réels.

## 2.4 Outils pour l'analyse du temps de réponse

Actuellement il existe plusieurs outils commerciaux et/ou académiques d'aide au calcul du pire temps de réponse de chaque tâche dans les systèmes temps réel. Nous donnons ci-après quelques exemples de logiciels :

- *Rapid RMA* de Tri-Pacific fournit une boîte à outils fondée sur PERTS – *Prototyping Environment for Real-Time Systems* –. Comme son nom l'indique, cette boîte à outils repose sur l'analyse RMA. Il autorise les concepteurs de tester, simuler et exécuter des modèles du logiciel sur de nombreux scénarios.

- *Time Wiz* de TimeSys est un outil d'analyse d'ordonnançabilité et de simulation qui est du même type que l'outil Rapid RMA. Il est fondé lui aussi fondé sur l'analyse RMA.

- *gRMA* (a Graphical tool for Rate Monotonic Analysis) est un logiciel libre basé sur RM, dont les sources sont disponibles afin d'étendre les analyses d'ordonnançabilité des systèmes temps réel.

- *ProtEx* étend le contexte d'étude afin de supporter les analyse de bout-en-bout. Cette boîte à outils permet d'analyser et prototyper des systèmes distribués temps réel.

- *TkRTS* est un outil permettant d'analyser l'ordonnançabilité des systèmes temps réel monoprocesseurs à l'aide des algorithmes classiques (RM, DM, etc.). il permet également d'analyser l'ordonnançabilité des files multi-niveaux.

- *Cheddar* est un outil développé à l'université de Brest. C'est un logiciel libre permettant d'analyser les systèmes monoprocesseurs avec les algorithmes classiques sous les hypothèses énoncées à la section 2.2.

- *MAST – Modeling and Analysis Suite for Real-Time Applications* – est une boîte à outils qui permet la modélisation, l'analyse et l'ordonnancement des tâches temps réel dans un environnement de conception *UML* [3].

- *SymTA/S Overview* est une boîte à outils utilisé pour la budgétisation, l'ordonnancement, la vérification et l'optimisation pour les processeurs, les unités de contrôle électronique (ECU), les bus de communication, les réseaux et les systèmes intégrés.

L'usage industriel de tels outils / logiciels impose une validation par une autorité compétente des matériels utilisés. A notre connaissance, aucune certification spécifique de ces logiciels ne permet de garantir leur fiabilité, par exemple aucun de ces logiciels n'intègre précisement les coûts temporels liés au système d'exploitation (OS) utilisé. Ainsi, le concepteur d'un système temps réel est obligé de valider son logiciel au sein de l'équipe de développement de son service ou bien d'en développer un intégralement dédié et sur mesure à ses besoins personnel.

[3] UML est l'acronyme pour Unified Modelling Language.

## 2.5 Prise en compte du coût de l'OS

Les résultats que nous avons présentés jusqu'ici font l'hypothèse implicite d'un système d'exploitation (OS) idéal gérant l'exécution des tâches selon un algorithme d'ordonnancement. En effet le coût de l'OS est constitué de deux parties : une partie constante ou *coût de l'ordonnanceur* facile à déterminer d'une part, associée à l'activation et la terminaison des tâches et une partie variable difficile à déterminer d'autre part, associée à l'occurrence de chaque préemption pendant l'exécution des tâches. Les résultats présentés jusqu'ici supposent l'utilisation dans l'OS d'un ordonnanceur pour lequel ni le coût temporel associé à l'activation et la terminaison des tâches d'une part, ni le coût temporel associé à l'occurrence de chaque préemption pendant l'exécution des tâches d'autre part, n'est considéré. Le but de cette section est de montrer comment ces différents coûts temporels dûs à l'OS peuvent être pris en compte en s'appuyant principalement sur trois références [KAS93, AKA95, Bim07].

### 2.5.1 Prise en compte du coût de l'ordonnanceur

Il existe plusieurs mécanismes d'activation des tâches [KAS93, Bim07]. Quelque soit le mécanisme utilisé, il y a un coût temporel associé aux changements de contexte au début et à la fin de l'exécution de chaque instance de chaque tâche. Ces coûts sont indépendants du nombre de préemptions (ayant elles aussi un coût temporel) subit par la tâche et peuvent être assimilés aux temps d'exécution de deux routines. L'équation 2.7 permet de prendre en compte les coûts temporels des routines de ces changements de contexte, ils augmentent essentiellement la pire durée d'exécution (WCET) de chaque tâche comme le montre la figure 2.5.

$$C_i^s = C_i^{s_{in}} + C_i + C_i^{s_{out}} \tag{2.7}$$

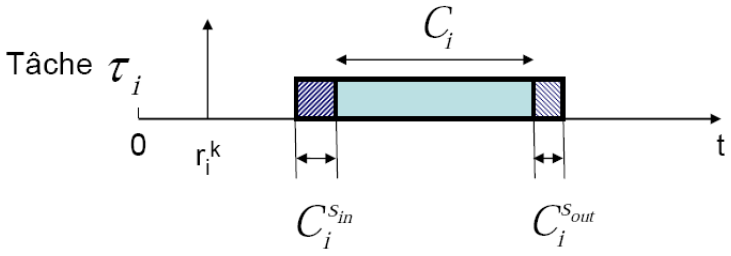


FIG. 2.5 – Illustration du coût de l'ordonnanceur

Dans l'équation 2.7,

$C_i$ : représente la pire durée d'exécution / WCET de la tâche $\tau_i$ dans laquelle est généralement approximé le coût temporel dû aux préemptions [LL73].
$C_i^{s_{in}}$ : représente la durée d'exécution de la routine du changement de contexte au début de l'exécution de la tâche $\tau_i$.
$C_i^{s_{out}}$ : représente la durée d'exécution de la routine du changement de contexte à la fin de l'exécution de la tâche $\tau_i$.
$C_i^s$ : représente la pire durée d'exécution / WCET de la tâche $\tau_i$ prenant en compte les coûts de changements de contexte.

L'équation 2.8 est une version simplifiée de l'équation 2.7 qui fait l'hypothèse que la durée d'exécution de la routine du changement de contexte est la même au début et à la fin de l'exécution pour toutes les tâches.

$$C_i^s = C_i + 2C^s \tag{2.8}$$

où $C_s$ représente la durée d'exécution de la routine du changement de contexte au début et à la fin d'une tâche quelconque.

Un mécanisme souvent utilisé pour l'activation des tâches consiste à considérer un dispositif qui génère des interruptions à un rythme lié à la période de chaque tâche : on suppose que les interruptions coïncident avec les périodes des tâches. On distingue deux types de mécanismes d'activation basés sur la génération d'interruptions : le *mécanisme intégré de gestion d'interruptions* et le *mécanisme non intégré de gestion d'interruptions*. Ces deux mécanismes ont un impact à la fois sur l'ordonnancement produit lorsque le système est ordonnançable, et sur le coût de l'OS obtenu. Cette assertion est illustrée en détail dans la sous section 2.5.2.

### Mécanisme intégré de gestion d'interruptions

Ce mécanisme de gestion d'interruptions est dit *intégré* car les priorités des interruptions générées sont égales aux priorités des tâches associées. Au début de la période d'une tâche, quand c'est à elle de s'exécuter, une interruption est générée suivie des informations supplémentaires telles que la priorité de la tâche à laquelle elle correspond. Si l'interruption a une priorité supérieure à celle de la tâche en cours d'exécution, alors elle est prise en compte. Sinon, l'interruption n'est pas prise en compte et donc reste en suspend, elle est mémorisée et traitée plus tard. Ainsi, seule une interruption de priorité supérieure à celle de la tâche en cours d'exécution peut être prise en compte et donc préempter celle-ci. L'avantage de ce mécanisme est qu'il nous permet d'éviter les phénomènes d'inversion de priorités ou de blocage des tâches [ZDE+93, SS94, But97] dûs à la prise en compte des interruptions de tâches moins prioritaires (voir figure 2.7). Si plusieurs interruptions arrivent simultanément, seule la plus prioritaire est prise en compte et les autres restent en suspend, elles sont mémorisées et traitées plus tard. La figure 2.6 illustre un exemple de mise en oeuvre de ce mécanisme où on considère un système de

trois tâches $\Gamma_3 = \{\tau_1, \tau_2, \tau_3\}$ avec $\tau_1$ désignant la tâche la plus prioritaire et $\tau_3$ la tâche la moins prioritaire. Ce mécanisme de gestion d'interruptions exige un matériel spécial dans le processeur permettant de mémoriser la priorité de la tâche en cours d'exécution dans un régistre et de comparer le contenu de ce registre avec la priorité de l'interruption de plus haute priorité en suspend.

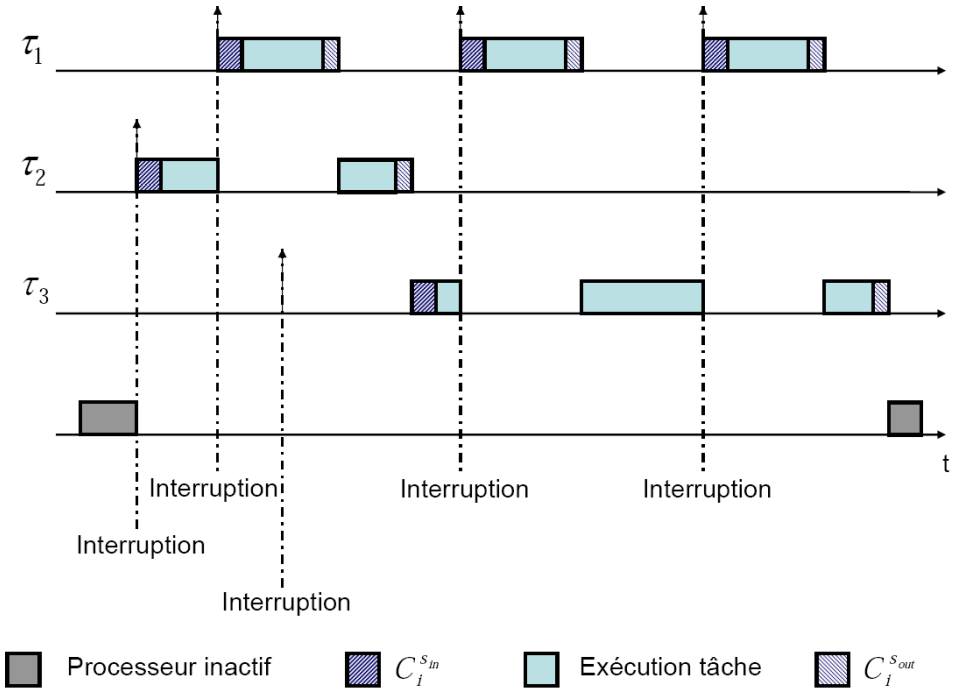


FIG. 2.6 – Illustration du mécanisme intégré de gestion d'interruptions.

### Mécanisme non intégré de gestion d'interruptions

Dans ce mécanisme de gestion d'interruptions, toutes les tâches sont aussi activées par des interruptions. Toutefois, les priorités des interruptions associées aux dates d'activation des tâches n'ont pas de correllation avec leurs priorités, en cela ce mécanisme est dit *non intégré*. Chaque fois qu'une tâche est activée, une interruption est produite et la tâche en cours d'exécution est interrompue quelque soit sa priorité. Si la tâche qui est activée a une priorité supérieure à celle de la tâche en cours d'exécution, alors une préemption a lieu. Sinon, la priorité de la tâche qui est activée est inférieure ou égale à celle de la tâche en cours d'exécution et donc elle ne préemptera pas celle-ci. Dans tous les cas, l'activation de la tâche cause donc une interruption nécessitant l'exécution

d'une routine associée à l'ordonnanceur et l'exécution de la tâche qui était en cours ne peut reprendre qu'à la fin de celle-ci. Ainsi, non seulement il peut y a avoir inversion de priorités des tâches mais il est aussi difficile d'anticiper le comportement du processeur lorsque cette approche est utilisée. La figure 2.7 illustre un exemple de mise en oeuvre de ce mécanisme où on considère le système de trois tâches $\Gamma_3 = \{\tau_1, \tau_2, \tau_3\}$ de la sous-section précédente, avec $\tau_1$ désignant toujours la tâche la plus prioritaire et $\tau_3$ la tâche la moins prioritaire.

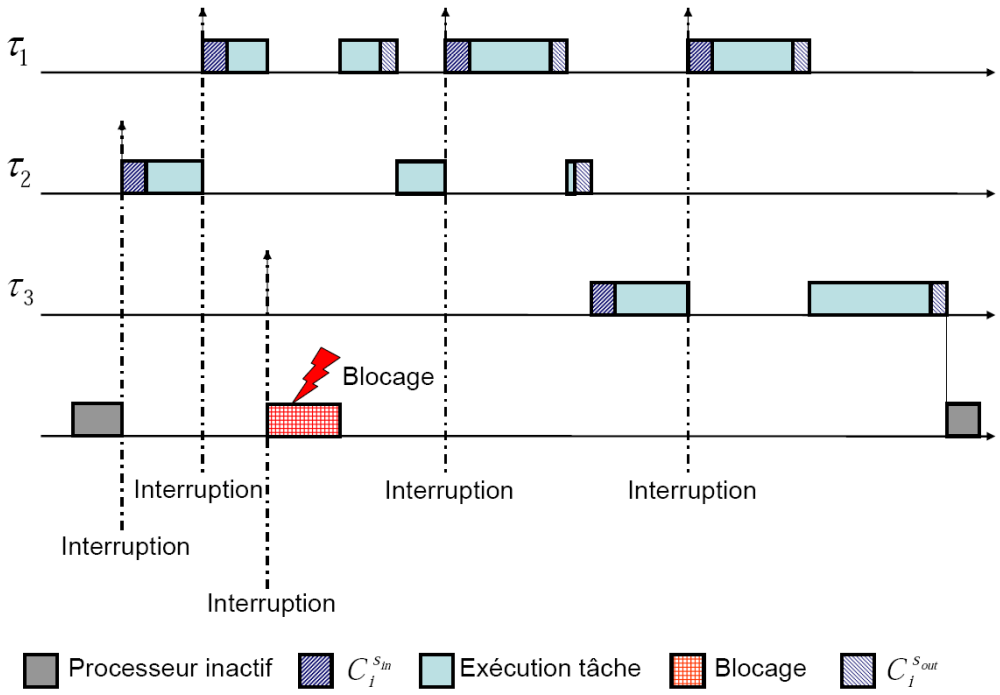


FIG. 2.7 – Illustration du mécanisme non intégré de gestion d'interruptions

### 2.5.2 Prise en compte du coût de la préemption

Contrairement à la section précédente, où prendre en compte le coût de l'ordonnanceur revient à rajouter une certaine quantité au début et à la fin de l'exécution de chaque instance pour chaque tâche, prendre en compte le coût exact dû à la préemption est un problème nettement plus difficile à résoudre [ERC95, AKA95]. En effet si nous pouvons connaître la valeur réelle du WCET d'une tâche (c'est-à-dire sans aucune approximation du coût de l'OS : ordonnanceur et préemptions), qui en soi est un domaine de recherche que nous n'aborderons pas dans cette thèse [CP00, BP05, HP08, WEE+08], alors lors-

qu'une tâche se fait préempter par une autre tâche plus prioritaire, son contexte doit être *restauré* au moment de sa re-sélection afin qu'elle puisse poursuivre son exécution. Cette restauration de contexte a un coût temporel "$\alpha$" à l'occurrence de chaque préemption qui rallonge d'une part le temps de réponse de la tâche considérée, et d'autre part peut causer par ricochet d'autres préemptions supplémentaires pour d'autres tâches moins prioritaires. L'opération de restauration du contexte d'une tâche est *atomique*, c'est-à-dire si une tâche est préemptée pendant cette opération de restauration de contexte, alors il faudra reprendre toute l'opération à *zéro* lors de la prochaine restauration de contexte de cette tâche. L'accumulation éventuelle du nombre de préemptions peut alors rendre un système non ordonnançable alors qu'il avait été initialement déclaré ordonnançable en négligeant leurs coûts. Sans nuire à la généralité, la quantité $\alpha$ peut être considérée constante pour chaque tâche puisqu'elle peut être assimilée au temps d'exécution d'une routine associée à l'ordonnanceur chaque fois que la tâche se fait préempter. La figure 2.8 illustre un exemple de prise en compte du coût exact de la préemption lorsque le mécanisme intégré de gestion d'interruptions est utilisé et la figure 2.9 illustre le même exemple lorsque le mécanisme non intégré de gestion d'interruptions est utilisé. On considère toujours le système de trois tâches $\Gamma_3 = \{\tau_1, \tau_2, \tau_3\}$ de la sous-section précédente, avec $\tau_1$ la tâche la plus prioritaire et $\tau_3$ la tâche la moins prioritaire. Dans cet exemple, $\alpha_1$ désigne le coût d'une préemption de $\tau_1$, $\alpha_2$ le coût d'une préemption de $\tau_2$ et $\alpha_3$ le coût d'une préemption de $\tau_3$. Dans la fenêtre considérée, le coût exact de la préemption vaut $(\alpha_2 + 2\alpha_3)$ pour le mécanisme intégré de gestion d'interruptions tandis qu'il vaut $(\alpha_1 + 2\alpha_2 + \alpha_3)$ pour le mécanisme non intégré de gestion d'interruptions.

Le coût exact de la préemption et par conséquent le coût exact du système d'exploitation (*OS*) obtenu est différent suivant le mécanisme de gestion d'interruptions utilisé (intégré ou non intégré). Dans le cas de l'utilisation du mécanisme non intégré de gestion d'intrruptions, le temps de blocage induit par la tâche $\tau_3$ (ayant la priorité la moins élevée du système) suite à son activation cause d'une part, directement une "préemption" de la tâche $\tau_1$ (ayant la priorité la plus élevée du système) et d'autre part, indirectement une préemption supplémentaire de la tâche $\tau_2$. Cette situation introduit nécessairement une difficulté pour l'établissement des conditions d'ordonnançabilité du système. Dans le cas de l'utilisation du mécanisme intégré de gestion d'intrruptions, cette difficulté est complètement évitée car seules les activations des tâches ayant une priorité supérieure à celle de la tâche en cours d'exécution sont prises en compte. Avec ce mécanisme, il n'y a pas de "préemption" de la tâche $\tau_1$ dûe à l'activation de la tâche $\tau_3$ ayant une priorité moins élevée. La tâche $\tau_1$ continue son exécution et l'activation de la tâche $\tau_3$ est prise en compte plus tard. Pour ces raisons ainsi que celles évoquées dans la section précédente, nous choisissons d'utiliser le mécanisme intégré de gestion d'interruptions dans toute la suite de cette thèse. Ce choix est fondamental pour la bonne compréhension de la suite de notre travail. Par ailleurs, à partir de maintenant et ceci pour toute la suite de cette thèse, chaque fois que nous parlerons du WCET $C_i$ d'une tâche $\tau_i$ pour un système

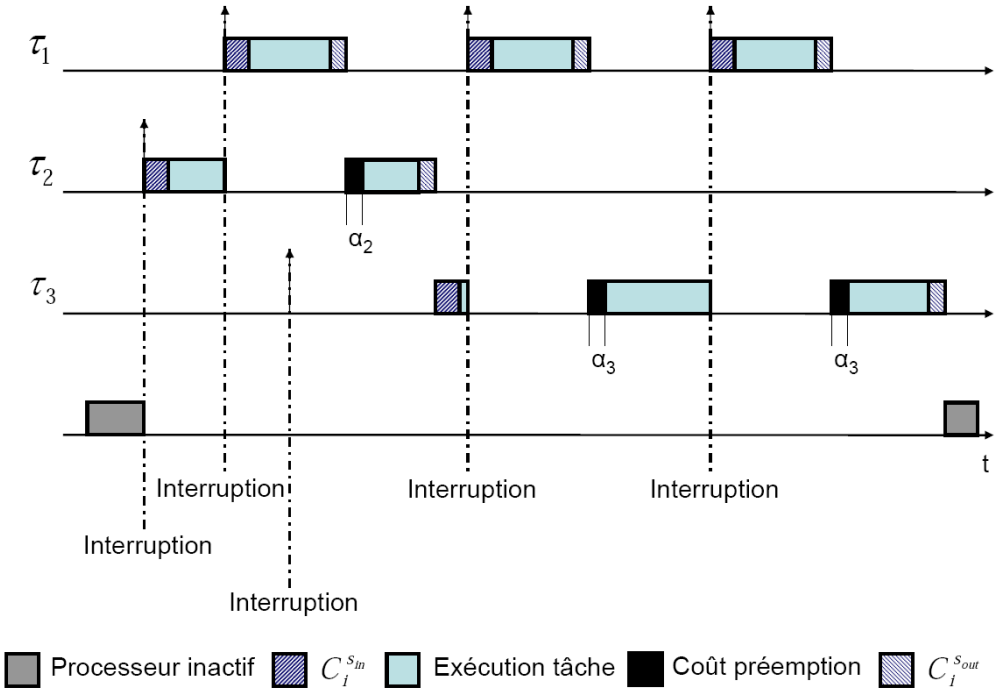


FIG. 2.8 – Illustration du coût exact de la préemption avec le mécanisme intégré de gestion d'interruptions

donné, nous supposerons qu'il correspond à la valeur donnée par l'équation 2.9.

$$C_i = C_i^{s_{in}} + C_i^{\circ} + C_i^{s_{out}} \tag{2.9}$$

Dans l'équation 2.9,

$C_i^{\circ}$ : représente la pire durée d'exécution / WCET de la tâche $\tau_i$ sans aucune approximation du coût temporel dû aux préemptions,
$C_i^{s_{in}}$ : représente la durée d'exécution de la routine du changement de contexte au début de l'exécution de la tâche $\tau_i$,
$C_i^{s_{out}}$ : représente la durée d'exécution de la routine du changement de contexte à la fin de l'exécution de la tâche $\tau_i$,
$C_i$ : représente la pire durée d'exécution / WCET de la tâche $\tau_i$ prenant en compte les coûts de changements de contexte.

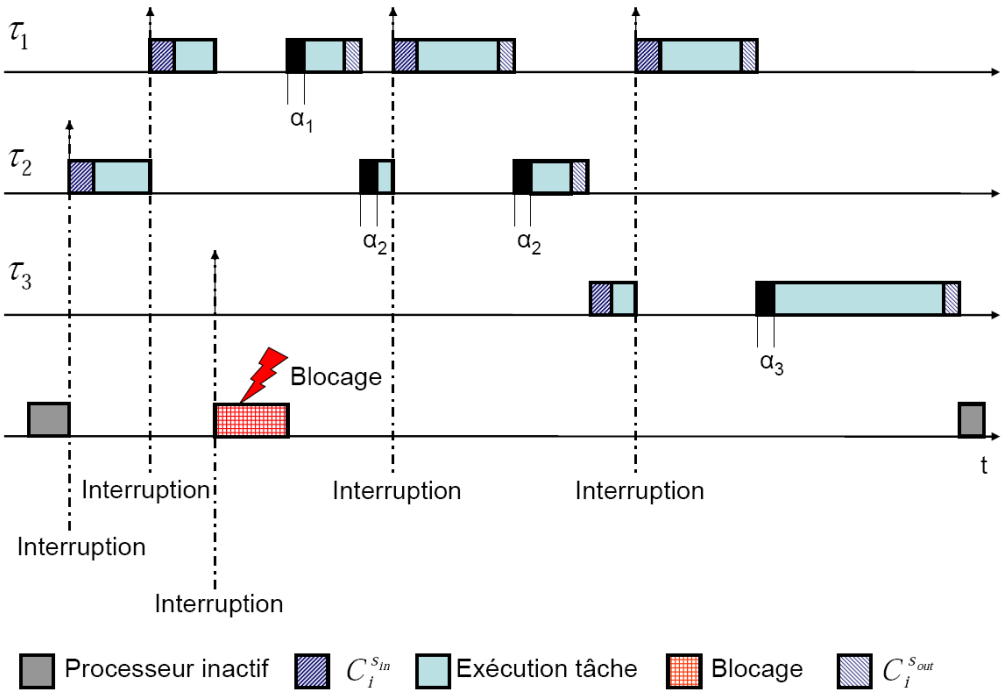


FIG. 2.9 – Illustration du coût exact de la préemption avec le mécanisme non intégré de gestion d'interruptions

## 2.6 Avantages et inconvénients des différentes techniques d'analyse d'ordonnançabilité

Comme nous l'avons déjà souligné dans le chapitre 1, la validation des systèmes temps réel durs impose de vérifier que toutes les tâches respectent leurs contraintes temporelles. Des techniques analytiques spécifiques aux systèmes temps réel ont été conçues pour répondre à cet objectif.

### 2.6.1 Les différentes techniques d'analyse

Afin de garantir que toutes les tâches d'un système temps réel dur donné respectent leurs contraintes temporelles, trois techniques analytiques pour le respect des contraintes temporelles existent et reposent :

1. **sur le facteur d'utilisation du processeur (Processor Utilization Factor Analysis)** [LL73, HT97]. Dans certains cas particuliers, cette approche permet de conclure si un système de tâches est ordonnançable ou non [LL73]. Il conduit gé-

néralement à des conditions suffisantes d'ordonnançabilité et est difficile à étendre pour prendre en compte des contraintes additionnelles telles que les coûts d'OS et de préemptions.

2. **sur la demande processeur (Processor Demand Analysis)**. Cette approche repose sur le calcul de la demande cumulée des exécutions des tâches activées et terminées dans un intervalle de temps (Demand Bound Function). Elle permet de construire des conditions d'ordonnançabilité [LSD89, JS93] en limitant l'étude d'ordonnançabilité à quelques instants d'une période d'activité du processeur mais ceux-ci ne sont pas toujours faciles à déterminer.

3. **sur l'analyse des temps de réponse (Response Time Analysis)** [JP86b, Leh90]. Cette analyse détermine les pires temps de réponse $R_i,\ i = 1, \cdots, n$ de chaque tâche [KRP$^+$93] et permet de valider l'ordonnancement de systèmes de tâches en vérifiant que $R_i \leq D_i$. Elle peut être utilisée dans des algorithmes complexes pour contrôler et garantir la qualité de l'ordonnancement.

Les deux dernières approches reposent sur les mêmes concepts et sont parfois désignées par le terme analyse de la demande de temps (*Time-Demand Analysis*) [Liu00]. Le point commun de ces approches est une analyse des périodes d'activité du processeur (*busy periods*), c'est-à-dire quand le processeur n'est pas oisif afin de déterminer le pire temps de réponse de chaque tâche. Dans la littérature, de nombreuses méthodes de calcul du pire temps de réponse ne conduisent pas à sa valeur exacte, mais à une borne supérieure à cause des différentes approximations souvent faites, notamment concernant le coût de l'OS (ordonnanceur et préemptions) subit par les tâches. Ainsi, comparer le pire temps de réponse à l'échéance de la tâche ($R_i \leq D_i,\ i = 1, 2, \cdots, n$) conduit souvent à :

1. *un algorithme de décision* si $R_i$ est le pire temps de réponse, c'est-à-dire une tâche est ordonnançable si, et seulement si, $R_i \leq D_i$.

2. *un algorithme de semi-décision* si $R_i$ est une borne supérieure du pire temps de réponse : si $R_i \leq D_i$, la tâche est ordonnançable, sinon on ne peut pas conclure.

Puisque les modèles de tâches considérés dans les trois techniques énoncées ci-dessus sont généralement simplistes au regard des applications industrielles [LL73], alors la simulation est souvent utilisée à postériori dans l'industrie pour contourner de telles difficultés. Cependant, les résultats obtenus même s'ils aident à comprendre la dynamique du système ne garantissent pas alors la validation des applications temps réel. En effet, les techniques de simulation n'intègrent pas le fait que les pires temps de réponse des tâches s'obtiennent sur des scénarios d'activation des tâches souvent difficiles à caractériser, ainsi que lorsque les tâches ne s'exécutent pas nécessairement avec leurs pires durées d'exécution.

### 2.6.2 Choix de notre approche d'analyse

Nous avons choisi l'approche basée sur le *pire temps de réponse* de chaque tâche car c'est celle qui se prête le plus naturellement à une extension permettant de prendre en compte des contraintes supplémentaires telles que le coût de l'OS (coûts liés à l'ordonnanceur et aux préemptions). De plus, cette apporche apporte un diagnostic plus détaillé du système que l'analyse de la demande processeur. Dans le cas où le système n'est pas ordonnançable, elle permet de cerner la raison précise. Nous cherchons à déterminer l'ordonnançabilité de chaque tâche individuellement et produire une condition nécessaire et suffisante d'ordonnançabilité de chaque tâche lorsque toutes les durées d'exécutions sont connues. Le calcul pratique du temps de réponse nécessite toujours de :

1. caractériser le(s) scénario(s) de première activation des tâches conduisant au pire temps de réponse de la tâche étudiée,
2. déterminer une fonction d'analyse de la durée cumulée des tâches correspondant au(x) scénario(s) de première activation.

Il n'est pas toujours facile de déterminer le(s) scénario(s) d'activation des tâches correspondant à "l'instant critique" pour toutes les tâches simultanément, en particulier lorsqu'on prend en compte le coût exact de l'OS. Par conséquent, il faut alors garantir à scénario de premier démarrage fixé, l'ordonnançabilité du système de tâches. En d'autres termes, nous supposons dans toute la suite de cette thèse que toutes les tâches sont concrètes. Nous rappelons que dans un système de tâches à priorité fixe, une tâche de priorité $i$ ne peut s'exécuter que s'il n'existe pas de tâche plus prioritaire dans l'état "prêt". Ainsi, l'exécution d'une tâche donnée est fortement liée à celle des tâches plus prioritaires qu'elles. Le concept de période d'activité doit donc être affiné pour tenir compte des tâches ayant un niveau de priorité supérieur à $i$.

**Définition 9 (Leh90)** *Une* période d'activité de niveau $i$ du processeur *est un intervalle de temps où le processeur n'exécute que des tâches ayant une priorité supérieure ou égale à* $i$ *(*$i$-level busy period*).*

Le calcul pratique du temps de réponse d'une tâche $\tau_i$ repose alors sur celui du calcul de la longueur d'une période d'activité, mais en se limitant aux tâches de priorités supérieures ou égales à $i$ [Goo98, RT02, Ber03, HD03].

### 2.6.3 Contraintes temps réel

L'analyse d'ordonnançabilité basée sur le pire temps de réponse de chaque tâche est de loin la plus développée dans la littérature [KRP$^+$93]. Deux types de contraintes temps réel sont usuellement considérés dans la littérature :

1. *extensions du modèle de tâche*. Les tâches considérées dans les paragraphes précédents sont indépendantes. La technique d'analyse du temps de réponse doit être étendue pour être exploitable sur des problèmes industriels dans lesquels les tâches sont dépendantes, avec des contraintes de périodicité strite, latence, etc.
2. *prise en compte du comportement intrinsèque du système d'exploitation (OS) et du matériel*. L'analyse d'ordonnançabilité doit intégrer, par exemple, les coûts temporels associés à l'OS à travers les changements de contextes (coûts de l'ordonnanceur) et ceux dûs aux préemptions.

Ces extensions conduisent généralement à redéfinir les notions de facteur d'utilisation du processeur et de temps de réponse des tâches afin de tenir compte des contraintes additionnelles. En effet, les extensions du modèle des tâches induisent généralement des anomalies d'ordonnancement lorsqu'un ordonnanceur à priorité fixe est utilisé. Une anomalie se caractérise par le fait que la relaxation d'une contrainte du problème peut rendre le système de tâches non ordonnançable [Gra69]. Par exemple, diminuer la durée d'exécution d'une tâche peut rendre le système non ordonnançable, alors que celui-ci est ordonnançable lorsque toutes les tâches s'exécutent avec leurs pires durées d'exécution, etc.

D'un point de vue de la complexité algorithmique [Tov02], le problème de décision associé au calcul du pire temps de réponse d'une tâche $\tau_i$ est de vérifier si $R_i \leq K$, où $K$ est une constante quelconque. On remarque ainsi, que ce problème de décision généralise celui de la condition d'ordonnançabilité de la tâche $\tau_i$ qui consiste à comparer le pire temps de réponse avec la constante $D_i$ : $R_i \leq D_i$. Par conséquent, si le problème de l'ordonnançabilité d'une tâche est $\mathcal{NP}$-Complet alors celui du calcul du temps de réponse l'est aussi. Dans les paragraphes suivants, nous considérons des contraintes temps réel séparément les uns des autres afin de souligner clairement l'impact de chacun d'eux dans la littérature.

### Echéance

Pendant les phases d'analyse d'ordonnançabilité et de validation d'un système temps réel dur, le respect des échéances de toutes les tâches est la contrainte la plus importante que cherche à satisfaire les concepteurs, ceci quelque soit le modèle utilisé. C'est la principale motivation pour laquelle autant de travaux théoriques ont été accomplis depuis ceux de Liu et Layland [LL73]. L'objectif étant de toujours proposer une modélisation se rapprochant le plus possible de la réalité physique afin de garantir le bon fonctionnement du système à l'exécution, les hypothèses simplificatrices souvent utilisées dans les modèles ont été relaxées les unes après les autres [CC86, SCGM99, BMR90, BGK97, ERC95, Cuc04, Bim07]. Toute fois, il reste encore de nombreuses améliorations à faire notamment concernant la réduction du pessimisme dans les différentes conditions d'ordonnançabilité proposées, la prise en compte des différents coûts liés à l'OS, etc.

### Précédence

Les problèmes d'ordonnancement impliquant des contraintes de précédence entre les tâches se retrouvent dans plusieurs domaines d'activité [Cuc04]. Un exemple typique consiste à considérer un système temps réel dans lequel certaines tâches communiquent : si la tâche $\tau_i$ est la tâche productrice d'une donnée et $\tau_j$ une tâche consommatrice de cette donnée, alors l'exécution de $\tau_j$ ne peut pas débuter avant la fin de l'exécution de $\tau_i$. Graphiquement les précédences sont souvent représentées par des arcs orientés reliant des sommets représentant les tâches. Etant donné que les précédences entre les tâches ne peuvent pas admettre de cycles, c'est-à-dire par exemple la tâche $\tau_i$ précède la tâche $\tau_j$ et la tâche $\tau_j$ précède la tâche $\tau_i$, alors ils sont généralement représentées par un graphe orienté sans cycle (voir chapitre 1 : figure 1.5). Plusieurs travaux dans la littérature traitent le problème de tâches dépendantes. Parmi eux, deux travaux sont pertinents : d'une part, les travaux de Cucu et al [Cuc04] utilisent le modèle de tâches sans dates de réveil non préemptif afin d'éviter les coûts dûs aux préemptions, mais ce choix limite la possibilité de trouver des ordonnancements valides. D'autre part, les travaux de Mangeruca et al [MBFSV06] utilisent le modèle de tâches classique, ils généralisent la notion de précédence entre les tâches, mais font l'hypothèse d'un système idéal, c'est-à-dire considère que tous les coûts temporels liés à l'OS sont nuls.

### Périodicité stricte

Dans la pratique, les applications basées sur des algorithmes provenant de l'automatique et/ou de traitement du signal sont généralement spécifiées à l'aide de bloc-diagrammes. Dans ces applications, l'ordonnancement des tâches exige un contrôle précis et continue de l'échantillonnage et du traitement des données. Les bloc-diagrammes sont souvent composés de fonctions produisant et consommant de données, et chaque fonction doit démarrer son exécution aussitôt que les données sont disponibles afin de maîtriser au mieux les temps de réponse et garantir les rapports entrées/sorties. A l'aide du modèle de tâches sans dates de réveil introduit dans le chapitre précédent (voir chapitre 1 : figure 1.4), Korst et al [KAL96, KAL97] arrivent à modéliser les circuits intégrés qui implantent des algorithmes de traitement du signal. Cucu et al [Cuc04] arrivent à résoudre les problèmes d'ordonnancement rencontrés dans le domaine de l'automatique et du contrôle/commande. Mais à chaque fois, soit on est restreint par la nature non préemptives des tâches, soit l'hypothèse d'un système idéal est fait.

### Latence

Les contraintes de latence dans les systèmes temps réel durs sont utiles dans la pratique car la notion d'échéance (deadline) du modèle classique n'est pas suffisante pour exprimer des contraintes temporelles entre les dates de début et de fin effectives de deux

tâches d'un système donnée. En effet, exprimer cette contrainte est souvent nécessaire dans plusieurs applications industrielles parce que la pertinence de certains résultats en dépend. En guise d'exemple, le temps s'écoulant entre l'appui sur la pédale de frein d'un véhicule et l'arrêt des roues freinées doit être borner puisque cette action peut entraîner l'exécution d'un ensemble de tâches réalisant la commande appropriée. Cucu, Pernet et Sorel [CPS07] ont proposé une définition de la latence appliquée à un couple quelconque de tâches lorsqu'elles sont toutes non préemptives, permettant de généraliser celles proposées par Gerber et al., ainsi que Kang et al [GPM97, DRM97]. Dans le même article [CPS07], les auteurs ont établi un lien entre la notion de chemin orienté dans un graphe d'algorithme et la notion de contrainte de latence appliquée à un couple de tâches de ce graphe et cela leurs a servi dans la recherche des conditions d'ordonnançabilité. De plus, ils ont proposé une méthodologie de réduction du nombre de contraintes du système. Cette méthodologie s'appuit sur la transformation d'un modèle de tâches basé sur les échéances vers un modèle de tâches basé sur les contraintes de latence. Leur contribution est à compléter car elle ne peut plus être appliquée lorsque la préemption est autorisée.

### Coût de l'OS

Les coûts dû à l'OS sont constitués des coûts associés à l'ordonnanceur et de ceux associés à l'occurrence de chaque préemption lors de l'exécution du système. Dans la littérature, il existe à notre connaissance très peu de travaux qui intègrent explicitement ces coûts dans les conditions d'ordonnançabilité proposées [KAS93, AKA95, ERC95, Bim07]. Si les contributions consernant la prise en compte du coût de l'ordonnanceur sont satisfaisantes, celles concernant la prise en compte du coût de la préemption peuvent, quant à elles, être encore améliorées.

- **Coût de l'ordonnanceur** : les systèmes d'exploitation (OS) souvent utilisés pour l'exécution des systèmes temps réels ne sont pas idéaux. Ils nécessitent lors de l'exécution d'un système de tâches donné un temps n'étant pas toujours négligeable pour démarrer et terminer chaque requête d'exécution de chaque tâche. Ces temps étant dépendant de l'architecture matérielle utisée doivent être intégrés dans les conditions d'ordonnançabilité de ces systèmes afin que celles-ci soient plus réalistes. Plusieurs travaux sont allés dans ce sens et parmi eux on peut citer les travaux de Katcher et al. [KAS93] et les ceux de Bimbard et al. [Bim07] qui ont proposés des contributions intéressantes. Dans les deux cas, de nouveaux résultats qui réflètent mieux la réalité physique ont été obtenus.

- **Coût de la préemption** : l'approximation du coût de la préemption dans le WCET des tâches comme c'est généralement le cas dans la littérature peut conduire à

un mauvais comportement du système alors que l'analyse d'ordonnançabilité a conclu que le système de tâches considéré est ordonnançable. Pour éviter ce problème les concepteurs de systèmes temps réel autorisent généralement des marges qui sont très diffiles à évaluer, et qui, dans tous les cas donnent lieu à un gaspillage des ressources puisque le pire temps de réponse d'une tâche est plus grand que son WCET dès que la tâche a été préemptée au moins une fois [CC86, CC89]. A ce jour, il existe très peu d'études dont le but principal est de compter le nombre exact de préemptions. Parmi ces études, les plus pertinentes sont les suivantes. A. Burns, K. Tindell et A. Wellings dans [AKA95] ont présenté une analyse qui permet de prendre en compte le coût global des préemptions dans les équations standards de calcul du pire temps de réponse de chaque tâche, mais les résultats obtenus sont pessimistes car les auteurs considèrent le nombre maximal de préemptions possible au lieu du nombre exact. Juan Echagüe, I. Ripoll et A. Crespo ont également essayé de résoudre le problème du nombre exact de préemptions dans [ERC95] en construisant l'ordonnancement tâche après tâche et en comptant le nombre de préemptions. Mais, ils n'ont pas pu déterminer le nombre exact de préemption. En effet, ils n'ont pas pris en compte le coût de chaque préemption au cours de l'analyse qu'ils ont proposé. Par conséquent, cela revient à l'examen du nombre minimum de préemptions parce que certaines préemptions ne sont pas prises en compte : celles qui sont dues à l'augmentation de la durée d'exécution de la tâche à cause du coût de la préemptions lui-même.

**Gigue**

La maîtrise de la gigue d'une tâche dans un système temps réel permet de caractériser la régularité d'exécution de toutes ses instances. Dans la littérature, la gigue semble être l'une des caractéristiques des tâches les plus difficiles à maîtriser lors de la conception des systèmes temps réel durs. Plusieurs travaux théoriques intéressants ont été proposés pour pallier cette difficulté [BCM97, BBGL99, Gro99, RT02], mais dans la plupart des cas, les conditions d'ordonnançabilité les plus pertinentes qui sont proposées proviennent d'une part d'algorithmes de complexité exponentielle, et d'autre part induisent des surdimensionnements de ressource.

## 2.7 Conclusion

Dans ce chapitre, nous avons commencé par présenter les hypothèses simplificatrices souvent faites dans les modèles utilisés pour la résolution des problèmes d'ordonnancement dans les systèmes temps réels durs. Ensuite, nous avons présenté l'état de l'art concernant les différentes techniques d'analyse, basées sur les algorithmes non oisifs préemptifs à priorités fixes, et nous avons donné les raisons motivant notre choix de

technique d'analyse. Nous avons présenté quelques outils commerciaux et académiques existants et nous avons souligné les challenges qui restent à résoudre, notamment concernant le respect des contraintes – échéance, précédence, périodicité stricte, latence, gigue – auxquelles est soumis le système. Nous avons souligné la difficulté inhérente à la prise en compte du coût de l'OS, lié au nombre exact de préemptions, dans les conditions d'ordonnançabilité existantes dans la littérature. Enfin, nous avons souligné l'importance et la nécessité de fournir des conditions necessaires et suffisantes d'ordonnançabilité intégrant tous ces paramètres. Ces nouvelles conditions garantiront toujours un bon fonctionnement du système à l'exécution et élimineront tout gaspillage de ressources.

# Chapitre 3

# Ordonnancement temps réel sans coût de la préemption

*Demain est moins à découvrir*
*qu'à inventer.*
**Gaston Berger**

## 3.1 Introduction

Dans les deux chapitres précédents (chapitre 1 et chapitre 2), nous avons présenté les notions de base nécessaires à la compréhension des termes que nous utiliserons dans ce chapitre pour l'analyse d'ordonnançabilité des systèmes temps réel durs. Nous avons présenté l'état de l'art des travaux relatifs à de tels systèmes afin de bien isoler le problème à résoudre et justifier les hypothèses que nous considérons dans la suite de cette thèse. Dans ce chapitre, nous allons présenter la première partie de notre contribution. Les modèles classiques n'étant pas adaptés pour prendre en compte le coût exact de la préemption, nous allons introduire un nouveau modèle pour résoudre le problème général de l'ordonnancement monoprocesseur de systèmes temps réel durs avec des contraintes multiples telles que la précédence, la périodicité stricte, la latence et la gigue, tout en tenant compte du coût exact du système d'exploitation (*OS*) pour tous les scénarii de premières activations de toutes les tâches (*simultané* ou *non simultané*). Fondé sur ce nouveau modèle, nous proposerons une analyse d'ordonnançabilité qui utilise la définition d'une opération binaire d'ordonnancement $\oplus$ dont les opérandes sont appelés otâches. Nous rappelons que le coût de l'OS est constitué de deux parties : une partie constante ou *coût de l'ordonnanceur*, facile à déterminer d'une part, associée à l'activation et la terminaison des tâches et une partie variable, difficile à déterminer d'autre part, associée à l'occurrence de chaque préemption pendant l'exécution des tâches. Nous montrerons la correspondance entre les notions introduites et les notions habituelles de l'ordonnancement temps réel classique et nous présenterons également des résultats concernant la réduction du domaine d'analyse d'ordonnançabilité d'un système donné lorsque le coût de la préemption n'est pas pris en compte. Enfin nous présenterons l'opé-

ration binaire d'ordonnancement $\oplus$ dans ce cadre lorsque les priorités sont attribuées suivant une politique imposée telle que *Rate Monotonic*, *Deadline Monotonic*, *Audsley*, etc.

## 3.2 Notion d'otâche

Dans cette section, nous allons commencer par introduire quelques définitions pour préparer le cadre de notre approche d'analyse d'ordonnançabilité des systèmes temps réel durs et ensuite nous allons présenter notre contribution à la théorie de l'ordonnancement temps réel sans coût de la préemption.

**Définition 10** *Un générateur $\Sigma$ est un ensemble fini de symboles.*

**Définition 11** *Une* otâche *sur le générateur $\Sigma$ est un multiensemble ordonné constitué d'un certain nombre (éventuellement zéro) d'éléments de $\Sigma$.*

Pour illustrer les définitions 10 et 11 qui sont deux définitions importantes pour une bonne compréhension de la suite de cette thèse, si $\Sigma = \{a, e\}$ est un générateur alors quelques otâches sur $\Sigma$ sont données par $\tau_1 = \{a\}$, $\tau_2 = \{e, a, a\}$, $\tau_3 = \{a, e, a\}$ et $\tau_4 = \{a, a, e, e, a\}$. La différence fondamentale que nous faisons dans la définition 11 entre la notion de *multiensemble ordonné* et la notion de *multiensemble* [MW85, Yag86, Bli91, Knu98] introduite par le mathématicien Hollandais *Nicolaas Govert de Bruijn* dans les années $1970$s est que dans notre cas l'ordre des éléments dans une otâche donnée est important. En effet cet ordre nous permet de faire la différence entre deux otâches quelconques et en particulier entre deux otâches issues des permutations des éléments d'une autre otâche. Par exemple, les deux otâches $\{a, a, a, a, e, e, e\}$ et $\{a, e, e, a, a, a, e\}$, vues comme des otâches issues des permutations des éléments de la otâche $\{a, a, e, a, e, a, e\}$, sont différentes. La *otâche nulle*, notée $\Lambda$, est l'ensemble ne contenant aucun symbole, c'est-à-dire $\{\}$ ou $\emptyset$.

**Définition 12** *Un* système d'otâches $\Gamma$ *est un ensemble d'otâches sur un générateur $\Sigma$ donné.*

La définition 12 est générale et autorise l'éventualité qu'un système d'otâches soit considéré comme étant une collection arbitraire d'otâches autant indépendantes que cohérentes. Aussi, elle autorise la considération de systèmes d'otâches fortement structurés tels que des systèmes d'otâches périodiques, des systèmes d'otâches apériodiques, des systèmes d'otâches hybrides, etc. La proximité des terminologies *otâche* et *tâche temps réel* d'une part, puis *système d'otâches* et *système de tâches temps réel* d'autre part, exprime l'idée d'une relation entre elles que nous détaillerons plus loin dans ce chapitre.

En effet nous montrerons qu'une tâche temps réel correspond à une otâche particulière et qu'un système de tâches temps réel correspond à un système d'otâches particulier. Avant d'aller plus loin, il serait intéressant de voir dans quelle mesure cela fait du sens de considérer un système de tâches temps réel (tel qu'un système de tâches périodiques, par exemple) comme un simple ensemble d'otâches particulier. Quand nous nous intéressons à l'étude de systèmes temps réel embarqués, nous manipulons nécessairement des tâches qui peuvent être : périodiques, sporadiques, ou apériodiques. Puisque l'étude d'un système de tâches périodiques est différente de celle d'un système de tâches apériodiques par exemple, nous avons jugé préférable de définir un seul ensemble d'otâches dans lequel chaque tâche temps réel a son correspondant de façon univoque. Cet ensemble autorise des éléments de tous les différents types, comme étant une collection d'otâches cohérentes et bien définies, dans lequel on peut se restreindre à un sous ensemble lorsqu'on souhaite étudier l'ordonnançabilité de celui-ci. Puisqu'à chaque système temps réel donné, on fait correspondre de façon bijective un système d'otâches, la conclusion concernant l'ordonnançabilité du système d'otâches permettra de conclure sur l'ordonnançabilité du système de tâches temps réel. Même si nous parlons habituellement d'un système temps réel à travers les tâches qui le constituent, nous devons noter que chaque tâche est régie par des règles et des paramètres qui caractérisent le système temps réel. Maintenant, nous devons établir l'équivalent de ces règles et paramètres pour un système d'otâches.

Quand nous étudions un système d'otâches, nous devons commencer par spécifier un générateur contenant tous les symboles légaux qui pourront être utilisés pour construire les otâches de ce système. Tous les générateurs utilisés dans cette thèse sont composés d'un nombre très réduit de symboles. Ils contiennent la plupart du temps deux, et occasionnellement un seul symbole. Les propriétés qui nous intéresserons vont découler de ces hypothèses.

**Propriété 1** *$\Lambda$ est une otâche sur $\Sigma$ quelque soit le générateur $\Sigma$ considéré.*

Pour éviter toute confusion, nous ne permettrons jamais que la lettre $\Lambda$ représente un élément de $\Sigma$.

**Définition 13** *Soit $\tau$ une otâche sur le générateur $\Sigma$. Le* cardinal *de $\tau$ est le nombre d'éléments dans cette otâche, nous noterons ce nombre $|\tau|$.*

Pour illustrer la définition 13, si $a, e \in \Sigma$ alors $|\Lambda| = 0$, $|\tau_1| = 1$, $|\tau_2| = |\tau_3| = 3$ et $|\tau_4| = 4$. Remarquons que lorsque nous écrivons $a$, nous faisons référence au symbole dans le générateur alors que $\{a\}$ fait référence à la otâche de cardinal $1$.

**Définition 14** *Soit un générateur $\Sigma$. L'*ensemble de toutes les otâches possibles sur $\Sigma$ *est noté $\Sigma^*$.*

**Propriété 2** *Tout système d'otâches* $\Gamma$ *sur le générateur* $\Sigma$ *est nécessairement un sous ensemble de* $\Sigma^*$.

Pour illustrer la définition 14 et la propriété 2, supposons que $\Sigma$ est un générateur à deux symboles $\{a, e\}$, c'est-à-dire $\Sigma = \{a, e\}$. Alors l'ensemble de toutes les otâches possibles sur $\Sigma$ est donné par

$$\Sigma^* = \{a, e\}^* = \{\Lambda, \{a\}, \{e\}, \{a, a\}, \{a, e\}, \{e, a\}, \{e, e\}, \{a, a, a\}, \{a, a, e\}, \ldots\}$$

Quelques exemples de systèmes d'otâches sur $\Sigma$ sont donnés par

$\Gamma_1 = \{\Lambda, \{a\}, \{a, a\}, \{a, a, e\}\}$
$\Gamma_2 = \{x \in \{a, e\}^* / \ |x| \leq 8\}$
$\Gamma_3 = \{x \in \{a, e\}^* / \ |x|$ est impair$\}$
$\Gamma_4 = \{x \in \{a, e\}^* / \ n_a(x) \geq n_e(x)\}$
$\Gamma_5 = \{x \in \{a, e\}^* / \ |x| \geq 2$ et $x$ commence et fini avec $e\}$

Dans le quatrième exemple $n_a(x)$ et $n_e(x)$ représentent respectivement le nombre de symboles "$a$" et de symboles "$e$", dans la otâche $x$.

Puisque les systèmes d'otâches sont par définition des ensembles d'otâches, de nouveaux systèmes d'otâches peuvent être construits en utilisant les opérations de base sur les ensembles.

**Propriété 3** *Si* $\Gamma_1$ *et* $\Gamma_2$ *sont deux systèmes d'otâches sur un générateur* $\Sigma$*, alors leur union, intersection, complémentaire et différence sont aussi des systèmes d'otâches sur le générateur* $\Sigma$.

Quand nous parlons de complémentaire d'un système d'otâches $\Gamma$ sur $\Sigma$, nous considérons que l'univers est l'ensemble $\Sigma^*$. Ainsi le complémentaire du système $\Gamma$ est donné par $\Gamma' = \Sigma^* \backslash \Gamma$ ([1]). Notons que deux systèmes d'otâches quelconques peuvent être définies sur un générateur commun.

**Propriété 4** *Si* $\Gamma_1 \subseteq \Sigma_1^*$ *et* $\Gamma_2 \subseteq \Sigma_2^*$*, alors* $\Gamma_1$ *et* $\Gamma_2$ *sont tous les deux des sous ensembles de* $(\Sigma_1 \cup \Sigma_2)^*$.

La propriété 4 pourrait être une source de confusion puisqu'à partir de maintenant le complémentaire de $\Gamma_1$ pourrait être soit $\Sigma_1^* \backslash \Gamma_1$, soit $(\Sigma_1 \cup \Sigma_2)^* \backslash \Gamma_1$. Chaque fois que cela sera nécessaire, cette ambiquité sera levée.
L'opération de *concaténation* sur les otâches nous permet également de construire de nouveaux systèmes d'otâches.

[1] $\Gamma' = \Sigma^* \backslash \Gamma$ implique que $\Gamma$ et $\Gamma'$ constituent une partition de $\Sigma^*$, c'est-à-dire $\Gamma \cup \Gamma' = \Sigma^*$ et $\Gamma \cap \Gamma' = \emptyset$

**Définition 15** *Soient $x$ et $y$ deux otâches de $\Sigma^*$. La* concaténation *de $x$ et $y$ est la otâche $xy$ obtenue en écrivant les symboles de $x$ et les symboles de $y$ consécutivement.*

Pour illustrer la définition 15, si $x = \{a, e, e\}$ et $y = \{e, a\}$ alors la otâche $xy$ est donnée par $xy = \{a, e, e, e, a\}$ et la otâche $yx$ est donnée par $yx = \{e, a, a, e, e\}$. D'où $xy \neq yx$ à cause de l'importance de l'ordre des éléments dans une otâche. Par conséquent, l'opération de concaténation n'est pas *commutative*, c'est-à-dire qu'étant donné un générateur $\Sigma$, il existe des otâches $x$ et $y$ sur $\Sigma$ telles que la otâche $xy$ est différente de la otâche $yx$.

**Propriété 5** *Pour toute otâche $x$ sur un générateur $\Sigma$, la concaténation de $x$ et $\Lambda$ vaut $x$ :*

$$x\Lambda = \Lambda x = x$$

Evidemment, l'opération de concaténation est *associative* et cela s'exprime par la propriété 6.

**Propriété 6** *Pour toutes otâches $x$, $y$, et $z$ sur un générateur $\Sigma$, $(xy)z = x(yz)$.*

L'intérêt principal de la propriété 6 est qu'elle nous permet de concaténer plusieurs otâches sans se soucier de l'ordre dans lequel les différentes opérations de concaténation seront effectuées.

**Définition 16** *Soient $x$ et $y$ deux otâches de $\Sigma^*$. S'il existe deux otâches $w$ et $z$, éventuellement nulles, telles que $y = wxz$, alors $x$ est une* sous otâche *de $y$. Nous appellerons l'opération permettant d'obtenir une sous otâche à partir d'une otâche une* extraction.

Pour illustrer la définition 16, la otâche $\{a, e, e\}$ est une sous otâche de chacune des otâches $\{e, e, a, e, e, a, a\}$, $\{a, a, a, e, e\}$, et $\{a, e, e\}$ puisque nous avons par exemple $\{e, e, a, e, e, a, a\} = \{e, e\}\{a, e, e\}\{a, a\}$, mais n'est pas une sous otâche de la otâche $\{a, e, a, e\}$.

**Définition 17** *Une otâche $p$ est un* préfixe *d'une otâche $y$ si et seulement si $p$ est une sous otâche de $y$ telle que*

$$\begin{cases} y = pw_1 \\ \nexists z \in \Sigma^* \backslash \{\Lambda\} \ / \ y = zpw_2 \end{cases}$$

*où $w_1$ et $w_2$ sont deux otâches de $\Sigma^*$.*

Pour illustrer la définition 17, les préfixes de la otâche $\{a, e, a, a\}$ sont donnés par $\Lambda$, $\{a\}$, $\{a, e\}$, $\{a, e, a\}$ et $\{a, e, a, a\}$ et rien d'autre. Par exemple la otâche $\{a, a\}$ n'est pas un préfixe de la otâche $\{a, e, a, a\}$.

**Définition 18** *Une otâche $s$ est un* suffixe *d'une otâche $y$ si et seulement si $s$ est une sous otâche finale de $y$, c'est-à-dire*

$$\begin{cases} y = w_1 s \\ \nexists z \in \Sigma^* \backslash \{\Lambda\} \ / \ y = w_2 s z \end{cases}$$

*où $w_1$ et $w_2$ sont deux otâches de $\Sigma^*$.*

Pour illustrer la définition 18, les suffixes de la otâche $\{a, e, a, a\}$ sont donnés par $\Lambda$, $\{a\}$, $\{a, a\}$, $\{e, a, a\}$ et $\{a, e, a, a\}$ et rien d'autre. Par exemple la otâche $\{e, a\}$ n'est pas un suffixe de la otâche $\{a, e, a, a\}$.

Maintenant que nous avons clairement défini l'opération de concaténation, nous pouvons l'appliquer aussi bien aux systèmes d'otâches qu'aux otâches elles-mêmes.

**Définition 19** *Soient $\Gamma_1, \Gamma_2 \subseteq \Sigma^*$ deux systèmes d'otâches sur $\Sigma$. La* concaténation de $\Gamma_1$ et $\Gamma_2$ *est le système d'otâches $\Gamma_1\Gamma_2$ obtenu en concaténant les otâches de $\Gamma_1$ à celles de $\Gamma_2$.*

$$\Gamma_1\Gamma_2 = \{xy / x \in \Gamma_1 \ \textit{et} \ y \in \Gamma_2\}$$

Pour illustrer la définition 19, si on considère deux systèmes d'otâches $\Gamma_1$ et $\Gamma_2$ sur le générateur $\Sigma = \{a, e\}$ tels que $\Gamma_1 = \{\{a, e, e\}, \{a, e, a, e\}\}$ et $\Gamma_2 = \{\{a, e\}, \{e, e, a, a\}\}$, alors le système d'otâches $\Gamma_1\Gamma_2$ obtenu en concaténant les otâches de $\Gamma_1$ à celles de $\Gamma_2$ est donné par

$$\Gamma_1\Gamma_2 = \{\{a, e, e\}, \{a, e, a, e\}\}\,\{\{a, e\}, \{e, e, a, a\}\} =$$
$$\{\{a, e, e, a, e\}, \{a, e, a, e, a, e\}, \{a, e, e, e, e, a, a\}, \{a, e, a, e, e, e, a, a\}\}$$

**Remarque 2** *Les operations de concaténation et d'extraction ont, par rapport aux operations classiques, union, intersection, complémentaire, et différence, la particularité qu'elles conservent l'ordre.*

Nous noterons à partir de maintenant en exposant le nombre de fois qu'un élément est concaténé. Ces éléments peuvent indifféremment être soit de simples otâches, ou même des systèmes d'otâches. Ainsi, si $\Sigma$ est un générateur et si $x \in \Sigma^*$, et $\Gamma \subseteq \Sigma^*$, alors

$$\begin{aligned} x^k &= xx\cdots x \\ \Sigma^k &= \Sigma\Sigma\cdots\Sigma = \{x \in \Sigma^* / |x| = k\} \\ \Gamma^k &= \Gamma\Gamma\cdots\Gamma \end{aligned}$$

où dans chaque cas, il y a $k$ facteurs qui sont concaténés. Un cas important est le cas particulier où $k = 0$ :

$$x^0 = \Lambda, \quad \Sigma^0 = \{\Lambda\}, \quad \Gamma^0 = \{\Lambda\}.$$

$\Gamma^k$ est l'ensemble de toutes les otâches qui peuvent être obtenues en concaténant $k$ fois des éléments de $\Gamma$. Maintenant, nous définissons l'ensemble de toutes les otâches qui peuvent être obtenues en concaténant un nombre *arbitraire* de fois des éléments du système $\Gamma$ :

$$\Gamma^* = \bigcup_{k=0}^{\infty} \Gamma^k$$

L'operation $^*$ dans cette expression est habituellement appelée l'étoile de Kleene : **Kleene**$^*$ en reconnaissance au mathématicien et logicien américain *Stephen Cole Kleene* [2]. Par définition, remarquons que $\Lambda$ est toujours un élément de $\Gamma^*$ quelque soit $\Gamma$ puisque $\Gamma^0 = \{\Lambda\}$. Finalement, nous noterons $\Gamma^+$ l'ensemble de toutes les otâches obtenues en concaténant un ou plusieurs éléments de $\Gamma$ :

$$\Gamma^+ = \bigcup_{k=1}^{\infty} \Gamma^k$$

Il est immédiat de vérifier que $\Gamma^+ = \Gamma^*\Gamma = \Gamma\Gamma^*$ et que les deux systèmes $\Gamma^*$ et $\Gamma^+$ pourraient, en effet, être égaux.

**Théorème 2** *Soit $\Sigma$ un générateur et $\Gamma$ un système d'otâches sur $\Sigma$.*

$$\textit{Si } \Gamma^2 \subseteq \Gamma, \textit{ alors } \Gamma^+ \subseteq \Gamma \tag{3.1}$$

**Preuve** : *– Par induction sur $k$ –*

Nous rappelons que $\Gamma^+ = \bigcup_{k=1}^{\infty} \Gamma^k$. Si nous montrons, pour tout $k \geq 1$, que chacun des systèmes d'otâches $\Gamma^k$ est un sous ensemble de $\Gamma$, alors il en découle que leur union est aussi un sous ensemble de $\Gamma$. Soit $\mathcal{P}(k)$ la proposition

$$\text{Si } \Gamma^2 \subseteq \Gamma, \text{ alors } \Gamma^k \subseteq \Gamma$$

Si nous montrons que $\mathcal{P}(k)$ est vraie pour tout $k \geq 1$, alors la proposition initiale est vraie.
*Première étape* : Nous devons montrer que $\mathcal{P}(1)$ est vraie, c'est-à-dire : si $\Gamma^2 \subseteq \Gamma$, alors $\Gamma^1 \subseteq \Gamma$. Ceci est immédiat (même sans l'hypothèse $\Gamma^2 \subseteq \Gamma$), puisque $\Gamma^1 = \Gamma$. Maintenant, montrons l'étape d'induction.
*Hypothèse d'induction* : $k \geq 2$ et si $\Gamma^2 \subseteq \Gamma$, alors $\Gamma^k \subseteq \Gamma$.
*Etape d'induction* : nous voulons montrer que si $\Gamma^2 \subseteq \Gamma$, alors $\Gamma^{k+1} \subseteq \Gamma$. En effet, en supposant que $\Gamma^2 \subseteq \Gamma$, nous avons $\Gamma^{k+1} = \Gamma\Gamma^k \subseteq \Gamma\Gamma = \Gamma^2 \subseteq \Gamma$. ■

[2] Stephen Cole Kleene, né le 5 janvier 1909 à Hartford et mort le 25 janvier 1994, est un mathématicien et logicien américain.

**Théorème 3** *Soit $\Sigma$ un générateur et $\Gamma$ un système d'otâches sur $\Sigma$. Pour construire $\Gamma^*$, il suffit que les trois assertions suivantes sont satisfaites :*

1. $\Lambda \in \Gamma^*$,
2. *pour tout $x \in \Gamma^*$ et pour tout $y \in \Gamma$, $xy \in \Gamma^*$,*
3. *toute otâche appartenant à $\Gamma^*$ peut être obtenue en utilisant les règles 1 et 2.*

**Preuve** :

Nous rappelons que $\Gamma^* = \bigcup_{k=0}^{\infty} \Gamma^k$. Pour tout $k \geq 0$, pour toute otâche $x \in \Gamma^k$, pour toute otâche $y \in \Gamma$, la otâche $xy$ est un élément de $\Gamma^{k+1}$ et par suite un élément de $\Gamma^*$. De plus, tout élément de $\Gamma^*$ peut être obtenu de cette façon sauf $\Lambda$, qui vient de $\Gamma^0$.

■

Pour illustrer le théorème 3, considérons le générateur $\Sigma = \{a, e\}$ et le système d'otâches $\Gamma = \{\{a\}, \{a, e\}\}$. Nous commençons avec $\Lambda$. Une application de la règle 2 ajoute les otâches $\Lambda\{a\} = \{a\}$ et $\Lambda\{a, e\} = \{a, e\}$ dans $\Gamma^1 = \Gamma$. Une seconde application nous donne les otâches dans $\Gamma^2$, qui sont $\Lambda\{a\}$, $\Lambda\{a, e\}$, $\{a\}\{a\}$, $\{a\}\{a, e\}$, $\{a, e\}\{a\}$, $\{a, e\}\{a, e\}$ ; par conséquent, à ce niveau, nous avons tous les éléments de $\Gamma^0 \cup \Gamma^1 \cup \Gamma^2$. Toute concaténation de $k$ éléments de $\Gamma$ peut être obtenue à partir de la définition en utilisant $k$ applications de la règle 2.

Les systèmes d'otâches les plus intéressants à étudier sont ceux constitués d'otâches infinies (en d'autres termes, ceux dont les otâches ont un cardinal infini). Cependant, pour pouvoir manipuler ce type de systèmes d'otâches, nous devons être capable de les spécifier — ou les décrire — de façon finie. A ce niveau nous pouvons distinguer deux approches possibles à ce problème, qui peuvent être illustrées par les exemples suivants :

$$\begin{array}{ll} 1^{ere} \text{ approche :} & \Gamma_1 = \{\{a, e\}, \{e, a, e\}\}^* \cup \{\{a, a\}\}\{\{e, e, a\}\}^* \\ 2^{eme} \text{ approche :} & \Gamma_2 = \{x \in \{a, e\}^* / \; n_a(x) \geq n_e(x)\} \end{array}$$

Dans le cas de $\Gamma_1$, nous décrivons le système d'otâches en disant comment une otâche arbitraire du système peut être construite : soit en concaténant un nombre arbitraire d'otâches, chacun étant soit un $\{a, e\}$ ou un $\{e, a, e\}$, ou en concaténant la otâche $\{a, a\}$ avec un nombre arbitraire de copies de la otâche $\{e, e, a\}$. Nous décrivons d'autre part $\Gamma_2$, en spécifiant une propriété qui caractérise les otâches du système. En d'autres termes, nous disons comment *reconnaître* une otâche du système : compter le nombre de $a$ et le nombre de $e$ et comparer les deux valeurs obtenues. Evidemment, la limite entre ces deux approches n'est pas toujours claire. La définition

$$\Gamma_3 = \{\{e\}y\{e\} \; / \; y \in \{a, e\}^*\}$$

qui est une autre façon de décrire le dernier système dans notre groupe de cinq exemples, peut être interprêtée comme une méthode de génération des otâches dans $\Gamma_3$ (commencer avec une otâche arbitraire $y$ et y adjoindre $e$ à chaque extrêmité, en supposant que le cardinal final est au moins égal à 2). Même une définition qui appartient à la première catégorie, comme celle de $\Gamma_1$, pourrait immédiatement suggérer une façon de déterminer si une otâche arbitraire appartient à $\Gamma_1$. De même, il pourrait exister une façon triviale de générer toutes les otâches appartenant à $\Gamma_2$, quoique la définition donnée soit clairement du second type. Dans la suite de cette thèse, nous nous limiterons à une spécification des systèmes d'otâches basée sur la première approche, c'est-à-dire décrivant le système d'otâches en disant comment une otâche arbitraire du système est construite. Par exemple, les otâches auxquelles nous nous intéresserons par la suite sont celles ayant un cardinal infini et dont pour chacune d'elles, on peut extraire une sous otâche de cardinal fini telle que la otâche soit une concaténation infini de la sous otâche à partir d'un certain rang.

Jusqu'ici dans la présentation de ce chapitre, nous n'avons pas fait intervenir la notion de "temps" qui est pourtant central dans cette thèse. A partir de maintenant, nous considérons dans toute la suite un indice parcourant un axe temporel orienté qui est une référence temporelle pour toutes les otâches. Sur cet axe, nous identifions un instante de référencee $t_0$, nous pouvons par exemple choisir $t_0 = 0$.

**Remarque 3** *En plus du cardinal et de l'ordre des éléments permettant de faire la différence entre deux otâches quelconques sur un générateur $\Sigma$, les dates de début "$r \in \mathbb{Z}$" et de fin "$f \in \mathbb{Z}$" relativement à l'instant de référence $t_0$ de chaque otâche sont importantes et peuvent aussi permettre de faire la différence entre deux otâches.*

En effet, si on considère l'instant de référence $t_0 = 0$ alors la otâche finie $\tau_1 = \{e, a, e\}$ débutant à la date $r_1 = 0$ et finissant à la date $f_1 = 3$ est différente de la otâche finie $\tau_2 = \{e, a, e\}$ débutant à la date $r_2 = 6$ et finissant à la date $f_2 = 9$. Par conséquent, à partir de maintenant, nous noterons les dates de début et de fin d'une otâche en indice : Par exemple l'expression $\{e, a, e\}_{(r,f)}$ indique que la otâche $\{e, a, e\}$ débute à la date $r$ et finit à la date $f$ où $f \geq r$. Nous supposerons toujours que si on considère une otâche $\tau_{(r,f)}$ alors on a :

$$f = r + |\tau| \tag{3.2}$$

**Propriété 7** *Pour toute otâche $\tau \neq \Lambda$ sur le générateur $\Sigma$ qui débute à la date $r$ et finit à la date $f$, il existe toujours un préfixe fini $x \neq \Lambda$ et un suffixe $y$ éventuellement égal à $\Lambda$ de $\tau$ tel que $|\tau| = |x| + |y|$ et $\tau$ est une concaténation de $x$ et $y$. En d'autres termes, on peut toujours écrire*

$$\tau_{(r,f)} = x_{(r,r+|x|)} y_{(r+|x|,r+|\tau|)}$$

**Remarque 4** *Dans l'équation 3.2, si $f = r$ alors $|\tau| = 0$. Cette assertion implique que la otâche $\tau_{(r,r)}$ correspond à la otâche ne contenant aucun élément, c'est-à-dire $\Lambda$. C'est la otâche qui débute à la date $r$ et finit à la date $r$.*

**Remarque 5** *Dans l'équation 3.2, si $\tau$ est une otâche de cardinal infini, c'est-à-dire $|\tau| = \infty$ alors $f = r + \infty = \infty$. Dans ce cas, il n'est pas nécessaire d'indiquer la valeur de $f$ puisqu'elle ne présente aucune ambiguïté.*

**Convention** : Si la remarque 5 est satisfaite alors on note $\tau_{(r,f)}$ par $\tau_r$ au lieu de $\tau_{(r,\infty)}$ pour signifier la otâche de cardinal infini débutant à la date $r$.

Pour illustrer cette convention, si nous considérons le générateur $\Sigma = \{a, e\}$ alors la otâche $\tau_1 = \{a, a, e, e, e, a, e, a, e, e, a, e, a, \cdots, e, e, a, e, a, \cdots\}_0$ désigne une otâche de cardinal infini débutant à la date $r_1 = 0$.

## 3.3 Otâches périodiques

**Définition 20** *Une* otâche périodique $\tau_{per}$ *sur un générateur $\Sigma$ est une otâche de cardinal infini ayant une date de début $r$ et telle qu'il existe une sous otâche $\tau$ de $\tau_{per}$ de cardinal fini $|\tau|$ et $\tau_{per}$ est une concaténation infinie de $\tau$ à partir d'un certain instant $\beta \geq r$. Nous noterons*

$$\tau_{per} = \zeta_{(r,\beta)}\tau_{\beta}^{\infty} \tag{3.3}$$

*où l'entier $\beta$ désigne le plus petit instant tel que la relation 3.3 soit satisfaite, $\zeta_{(r,\beta)}$ désigne la otâche finie de cardinal $\beta - r$ débutant à la date $r$ et finissant à la date $\beta$ et $\tau_{\beta}^{\infty}$ désigne la otâche de cardinal infini, concaténation infinie de la otâche $\tau$ débutant à la date $\beta$.*

**Convention** : Dans la relation 3.3, les otâches $\zeta_{(r,\beta)}$ et $\tau_{\beta}^{\infty}$ désignent respectivement *la partie initiale* et *la partie périodique* de la otâche périodique $\tau_{per}$.

**Définition 21** *Un* système d'otâches périodiques $\Gamma$ *sur un générateur $\Sigma$ est un ensemble constitué uniquement d'otâches périodiques sur $\Sigma$.*

Pour illustrer les définitions 20 et 21, si nous considérons $\Sigma = \{a, e\}$ alors $\tau_1 = \{a, a, e, \underbrace{e, e, a, e, a}_{\tau}, \underbrace{e, e, a, e, a}_{\tau}, \cdots, \underbrace{e, e, a, e, a}_{\tau}, \cdots\}_0 = \{a, a, e\}_{(0,3)}\{e, e, a, e, a\}_3^{\infty}$ est une otâche périodique sur $\Sigma$ dont la partie initiale est $\{a, a, e\}_{(0,3)}$ et la partie périodique est $\{e, e, a, e, a\}_3^{\infty}$. Un système d'otâches périodiques sur $\Sigma$ est alors par exemple donné par $\Gamma = \{\{a, a, e\}_{(0,3)}\{e, e, a, e, a\}_3^{\infty}, \{a, e, e, a\}_{(12,16)}\{e, a, a, e, e, a, a, a\}_{16}^{\infty}\}$.

A partir de maintenant, étant donnée une otâche périodique, nous distinguerons deux formes différentes pour son écriture : la *forme factorisée* et la *forme développée*. Pour la otâche $\tau_1$ par exemple : $\{a, a, e\}_{(0,3)}\{e, e, a, e, a\}_3^{\infty}$ désignera la *forme factorisée* et $\{a, a, e, \underbrace{e, e, a, e, a}_{\tau}, \underbrace{e, e, a, e, a}_{\tau}, \cdots, \underbrace{e, e, a, e, a}_{\tau}, \cdots\}_0$ désignera la *forme développée*.

**Remarque 6** *Soit $\tau_{per}$ une otâche périodique sur $\Sigma$. Si l'existence de l'entier $\beta$ tel que $\tau_{per} = \zeta_{(r,\beta)}\tau_{\beta}^{\infty}$ est unique par définition, celle de la sous otâche finie $\tau$ n'est pas unique. En effet, la otâche $\tau_1$ peut aussi s'écrire $\tau_1 = \{a,a,e\}_{(0,3)}\{\underline{e,e,a,e,a}_{\tau}, \underline{e,e,a,e,a}_{\tau}\}_{3}^{\infty}$ et on a $5 = |\{e,e,a,e,a\}| \neq |\{e,e,a,e,a,e,e,a,e,a\}| = 10$.*

**Définition 22** *Soit $\tau_{per} = \zeta_{(r,\beta)}\tau_{\beta}^{\infty}$ une otâche périodique sur un générateur $\Sigma$. Le* motif *de $\tau_{per}$ est la sous otâche $\tau_{min}$ de $\tau$ de cardinal minimal telle que $\tau_{per} = \zeta_{(r,\beta)}(\tau_{min})_{\beta}^{\infty}$.*

**Théorème 4** *Le motif de toute otâche périodique $\tau_{per}$ sur un générateur $\Sigma$ existe toujours et il est unique.*

**Preuve** :

Soit $\tau_{per}$ une otâche périodique sur un générateur $\Sigma$. L'existence d'un motif de $\tau_{per}$ découle directement de la définition 20. Maintenant il reste à montrer l'unicité. Soient $\tau_1$ et $\tau_2$ deux motifs de $\tau_{per}$, on a $\tau_{per} = \zeta_{(r,\beta)}(\tau_1)_{\beta}^{\infty} = \zeta_{(r,\beta)}(\tau_2)_{\beta}^{\infty}$. Puisque l'ordre des éléments d'une otâche est important, il suffit de montrer que $|\tau_1| = |\tau_2|$. En effet, si $\tau_1$ est un motif de $\tau_{per} = \zeta_{(r,\beta)}(\tau_2)_{\beta}^{\infty}$ alors $|\tau_1| \leq |\tau_2|$ grâce à la définition 22. De même, si $\tau_2$ est un motif de $\tau_{per} = \zeta_{(r,\beta)}(\tau_1)_{\beta}^{\infty}$ alors $|\tau_2| \leq |\tau_1|$ toujours grâce à la définition 22. Il en découle donc que $|\tau_1| = |\tau_2|$. ■

Pour illustrer la définition 22, le motif de la otâche périodique $\tau_1$ est donné par $\tau_{min} = \{e,e,a,e,a\}$. Nous notons au passage que le cardinal $|\tau_{min}|$ du motif de la otâche périodique $\tau_{per} = \zeta_{(r,\beta)}\tau_{\beta}^{\infty}$ divise nécessairement $|\tau|$.

**Propriété 8** *Soient $\tau_1 = \zeta_{i(r_i,\beta_i)}(\tau_i)_{\beta_i}^{\infty}$ et $\tau_2 = \zeta_{j(r_j,\beta_j)}(\tau_j)_{\beta_j}^{\infty}$ deux otâches périodiques sur un générateur $\Sigma$. $\tau_1$ et $\tau_2$ sont* égales *si et seulement si d'une part $\zeta_{i(r_i,\beta_i)} = \zeta_{j(r_j,\beta_j)}$ et d'autre part $\tau_1$ et $\tau_2$ ont le même* motif.

**Définition 23** *Soient $\tau_1 = \zeta_{i(r_i,\beta_i)}(\tau_i)_{\beta_i}^{\infty}$ et $\tau_2 = \zeta_{j(r_j,\beta_j)}(\tau_j)_{\beta_j}^{\infty}$ deux otâches périodiques sur un générateur $\Sigma$. Les otâches $\tau_1$ et $\tau_2$ sont* équivalentes *si et seulement si elles ont la même forme développée.*

Dans la suite de cette thèse, toute otâche périodique $\tau_{per}$ sur un générateur $\Sigma$ sera notée $\tau_{per} = \zeta_{(r,\beta)}\tau_{\beta}^{\infty}$ où $\zeta_{(r,\beta)}$ est la partie initiale, $\tau$ le motif et $T = |\tau|$ la *période* de $\tau_{per}$.

**Théorème 5** *Une otâche périodique de période $T$ sur un générateur $\Sigma$ est équivalente à une infinité d'otâches périodiques de période $T$ sur $\Sigma$.*

**Preuve** :

Soit $\tau_{per} = \zeta_{(r,\beta)}\tau_{\beta}^{\infty}$ une otâche périodique sur un générateur $\Sigma$. Puisqu'on a $\tau \neq \Lambda$ et $|\tau|$ fini par définition, alors par la propriété 7, il existe un préfixe fini $x \neq \Lambda$ et un suffixe fini $y$ de $\tau$ tels que $|\tau| = |x| + |y|$ et $\tau_{(\beta,\beta+|\tau|)} = x_{(\beta,\beta+|x|)}y_{(\beta+|x|,\beta+|\tau|)}$. Ainsi :

$$\begin{array}{rcl} \tau_{per} & = & \zeta_{(r,\beta)}\tau_{\beta}^{\infty} \\ & = & \zeta_{(r,\beta)}(xy)_{\beta}^{\infty} \\ & = & \zeta_{(r,\beta)}(xy)(xy)(xy)\cdots(xy)(xy)\cdots \\ & = & \zeta_{(r,\beta)}x(yx)(yx)(yx)\cdots(yx)(yx)\cdots \\ & = & \left(\zeta_{(r,\beta)}x_{(\beta,\beta+|x|)}\right)(yx)_{\beta+|x|}^{\infty} = \tau_{per}^{'} \end{array}$$

La otâche $\tau_{per}^{'}$ est une otâche périodique de période $T$ dont la partie initiale est la otâche $\zeta_{(r,\beta)}x_{(\beta,\beta+|x|)}$ et la partie périodique $(yx)_{\beta+|x|}^{\infty}$. Puisque les otâches $x$ et $y$ sont quelconques, on peut répéter ce processus autant de fois que l'on souhaite et ainsi le théorème est prouvé. ■

**Remarque 7** *Soit $\tau_{per} = \zeta_{(r,\beta)}\tau_{\beta}^{\infty}$ une otâche périodique sur le générateur $\Sigma$. Pour tout entier $k \in \mathbb{N}$, on a aussi*

$$\tau_{per} = \zeta_{(r,\beta)}\tau_{\beta}^{\infty} = \zeta_{(r,\beta)}\underbrace{\tau\tau\tau\cdots\tau}_{kfois}\tau_{\beta+k|\tau|}^{\infty} = \zeta_{(r,\beta)}\tau_{(\beta,\beta+k|\tau|)}^{k}\tau_{\beta+k|\tau|}^{\infty} \tag{3.4}$$

*Dans l'expression 3.4, $\tau_{(\beta,\beta+k|\tau|)}^{k}$ désigne la otâche finie débutant à la date $\beta$ et finissant à la date $\beta + k|\tau|$, elle correspond à la otâche $\tau$ concaténée $k$ fois.*

Dans toute la suite de cette thèse, nous supposons toujours que le générateur est constitué de deux symboles "$a$" et "$e$" : $\Sigma = \{a, e\}$. Relativement au contexte de la théorie de l'ordonnancement des systèmes temps réel, nous identifierons toujours le symbole "$a$" à une unité de temps *disponible* et le symbole "$e$" à une unité de temps *exécutée* ou *exécutable* suivant les cas que nous détaillerons plus tard. Nous supposons que le motif $\tau$ de toute otâche périodique $\tau_{per} = \zeta_{(r,\beta)}\tau_{\beta}^{\infty}$ contient au moins un symbole "$e$".

**Convention** : Si $\tau_{per} = \zeta_{(r,\beta)}\tau_{\beta}^{\infty}$ est une otâche périodique et le motif de sa partie périodique vaut $\tau = \{a\}$ alors on considère que $\tau_{per}$ est égale à la otâche de cardinal fini $\zeta_{(r,\beta)}$ :

$$\tau_{per} = \zeta_{(r,\beta)}\{a\}_{\beta}^{\infty} = \zeta_{(r,\beta)} \tag{3.5}$$

De même nous supposons que le motif $\tau$ de toute otâche périodique $\tau_{per} = \zeta_{(r,\beta)}\tau_{\beta}^{\infty}$ contient au moins un symbole "$a$". En effet, si c'est pas le cas, en identifiant le symbole "$e$" à une unité de temps *exécutée* ou *exécutable*, la otâche périodique $\tau_{per} = \zeta_{(r,\beta)}\{e\}_{\beta}^{\infty}$ correspond à une otâche dont les éléments ne changent plus à partir d'une certaine date : la date $\beta$. Cette situation n'est pas intéressante puisque notre but est de pouvoir composer

des otâches en remplassant les unités de temps disponibles d'une otâche, les symboles "$a$", par les unités de temps exécutables d'une autre otâche, les symboles "$e$".

En définitive, les hypothèses ci-dessus impliquent que les otâches de cardinaux finis $\{a, e\}$ et $\{e, a\}$ sont toujours des sous otâches du motif d'une otâche périodique. Par conséquent, la période de toute otâche périodique est supérieure ou égale à 2.

**Définition 24** *Soit $\tau_{per} = \zeta_{(r,\beta)}\tau_{\beta}^{\infty}$ une otâche périodique sur $\Sigma = \{a, e\}$, on dira que $\tau_{per}$ est* canonique *si et seulement si le premier élément de la otâche de cardinal fini $\tau$ est le symbole "$e$".*

Pour illustrer la définition 24, la otâche périodique $\tau_1 = \{a, a, e\}_{(0,3)}\{e, e, a, e, a\}_3^{\infty}$ est canonique puisque le premier élément de la otâche finie $\{e, e, a, e, a\}$ est le symbole "$e$", tandis que la otâche périodique $\tau_2 = \{a, a, e\}_{(0,3)}\{a, a, e, e, a, e, a\}_3^{\infty}$ ne l'est pas puisque le premier élément de la otâche finie $\{a, a, e, e, a, e, a\}$ est le symbole "$a$". Grâce au théorème 5, toute otâche périodique dont le motif contient le symbole "$e$" est équivalente à une otâche périodique canonique. En effet, à titre d'exemple, on a :

$$\begin{aligned}
\tau_2 &= \{a, a, e\}_{(0,3)}\{a, a, e, e, a, e, a\}_3^{\infty} \\
&= \{a, a, e\}_{(0,3)}(\{a, a\}\{e, e, a, e, a\})_3^{\infty} \\
&= \{a, a, e\}_{(0,3)}\{a, a\}_{(3,5)}(\{e, e, a, e, a\}\{a, a\})_5^{\infty} \\
&= \{a, a, e, a, a\}_{(0,5)}\{e, e, a, e, a, a, a\}_5^{\infty}
\end{aligned}$$

Jusqu'ici, nous nous sommes surtout intéressés aux otâches périodiques et aux systèmes d'otâches périodiques car c'est ceux qui nous intéressent au premier chef. Avant de nous y restreindre définitivement, notons qu'à part les otâches périodiques, les otâches apériodiques et les otâches sporadiques nous permettent de construire une partition de l'ensemble $\Sigma^*$.

**Définition 25** *Une otâche* apériodique $x$ *sur le générateur $\Sigma = \{a, e\}$ est une otâche de cardinal fini ou non constitué d'un nombre fini de symboles "$e$". Si $\mathcal{A}$ désigne l'ensemble de toutes les otâches apériodiques sur $\Sigma = \{a, e\}$, alors $\mathcal{A}$ est donné par :*

$$\mathcal{A} = \{x \in \Sigma^* : n_e(x) \textit{est fini}\}$$

*où $n_e(x)$ représente le nombre de symboles "$e$" de la otâche $x$.*

**Définition 26** *Une otâche* sporadique $x$ *sur le générateur $\Sigma = \{a, e\}$ est une otâche de cardinal infini qui n'est ni périodique, ni aporadique. Si $\mathcal{S}$ désigne l'ensemble de toutes les otâches sporadiques sur $\Sigma = \{a, e\}$, alors $\mathcal{S}$ est donné par :*

$$\mathcal{S} = \{x \in \Sigma^* : x \notin \mathcal{P} \cup \mathcal{A}\} = \Sigma^* \backslash (\mathcal{P} \cup \mathcal{A})$$

*où $\mathcal{P}$ représente l'ensemble de toutes les otâches périodiques sur $\Sigma = \{a, e\}$.*

**Remarque 8** *Toute otâche $\tau$ sur le générateur $\Sigma$ de cardinal fini est une otâche apériodique sur $\Sigma$.*

A partir de maintenant, nous nous intéressons principalement à des systèmes d'otâches périodiques $\Gamma$ sur le générateur $\Sigma$ sauf mention explicite du contraire. Pour chaque otâche périodique $\tau_{per} = \zeta_{(r,\beta)}\tau_{\beta}^{\infty}$, nous considérons toujours la otâche périodique canonique $\tau_{per}^{'} = \zeta_{(r,\beta')}\tau_{\beta'}^{\infty}$ équivalente où $\beta^{'}$ est le plus petit entier supérieur à $\beta$ tel que la otâche $\tau_{per}^{'}$ est canonique. La figure 3.1 illustre la représention graphique d'une otâche périodique. Dans cette figure, chaque case hachurée correspond au symbole "$e$" et chaque case non hachurée au symbole "$a$" du générateur $\Sigma$. La partie initiale, qui est finie, est comprise entre les dates $r$ et $\beta$. Le motif de la partie périodique, qui se répète infiniment, est comprise entre les dates $\beta$ et $\beta + T$.

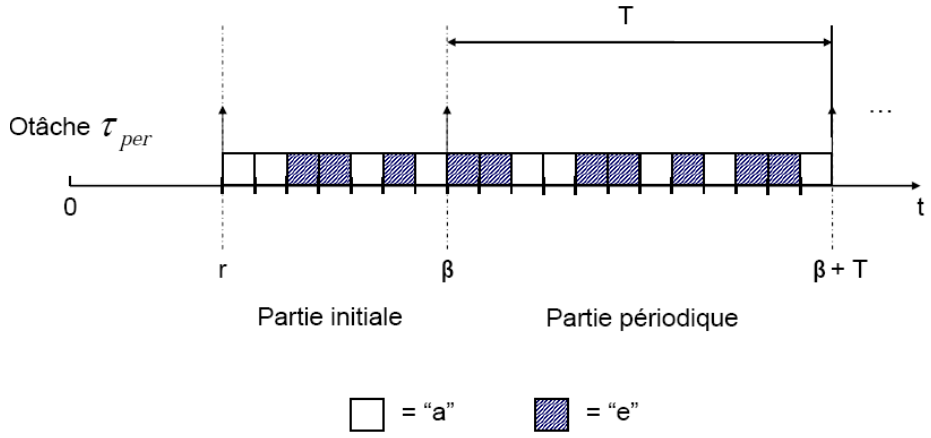


FIG. 3.1 – Illustration graphique d'une otâche périodique de période $T$.

**Définition 27** *L'*échéance relative $D$ *d'une otâche périodique $\tau_{per} = \zeta_{(r,\beta)}\tau_{\beta}^{\infty}$ est une valeur entière égale à $\beta - r$ pour la partie initiale de $\tau_{per}$ et égale au cardinal d'un unique préfixe contenant tous les symboles "$e$" du motif de la partie périodique de $\tau_{per}$.*

$$D = \begin{cases} \beta - r & \textit{pour la partie initiale} \\ |\tau_0| & \textit{pour la partie périodique} \end{cases}$$

*où $\tau_0$ est un unique préfixe contenant tous les symboles "$e$" du motif de la partie périodique de $\tau_{per}$.*

**Remarque 9** *Soit $\tau_{per} = \zeta_{(r,\beta)}\tau_{\beta}^{\infty}$ une otâche périodique canonique, si le dernier élément du motif de $\tau_{per}$ est le symbole "$e$" alors le seul préfixe contenant tous les symboles*

*"$e$" du motif est $\tau$ lui même. Dans ce cas, on a forcément $D = |\tau|$ pour la partie périodique de $\tau_{per}$.*

Pour chaque otâche périodique $\tau_{per} = \zeta_{(r,\beta)}\tau_{\beta}^{\infty}$, puisque la définition de l'échéance relative ne présente aucune ambiguïté pour la partie initiale, il n'est pas nécessaire de la représenter graphiquement. Cependant, pour la partie périodique, l'échéance sera représentée par un crochet : ⌟. La figure 3.2 illustre la représention graphique d'une otâche périodique avec échéance $D$. Dans cette figure, $D$ peut prendre $5$ valeurs possibles relativement à la position du dernier symbole "$e$" dans la partie périodique de la otâche. Ces valeurs sont $\{T, T-1, T-2, T-3, T-4\}$. Notons que dans notre modèle, la valeur de l'échéance relative de la partie périodique de toute otâche périodique est inférieure ou égale à sa période par définition. En effet l'échéance relative de la partie périodique de toute otâche périodique est égale au cardinal d'un unique préfixe de son motif, sa valeur est par conséquent inférieure ou égale au cardinal du motif lui-même.

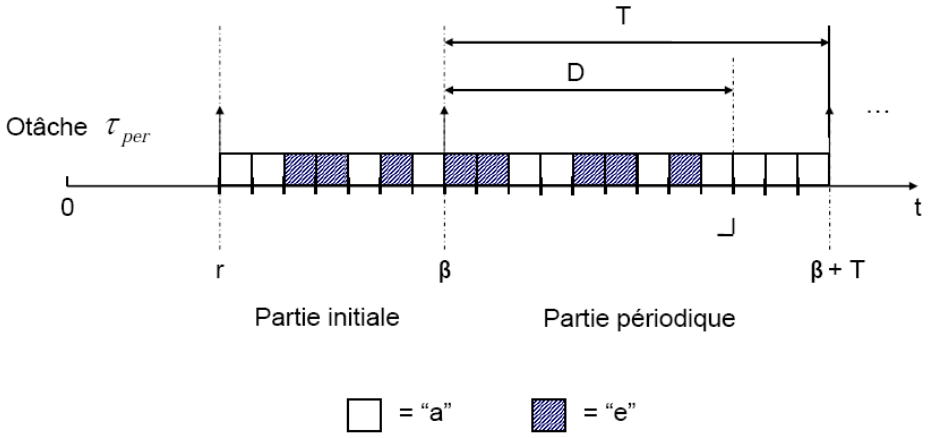


FIG. 3.2 – Illustration graphique d'une otâche périodique avec échéance $D$.

Puisque la partie initiale et le motif de chaque otâche périodique $\tau_{per} = \zeta_{(r,\beta)}\tau_{\beta}^{\infty}$ sont finis par définition, alors le nombre de séquences du symbole "$e$" dans la partie initiale et le nombre de séquences du symbole "$e$" dans le motif de $\tau_{per}$ sont aussi finis. Si $n_{in}$ et $n_p$ désignent respectivement le nombre de séquences de "$e$" dans la partie initiale et dans le motif de $\tau_{per}$ alors les ensembles

$$\begin{cases} SEQ_{in} = \{seq_{in}^{l} : \ l \in \{1, \cdots, n_{in}\}\} \\ \\ SEQ_{p} = \{seq_{p}^{l} : \ l \in \{1, \cdots, n_{p}\}\} \end{cases}$$

où $seq_{in}^{l}$ désigne la $l^{ieme}$ séquence de symboles "$e$" de la partie intiale, et $seq_{p}^{l}$ désigne la $l^{ieme}$ séquence de symboles "$e$" de la partie périodique sont aussi finis. Considérons les

applications $r_{in}$ et $r_p$ suivantes définies respectivement par rapport aux dates $r$ et $\beta$.

– Pour la partie initiale de $\tau_{per}$ :

$$r_{in} : SEQ_{in} \longrightarrow \{0, 1, \cdots, \beta - r - 1\}$$

$$seq_{in}^{l} \longmapsto r_{in}(seq_{in}^{l}) = r_{in}^{l} \text{ tel que } r + r_{in}^{l} \text{ est la date de début de } seq_{in}^{l}$$

– Pour le motif de $\tau_{per}$ :

$$r_p : SEQ_p \longrightarrow \{0, 1, \cdots, |\tau| - 1\}$$

$$seq_{p}^{l} \longmapsto r_p(seq_{p}^{l}) = r_{p}^{l} \text{ tel que } \beta + r_{p}^{l} \text{ est la date de début de } seq_{p}^{l}$$

Les applications $r_{in}$ et $r_p$ nous permettent de donner les définitions 28 et 29 ci-dessous concernant les dates de sous-activations et les sous-durées d'exécution pour une otâche périodique $\tau_{per}$ appartenant à $\mathcal{P}$.

**Définition 28** *Soit une otâche périodique $\tau_{per} = \zeta_{(r,\beta)}\tau_{\beta}^{\infty}$, on appelle* date de sous-activation de rang $l$ de la partie initiale de $\tau_{per}$ *et on note $r_{in}^{l}$ (resp.* date de première sous-activation de rang $l$ de la partie périodique de $\tau_{per}$ *et on note $r_{p}^{l,1}$) le nombre entier $r_{in}^{l} = r + r_{in}(seq_{in}^{l})$ (resp. le nombre entier $r_{p}^{l,1} = \beta + r_p(seq_{p}^{l})$).*

**Remarque 10** *Puisque la otâche $\tau_{per} = \zeta_{(r,\beta)}\tau_{\beta}^{\infty}$ est périodique de période $T = |\tau|$, alors la date $r_{p}^{l,k} = \beta + r_p(seq_{p}^{l}) + (k-1) \cdot T = r_{p}^{l,1} + (k-1) \cdot T,\ k \geq 1$ est aussi une date de sous-activation de $\tau_{per}$, elle correspond à la* date de la $k^{ieme}$ sous-activation de rang $l$ de la partie périodique de $\tau_{per}$.

**Définition 29** *Soit une otâche périodique $\tau_{per} = \zeta_{(r,\beta)}\tau_{\beta}^{\infty}$, on appelle* sous-durée d'exécution de rang $l$ de la partie initiale *(resp.* sous-durée d'exécution de rang $l$ de la partie périodique*) et on note $C_{in}^{l}$ (resp. $C_{p}^{l}$) le cardinal de la sous otâche constituée uniquement de symboles "$e$" correspondant à la séquence $seq_{in}^{l}$ (resp. $seq_{p}^{l}$).*

La figure 3.3 illustre la définition 28 ainsi que la définition 29. Elle précise aussi graphiquement les notions de date de sous-activation et de sous-durée d'exécution d'une otâche périodique.

Pour chaque otâche périodique $\tau_{per} = \zeta_{(r,\beta)}\tau_{\beta}^{\infty} \in \mathcal{P}$ de période $T \in \mathbb{N}$, illustrée par la figure 3.3, nous nous intéressons maintenant à sa partie périodique puisque seul le motif se répète *ad vitam aeternam* par définition.

Nous introduisons la notion de *facteur d'utilisation permanent sans coût de préemption* de $\tau_{per}$ relativement à son motif et nous la définissons comme suit

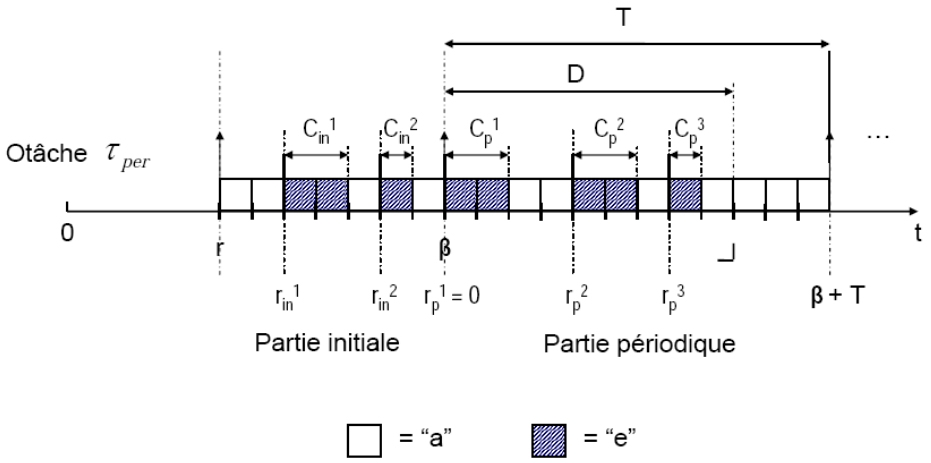


FIG. 3.3 – Illustration des dates de sous-activation et des sous-durées d'exécution d'une otâche périodique.

**Définition 30** *Le* facteur d'utilisation permanent du processeur sans coût de préemption $U_{per}$ *d'une otâche periodique* $\tau_{per} = \zeta_{(r,\beta)}\tau_\beta^\infty$ *de période* $T$ *est donné par la relation*

$$U_{per} = \frac{C_p}{T} \ \textit{avec} \ C_p = \sum_{l=1}^{n_p} C_p^l \tag{3.6}$$

*Dans la relation 3.6,* $C_p$ *et* $n_p$ *désignent respectivement la somme des sous-durées d'exécution de rang* $l$ *et le nombre de séquences du symbole "*$e$*" dans le motif de* $\tau_{per}$*.*

**Définition 31** *Le* facteur d'utilisation permanent du processeur sans coût de préemption *d'un système* $\Gamma_n$ *constitué de* $n$ *otâches périodiques* $\tau_i$ *de période* $T_i$ *est donné par la relation*

$$U_n = \sum_{i=1}^{n} \frac{C_{p_i}}{T_i} \ \textit{avec} \ C_{p_i} = \sum_{l=1}^{n_{p_i}} C_{p_i}^l \tag{3.7}$$

*Dans la relation 3.7,* $C_{p_i}$ *et* $n_{p_i}$ *désignent respectivement la somme des sous-durées d'exécution de rang* $l$ *et le nombre de séquences du symbole "*$e$*" dans le motif de la otâche* $\tau_i \in \Gamma_n$*.*

**Remarque 11** *Si* $n_{p_i} = 1$ *pour toutes les otâches (*$i = 1, \cdots, n$*) de* $\Gamma_n$ *alors la définition du facteur d'utilisation permanent du processeur sans coût de préemption ci-dessus est équivalente à celle du facteur d'utilisation classique du processeur définit dans le chapitre précédent.*

Une conséquence immédiate de la définition 31 est que si le *facteur d'utilisation permanent sans coût de préemption* d'un système de $n$ otâches périodiques est strictement supérieur à $1$ alors aucun algorithme à priorités fixes ne pourra fournir un *ordonnancement valide*. Nous reviendrons plus en détail sur ces deux notions pour les otâches plus tard dans cette thèse. Cette assertion se justifie par le fait que globalement le nombre de symboles "$e$" correspondant à un système d'otâches ne devrait jamais être supérieur au nombre d'unités de temps dans la phase permanente du processeur puisque le but de l'approche proposée ici est d'exploiter les symboles "$a$" appartenant à une otâche donnée avec les "$e$" d'une autre otâche en remplaçant les premiers par les seconds. Les principes généraux de ce processus seront expliqués en détail dans la section 3.5. Avant d'aller plus loin dans notre présentation, il serait intéressant de montrer le lien existant entre tâche périodique et une otâche périodique.

### 3.3.1 Correspondance entre tâches périodiques et otâches périodiques

Comme c'est le cas pour les systèmes d'otâches périodiques, nous rappelons que les systèmes temps réel auxquels nous nous intéressons dans cette thèse sont composés de tâches qui sont à la fois infinies (constituées d'une infinité d'instances) et périodiques. L'étude de tels systèmes à l'aide de systèmes d'otâches périodiques nécessite que chaque tâche périodique du système temps réel soit descriptible sous la forme d'une otâche périodique de façon univoque, c'est-à-dire qu'à deux tâches périodiques distinctes, doivent correspondre deux otâches périodiques distinctes. Dans toute la suite de notre présentation, nous choisissons de construire une telle correspondance en décrivant comment les otâches peuvent être *générées* à partir des caractérisques temporelles des tâches temps réel et des opérations sur des otâches plus simples.

Soit $\Gamma_n = \{\tau_1, \tau_2, \cdots, \tau_n\}$ un système de $n$ tâches temps réel périodiques où la tâche $\tau_i = (r_i^1, C_i, D_i, T_i)$, $i = 1, 2, \cdots, n$ et $C_i \leq D_i \leq T_i$. En nous basant sur les caractéristiques d'une tâche périodique et l'équation 2.9 du chapitre 2, $r_i^1$ est la date de première activation, $C_i$ le WCET intégrant la durée d'exécution de la routine de changement de contexte au début de l'exécution de la tâche $\tau_i$ et la durée d'exécution de la routine à la fin de l'exécution de la tâche $\tau_i$, $D_i$ l'échéance relative et $T_i$ la période de $\tau_i$. Au système $\Gamma_n$, nous faisons correspondre le système d'otâches périodiques $O\Gamma_n = \{o\tau_1, o\tau_2, \cdots, o\tau_n\}$ où $o\tau_i$, $i = 1, 2, \cdots, n$ est la otâche périodique correspondante à la tâche périodique $\tau_i$. Nous rappelons que le générateur $\Sigma$ que nous considérons est constitué de deux symboles "$a$" et "$e$" : $\Sigma = \{a, e\}$ où nous identifions le symbole "$a$" à une unité de temps disponible et le symbole "$e$" à une unité de temps exécutée ou exécutable.

Puisque notre objectif principal est l'analyse d'ordonnançabilité d'un système de tâches périodiques, nous allons considérer le système d'otâches correspondant. Pour ce système, nous allons composer les otâches à l'aide d'une *opération binaire d'ordonnan-*

*cement associative* que nous noterons $\oplus$ afin d'obtenir une otâche permettant de décider de l'ordonnançabilité du système. Lorsque nous écrirons $x \oplus y$ où $x$ et $y$ sont deux otâches périodiques, cela signifiera par convention que le premier opérande (c'est-à-dire, la otâche $x$) a une priorité supérieure à celle du second opérande (c'est-à-dire, la otâche $y$), par conséquent l'opération $\oplus$ n'est pas *commutative*, c'est-à-dire $x \oplus y \neq y \oplus x$. Dans l'expression $x \oplus y$, les symboles "$e$" de la otâche $x$ sont dits *exécutés* et ceux de la otâche $y$ sont dits *exécutables*. L'idée intuitive que nous proposons pour effectuer l'opération sera donc d'exploiter les symboles "$a$" de la otâche $x$ avec les "$e$" de la otâche $y$. Les dates de sous-activation et les sous-durées d'exécution donnent respectivement les positions et les longueurs des séquences de symboles "$e$" de la otâche $x$ (exécutés) et celles de la otâche $y$ (exécutables). La date de sous-activation de chaque séquence de symboles "$e$" exécutables donne la date de leurs exécutions éventuelles au plus tôt. Ces exécutions sont éventuelles parce que lors de la composition des otâches $x$ et $y$ à l'aide de l'opération d'ordonnancement $\oplus$, les symboles "$e$" de $y$ ne pourront pas modifier ni la position, ni la longueur des symboles "$e$" de la otâche $x$ à cause du fait que celle-ci est plus prioritaire. Nous insistons une fois de plus sur le fait que chaque fois que nous parlons d'otâches ou de systèmes d'otâches, nous les considèrons sur le générateur $\Sigma = \{a, e\}$.

A la tâche temps réel périodique $\tau_i$, nous faisons correspondre la otâche périodique $o\tau_i$ définie par la relation 3.8.

$$o\tau_i = \left\{ \underbrace{\overbrace{e, e, \cdots, e}^{C_i}, a, a, a, \cdots, a}_{D_i} \underbrace{, a, \cdots, a, a}_{\lrcorner} \right\}_{r_i^1}^{\infty}, \quad i = 1, 2, \cdots, n. \tag{3.8}$$

où $r_i^1$ désigne que le motif de la otâche $o\tau_i$ débute à la date $r_i^1$, correspondant à la date de première activation de la tâche $\tau_i$. Il en découle que la otâche $o\tau_i$ est une otâche particulière sur $\Sigma$ car elle est *canonique*, constituée d'une partie périodique et n'a pas de partie initiale non triviale : sa partie initiale est égale à la otâche nulle $\Lambda$. De plus elle est *régulière*, c'est-à-dire que son motif contient une seule et unique séquence de $C_i$ symboles "$e$" suivie d'une seule et unique séquence de $T_i - C_i$ symboles "$a$". Les $D_i$ premiers symboles du motif (matérialisés par le caractère : $\lrcorner$ ) représentent l'échéance relative de la otâche $o\tau_i$. La valeur de $D_i$ délimite l'intervalle avant lequel il faudrait absolument avoir exécuté les $C_i$ symboles "$e$" de $o\tau_i$ lors de l'application de l'opération d'ordonnancement $\oplus$. Dans l'égalité 3.8 chaque répétition du motif à partir de la date $r_i^1$ correspond à une instance de la tâche $\tau_i$. Le motif de rang $k$ débutant à la date $r_i^k =$

$r_i^1 + (k-1)T_i$ correspond à la $k^{ieme}$ instance. La figure 3.4 illustre la correspondante entre une tâche périodique et une otâche périodique.

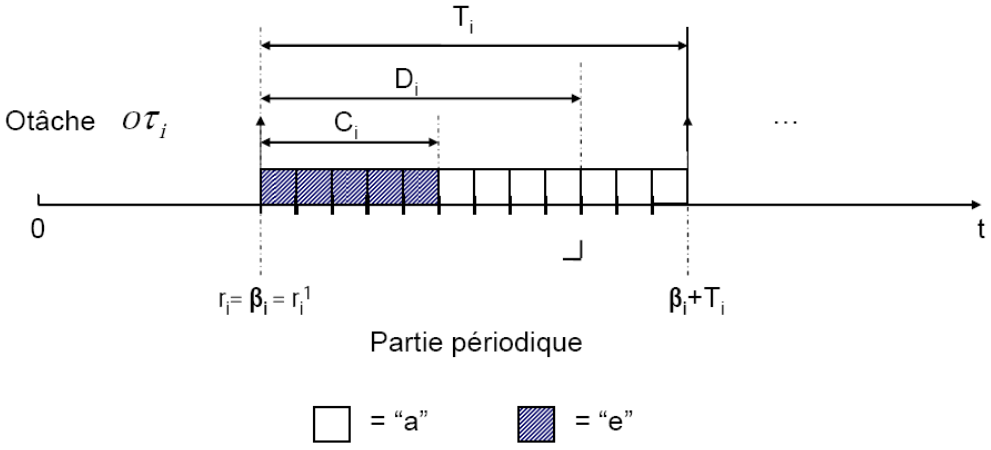


FIG. 3.4 – Correspondance entre une tâche périodique et une otâche périodique.

Lorsque l'échéance relative d'une tâche périodique est égale à sa période, nous ne l'a représenterons pas dans la otâche périodique correspondante car dans ce cas la tâche est dite à échéance sur activation et sa valeur ne présente aucune ambiguïté, elle est égale au cardinal du motif de la otâche périodique. A partir de maintenant, chaque fois que nous parlerons de la tâche périodique $\tau_i$, nous sous-entendrons que la otâche périodique correspondante $o\tau_i$ est construite grâce à la relation 3.8 et ainsi nous pourrons directement la manipuler sans plus avoir à rappeler le principe de sa construction. Ci-après, nous donnons quelques exemples précisant l'idée de la correspondance entre tâches périodiques et otâches périodiques sur le générateur $\Sigma = \{a, e\}$. Soient $\tau_1 = (0, 2, 5, 5)$ et $\tau_2 = (4, 3, 6, 8)$ deux tâches périodiques, alors grâce à la relation 3.8, les otâches périodiques correspondantes sont respectivement données par $o\tau_1 = \{e, e, a, a, a\}_0^\infty$ et $o\tau_2 = \{e, e, e, a, a, a \llcorner a, a\}_4^\infty$.

**Définition 32** *L'*opération binaire d'ordonnancement $\oplus$ *sur* $\Sigma$ *est une* application *de* $\Sigma^* \times \Sigma^*$ *dans* $\Sigma^*$ *:*

$$\oplus : \Sigma^* \times \Sigma^* \longrightarrow \Sigma^*$$

Dans la définition précédente, l'opération $\oplus$ est une application puisqu'on peut supposer sans nuire à la généralité que si $x \oplus y$ n'est pas définie, alors $x \oplus y = \Lambda$. Maintenant, puisque $\mathcal{P}$ désigne le sous ensemble de $\Sigma^*$ constitué de toutes les otâches périodiques, alors l'opération d'ordonnancement des otâches périodiques correspond à la restriction de l'opération $\oplus$ aux éléments de $\mathcal{P}$. Cette restriction correspond à une opération interne

dans $\mathcal{P}$ lorsqu'elle est définie, c'est-à-dire lorsque le résultat de l'opération est non trivial : différent de la otâche nulle $\Lambda$. Cette assertion se traduit par la relation 3.9 lorsque $x \oplus y \neq \Lambda$.

$$\text{Si } x \in \mathcal{P} \text{ et } y \in \mathcal{P} \text{ alors } x \oplus y \in \mathcal{P} \tag{3.9}$$

La relation 3.9 sera montrée plus loin dans ce chapitre à travers le théorème 7 et lorsque nous aurons définit les étapes à suivre pour effectuer l'opération $\oplus$. L'idée intuitive derrière cette relation consiste à dire que tout résultat *non trivial* de la composition de deux otâches périodiques est une otâche périodique. Puisque tout système d'otâches périodiques $O\Gamma_n = \{o\tau_1, o\tau_2, \cdots, o\tau_n\}$ est inclut dans le sous ensemble $\mathcal{P}$ de $\Sigma^*$, alors l'opération d'ordonnancement du système $O\Gamma_n$ correspond à la restriction de l'opération $\oplus$ dans $\mathcal{P}$ aux éléments de $O\Gamma_n \cup \{\Lambda\}$. Cette fois, il est important de noter que la restriction de l'opération $\oplus$ à l'ensemble $O\Gamma_n \cup \{\Lambda\}$ est *externe*. En effet nous pouvons avoir, et d'ailleurs cela sera très souvent le cas, $x \in O\Gamma_n$, $y \in O\Gamma_n$ et $x \oplus y \notin O\Gamma_n \cup \{\Lambda\}$. Par conséquent, l'opération d'ordonnancement de $O\Gamma_n$ est alors une application *surjective* de $\mathcal{P} \times (O\Gamma_n \cup \{\Lambda\})$ dans $\mathcal{P}$.

Pour tout système $O\Gamma_n$ constitué de $n \geq 1$ otâches périodiques, l'opération d'ordonnancement $\oplus$ étant une opération binaire, elle sera utilisée autant de fois qu'il y a d'otâches dans le système $O\Gamma_n$ afin de garantir ou non l'ordonnançabilité de celui-ci relativement à un algorithme de choix des priorités des otâches tel que *Rate Monotonic*, *Deadline Monotonic*, etc. De plus, comme l'opération $\oplus$ n'est pas commutative par convention, l'ordre d'application récursive de ces opérations sera fixé par le choix des priorités des otâches. Ils seront toujours appliqués de la otâche la plus prioritaire vers la otâche la moins prioritaire en produisant à chaque étape un résultat intermédiaire qui correspondra à la otâche la plus prioritaire, c'est-à-dire le premier opérande, de l'application suivante de l'opération $\oplus$. Cette assertion est soulignée par la propriété 9.

**Propriété 9** *Soit le système d'otâches $O\Gamma_n = \{o\tau_1, o\tau_2, \cdots, o\tau_n\}$ où les otâches sont rangées suivant les priorités décroissantes relativement à la politique d'ordonnancement* Ordo. *Si $\mathcal{R}_n$ représente le résultat de l'ordonnancement de $O\Gamma_n$, alors $\mathcal{R}_n$ est obtenu par itération successive de la suite*

$$\begin{cases} \mathcal{R}_1 = \Lambda \oplus o\tau_1 = o\tau_1 \\ \mathcal{R}_i = \mathcal{R}_{i-1} \oplus o\tau_i, \quad 2 \leq i \leq n \end{cases}$$

Grâce à la propriété 9, on avons :

$$\mathcal{R}_n = (((\Lambda \oplus o\tau_1) \oplus o\tau_2) \oplus \cdots \oplus o\tau_{n-1}) \oplus o\tau_n \tag{3.10}$$

que nous noterons également :

$$\mathcal{R}_n = \bigoplus_{i=1}^{n} o\tau_i$$

**Remarque 12** *Etant donné un système d'otâches, le choix des priorités des otâches détermine un ordre total d'application des opérations $\oplus$ pour ce système.*

Nous expliquons en détail dans la section 3.5 comment obtenir le résultat de la composition de deux otâches périodiques grâce à l'opération $\oplus$.

**Définition 33** *Soit un système de $n$ otâches périodiques $O\Gamma_n = \{o\tau_1, o\tau_2, \cdots, o\tau_n\}$ où les otâches sont rangées suivant les priorités décroissantes relativement à une politique d'ordonnancement* Ordo *donnée.*
*La otâche $o\tau_i, 1 \leq i \leq n$ est dite* ordonnançable *relativement à la politique* Ordo *si et seulement si*

$$\mathcal{R}_i \neq \Lambda \tag{3.11}$$

*Le système $O\Gamma_n$ est dit* ordonnançable *relativement à la politique* Ordo *si et seulement si toutes ses otâches sont ordonnançables, c'est-à-dire*

$$\mathcal{R}_i \neq \Lambda \quad \forall i \in \{1, 2, \cdots, n\} \tag{3.12}$$

*Si l'équation 3.12 n'est pas satisfaite alors le système $O\Gamma_n$ est dit* non ordonnançable.

### 3.3.2 Décomposition d'une otâche périodique

Etant donné le générateur $\Sigma = \{a, e\}$, nous rappelons que nous identifions le symbole "$a$" à une unité de temps libre et le symbole "$e$" à une unité de temps exécutée ou exécutable suivant qu'il appartient respectivement au premier opérande ou au second opérande de l'opération $\oplus$. Le but de cette sous-section est de présenter la décomposition d'une otâche périodique quelconque.

Soit $\tau_{per} = \zeta_{(r,\beta)}\tau_\beta^\infty \in \mathcal{P}$ une otâche périodique sur $\Sigma$ de période $T = |\tau|$ et d'échéance relative $D$. Soit $n_p$ (resp. $r_p^l$, $l \in \{1, \cdots, n_p\}$) le nombre de séquences du symbole "$e$" dans le motif de la otâche périodique $\tau_{per}$ (resp. l'image de la séquence $seq_p^l$, $l \in \{1, \cdots, n_p\}$ par l'application $r_p$). Nous rappelons que $r_p^l$ correspond à l'indice tel que l'instant $r_p^{l,1} = \beta + r_p^l$ est la date de première sous-activation de rang $l$ de la séquence $seq_p^l$). La otâche $\tau_{per}$ peut être *décomposée* en $n_p$ otâches périodiques canoniques de période $T$ dont l'échéance relative de la partie périodique de chaque otâche est respectivement donnée par :

$$D_{per,l} = D - r_p^l, \;\; l \in \{1, \cdots, n_p\} \tag{3.13}$$

Dans ce cas, nous dirons que la otâche périodique $\tau_{per}$ est $n_p$*-décomposable*. Si la otâche périodique $\tau_{per}$ est $n_p$-décomposable, alors nous obtenons la décomposition de $\tau_{per}$ à l'aide de l'application $\pi$ de $\mathcal{P}$ vers $\mathcal{P}^{n_p}$ suivante :

$$\begin{array}{rcl} \pi : \mathcal{P} & \longrightarrow & \mathcal{P}^{n_p} \\ \tau_{per} & \longmapsto & \tau_{per}^{\odot} = (x_1; x_2; \cdots; x_{n_p}) \end{array} \tag{3.14}$$

où $x_l$, $l \in \{1, \cdots, n_p\}$ est la otâche périodique canonique régulière ayant les caractéristiques suivantes : la partie périodique débute à la date $\beta + r_p^l$, la période vaut $T$ et l'échéance relative de la partie périodique vaut $D_{per,l} = D - r_p^l$, $l \in \{1, \cdots, n_p\}$. La otâche $x_l$ est régulière, par conséquent elle contient une seule séquence de symbole "$e$" dont le cardinal de la sous otâche correspondante vaut $C_p^l$, c'est la sous-durée d'exécution de rang $l$ de la partie périodique de $\tau_{per}$. En guise d'exemple, la figure 3.6 illustre graphiquement la décomposition de la otâche périodique de la figure 3.5.

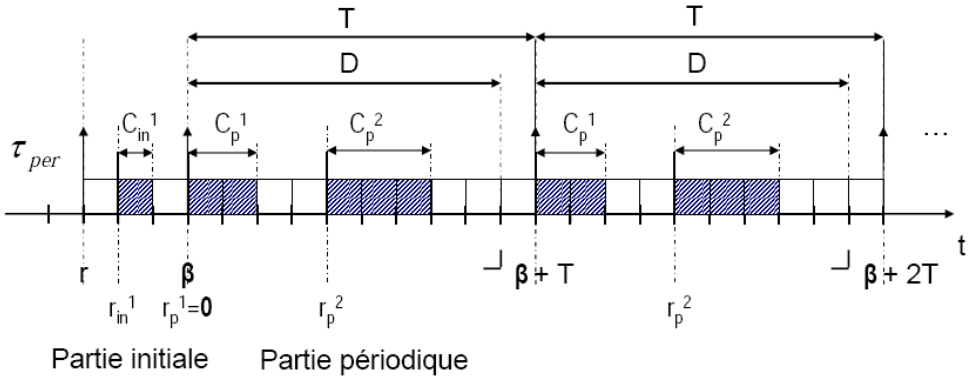


FIG. 3.5 – Illustration d'une tâche périodique à décomposer.

**Théorème 6** *Soit $\Sigma$ un générateur et $o\tau$ une otâche sur $\Sigma$. Si la otâche $o\tau$ est périodique de période $T$ finie alors elle peut toujours être décomposée en une suite finie d'otâches périodiques canoniques de même période, $T$.*

**Preuve** :

La preuve du théorème 6 découle directement du fait que la période de la otâche $o\tau$ est finie et de la propriété de décomposition d'une otâche. ■

Afin de donner un exemple numérique de la décomposition d'une otâche périodique, considérons la otâche périodique $\tau_0$ donnée par l'égalité

$$\tau_0 = \{e, a, a\}_{(r,\beta)}\{e, e, a, a, e, e, e, a \lrcorner a, a\}_{\beta}^{\infty}$$

La période de $\tau_0$ vaut $T = |\{e, e, a, a, e, e, e, a, a, a\}| = 10$ et l'échéance relative de la partie périodique vaut $D = |\{e, e, a, a, e, e, e, a\}| = 8$. Grâce aux notions introduites précédemment, la otâche périodique $\tau_0$ est 2-décomposable car le motif de sa partie périodique contient deux séquences de symboles "$e$" avec $r_p^1 = 0$ et $r_p^2 = 4$. Les sous-durées d'exécution de rang 1 et 2 de la partie périodique de $\tau_0$ valent respectivement

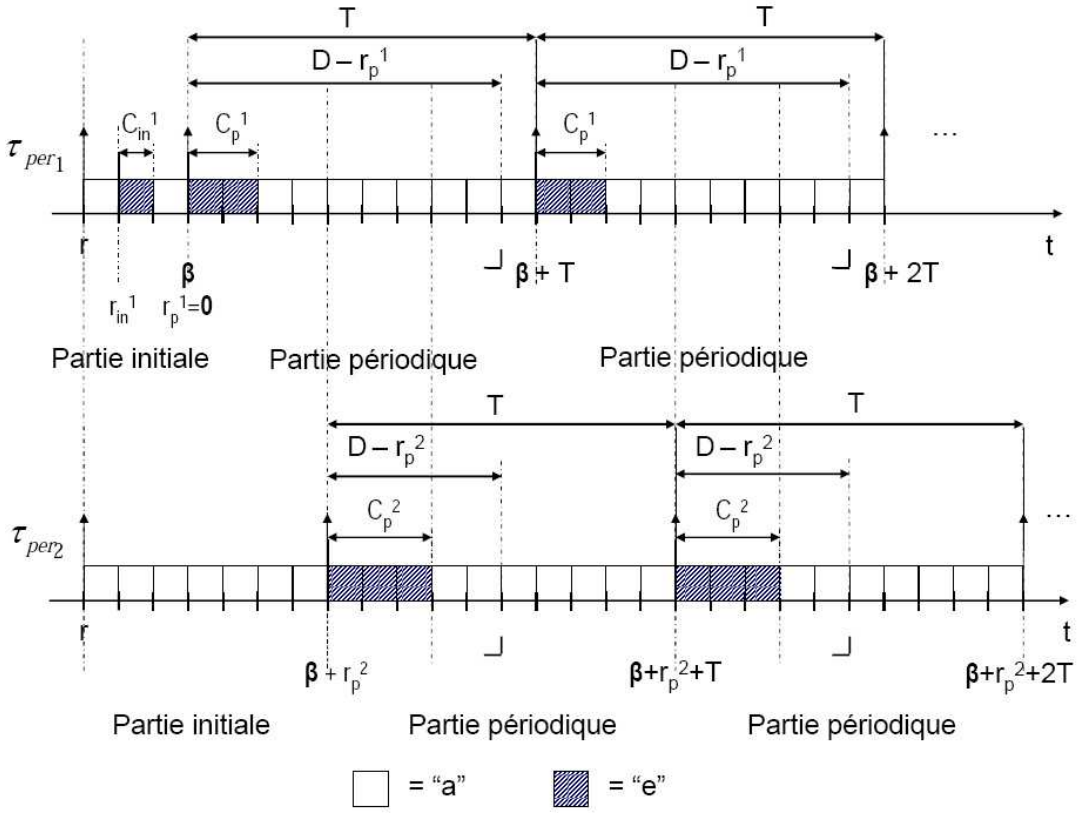


FIG. 3.6 – Illustration de la décomposition d'une otâche périodique.

$C_p^1 = 2$ et $C_p^2 = 3$. Ainsi, nous avons :

$$\begin{aligned}
\tau_0^{\odot} = \pi(\tau_0) &= \pi\left(\{e,a,a\}_{(r,\beta)}\{e,e,a,a,e,e,e,a\llcorner a,a\}_{\beta}^{\infty}\right)\\
&= \left(\{e,a,a\}_{(r,\beta)}\{e,e,a,a,a,a,a,a\llcorner a,a\}_{\beta}^{\infty};\{a,a,a,a,e,e,e,a\llcorner a,a\}_{\beta}^{\infty}\right)\\
&= \left(\{e,a,a\}_{(r,\beta+r_p^1)}\{e,e,a,a,a,a,a,a\llcorner a,a\}_{\beta+r_p^1}^{\infty};\right.\\
&\qquad \left.\{a,a,a,a\}_{(\beta,\beta+r_p^2)}\{e,e,e,a\llcorner a,a,a,a,a,a\}_{\beta+r_p^2}^{\infty}\right)\\
&= \left(\tau_{per,1};\tau_{per,2}\right)
\end{aligned}$$

où la otâche périodique $\tau_{per,1}$ vaut $\tau_{per,1} = \{e,a,a\}_{(r,\beta)}\{e,e,a,a,a,a,a,a\llcorner\, a,a\}_{\beta}^{\infty}$ et la otâche périodique $\tau_{per,2}$ vaut $\tau_{per,2} = \{a,a,a,a\}_{(\beta,\beta+4)}\{e,e,e,a\llcorner\, a,a,a,a,a,a\}_{\beta+4}^{\infty}$. Les otâches périodiques $\tau_{per,1}$ et $\tau_{per,2}$ sont bien canoniques régulières grâce entre autre à l'application du théorème 5. La période vaut $T = 10$ pour les deux otâches cano-

niques régulières et les échéances relatives des parties périodiques de ces deux otâches périodiques sont respectivement données par

$$D_{per,1} = D - r_p^1 = 8 \text{ et } D_{per,2} = D - r_p^2 = 4$$

**Remarque 13** *Une otâche périodique* $1$*-décomposable sur le générateur* $\{a, e\}$ *est une otâche dont le motif contient une seule séquence de symbole "*$e$*". Nous qualifierons de telles otâches de* régulières*. Une otâche périodique* $k$*-décomposable avec* $k > 1$ *sera dite* irrégulière*.*

Ainsi la otâche périodique contenant une seule séquence de symbole "$e$" donnée par $\tau_{per,2} = \{a, a, a, a, e, e, e, a\_\rfloor a, a\}_\beta^\infty = \{a, a, a, a\}_{(\beta,\beta+4)}\{e, e, e, a\_\rfloor a, a, a, a, a, a\}_{\beta+4}^\infty$ est une otâche régulière tandis que $\tau_0$ est irrégulière puisqu'elle est $2$-décomposable. Grâce au principe de décomposition d'une otâche périodique, toute otâche périodique irrégulière est décomposable en une suite finie d'otâches régulières. Cette décomposition nous permettra de raisonner par la suite, sans nuire à la généralité, sur des otâches périodiques canoniques régulières pour la composition des otâches périodiques avec l'opération $\oplus$.

## 3.4 Intervalle d'analyse d'un système d'otâches périodiques

Chaque otâche d'un système d'otâches périodiques étant par définition de cardinal infini, il est nécessaire de définir un intervalle *fini* pour permettre l'analyse complète d'un système d'otâches périodiques, c'est le but de cette section. Dans le système d'otâches périodiques $O\Gamma_n$ correspondant au système de tâches périodiques $\Gamma_n$, la priorité de la otâche $o\tau_i$, $i = 1, \cdots, n$ est la même que celle de la tâche $\tau_i$ qui lui est associée. De plus les otâches héritent des contraintes auxquelles sont soumises les tâches le cas échéant (échéances, dépendances, périodicités strictes, latences, etc). Nous définirons en détail l'équivalence de chaque terme plus loin dans cette thèse (voir chapitre 5). Tant que nous n'en faisons pas mention explicitement, nous supposons que les tâches (resp. les otâches) sont indépendantes et donc les otâches correspondantes le sont aussi. Nous supposons de plus sauf mention explicite du contraire que la seule contrainte à laquelle les tâches et les otâches sont soumises est le respect des échéances relatives.

L'opération $\oplus$ appliquée à deux otâches périodiques $x$ et $y$, telle que $x \oplus y$ est le résultat obtenu, consiste à exploiter les symboles "$a$" de la otâche $x$ la plus prioritaire en remplaçant certains par les symboles "$e$" de la otâche $y$ tout en respectant les échéances relatives. Dans le cas d'un système d'otâches on applique autant de fois l'opération $\oplus$ qu'il y a d'otâches dans le système conformément à la priorité des otâches, bien sûr tout en respectant les échéances relatives. On traite de cette manière un problème classique d'ordonnancement.

Dans la suite de la thèse nous avons besoin de l'extension du théorème 1 du chapitre 2 aux systèmes d'otâches périodiques. Cette extension est obtenue grâce au théorème 7.

**Théorème 7** *Soit $O\Gamma_n = \{o\tau_1, o\tau_2, \cdots, o\tau_n\}$ un système de $n$ otâches périodiques tel que $o\tau_i = \zeta_{i(r_i,\beta_i)}(\tau_{0,i})_{\beta_i}^{\infty}$, avec $i = 1, \cdots, n$, rangées par priorités décroissantes relativement à une politique d'ordonnancement, c'est-à-dire $i < j$ implique que la priorité de $o\tau_i$ est supérieure à la priorité de $o\tau_j$. Soit $(s_i^{'})_{i\in\mathbb{N}^*}$ la suite définie par*

$$\begin{cases} s_1^{'} = \beta_1 \\ \\ s_i^{'} = \beta_i + \left\lceil \dfrac{(s_{i-1} - \beta_i)^+}{T_i} \right\rceil \cdot T_i, \quad 2 \leq i \leq n \end{cases}$$

*S'il existe un ordonnancement valide de $O\Gamma_n$ jusqu'à la date $s_n^{'} + H_n$ où $H_n = ppcm\{T_i \mid i = 1, \cdots, n\}$, $T_i = |\tau_{0,i}|$ et $x^+ = max(x, 0)$, alors cet ordonnancement est valide et périodique de période $H_n$ à partir de $s_n^{'}$.*

**Preuve** :

La preuve du théorème 7 est similaire à celle du théorème 1 du chapitre 2. En effet l'échéance relative de chaque otâche est inférieure ou égale à sa période pour sa partie périodique. De plus la preuve n'implique que les dates de début de la partie périodique (ce qui correspond aux dates de premières activations pour les tâches) et les cardinaux des motifs des otâches (ce qui correspond aux périodes pour les tâches). ■

Une conséquence directe du théorème 7 consiste à dire que dans le cas d'un ordonnancement valide, la otâche $\mathcal{R}_i$ (résultat de l'ordonnancement des $i$ otâches périodiques les plus prioritaires) est périodique de période $T_{\mathcal{R}_i} = ppcm\{T_j \mid j = 1, \cdots, i\}$ à partir de $s_i^{'}$. Ainsi l'intervalle précédent $s_i^{'}$ contient nécessairement la *phase transitoire*, correspondant à la partie initiale de $\mathcal{R}_i$, et l'intervalle démarrant à la date $s_i^{'}$ et de longueur $H_i$ est isomorphe à la *phase permanente de niveau $i$*, correspondant à la partie périodique de $\mathcal{R}_i$. Puisque la phase transitoire est toujours finie grâce à l'existence de la phase permanente qui se répète à partir d'un certain rang, la date de début *minimale* ou *au plus tôt* de la phase permanente sera déduite par application du théorème 5.

Pour un système de $n$ otâches périodiques pour lequel il existe un ordonnancement valide, puisque la phase permanente se répète indéfiniment, nous la représenterons graphiquement à l'aide d'un *disque circulaire orienté* appelé le *Dameid* ayant un instant de référence $t_0 = 0$ correspondant à l'instant de référence de l'axe temporel définit précédemment pour les otâches. Le sens positif du *Dameid* correspond au sens trigonométrique, c'est-à-dire au sens inverse (ou contraire) des aiguilles d'une montre. La circonférence du *Dameid* vaut $H_n = ppcm\{T_i \mid i = 1, \cdots, n\}$ où $T_i$ désigne la période de la $i^{ieme}$ otâche et $n$ désigne le nombre d'otâches dans le système. Dans le *Dameid*, les

différentes dates de début de la partie périodique de chaque otâche $o\tau_i = \zeta_{i(r_i,\beta_i)}(\tau_{0,i})_{\beta_i}^{\infty}$ de période $T_i = |\tau_{0,i}|$ sont déterminées sans ambiguïté par la valeur de $\beta_i$ de la otâche relativement à la date de référence $t_0$ et le rapport $\frac{H_n}{T_i}$. En guise d'exemple, la figure 3.7 illustre pour un système constitué de $4$ otâches les dates de début de la partie périodique d'une otâche périodique dont la partie initiale de chaque otâche est égale à $\Lambda$. Dans cette figure, le date de début la partie périodique de la otâche $t_1$ est $-2$ tandis que celle de la otâche $t_3$ est $0$.

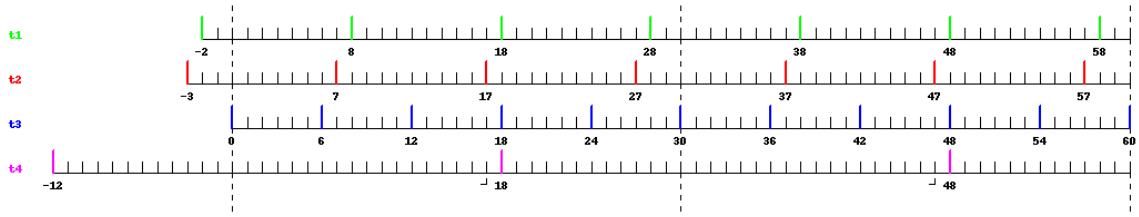


FIG. 3.7 – Illustration des dates de début de la partie périodique d'une otâche périodique.

La figure 3.8 précise notre idée de la construction du *Dameid*. Cette figure illustre pour le même système constitué de $4$ otâches ci-dessus (voir figure 3.7) la correspondance des dates de début de la partie périodique de chaque otâche dans le *Dameid* relativement à la phase permanente. L'intuition principale derrière cette nouvelle représentation est de réduire au mieux l'intervalle d'analyse d'un système d'otâches périodiques donné et donc d'un système de tâches périodiques.

## 3.5 Opération $\oplus$ d'ordonnancement d'un système d'otâches périodiques sans coût de préemption

Maintenant que nous avons défini ce qu'est une otâche périodique et que nous avons montré quelques propriétés vérifiées par celle-ci, le but de cette section est de décrire en détail l'opération $\oplus$ d'ordonnancement d'un système d'otâches périodiques sans coût de préemption.

Soient $\Gamma_n = \{\tau_1, \tau_2, \cdots, \tau_n\}$ un système de $n$ tâches temps réel périodiques indépendantes où $\tau_i = (r_i^1, C_i, D_i, T_i),\ i = 1, 2, \cdots, n$ et $O\Gamma_n = \{o\tau_1, o\tau_2, \cdots, o\tau_n\}$ le système d'otâches périodiques correspondant. Puisque dans le cadre de cette thèse les politiques d'ordonnancement que nous considérons sont toujours à priorités fixes, alors nous rappelons que l'opération d'ordonnancement $\oplus$ que nous allons définir dans cette section est non commutative (c'est-à-dire $x \oplus y \neq x \oplus y,\ \forall\ x, y$ deux otâches périodiques) car chaque otâche, lorsqu'elle décrit une tâche temps réel, hérite de la priorité de

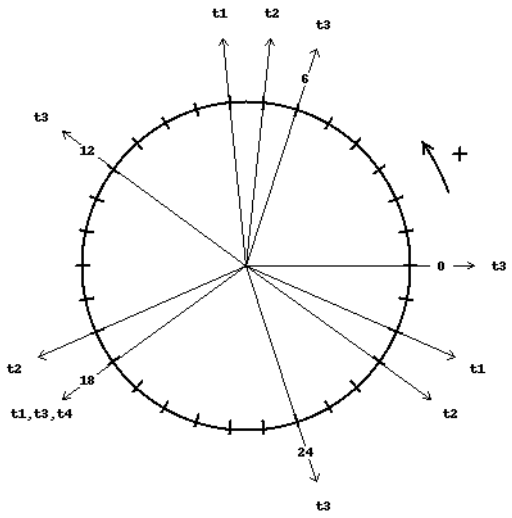


FIG. 3.8 – Illustration de la correspondance des dates de début de la partie périodique d'une otâche périodique dans le *Dameid*.

celle-ci. De même, nous rappelons que par convention, l'expression $x \oplus y$ signifie que le premier opérande (c'est-à-dire, la otâche $x$) a une priorité supérieure à celle du second opérande (c'est-à-dire, la otâche $y$). Notons qu'à tout moment, l'opération $\oplus$ implique deux otâches périodiques $o\tau_i = \zeta_{i(r_i,\beta_i)}(\tau_{0,i})_{\beta_i}^{\infty}$ et $o\tau_j = \zeta_{j(r_j,\beta_j)}(\tau_{0,j})_{\beta_j}^{\infty}$ de priorités adjacentes (c'est-à-dire aucune otâche appartenant au système d'otâches considéré n'a une priorité comprise entre celle de $o\tau_i$ et celle de $o\tau_j$). Ainsi nous pouvons définir le déphasage entre les otâches périodiques $o\tau_i$ et $o\tau_j$ par $\gamma_{ij} = r_j - r_i$ comme c'est le cas pour les tâches temps réel correspondantes.

Suivant le signe de l'entier $\gamma_{ij} \in \mathbb{Z}$, l'opération d'ordonnancement $\oplus$ a une définition différente. En effet, si $\gamma_{ij}$ est positif alors $r_j \geq r_i$ et donc l'ordonnancement de la première unité de temps de $o\tau_j$ (c'est-à-dire, son premier symbole "$e$") nécessite d'attendre au moins $\gamma_{ij}$ unités de temps après la date de début de la otâche $o\tau_i$. Par contre, si $\gamma_{ij}$ est strictement négatif alors $r_j < r_i$. Dans ce cas, l'ordonnancement de la première unité de temps de $o\tau_j$ peut avoir lieu immédiatement après sa date de début $r_j$. Pour illustrer nos propos, nous allons donner la description détaillée de l'opération $\oplus$. Cette description comporte trois grandes procédures. La première procédure consiste à *normaliser* les deux opérandes, c'est-à-dire référencer les deux opérandes par rapport à la même origine temporelle. La deuxième procédure consiste à décomposer le second opérande en une suite finie d'otâches canoniques régulières grâce à l'application $\pi$ et la troisième procédure consiste à effectuer l'opération d'ordonnancement proprement dite, c'est-à-dire remplacer les unités de temps disponibles "$a$" laissées par la otâche la plus prioritaire $o\tau_i$

par les unités de temps exécutable "$e$" de la otâche moins prioritaire $o\tau_j$. Chacune de ces trois procédures est expliquée dans la suite de cette thèse (voir procédure 1, procédure 2 et procédure 3). Nous rappelons que les unités de temps exécutables (les symboles "$e$" appartennant de la otâche $o\tau_j$) ne peuvent pas modifier ni la position ni le nombre des symboles "$e$" de la otâche $o\tau_i$ car le premier opérande est toujours plus prioritaire que le second par convention. De plus, notons que pour chaque otâche $o\tau_i = \zeta_{i(r_i,\beta_i)}(\tau_{0,i})^{\infty}_{\beta_i}$, nous avons l'inégalité 3.15 qui est toujours satisfaite.

$$r_i \leq \beta_i \leq s_i^{'} \tag{3.15}$$

Comme nous l'avons déjà mentionné, la correspondance entre les tâches périodiques et otâches périodiques est donnée par la relation suivante :

$$\tau_i = (r_i^1, C_i, D_i, T_i) \equiv \left\{ \underbrace{\overbrace{e, e, \cdots, e}^{C_i}, a, a, a, \cdots, a}_{D_i} \underbrace{, a, \cdots, a, a}_{T_i} \right\}^{\infty}_{r_i^1} = o\tau_i, i = 1, 2, \cdots, n.$$

Soient deux otâches périodiques $o\tau_i = \zeta_{i(r_i,\beta_i)}(\tau_{0,i})^{\infty}_{\beta_i}$ et $o\tau_j = \zeta_{j(r_j,\beta_j)}(\tau_{0,j})^{\infty}_{\beta_j}$ de priorités adjacentes et ayant chacune les caractérisques illustrées par la figure 3.3. Nous supposons que la otâche $o\tau_i$ a une priorité supérieure et adjacente à celle de la otâche $o\tau_j$. Maintenant, nous avons tous les éléments pour formellement définir la otâche $\mathcal{R}_j$ résultat de l'opération $o\tau_i \oplus o\tau_j$, c'est-à-dire :

$$\mathcal{R}_j = o\tau_i \oplus o\tau_j$$

Soit $\epsilon = min(r_i, r_j)$. Puisque $\epsilon$ est toujours bien défini à cause de l'aspect périodique de $o\tau_i$ et $o\tau_j$, alors nous choisissons sans nuire à la généralité $\epsilon$ comme *origine relative* de temps pour normaliser les deux opérandes de l'opération $\oplus$. L'action de normalisation de ces deux opérandes consiste à concaténer les préfixes $\{a\}^{|\gamma_{ij}|}_{(\epsilon,r_i)}$ et $\{a\}^{|\gamma_{ij}|}_{(\epsilon,r_j)}$ respectivement aux otâches $o\tau_i$ et $o\tau_j$. On obtient alors les otâches périodiques *normalisées*

$$\begin{cases} no\tau_i = \{a\}^{|\gamma_{ij}|}_{(\epsilon,r_i)} o\tau_i \\ \\ no\tau_j = \{a\}^{|\gamma_{ij}|}_{(\epsilon,r_j)} o\tau_j \end{cases}$$

La définition de l'opération $o\tau_i \oplus o\tau_j$ est donnée par trois grandes procédures décrites ci-dessous : la procédure 1, la procédure 2 et la procédure 3.

**Procédure 1 :** ***Normaliser les opérandes et réécrire le premier opérande.***

1: Concaténer les préfixes $\{a\}_{(\epsilon,r_i)}^{|\gamma_{ij}|}$ et $\{a\}_{(\epsilon,r_j)}^{|\gamma_{ij}|}$ respectivement aux otâches $o\tau_i$ et $o\tau_j$ : ce calcul permet de référencer les deux opérandes par rapport à la même origine de temps $\epsilon$. Nous obtenons alors

$$o\tau_i \oplus o\tau_j = \{a\}_{(\epsilon,r_i)}^{|\gamma_{ij}|}\zeta_{i(r_i,\beta_i)}(\tau_{0,i})_{\beta_i}^{\infty} \oplus \{a\}_{(\epsilon,r_j)}^{|\gamma_{ij}|}\zeta_{j(r_j,\beta_j)}(\tau_{0,j})_{\beta_j}^{\infty} = no\tau_i \oplus no\tau_j \quad (3.16)$$

2: Déterminer les dates $s_i^{'}$ et $s_j^{'}$ permettant de définir la phase transitoire et la phase permanente en utilisant la relation 3.17. Puisque $o\tau_i$ est la otâche ayant la priorité la plus élévée par définition, alors

$$\begin{cases} s_i^{'} = \beta_i \\ \\ s_j^{'} = \beta_j + \left\lceil \dfrac{(s_i^{'} - \beta_j)^+}{T_j} \right\rceil \cdot T_j \end{cases} \quad (3.17)$$

L'intervalle donnée par $[\epsilon, s_j^{'}]$ définit la phase transitoire et l'intervalle donnée par $[s_j^{'}, s_j^{'} + ppcm(T_i, T_j)]$ définit la phase permanente.

3: Déterminer la valeur de la quantité $\sigma_i = \left( \left\lceil \dfrac{s_j^{'} - s_i^{'}}{T_i} \right\rceil - 1 \right) + \dfrac{ppcm(T_i, T_j)}{T_i}$. Grâce à la valeur de $\sigma_i$, à la remarque 7 et au théorème 7, écrire $no\tau_i$ comme la concaténation d'une otâche finie *ph-trans* de cardinal $|$*ph-trans*$| = s_j^{'} - \epsilon$ et d'une partie périodique *ph-perm* dont la date de première activation est $s_j^{'}$ et dont la période est $T_i \vee T_j = ppcm(T_i, T_j)$. Les otâches *ph-trans* et *ph-perm* déterminent respectivement la phase transitoire et la phase permanente. Le premier opérande devient

$$no\tau_i = \textit{ph-trans ph-perm}. \quad (3.18)$$

4: Réécrire le premier opérande. Puisque le cardinal $|$*ph-trans*$|$ de la otâche finie *ph-trans* peut aussi s'écrire $|$*ph-trans*$| = s_j^{'} - \epsilon = (r_j - \epsilon) + (\beta_j - r_j) + (s_j^{'} - \beta_j)$, alors la otâche *ph-trans* peut être écrite comme concaténation de trois autres otâches finies

$$\textit{ph-trans} = (tr_1)_{(\epsilon,r_j)}(tr_2)_{(r_j,\beta_j)}(tr_3)_{(\beta_j,s_j^{'})} \quad (3.19)$$

On obtient alors :

$$no\tau_i = (tr_1)_{(\epsilon,r_j)}(tr_2)_{(r_j,\beta_j)}(tr_3)_{(\beta_j,s_j^{'})}\textit{ph-perm} \quad (3.20)$$

□

**Procédure 2 :** ***Décomposer le second opérande et mésoïdifier le premier opérande.***

1: Au terme de la procédure 1, nous avons obtenu $o\tau_i \oplus o\tau_j = no\tau_i \oplus no\tau_j$ où $no\tau_i$ est donné par l'équation 3.20. Grâce à l'application de décomposition $\pi$, si $n_j$ désigne le nombre de séquences du symbole "$e$" dans la partie périodique du second opérande, alors $no\tau_i \oplus no\tau_j = no\tau_i \oplus no\tau_j^{\odot}$ où $no\tau_j^{\odot}$ est la décomposition de $no\tau_j$.

$$no\tau_j^{\odot} = \pi(no\tau_j) = (no\tau_{j,1}; no\tau_{j,2}; \cdots; no\tau_{j,n_j}) \tag{3.21}$$

Dans la décomposition 3.21, nous rappelons que toutes les otâches périodiques $no\tau_{j,k};\ k = 1, 2, \cdots, n_j$ sont canoniques et régulières, ainsi on définit

$$no\tau_i \oplus no\tau_j^{\odot} = (((no\tau_i \oplus no\tau_{j,1}) \oplus no\tau_{j,2}) \oplus \cdots) \oplus no\tau_{j,n_j} \tag{3.22}$$

Grâce à l'égalité 3.22, effectuer l'opération $\oplus$ entre deux otâches périodiques normalisées quelconques revient à l'appliquer $n_j$ fois. L'opération $\oplus$ est alors toujours effectuée entre deux otâches dont le premier opérande est une otâche périodique et le second opérande est une otâche périodique canonique régulière. La durée d'exécution de la otâche régulière $no\tau_{j,l}, l = 1, 2, \cdots, n_j$ vaut $C_j^l$, c'est-à-dire la sous-durée d'exécution de rang $l$ de la otâche $o\tau_j$. Après toutes les opérations effectuées sur les opérandes jusqu'ici, nous avons :

$$\begin{array}{rcl} \mathcal{R}_j & = & o\tau_i \oplus o\tau_j = no\tau_i \oplus no\tau_j^{\odot} \\ & = & (((no\tau_i \oplus no\tau_{j,1}) \oplus no\tau_{j,2}) \oplus \cdots) \oplus no\tau_{j,n_j} \end{array} \tag{3.23}$$

Le résultat $\mathcal{R}_j$ de l'opération 3.23 est obtenu par itération successive de la suite

$$\begin{cases} \mathcal{R}_{j,1} = no\tau_i \oplus no\tau_{j,1} \\ \mathcal{R}_{j,l} = \mathcal{R}_{j,l-1} \oplus no\tau_{j,l}, \quad 2 \leq l \leq n_j \end{cases} \tag{3.24}$$

Ainsi, à la fin du processus 3.24, nous obtenons $\mathcal{R}_j = \mathcal{R}_{j,n_j}$. Notons que nous avons directement $\beta_{j,1} = \beta_j$ lorsque la otâche $o\tau_j$ est sous sa forme canonique.
Si à une itération donnée dans ce processus nous avons $\mathcal{R}_{j,l} = \Lambda,\ 2 \leq l \leq n_j$ alors la otâche $o\tau_j$ sera dite *non ordonnançable* et par conséquent le système d'otâches aussi, cf. la définition 33.

2: Mésoïdifier le premier opérande de l'opération $\oplus$ à chacune de ses itérations dans le calcul de $\mathcal{R}_j$. Cette opération consiste à écrire la otâche finie $(tr_3)_{(\beta_j, s_j')}$ et le motif de la otâche *ph-perm* dans l'égalité 3.20 respectivement comme des concaténations de $\sigma_{tr_3} = \left\lceil \dfrac{(s_i' - \beta_j)^+}{T_j} \right\rceil$ et $\sigma_{perm} = \dfrac{ppcm(T_i, T_j)}{T_j}$ otâches finies de cardinaux $T_j$ chacun. On obtient alors

$$no\tau_i = (tr_1)_{(\epsilon, r_j)} (tr_2)_{(r_j, \beta_j)} \left(\mathcal{M}_{tr_3,1} \cdots \mathcal{M}_{tr_3,\sigma_{tr_3}}\right) \left(\mathcal{M}_{perm,1} \cdots \mathcal{M}_{perm,\sigma_{perm}}\right)_{s_j'}^{\infty} \tag{3.25}$$

Dans l'équation 3.25, toutes les otâches $\mathcal{M}_{tr_3,k}$, avec $k = 1, \cdots, \sigma_{tr_3}$ et toutes les otâches $\mathcal{M}_{perm,l}$, avec $l = 1, \cdots, \sigma_{perm}$, sont finies de cardinaux $T_j$ et sont appelées des $T_j$-*mésoids* car elles font intervenir la période du second opérande $no\tau_j$. Chaque $T_j$-*mésoid* correspond à une instance de la tâche $\tau_j$.

□

La procédure 3 explique l'opération d'ordonnancement $\oplus$ proprement dite qui est itérée autant de fois que nécessaire pour obtenir $\mathcal{R}_j$ dans la procédure 2. En effet il suffit d'expliquer les principes permettant l'obtention du résultat de l'opération $\mathcal{R}_{j,1} = no\tau_i \oplus no\tau_{j,1}$ puis répéter le même processus itérativement pour $l = 2, \cdots, n_j$.

**Procédure 3 :** ***Effectuer l'opération d'ordonnancement $\oplus$ proprement dite.***

1: Extraire les *domaines d'échéance* : $\mathcal{D}_{(r_j,\beta_j)}$, puis $\mathcal{D}_{tr_3,k}$, où $k = 1, \cdots, \sigma_{tr_3}$ et enfin $\mathcal{D}_{perm,l}$, où $l = 1, \cdots, \sigma_{perm}$ de chaque $T_j$-*mésoid* grâce à l'égalité 3.25 et la définition 16. Cette opération consiste chaque fois à considérer d'une part la otâche finie $(tr_2)_{(r_j,\beta_j)}$ et d'autre part l'extraction du préfixe constitué des $D_j$ premiers symboles de chaque $T_j$-*mésoid*. Cette opération est toujours possible puisque $D_j \leq T_j$ pour la partie périodique de chaque otâche par définition.

2: Extraire de chaque domaine d'échéance l'*univers* associé. Les univers correspondent aux otâches finies $\mathcal{U}_{r_j,\beta_j}$, puis $\mathcal{U}_{tr_3,k}$, avec $k = 1, \cdots, \sigma_{tr_3}$ et $\mathcal{U}_{perm,l}$, avec $l = 1, \cdots, \sigma_{perm}$ constituées uniquement des symboles "$a$" du domaine d'échéance correspondant (c'est-à-dire, les unités de temps disponibles). Chaque univers est obtenu par concaténation de toutes sous otâches constituées uniquement de symbôles "$a$" du domaine d'échéance.

3: Soit $\mathcal{E}_{r_j,\beta_j}$ la sous otâche constituée uniquement des symboles "$e$" de la partie initiale de $no\tau_{j,1}$. Puisque $|\mathcal{E}_{r_j,\beta_j}| = \sum_{m=1}^{n_{in,j}} C_{in,j}^m$ et $C_j^1$ désignent respectivement le cardinal de $\mathcal{E}_{r_j,\beta_j}$ et la durée d'exécution de la partie périodique de la otâche $no\tau_{j,1}$, alors la otâche $no\tau_{j,1}$ est ordonnançable sans prise en compte du coût de la préemption si et seulement si $|\mathcal{E}_{r_j,\beta_j}| \leq |\mathcal{U}_{r_j,\beta_j}|$, $C_j^1 \leq |\mathcal{U}_{tr_3,k}|, \quad \forall k \in \{1, \cdots, \sigma_{tr_3}\}$ et enfin $C_j^1 \leq |\mathcal{U}_{perm,l}|, \quad \forall l \in \{1, \cdots, \sigma_{perm}\}$ c'est-à-dire,

$$\begin{cases} |\mathcal{E}_{r_j,\beta_j}| \leq |\mathcal{U}_{r_j,\beta_j}|, \\ C_j^1 \leq \min_{\substack{k \in \{1,\cdots,\sigma_{tr_3}\}, \\ l \in \{1,\cdots,\sigma_{perm}\}}} (|\mathcal{U}_{tr_3,k}|, |\mathcal{U}_{perm,l}|) \end{cases} \tag{3.26}$$

4: Si l'équation 3.26 est satisfaite, alors la otâche $\mathcal{R}_{j,1}$, résultat de l'opération $no\tau_i \oplus no\tau_{j,1}$, est obtenue à partir de l'égalité 3.25, en remplaçant respectivement les $|\mathcal{E}_{r_j,\beta_j}|$ premiers symboles "$a$" de la otâche finie $(tr_2)_{(r_j,\beta_j)}$ et les $C_j^1$ premiers symboles "$a$" de chaque $T_j$-*mésoid* par des symboles "$e$". Nous obtenons ainsi $\mathcal{R}_{j,1} \neq \Lambda$ et nous pouvons passer à l'opération du rang suivant de la otâche $o\tau_j$ et ainsi de suite jusqu'au rang $n_j$.

5: Dans le processus de remplacement des symboles, les *temps de réponse de rang* 1 de la otâche $o\tau_{j,1}$ correspondent chaque fois au cardinal du préfixe définit par l'indice du dernier symbole "$a$" remplacé. Nous noterons respectivement $R_{r_j,\beta_j}$, $R^1_{tr_3,k}, k \in \{1, \cdots, \sigma_{tr_3}\}$, et $R^1_{perm,l}, l \in \{1, \cdots, \sigma_{perm}\}$ les différentes valeurs obtenues. Le *pire temps de réponse de rang* 1 de la otâche $o\tau_{j,1}$ correspond au maximum des temps de réponse de rang 1.

6: Si l'équation 3.26 n'est pas satisfaite alors $\mathcal{R}_{j,1} = \Lambda$. La otâche $o\tau_j$ et le système d'otâches sont déclarés *non ordonnançables* à cause d'une échéance dépassée.

□

**Définition 34** *Les* temps de réponse *de la otâche $o\tau_j$, lorsque $\mathcal{R}_j = \mathcal{R}_{j,n_j} \neq \Lambda$ est écrite sous sa forme mésoïdifiée (cf. équation 3.25), correspondent chaque fois au cardinal du préfixe de la otâche définie par l'indice du dernier symbole "$a$" remplacé. En d'autres termes, ils valent respectivement $R_{r_j,\beta_j}$, $r_p^{n_j} + R^{n_j}_{tr_3,k}, k \in \{1, \cdots, \sigma_{tr_3}\}$, et $r_p^{n_j} + R^{n_j}_{perm,l}, l \in \{1, \cdots, \sigma_{perm}\}$ où nous rappelons que $r_p^{n_j}$ désigne l'indice de début de la dernière séquence de symboles "$e$" de $o\tau_j$.*

**Définition 35** *Le* pire temps de réponse *de la otâche $o\tau_j$ correspond au maximum de ses différents temps de réponses.*

Maintenant, nous allons illustrer la définition de l'opération $\oplus$ sans coût de préemption à l'aide d'un exemple en suivant les procédures 1, 2 et 3.

### Exemple

Soit $O\Gamma_2 = \{o\tau_1, o\tau_2\}$ un système constitué de deux otâches périodiques à échéances sur activations tel que

$$\begin{cases} o\tau_1 = \{e, a\}_{(2,4)}\{a, e, a, a\}_4^\infty \\ \\ o\tau_2 = \{e, a, a\}_{(0,3)}\{e, a, a, e, e, a\}_3^\infty \end{cases}$$

Comme la otâche $o\tau_2$ est canonique tandis que la otâche $o\tau_1$ ne l'est pas, l'écriture de $o\tau_1$ sous sa forme canonique grâce au théorème 5 donne :

$$\begin{cases} o\tau_1 = \{e,a,a\}_{(2,5)}\{e,a,a,a\}_5^\infty \\ \\ o\tau_2 = \{e,a,a\}_{(0,3)}\{e,a,a,e,e,a\}_3^\infty \end{cases}$$

Ainsi pour ce système d'otâches, nous avons $o\tau_1 = \zeta_{1(r_1,\beta_1)}(\tau_{0,1})_{\beta_1}^\infty$ où $\zeta_1 = \{e,a,a\}$, $r_1 = 2$, $\beta_1 = 5$, $\tau_{0,1} = \{e,a,a,a\}$ et la période est $T_1 = |\{e,a,a,a\}| = 4$. De même, nous avons $o\tau_2 = \zeta_{2(r_2,\beta_2)}(\tau_{0,2})_{\beta_2}^\infty$ où $\zeta_2 = \{e,a,a\}$, $r_2 = 0$, $\beta_2 = 3$, $\tau_{0,2} = \{e,a,a,e,e,a\}$ et la période est $T_2 = |\{e,a,a,e,e,a\}| = 6$. Si nous supposons que la otâche $o\tau_1$ est plus prioritaire que la otâche $o\tau_2$, alors grâce la propriété 9, le résultat de l'ordonnancement est donné par le calcul suivant :

$$\begin{cases} \mathcal{R}_1 = \Lambda \oplus o\tau_1 = o\tau_1 \\ \mathcal{R}_2 = \mathcal{R}_1 \oplus o\tau_2 \end{cases}$$

Grâce à la définition 31, le facteur d'utilisation permanent du processeur sans coût de préemption du système vaut $U_2 = 1/4 + (1+2)/6 = 3/4 = 0.75$.

Pour calculer $\mathcal{R}_2 = o\tau_1 \oplus o\tau_2$ il faut commencer par normaliser les deux opérandes. On a $r_1 = 2$ et $r_2 = 0$, donc $|\gamma_{12}| = |0-2| = 2$ et $\epsilon = min(2,0) = 0$. Ainsi la normalisation des deux opérandes est donnée par :

$$\begin{cases} no\tau_1 & = \{a\}_{(0,2)}^2 o\tau_1 \\ & = \{a,a\}_{(0,2)}\{e,a,a\}_{(2,5)}\{e,a,a,a\}_5^\infty \\ & = \{a,a,e,a,a\}_{(0,5)}\{e,a,a,a\}_5^\infty \\ \\ no\tau_2 & = \{a\}_{(0,0)}^2\{e,a,a\}_{(0,3)}\{e,a,a,e,e,a\}_3^\infty \\ & = \ \Lambda\ \{e,a,a\}_{(0,3)}\{e,a,a,e,e,a\}_3^\infty \\ & = \{e,a,a\}_{(0,3)}\{e,a,a,e,e,a\}_3^\infty \end{cases}$$

Maintenant que nous avons normalisé les deux opérandes de l'opération $\oplus$ il faut décomposer le second opérande. Grâce à l'application $\pi$, cette décomposition est donnée par $\pi(no\tau_2) = no\tau_2^\odot$ avec

$$\begin{cases} no\tau_2^\odot & = \big(\{e,a,a\}_{(0,3)}\{e,a,a,a,a,a\}_3^\infty; \{a,a,a,e,e,a\}_3^\infty\big) \\ & = \big(\{e,a,a\}_{(0,3)}\{e,a,a,a,a,a\}_3^\infty; \{a,a,a\}_{(3,6)}\{e,e,a,a,a,a\}_6^\infty\big) \end{cases}$$

Ainsi, $\mathcal{R}_2 = o\tau_1 \oplus o\tau_2$ vaut :

$$\begin{aligned} \mathcal{R}_2 &= no\tau_1 \oplus no\tau_2 \\ &= no\tau_1 \oplus no\tau_2^\odot \\ &= \big((no\tau_1 \oplus no\tau_{2,1}) \oplus no\tau_{2,2}\big) \end{aligned}$$

où les otâches $no\tau_{2,1}$ et $no\tau_{2,2}$ sont données par $no\tau_{2,1} = \{e,a,a\}_{(0,3)}\{e,a,a,a,a,a\}_3^\infty$ et $no\tau_{2,2} = \{a,a,a\}_{(3,6)}\{e,e,a,a,a,a\}_6^\infty$. Cette décomposition impose donc deux itérations pour conclure sur l'ordonnançabilité de la otâche $o\tau_2$.

**Première itération :** Calcul de l'opération $\mathcal{R}_{2,1} = no\tau_1 \oplus no\tau_{2,1}$

Nous rappelons que les otâches $no\tau_1$ et $no\tau_{2,1}$ sont respectivement données par $no\tau_1 = \{a,a,e,a,a\}_{(0,5)}\{e,a,a,a\}_5^\infty$ et $no\tau_{2,1} = \{e,a,a\}_{(0,3)}\{e,a,a,a,a,a\}_3^\infty$. Grâce à l'équation 3.17, nous avons :

$$\begin{cases} s_1^{'} = 5 \\ \\ s_2^{'} = 3 + \left\lceil \dfrac{(5-3)^+}{6} \right\rceil \cdot 6 = 9 \end{cases}$$

Donc l'intervalle $[0,9]$ détermine la phase transitoire et l'intervalle $[9,21]$ détermine la phase permanente.

Le calcul de $\sigma_1$ est donné par $\sigma_1 = \left(\left\lceil \dfrac{9-5}{4} \right\rceil - 1\right) + \dfrac{ppcm(4,6)}{4} = 3$. Grâce à la valeur de $\sigma_1$, la remarque 7 et le théorème 7, on peut écrire $no\tau_1$ comme la concaténation de la otâche finie *ph-trans*$_{2,1}$ de cardinal $|$*ph-trans*$_{2,1}| = 9$ et d'une partie périodique *ph-perm*$_{2,1}$ dont la date de première activation est $9$ et dont la période est $ppcm(T_1,T_2) = 12$. On obtient alors :

$$\begin{aligned} no\tau_1 &= \{a,a,e,a,a\}_{(0,5)}\{e,a,a,a\}_5^\infty \\ &= \{a,a,e,a,a\}_{(0,5)}(\{e,a,a,a\}\{e,a,a,a\}\{e,a,a,a\}\{e,a,a,a\}_{17}^\infty) \\ &= \{a,a,e,a,a,e,a,a,a\}_{(0,9)}\{e,a,a,a,e,a,a,a,e,a,a,a\}_9^\infty \\ &= \left(\{a,a,e\}_{(0,3)}\{a,a,e,a,a,a,a\}_{(3,9)}\right)\{e,a,a,a,e,a,a,a,e,a,a,a\}_9^\infty \end{aligned}$$

Par identification par rapport à l'égalité 3.20, la otâche normalisée $no\tau_1$ est bien de la forme $no\tau_1 = (tr_1)_{(\epsilon,r_2)}(tr_2)_{(r_2,\beta_2)}(tr_3)_{(\beta_2,s_2')}$*ph-perm* où

$$\begin{cases} (tr_1)_{(\epsilon,r_2)} &= (tr_1)_{(0,0)} = \Lambda \\ \\ (tr_2)_{(r_2,\beta_2)} &= \{a,a,e\}_{(0,3)} \\ \\ (tr_3)_{(\beta_2,s_2')} &= \{a,a,e,a,a,a\}_{(3,9)} \\ \\ \textit{ph-perm} &= \{e,a,a,a,e,a,a,a,e,a,a,a\}_9^\infty \end{cases}$$

Pour effectuer la mésoïdification de $no\tau_1$, les calculs de $\sigma_{tr_3}$ et $\sigma_{perm}$ donnent respectivement $\sigma_{tr_3} = \left\lceil \dfrac{(5-3)^+}{6} \right\rceil = 1$ et $\sigma_{perm} = \dfrac{ppcm(4,6)}{6} = 2$. Ainsi on a :

$$\begin{aligned} no\tau_1 &= \left(\{a,a,e\}_{(0,3)}\{a,a,e,a,a,a\}_{(3,9)}\right)\{e,a,a,a,e,a,a,a,e,a,a,a\}_9^\infty \\ &= \{a,a,e\}_{(0,3)}\{a,a,e,a,a,a\}_{(3,9)}\left(\{e,a,a,a,e,a\}\{a,a,e,a,a,a\}\right)_9^\infty \\ &= \{a,a,e\}_{(0,3)}\left(\mathcal{M}_{tr_3,1}\right)_{(3,9)}\left(\mathcal{M}_{perm,1}\mathcal{M}_{perm,2}\right)_9^\infty \end{aligned}$$

où les 6-*mésoids* $\mathcal{M}_{tr_3,1}, \mathcal{M}_{perm,1}$ et $\mathcal{M}_{perm,2}$ sont donnés par :

$$\begin{cases} \mathcal{M}_{tr_3,1} &= \{a,a,e,a,a,a\} \\ \mathcal{M}_{perm,1} &= \{e,a,a,a,e,a\} \\ \mathcal{M}_{perm,2} &= \{a,a,e,a,a,a\} \end{cases}$$

Puisque les otâches sont à échéances sur activations (c'est-à-dire, l'échéance relative de chaque otâche est égale à sa période), alors l'extraction des *domaines d'échéance* est donnée par :

$$\begin{cases} \mathcal{D}_{(0,3)} &= \{a,a,e\}_{(0,3)} \\ \mathcal{D}_{tr_3,1} &= \{a,a,e,a,a,a\} \\ \mathcal{D}_{perm,1} &= \{e,a,a,a,e,a\} \\ \mathcal{D}_{perm,2} &= \{a,a,e,a,a,a\} \end{cases}$$

Les univers $\mathcal{U}_{0,3}, \mathcal{U}_{tr_3,1}, \mathcal{U}_{perm,1}$, et $\mathcal{U}_{perm,2}$ sont alors donnés par :

$$\begin{cases} \mathcal{U}_{0,3} &= \{a,a\}_{0,3} \\ \mathcal{U}_{tr_3,1} &= \{a,a\}\{a,a,a\} = \{a,a,a,a,a\} \\ \mathcal{U}_{perm,1} &= \{a,a,a\}\{a\} = \{a,a,a,a\} \\ \mathcal{U}_{perm,2} &= \{a,a\}\{a,a,a\} = \{a,a,a,a,a\} \end{cases}$$

Grâce aux égalités ci-dessus, $|U_{0,3}| = 2$, $|\mathcal{U}_{tr_3,1}| = 5$, $|\mathcal{U}_{perm,1}| = 4$ et $|\mathcal{U}_{perm,2}| = 5$. Comme $no\tau_{2,1} = \{e,a,a\}_{(0,3)}\{e,a,a,a,a,a\}_3^\infty$, alors $|\mathcal{E}_{0,3}| = 1$ et $C_2^1 = 1$. Ainsi la otâche $no\tau_{2,1}$ est ordonnançable puisque la relation 3.26 est satisfaite.

$$\begin{cases} |\mathcal{E}_{0,3}| = 1 \leq 2 = |\mathcal{U}_{r_j,\beta_j}| \\ |\mathcal{E}_j| = 1 \leq 4 = \min(|\mathcal{U}_{tr_3,1}|, |\mathcal{U}_{perm,l}|, |\mathcal{U}_{perm,2}|) \end{cases}$$

La otâche $\mathcal{R}_{2,1} = no\tau_1 \oplus no\tau_{2,1}$ est obtenue à partir de l'égalité 3.25 et est donnée par :

$$\begin{aligned}
\mathcal{R}_{2,1} &= no\tau_1 \oplus no\tau_{2,1} \\
&= \{a,a,e\}_{(0,3)}\{a,a,e,a,a,a\}_{(3,9)}\left(\{e,a,a,a,e,a\}\{a,a,e,a,a,a\}\right)_9^{\infty} \oplus no\tau_{2,1} \\
&= \{\mathbf{e},a,e\}_{(0,3)}\{\mathbf{e},a,e,a,a,a\}_{(3,9)}\left(\{e,\mathbf{e},a,a,e,a\}\{\mathbf{e},a,e,a,a,a\}\right)_9^{\infty} \\
&= \{e,a,e,e,a,e,a,a,a\}_{(0,9)}\{e,e,a,a,e,a,e,a,e,a,a,a\}_9^{\infty} \\
&= \{e,a,e\}_{(0,3)}\{e,a,e,a,a,a,e,e,a,a,e,a\}_3^{\infty}
\end{aligned}$$

La dernière égalité ci-dessus est obtenue par application du théorème 5. Du point de vue des différents temps de réponse, ils sont respectivement donnés par les égalités suivantes : $R_{0,3} = |\{e\}| = 1$, $R^1_{tr_3,1} = |\{e\}| = 1$, $R^1_{perm,1} = |\{e,e\}| = 2$ et $R^1_{perm,2} = |\{e\}| = 1$ et se réduisent finalement grâce au théorème 5 à $R_{0,3} = 1$ puis $R^1_{perm,1} = 1$ et $R^1_{perm,2} = 2$ unités de temps. Puisque $\mathcal{R}_{2,1} \neq \Lambda$, nous pouvons passer à l'itération suivante, c'est-à-dire au calcul de l'opération $\mathcal{R}_{2,1} \oplus no\tau_{2,2}$.

**Seconde itération :** Calcul de l'opération $\mathcal{R}_2 = \mathcal{R}_{2,2} = \mathcal{R}_{2,1} \oplus no\tau_{2,2}$

Grâce au résultat de l'itération précédente, nous rappelons que les otâches $\mathcal{R}_{2,1}$ et $no\tau_{2,2}$ sont respectivement données par $\mathcal{R}_{2,1} = \{e,a,e\}_{(0,3)}\{e,a,e,a,a,a,e,e,a,a,e,a\}_3^{\infty}$ et $no\tau_{2,2} = \{a,a,a\}_{(3,6)}\{e,e,a,a,a,a\}_6^{\infty}$. Après un calcul analogue à celui de la première itération, nous pouvons donc conclure que la otâche $no\tau_{2,2}$ est ordonnançable et le résultat est donné par :

$$\begin{aligned}
\mathcal{R}_{2,2} &= \mathcal{R}_{2,1} \oplus no\tau_{2,2} \\
&= \{e,a,e\}_{(0,3)}\{e,a,e,a,a,a,e,e,a,a,e,a\}_3^{\infty} \oplus \{a,a,a\}_{(3,6)}\{e,e,a,a,a,a\}_6^{\infty} \\
&= \{e,a,e,e,a,e\}_{(0,6)}\{a,a,a,e,e,a,a,e,a,e,a,e\}_6^{\infty} \oplus \{a\}^6_{(0,6)}\{e,e,a,a,a,a\}_6^{\infty} \\
&= \{e,a,e,e,a,e\}_{(0,6)}\left(\{a,a,a,e,e,a\}\{a,e,a,e,a,e\}\right)_6^{\infty} \oplus \{a\}^6_{(0,6)}\{e,e,a,a,a,a\}_6^{\infty} \\
&= \{e,a,e,e,a,e\}_{(0,6)}\left(\{\mathbf{e},\mathbf{e},a,e,e,a\}\{\mathbf{e},e,\mathbf{e},e,a,e\}\right)_6^{\infty} \\
&= \{e,a,e,e,a,e\}_{(0,6)}\{e,e,a,e,e,a,e,e,e,e,a,e\}_6^{\infty} \\
&= \{e,a,e\}_{(0,3)}\{e,a,e,e,e,a,e,e,a,e,e,e\}_3^{\infty} = \mathcal{R}_2
\end{aligned}$$

**Remarque 14** *Le cardinal de la partie initiale du résultat est toujours supérieur ou égal à* 3 *à cause de l'inégalité 3.15. Dans notre cas* $r_{min} = min(r_{o\tau_1}, r_{o\tau_2}) = 0$ *et* $\beta_{min} = min(\beta_{o\tau_1}, \beta_{o\tau_2}) = 3$.

Comme dans la première itération, la dernière égalité ci-dessus est obtenue grâce au théorème 5. De même, du point de vue des temps de réponse, nous avons $R_{0,6} = 0$, puis $R^2_{perm,1} = 2$ et $R^1_{perm,2} = 3$ unités de temps. Puisque $r^2_p = 3$, alors par comparaison à la otâche $\mathcal{R}_{2,1}$, les différents temps de réponse de la otâche $o\tau_2$ sont respectivement donnés par 1 unité de temps dans la phase transitoire, puis $3 + |\{e,e\}| = 3 + 2 = 5$ et $3 + |\{e,e,e\}| = 3 + 3 = 6$ dans la phase permanente. Le pire temps de réponse de la

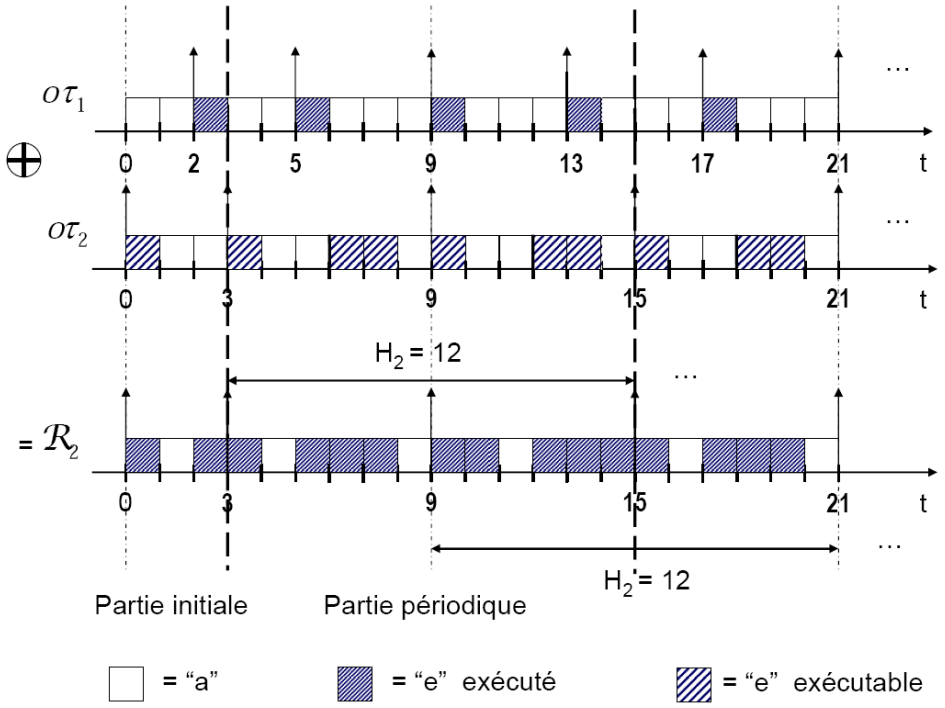


FIG. 3.9 – Illustration du résultat de l'opération $\oplus$ sans coût de préemption.

otâche $o\tau_2$ est donc atteint dans la phase permanente et vaut $6$ unités de temps. La figure 3.9 illustre le résultat final de l'opération $o\tau_1 \oplus o\tau_2$ sans coût de préemption.

En définitive lorsque nous appliquons l'opération $\oplus$ à deux otâches $o\tau_1$ et $o\tau_2$ d'un système donné, le résultat obtenu comporte les trois parties illustrées par la figure 3.10. La partie initiale et la phase transitoire définissent la partie initiale du résultat et la phase permanente est égale à la partie périodique du résultat. Ce résultat est alors réécrit sous la forme d'une otâche périodique habituelle (partie initiale et partie périodique) grâce au théorème 5.

## 3.6 Simplification pour le cas des tâches périodiques

Dans la section précédente nous avons parlé de l'étude d'ordonnançabilité de systèmes d'otâches périodiques et maintenant nous allons parler d'un cas plus simple. Pour le cas particulier de l'étude d'ordonnançabilité d'un système temps réel constitués de tâches périodiques à l'aide des systèmes d'otâches périodiques, nous allons considérer dans toute la suite de cette thèse seulement des ôtaches canoniques dont la partie initiale

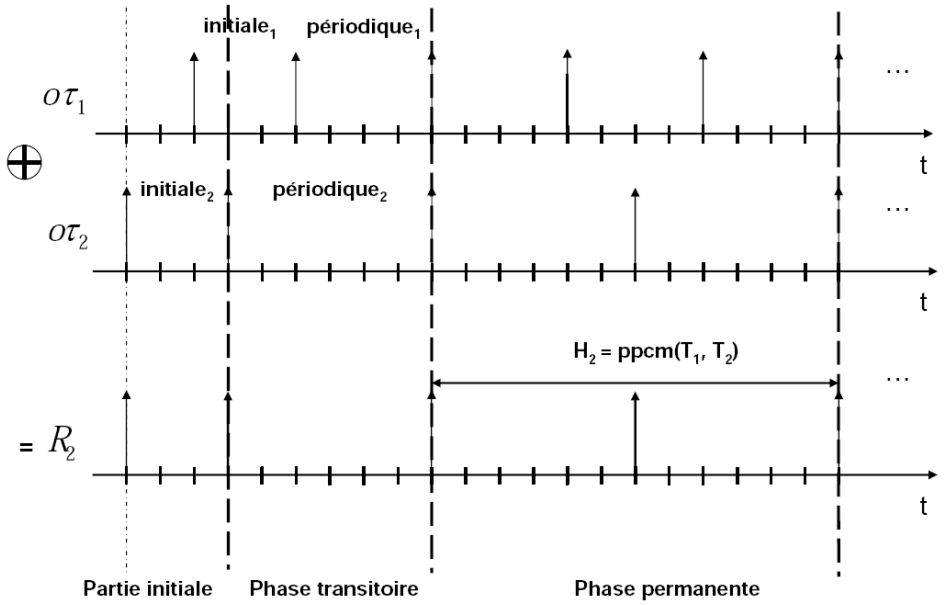


FIG. 3.10 – Illustration des parties du résultat de l'opération $\oplus$.

ne pose aucun problème du point de vue de l'ordonnançabilité. De plus, sans nuire à la généralité, nous allons supposer que cette partie initiale vaut toujours $\Lambda$ pour toutes les otâches du système considéré. Ainsi, chaque otâche sera restreinte à sa partie périodique. La conséquence directe de cette hypothèse étant que, grâce à la décomposition d'une otâche périodique par l'application $\pi$, il suffira de raisonner sur des otâches canoniques régulières pour le second opérande de l'opération $\oplus$, c'est-à-dire les otâches dont la correspondance est illustrée par la figure 3.8. Lorsque le second opérande correspond à une tâche périodique du type $\tau_j = (r_j^1, C_j, D_j, T_j)$, la procédure 3 est simplifiée et remplacée par la procédure 4.

**Procédure 4 :** ***Procédure 3 simplifiée lorsque le second opérande correspond à une tâche.***

1: Extraire les *domaines d'échéance* : $\mathcal{D}_{tr_3,k}$, où $k = 1, \cdots, \sigma_{tr_3}$ et $\mathcal{D}_{perm,l}$, où $l = 1, \cdots, \sigma_{perm}$ de chaque $T_j$*-mésoid* grâce à l'égalité 3.25 et à l'aide de la définition 16. Cette opération consiste chaque fois à considérer l'extraction du préfixe constitué des $D_j$ premiers symboles de chaque $T_j$*-mésoid*. Pour chaque $T_j$*-mésoid*, cette opération est toujours possible puisque $D_j \leq T_j$ par définition.

2: Extraire de chaque domaine d'échéance obtenu à l'étape précédente l'*univers* associé. Les univers correspondent aux otâches finies $\mathcal{U}_{tr_3,k}$, où $k = 1, \cdots, \sigma_{tr_3}$ et $\mathcal{U}_{perm,l}$, où $l = 1, \cdots, \sigma_{perm}$ constituées uniquement des symboles "$a$" du domaine d'échéance correspondant (c'est-à-dire, les unités de temps disponibles). Chaque univers est obtenu par concaténation de toutes sous otâches constituées uniquement de symbôles "$a$" du domaine d'échéance.

3: Puisque $C_j$ désigne la durée d'exécution de la partie périodique de la otâche $no\tau_j$, alors la otâche $no\tau_j$ est ordonnançable sans prise en compte du coût de la préemption si et seulement si

$$C_j \leq \min_{\substack{k \in \{1, \cdots, \sigma_{tr_3}\}, \\ l \in \{1, \cdots, \sigma_{perm}\}}} (|\mathcal{U}_{tr_3,k}|, |\mathcal{U}_{perm,l}|) \tag{3.27}$$

4: Si l'équation 3.27 est satisfaite, alors la otâche $\mathcal{R}_j = no\tau_i \oplus no\tau_j$ est obtenue à partir de l'égalité 3.25, en remplaçant les $C_j$ premiers symboles "$a$" de chaque $T_j$-*mésoid* par des symboles "$e$". Nous obtenons ainsi $\mathcal{R}_j \neq \Lambda$.
Dans le processus de remplacement des symboles, les *temps de réponse* de la otâche $o\tau_j$ correspondent chaque fois au cardinal du préfixe définit par l'indice du dernier symbole "$a$" remplacé. Le *pire temps de réponse* de la otâche $o\tau_j$ correspond au maximum des temps de réponse.

5: Si l'équation 3.27 n'est pas satisfaite, alors $\mathcal{R}_j = \Lambda$. La otâche $o\tau_j$ et le système d'otâches sont déclarés *non ordonnançables*.

□

Sous l'hypothèse qu'on ne prend pas en compte le coût lié à la préemption comme c'est le cas dans ce chapitre, *Joël Goossens* a démontré dans sa thèse (chapitre 2) [Goo98] que le pire temps de réponse de chaque tâche périodique d'un système ordonnançable apparaît forcément dans la phase permanente (*ph-perm*) de l'ordonnancement final. Ainsi, il suggère de négliger la phase transitoire (*ph-trans*) et de considérer seulement la phase permanente pour garantir l'ordonnançabilité de chaque tâche. Cette assertion permet de diminuer le nombre de tests à effectuer dans l'équation 3.27 de la procédure 4 car cette équation se réduirait à la condition 3.28 pour garantir l'ordonnançabilité de la tâche $\tau_j$.

$$C_j \leq \min_{l \in \{1, \cdots, \sigma_{perm}\}} (|\mathcal{U}_{perm,l}|) \tag{3.28}$$

La condition 3.28 suggère la possibilité, dans ce cas, de restreindre considérablement l'intervalle d'analyse s'il est possible de déterminer directement la phase permanente du système.

Dans la procédure 1 nous avons vu que la phase permanente d'un système ne dépend que du déphasage de chaque tâche par rapport aux autres tâches (cf. équations 3.16, 3.17 et 3.18).

**Théorème 8** *Soit $O\Gamma_n = \{o\tau_1, o\tau_2, \cdots, o\tau_n\}$ un système de $n$ otâches périodiques tel que $o\tau_i = \Lambda(\tau_{0,i})_{\beta_i}^{\infty}$, avec $i = 1, \cdots, n$, rangées par priorités décroissantes relativement à une politique d'ordonnancement, c'est-à-dire $i < j$ implique que la priorité de $o\tau_i$ est supérieure à la priorité de $o\tau_j$. Si on ne prend pas en compte le coût de la préemption alors un intervalle de longueur $H_n = ppcm\{T_i \mid i = 1, \cdots, n\}$ est suffisant pour garantir l'ordonnançabilité de chaque otâche et ceci quelque soit le scénario de premier démarrage de toutes les otâches.*

**Preuve** :
La preuve du théorème découle directement du remplissage des durées d'exécution des otâches dans le *Dameid* suivant les priorités des otâches et non suivant leurs dates de première activation qui causent l'existence d'une phase transitoire dans la représentation linéaire de l'ordonnancement / GantChart. ■

Maintenant nous allons rajouter les durées d'exécution des tâches (resp. des otâches) et expliquer comment le *Dameid* permet de représenter des ordonnancements.

Les symboles "$a$" du premier opérande (les unités de temps disponibles) qui sont remplacés par les symboles "$e$" du second opérande (les unités de temps exécutables) pour obtenir un résultat après une opération $\oplus$ déterminent de façon exacte les symboles "$a$" qui ne sont pas remplacés relativement aux priorités des opérandes. Puisque les priorités des tâches (resp. des otâches) déterminent l'*ordre total* permettant d'effectuer l'opération $\oplus$ suivant une politique d'ordonnancement donnée, il s'en suit que la représentation circulaire, le *Dameid*, de circonférence correspondante au *plus petit commun multiple* ($ppcm$) des périodes de toutes les tâches que nous avons introduite permet de construire directement la phase permanente du système si celui-ci est ordonnançable. En effet, le *Dameid* peut être construit de façon totalement indépendante. Dans cette représentation, les durées d'exécution des tâches (resp. des otâches) sont représentées par des secteurs angulaires dont l'unité est $\frac{1}{H_n}$ et $H_n = ppcm(T_1, T_2, \cdots, T_n)$. La figure 3.13 illustre un exemple pour le système dont l'ordonnancement et la courbe du temps de réponse de chaque tâche (resp. des otâches) en fonction du temps sont respectivement illustrés par la figure 3.11 et la figure 3.12. Pour ce système dont les caractéristiques sont résumées dans la table 3.1, nous supposons que la tâche $t_1$ a une priorité supérieure à celle de la tâche $t_2$. Dans la figure 3.11, la phase permanente est illustrée par la zone mise en évidence (zone bleue). La courbe du temps de réponse de chaque tâche en fonction du temps (figure 3.12) montre qu'à partir de la date $t = 15$, le temps de réponse de chaque tâche est constant. On retrouve ce résultat en construisant le *Dameid*. En effet le

plus petit commun multiple des périodes des deux tâches vaut $H_2 = ppcm(5, 15) = 15$. Les dates d'activation de la tâche $t_1$ dans le *Dameid* relativement à sa date de première activation sont $r_1^1 = 4$, $r_1^2 = 9$ et $r_1^3 = 14$. Quant à la tâche $t_2$, elle a une seule date d'activation qui vaut $r_2^1 = 0$ car nous avons sa période $T_2 = H_2 = 15$. Puisque la tâche $t_1$ est plus prioritaire que la tâche $t_2$, à chacune de ses activations, c'est-à-dire aux dates $r_1^1$, $r_1^2$ et $r_1^3$, un secteur correspondant à sa durée d'exécution ($C_1 = 2$ unités de temps) est rempli. Comme la tâche $t_2$ a une priorité moins élevée que celle de la tâche $t_1$, le remplissage du secteur de la circonférence correspondant à sa durée d'exécution ($C_2 = 4$ unités de temps) ne peut se faire qu'entre les instants $1$ et $4$ puis les instants $6$ et $7$. Le *Dameid* construit directement la phase permanente du système : dans la figure 3.11, la tâche $t_2$ a deux temps de réponses distincts, $4$ unités de temps à la première activation et $7$ unités de temps par la suite tandis que dans la représentation circulaire grâce au *Dameid*, elle a un seul temps de réponse, $7$ unités de temps, ce qui correspond à son temps de réponse dans la phase permanente.

| Tâche | $r_i^1$ | $C_i$ | $D_i$ | $T_i$ |
|---|---|---|---|---|
| $t_1$ | 4 | 2 | 5 | 5 |
| $t_2$ | 0 | 4 | 15 | 15 |

TAB. 3.1 – Caractéristiques des tâches

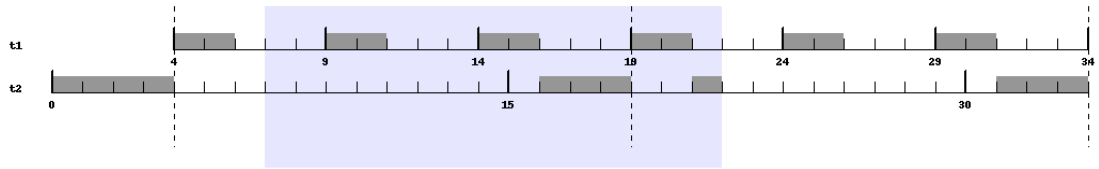


FIG. 3.11 – Illustration de l'ordonnancement linéaire des tâches.

Cette nouvelle représentation est d'autant plus intéressante car elle est plus compacte et elle met mieux en évidence (en comparaison avec la représentation linéaire / GantChart) *les unités de temps libres* de la phase permanente dans le résultat obtenu. Dans la section 4.3 du chapitre 4, contrairement à ce qui avait été suggéré par *Joël Goossens* dans sa thèse [Goo98], nous allons prouver que *nous ne pouvons pas* et surtout *nous ne devons pas* nous affranchir de la phase transitoire de l'ordonnancement d'un système donné lorsqu'on prend en compte le coût temporel lié à la préemption pour guarantir ou non son ordonnançabilité.

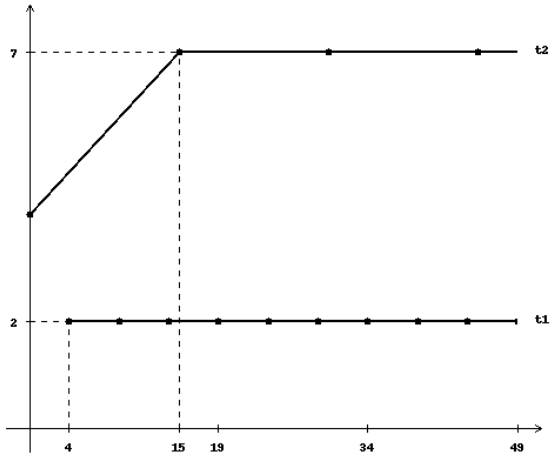


FIG. 3.12 – Courbe du temps de réponse en fonction du temps.

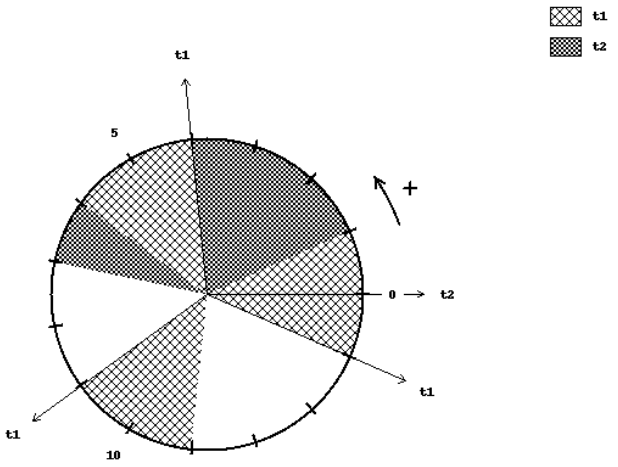


FIG. 3.13 – Illustration de l'ordonnancement circulaire des tâches par le *Dameid*.

### Exemple

Dans cet exemple, nous allons appliquer la définition de l'opération $\oplus$ sans coût de préemption au problème d'ordonnancement d'un système de tâches périodiques. On considère $\Gamma_3 = \{\tau_1, \tau_2, \tau_3\}$ un système constitué de trois tâches périodiques indépendantes où $\tau_1$ est la tâche la plus prioritaire et $\tau_3$ la tâche la moins prioritaire. Les caractéristiques des tâches sont résumées dans la table 3.2.

| Tâche | $r_i^1$ | $C_i$ | $D_i$ | $T_i$ |
|---|---|---|---|---|
| $\tau_1$ | 0 | 3 | 7 | 15 |
| $\tau_2$ | 5 | 2 | 6 | 6 |
| $\tau_3$ | 3 | 4 | 10 | 10 |

TAB. 3.2 – Caractéristiques des tâches

Nous remarquons que ce choix des priorités des tâches ne correspond ni à celui de l'algorithme $RM$ (c'est-à-dire, plus la période d'une tâche est petite plus sa priorité est grande), ni à celui de l'algorithme $DM$ (c'est-à-dire, plus l'échéance relative d'une tâche est petite plus sa priorité est grande). En effet, suivant l'algorithme $RM$, les tâches $\tau_2$ et $\tau_1$ auraient respectivement la priorité la plus élévée et la plus basse alors que suivant l'algorithme $DM$, c'est plutôt les tâches $\tau_2$ et $\tau_3$ qui auraient respectivement la priorité la plus élévée et la plus basse. Grâce à la correspondance 3.8, les otâches $o\tau_1$, $o\tau_2$ et $o\tau_3$ correspondants respectivement aux tâches $\tau_1$, $\tau_2$ et $\tau_3$ sont données par la relation ci-dessous où $o\tau_1$ est la otâche la plus prioritaire et $\tau_3$ la otâche la moins prioritaire.

$$\begin{cases} o\tau_1 = \{e,e,e,a,a,a,a\llcorner\, a,a,a,a,a,a,a,a\}_0^\infty \\ o\tau_2 = \{e,e,a,a,a,a\}_5^\infty \\ o\tau_3 = \{e,e,e,e,a,a,a,a,a,a\}_3^\infty \end{cases}$$

Pour chacune de ces otâches, la partie initiale vaut $\Lambda$ et nous avons $r_i = \beta_i,\ i = 1,2,3$. Le facteur d'utilisation permanent du processeur sans coût de préemption du système vaut $U_3 = 3/15{+}2/6{+}4/10 = 28/30 = 0.9333$. Grâce à tout ce que nous avons présenté jusqu'ici, le résultat $\mathcal{R}_3$ de l'ordonnancement du système $O\Gamma_3 = \{o\tau_1, o\tau_2, o\tau_3\}$ est obtenu par itération successive de la suite

$$\begin{cases} \mathcal{R}_1 = \Lambda \oplus o\tau_1 = o\tau_1 \\ \mathcal{R}_i = \mathcal{R}_{i-1} \oplus o\tau_i, \quad i = 2,3 \end{cases}$$

**Première itération :** Calcul de l'opération $\mathcal{R}_2 = o\tau_1 \oplus o\tau_2$

Pour effectuer cette opération, il faut commencer par normaliser les deux opérandes. Les otâches normalisées $no\tau_1$ et $no\tau_2$ correspondantes sont respectivement données par $no\tau_1 = \{e,e,e,a,a,a,a\llcorner\, a,a,a,a,a,a,a,a\}_0^\infty$ pour le premier opérande et $no\tau_2 = \{a,a,a,a,a\}_{(0,5)}\{e,e,a,a,a,a\}_5^\infty$ pour le second opérande. Grâce à l'équation 3.17,

nous avons :

$$\begin{cases} s_1^{'} = 0 \\ \\ s_2^{'} = 5 + \left\lceil \dfrac{(0-5)^+}{6} \right\rceil \cdot 6 = 5 \end{cases}$$

Nous avons $H_2 = ppcm(15, 6) = 30$, donc l'intervalle $[0, 5]$ détermine la phase transitoire et l'intervalle $[5, 35]$ détermine la phase permanente.

Le calcul de $\sigma_1$ est donné par $\sigma_1 = \left( \left\lceil \dfrac{5-0}{15} \right\rceil - 1 \right) + \dfrac{ppcm(15,6)}{15} = 2$. Grâce à la valeur de $\sigma_1$, la remarque 7 et le théorème 7, nous pouvons écrire $no\tau_1$ comme la concaténation d'une otâche finie *ph-trans* de cardinal $|$*ph-trans*$| = 5$ et d'une partie périodique *ph-perm* dont la date de première activation est $5$ et dont la période est $ppcm(15, 6) = 30$. On obtient alors :

$$\begin{aligned} no\tau_1 &= \{e, e, e, a, a, a, a, a, a, a, a, a, a, a\}_0^\infty \\ &= \{e, e, e, a, a\}_{(0,5)} \\ &\quad \{a, a, a, a, a, a, a, a, a, a, a, e, e, e, a, a, a, a, a, a, a, a, a, a, a, a, a, e, e, e, a, a\}_5^\infty \end{aligned}$$

Par identification par rapport à l'égalité 3.20, la otâche normalisée $no\tau_1$ est bien de la forme $no\tau_1 = (tr_1)_{(\epsilon,r_2)}(tr_2)_{(r_2,\beta_2)}(tr_3)_{(\beta_2,s_2')}$***ph-perm*** où $(tr_1)_{(\epsilon,r_2)} = \{e, e, e, a, a\}_{(0,5)}$, $(tr_2)_{(r_2,\beta_2)} = \Lambda$ puisque nous avons $r_2 = \beta_2$, $(tr_3)_{(\beta_2,s_2')} = \Lambda$ puisque nous avons $s_2^{'} = \beta_2$ et enfin nous avons

$$\textit{ph-perm} = \{a, a, a, a, a, a, a, a, a, a, a, e, e, e, a, a, a, a, a, a, a, a, a, a, a, a, a, e, e, e, a, a\}_5^\infty$$

Pour effectuer la mésoïdification de $no\tau_1$, les calculs de $\sigma_{tr_3}$ et $\sigma_{perm}$ donnent respectivement $\sigma_{tr_3} = \left\lceil \dfrac{(0-5)^+}{6} \right\rceil = 0$ et $\sigma_{perm} = \dfrac{ppcm(15,6)}{6} = 5$. Ainsi nous avons :

$$\begin{aligned} no\tau_1 &= \{e, e, e, a, a\}_{(0,5)} \\ &\quad \{a, a, a, a, a, a, a, a, a, a, a, e, e, e, a, a, a, a, a, a, a, a, a, a, a, a, a, e, e, e, a, a\}_5^\infty \\ &= \{e, e, e, a, a\}_{(0,5)}(\{a, a, a, a, a, a\}\{a, a, a, a, e, e\}\{e, a, a, a, a, a\} \\ &\quad \{a, a, a, a, a, a\}\{a, e, e, e, a, a\})_5^\infty \end{aligned}$$

$$= \{e, e, e, a, a\}_{(0,5)} \, (\mathcal{M}_{perm,1}\mathcal{M}_{perm,2}\mathcal{M}_{perm,3}\mathcal{M}_{perm,4}\mathcal{M}_{perm,5})_5^\infty$$

où les 6-***mésoids*** $\mathcal{M}_{perm,1}$, $\mathcal{M}_{perm,2}$, $\mathcal{M}_{perm,3}$, $\mathcal{M}_{perm,4}$ et $\mathcal{M}_{perm,5}$ sont donnés par $\mathcal{M}_{perm,1} = \{a, a, a, a, a, a\}$, $\mathcal{M}_{perm,2} = \{a, a, a, a, e, e\}$, $\mathcal{M}_{perm,3} = \{e, a, a, a, a, a\}$, $\mathcal{M}_{perm,4} = \{a, a, a, a, a, a\}$ et $\mathcal{M}_{perm,5} = \{a, e, e, e, a, a\}$. Chaque 6-*mésoid* correspond à une instance de la tâche $\tau_2$. Puisque la tâche $\tau_2$ est à échéance sur activation (c'est-à-dire $D_2 = T_2$), alors les *domaines d'échéance* sont aussi donnés par $\mathcal{D}_{perm,1} = \{a, a, a, a, a, a\}$, $\mathcal{D}_{perm,2} = \{a, a, a, a, e, e\}$, $\mathcal{D}_{perm,3} = \{e, a, a, a, a, a\}$, $\mathcal{D}_{perm,4} =$

$\{a,a,a,a,a,a\}$ et $\mathcal{D}_{perm,5} = \{a,e,e,e,a,a\}$. Ainsi les univers $\mathcal{U}_{perm,1}, \mathcal{U}_{perm,2}, \mathcal{U}_{perm,3}$, $\mathcal{U}_{perm,4}$ et $\mathcal{U}_{perm,5}$ sont donnés par :

$$\begin{cases} \mathcal{U}_{perm,1} & = \mathcal{D}_{perm,2} = \{a,a,a,a,a,a\} \\ \mathcal{U}_{perm,2} & = \{a,a,a,a\} \\ \mathcal{U}_{perm,3} & = \{a,a,a,a,a\} \\ \mathcal{U}_{perm,4} & = \mathcal{D}_{perm,4} = \{a,a,a,a,a,a\} \\ \mathcal{U}_{perm,5} & = \{a\}\{a,a\} = \{a,a,a\} \end{cases}$$

Grâce aux égalités ci-dessus, nous avons $|\mathcal{U}_{perm,1}| = 6$, $|\mathcal{U}_{perm,2}| = 4$, $|\mathcal{U}_{perm,3}| = 5$, $|\mathcal{U}_{perm,4}| = 6$ et $|\mathcal{U}_{perm,5}| = 3$. Comme $no\tau_2 = \{a,a,a,a,a\}_{(0,5)}\{e,e,a,a,a,a\}_5^\infty$, nous avons $C_2 = 2$ et par conséquent la otâche $o\tau_2$ est ordonnançable puisque la relation 3.26 est satisfaite.

$$C_2 = 2 \leq 3 = \min_{1 \leq j \leq 5}(|\mathcal{U}_{perm,j}|)$$

La otâche $\mathcal{R}_2 = no\tau_1 \oplus no\tau_2$ est obtenue à partir de l'égalité 3.25 et est donnée par :

$$\begin{aligned} \mathcal{R}_2 &= no\tau_1 \oplus no\tau_2 \\ &= \{e,e,e,a,a\}_{(0,5)}(\{a,a,a,a,a,a\}\{a,a,a,a,e,e\}\{e,a,a,a,a,a\} \\ &\qquad \{a,a,a,a,a,a\}\{a,e,e,e,a,a\})_5^\infty \oplus \{a,a,a,a,a\}_{(0,5)}\{e,e,a,a,a,a\}_5^\infty \\ &= \{e,e,e,a,a\}_{(0,5)}(\{\mathbf{e},\mathbf{e},a,a,a,a\}\{\mathbf{e},\mathbf{e},a,a,e,e\}\{e,\mathbf{e},\mathbf{e},a,a,a\} \\ &\qquad \{\mathbf{e},\mathbf{e},a,a,a,a\}\{\mathbf{e},e,e,e,\mathbf{e},a\})_5^\infty \\ &= \{e,e,e,a,a\}_{(0,5)} \\ &\qquad \{e,e,a,a,a,a,e,e,a,a,e,e,e,e,e,a,a,a,e,e,a,a,a,a,e,e,e,e,e,a\}_5^\infty \\ &= \{e,e,e,a\}_{(0,4)} \\ &\qquad \{a,e,e,a,a,a,a,e,e,a,a,e,e,e,e,e,a,a,a,e,e,a,a,a,a,e,e,e,e,e\}_4^\infty \end{aligned}$$

Une fois de plus, la dernière égalité de la série d'égalités ci-dessus est obtenue grâce au théorème 5. Cette égalité permet aussi de déterminer la date de début minimale (au plus tôt) de la phase permanente lorsque le système est ordonnançable. Du point de vue des temps de réponse, nous avons $R_2^1 = 2$, $R_2^2 = 2$, $R_2^3 = 3$, $R_2^4 = 2$ et $R_2^5 = 5$ unités de temps où $R_2^k$ désigne le temps de réponse de la tâche $\tau_2$ dans sa $k^{ieme}$ instance. Le pire temps de réponse de la tâche $\tau_2$ vaut alors $R_2 = 5$ unités de temps et il est atteint pour la première fois dans la cinquième instance de $\tau_2$.

**Seconde itération :** Calcul de l'opération $\mathcal{R}_3 = \mathcal{R}_2 \oplus o\tau_3$

Grâce au résultat de l'itération précédente, les otâches normalisées $n\mathcal{R}_2$ et $no\tau_3$ correspondants respectivement aux otâches périodiques $\mathcal{R}_2$ et $o\tau_3$ sont données par $n\mathcal{R}_2 = \{e,e,e,a,a\}_{(0,5)}\{e,e,a,a,a,a,e,e,a,a,e,e,e,e,e,a,a,a,e,e,a,a,a,a,e,e,e,e,e,a\}_5^\infty$ et $no\tau_3 = \{a,a,a\}_{(0,3)}\{e,e,e,e,a,a,a,a,a,a\}_3^\infty$. Après un calcul analogue à celui de

la première itération, nous pouvons donc conclure que la otâche $no\tau_3$ est ordonnançable et le résultat est donné par :

$$
\begin{aligned}
\mathcal{R}_3 &= n\mathcal{R}_2 \oplus no\tau_3 \\
&= \{e,e,e,a,a\}_{(0,5)} \\
&\qquad \{e,e,a,a,a,a,e,e,a,a,e,e,e,e,e,a,a,a,e,e,a,a,a,a,e,e,e,e,e,a\}_5^{\infty} \\
&\qquad \oplus \{a,a,a\}_{(0,3)}\{e,e,e,e,a,a,a,a,a,a\}_3^{\infty} \\
&= \{e,e,e\}_{(0,3)}\{a,a,e,e,a,a,a,a,e,e\}_{(3,13)} \\
&\qquad \{a,a,e,e,e,e,e,a,a,a,e,e,a,a,a,a,e,e,e,e,e,a,e,e,a,a,a,a,e,e\}_{13}^{\infty} \\
&\qquad \oplus \{a,a,a\}_{(0,3)}\{e,e,e,e,a,a,a,a,a,a\}_3^{\infty} \\
&= \{e,e,e\}_{(0,3)}\{a,a,e,e,a,a,a,a,e,e\}_{(3,13)} \\
&\qquad (\{a,a,e,e,e,e,e,a,a,a\}\{e,e,a,a,a,a,e,e,e,e\}\{e,a,e,e,a,a,a,a,e,e\})_{13}^{\infty} \\
&\qquad \oplus \{a,a,a\}_{(0,3)}\{e,e,e,e,a,a,a,a,a,a\}_3^{\infty} \\
&= \{e,e,e\}_{(0,3)}\{\mathbf{e},\mathbf{e},e,e,\mathbf{e},\mathbf{e},a,a,e,e\}_{(3,13)} \\
&\qquad (\{\mathbf{e},\mathbf{e},e,e,e,e,e,\mathbf{e},\mathbf{e},a\}\{e,e,\mathbf{e},\mathbf{e},\mathbf{e},\mathbf{e},e,e,e,e\}\{e,\mathbf{e},e,e,\mathbf{e},\mathbf{e},\mathbf{e},a,e,e\})_{13}^{\infty} \\
&= \{e,e,e,e,e,e,e,e,e,a,a,e,e\}_{(0,13)} \\
&\qquad \{e,e,e,e,e,e,e,e,e,a,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,a,e,e\}_{13}^{\infty} \\
&= \{e,e,e,e,e,e,e,e,e,a\}_{(0,10)} \\
&\qquad \{a,e,e,e,e,e,e,e,e,e,e,e,a,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e\}_{10}^{\infty}
\end{aligned}
$$

Du point de vue des temps de réponse, nous avons $R_3^1 = 6$, $R_3^2 = 9$, $R_3^3 = 6$ et $R_3^4 = 7$ unités de temps où $R_3^2 = 9$, $R_3^3 = 6$ et $R_3^4 = 7$ unités de temps désignent les temps de réponse de la tâche $\tau_3$ dans la phase permanente. Par conséquent, le pire temps de réponse de la tâche $\tau_3$ vaut $R_2 = 9$ unités de temps et il est atteint pour la première fois à la deuxième activation de $\tau_3$. La figure 3.14 résume les résultats de cet exemple et la figure 3.15 illustre la courbe du temps de réponse de chaque otâche et donc de chaque tâche en fonction du temps. Dans la figure 3.14, *PTR* signifie *Pire Temps de Réponse*, la phase permanente correspond à la zone mise en évidence dans l'ordonnançement (zone bleue) et la phase transitoire correspond à l'intervalle précédent cette zone.

## 3.7 Conclusion

Dans ce chapitre, nous avons commencé par introduire quelques définitions pour préparer le cadre de notre approche d'analyse d'ordonnançabilité des systèmes temps réel durs. Nous avons introduit la notion d'otâche et de système d'otâches. Nous avons présenté la notion d'otache périodique et nous avons montré la correspondance entre une otâche et une tâche périodique. Ensuite, nous avons introduit la décomposition d'une otâche périodique comme une suite d'otâches périodiques canoniques et régulières. Nous avons présenté un moyen permettant de réduire l'intervalle d'analyse pour l'ordonnancement temps réel sans coût de la préemption et enfin nous avons présenté l'opération

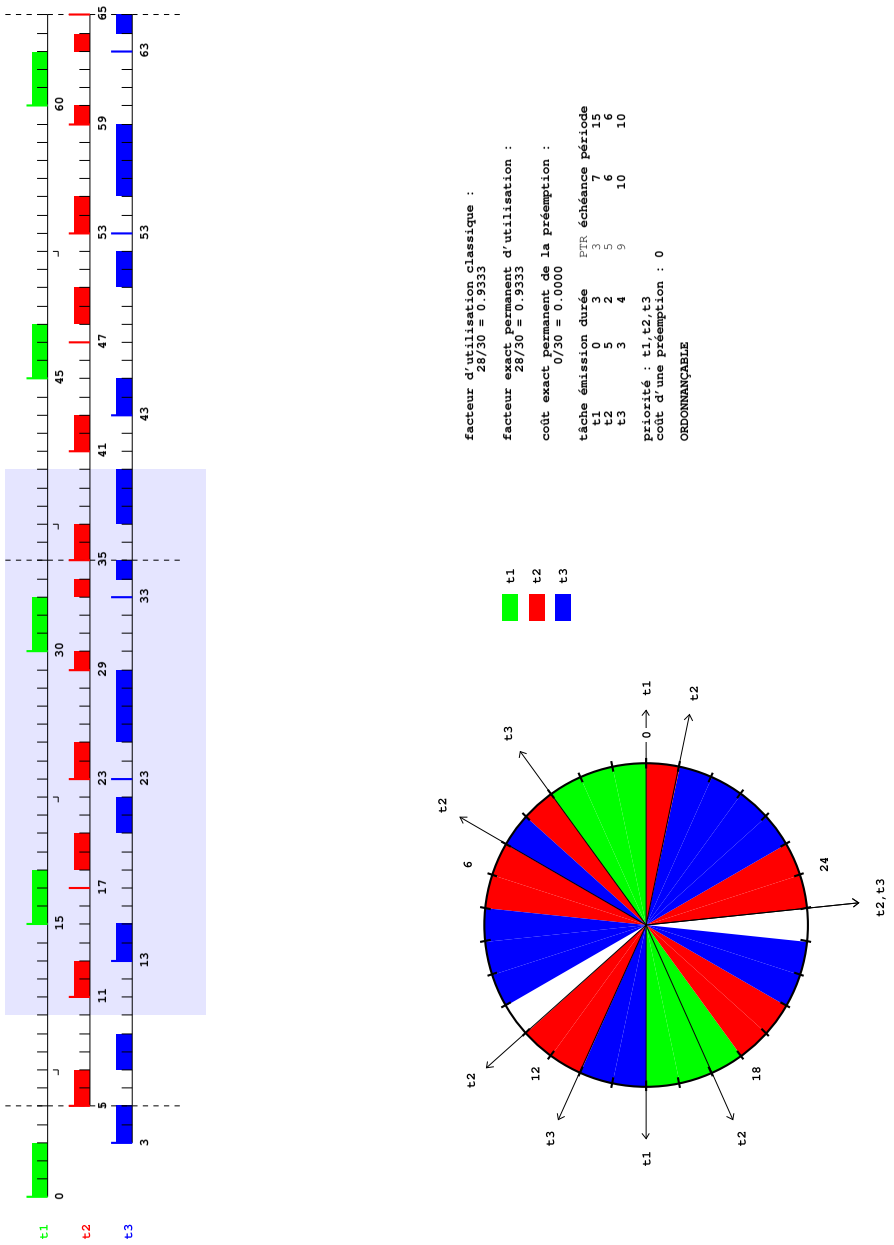


FIG. 3.14 – Résumé des résultats de l'exemple sans coût de préemption.

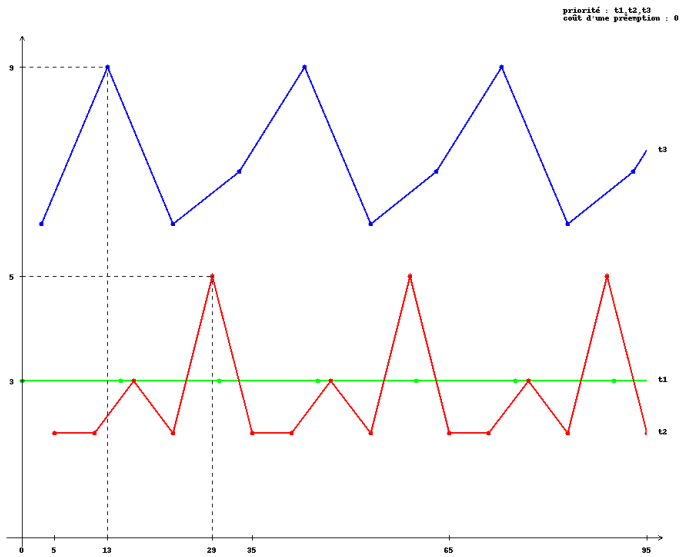


FIG. 3.15 – Courbe du temps de réponse en fonction du temps.

d'ordonnancement d'otâches périodiques sans coût de préemption lorsque les priorités des otâches sont imposées.

# Chapitre 4

# Ordonnancement temps réel avec coût exact de la préemption

*Se donner du mal pour les petites choses,*
*c'est parvenir aux grandes, avec le temps.*
**Samuel Beckett**

## 4.1 Introduction

Dans le chapitre 3, nous avons présenté quelques définitions pour préparer le cadre de notre approche d'analyse d'ordonnançabilité des systèmes temps réel durs. Parmi les notions nouvelles, nous avons introduit les notions d'otâche et de système d'otâches, puis nous avons fait la correspondance entre une otâche périodique et une tâche périodique. Nous avons présenté l'opération d'ordonnancement $\oplus$ pour un système d'otâches périodiques sans coût de préemption lorsque les priorités des otâches sont imposées (par exemple, suivant Rate Monotonic / RM, Deadline Monotonic / DM, Audsley ou encore tout autre politique de choix des priorités des otâches). Dans ce chapitre, nous allons présenter la deuxièmee partie de notre contribution. Nous allons commencer par montrer l'impact de la prise en compte du coût de la préemption sur l'ordonnancement et l'ordonnançabilité d'un système donné, puis nous allons présenté l'opération d'ordonnancement $\oplus$ d'un système d'otâches périodiques avec *coût exact de préemption* lorsque les priorités des otâches sont imposées. Par la suite, nous allons relacher cette contrainte et nous allons montrer l'impact du choix des priorités sur l'ordonnancement et sur l'analyse d'ordonnançabilité d'un système. Nous allons montrer que le choix des priorités suivant l'algorithme d'*Audsley* n'est plus applicable, ni optimal lorsqu'on prend en compte le coût de la préemption. Enfin, nous allons présenter un algorithme optimal de choix des priorités des otâches pour un système donné au sens où s'il existe un choix de priorités conduisant à un ordonnancement valide alors le choix des priorités des otâches proposé conduira également à un ordonnancement valide.

## 4.2 Impact du coût de la préemption

Dans la section 3.5 du chapitre 3, afin de faire l'étude d'ordonnançabilité des systèmes temps réel, nous avons explicitement fait l'hypothèse qu'on pouvait, sans nuire à la généralité, considérer la partie initiale de chaque otâche périodique égale à $\Lambda$. De plus, grâce à la propriété de décomposition d'une otâche périodique quelconque en une suite finie d'otâches canoniques régulières, nous avons montré qu'on pouvait, toujours sans nuire à la généralité, supposer que le second opérande de l'opération $\oplus$ est canonique régulière. Par conséquent, dans ce chapitre, nous allons principalement nous intéresser à la définition de l'opération $\mathcal{R}_j = o\tau_i \oplus o\tau_j$ où $o\tau_i$ est une otâche périodique et $o\tau_j$ est une otâche périodique canonique régulière dont la partie initiale vaut $\Lambda$.

Comme nous l'avons déjà souligné dans le chapitre 2, la difficulté de la prise en compte du coût exact du système d'exploitation (*OS*) dans la condition d'ordonnançabilité d'un système temps réel est réduite à celle de la prise en compte du coût de la préemption. Dans le cas où on le prend en compte lorsqu'on effectue l'opération $\oplus$, le premier opérande ne peut pas être préemptée parce qu'il a la priorité la plus élevée alors que le second opérande peut être préempté. Une *préemption* éventuelle du second opérande, c'est-à-dire de la otâche $o\tau_j$ lorsqu'on effectue l'opération $\mathcal{R}_j = o\tau_i \oplus o\tau_j$, correspond à une *transition* $(a \rightarrow e)$ dans chaque $T_j$*-mésoid* du premier opérande, c'est-à-dire de la otâche $o\tau_i$. Ainsi, la condition fournie par l'équation 3.27 du chapitre 3 n'est plus que *nécessaire*. En effet, à chaque occurrence d'une préemption d'une tâche temps réel lors de son exécution (resp. d'une otâche canonique régulière lors du remplacement des symboles "$a$" du premier opérande par des symboles "$e$" exécutables), un coût temporel correspondant à $\alpha$ unités de temps (resp. $\alpha$ symboles "$e$" exécutables), $\alpha \in \mathbb{N}$, se rajoute à la durée d'exécution restante pour cette tâche (resp. au nombre de symboles "$e$" exécutables pour cette otâche). Pour distinguer ces symboles "$e$" particuliers liés à la préemption de ceux correspondants à la otâche elle même, nous les noterons "**ĕ**" et graphiquement, nous les représenterons par un slot noir "■" dans la représentation linéaire (*GantChart*) et un *secteur noir* dans la représentation circulaire (*Dameid*). Cette quantité $\alpha$ correspond précisément à la durée de la *restauration du contexte* de la tâche (resp. de la otâche) qui a été préemptée. Comme nous l'avons déjà souligné dans le chapitre 2, l'opération de restauration du contexte d'une tâche (resp. d'une otâche) est *atomique*, c'est-à-dire si une tâche (resp. une otâche) est préemptée pendant cette opération de restauration de contexte, alors il faudra reprendre toute l'opération à *zéro* lors de la prochaine restauration de contexte.

Puisqu'une tâche (resp. une otâche) peut être préemptée plusieurs fois lors de son exécution (resp. lors du remplacement des symboles "$a$" du premier opérande par des symboles "$e$" exécutables), il est évident qu'une mauvaise quantification de ce coût temporel (resp. du nombre de symboles "**ĕ**" supplémentaires) qui s'additionne à la durée d'exécution de chaque tâche (resp. au nombre de symboles exécutables de chaque

otâche) lorsqu'elle est préemptée peut conduire à des conclusions erronées concernant l'ordonnançabilité du système de tâches (resp. d'otâches). Une conclusion erronée peut conduire à un mauvais fonctionnement du système lors de son exécution effectif. En effet, un système qui avait été déclaré ordonnançable grâce à l'analyse d'ordonnançabilité pourrait ne pas l'être en définitive à cause du coût de la préemption. Lorsque ce n'est pas la conclusion sur l'ordonnançabilité du système qui est erronnée, c'est la marge qui est d'une part approximative et d'autre part difficile à évaluer de façon rigoureuse qu'autorise habituellement les concepteurs de systèmes temps réel pour contourner le problème qui est trop grande. Cette marge approximative est dûe au fait que le pire temps de réponse d'une tâche (resp. d'une otâche) est strictement supérieur à sa durée d'exécution lorsqu'elle est préemptée dans une instance (resp. dans un $T_j$*-mésoid*). Ainsi, cette marge conduit inévitablement à un gaspillage inutile des ressources car la difficulté majeure est la détermination du *nombre exact* de préemptions de chaque tâche (resp. chaque otâche). En effet il faut prendre en compte le coût de chaque préemption lors de l'analyse et le fait que la restauration du contexte d'une tâche (resp. d'une otâche) est *atomique*.

Avant de donner la définition de l'opération d'ordonnancement $\oplus$ avec coût exact de préemption, nous allons énoncer deux théorèmes importants. Le premier théorème concerne l'instant critique et le second théorème concerne l'importance de la phase transitoire lorsque le coût de la préemption est pris en compte lors de l'analyse d'ordonnançabilité d'un système. Ensuite nous proposerons un nouvel algorithme qui prend en compte le nombre exact de préemptions d'une tâche dans chaque instance (resp. d'une otâche dans chaque $T_j$*-mésoid*). Enfin, nous proposerons de nouvelles conditions d'ordonnançabilité qui prennent en compte le coût exact dû aux préemptions lors de l'analyse d'ordonnançabilité. Ces nouvelles conditions garantiront toujours un bon fonctionnement du système lors de son exécution effectif et élimineront les gaspillages de la ressource (CPU) puisque les marges superflues sont inutiles. Par soucis de clarté et sans nuire à la généralité, même si cette hypothèse n'est pas réaliste, nous allons considérer dans tous les exemples un coût de préemption $\alpha$ *constant* pour chaque tâche (resp. otâche). Dans certains exemples le coût $\alpha$ d'une préemption sera arbitrairement élevé par rapport aux durées d'exécution des tâches (resp. des otâches). Ce coût élevé associé à l'occurrence d'une préemption sera utilisé pour illustrer l'impact et surtout le risque de ne pas prendre en compte le coût temporel lié aux préemptions dans la condition d'ordonnançabilité du système de tâches (resp. d'otâches) périodiques considéré.

**Théorème 9** *Soit $\Gamma_n$ un système constitué de $n$ tâches périodiques indépendantes $\tau_i = (r_i^1, C_i, D_i, T_i),\ i = 1, \cdots, n$ et $O\Gamma_n$ le système d'otâches correspondant. L'*instant critique *de $\Gamma_n$ (resp. de $O\Gamma_n$) lorsque le coût de la préemption de chaque tâche (resp. de chaque otâche) est pris en compte n'est pas forcément tel que la date de première activation de toutes les tâches (resp. otâches) soit simultané, c'est-à-dire $r_i^1 = 0, i = 1, \cdots, n$.*

**Preuve** :

Soit $\Gamma_2$ le système constitué de 2 tâches périodiques indépendantes dont les caractéristiques sont données dans la table 5.1. Il suffit de considérer $\alpha = 1$ unité de temps pour toutes les tâches. Nous supposons que la tâche $\tau_1$ a une priorité supérieure à celle de la tâche $\tau_2$. Nous résumons les résultats obtenus dans la figure 4.2, la courbe du temps de réponse de chaque tâche est illustrée par la figure 4.1.

| Tâche | $r_i^1$ | $C_i$ | $D_i$ | $T_i$ |
|---|---|---|---|---|
| $\tau_1$ | 0 | 2 | 5 | 5 |
| $\tau_2$ | 0 | 2 | 8 | 8 |

TAB. 4.1 – Caractéristiques des tâches

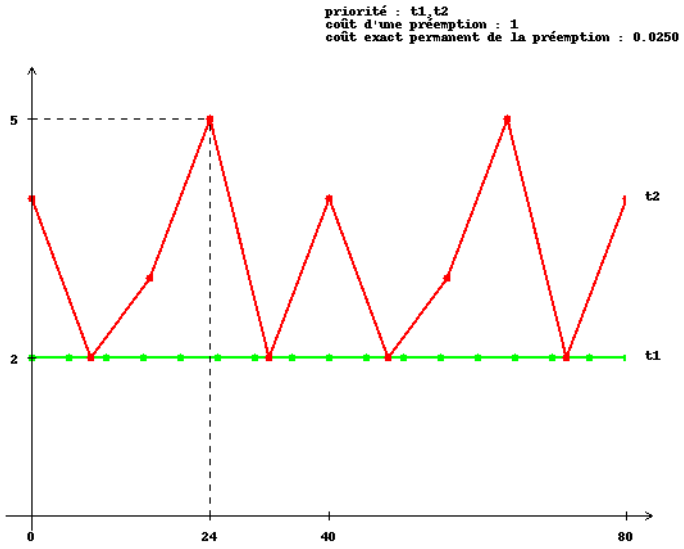


FIG. 4.1 – L'instant critique ne correspond pas forcément au scénario synchrone.

Dans la figure 4.2, le temps de réponse (4 unités de temps) de la première instance de la tâche $\tau_2$ (correspondant à l'instant critique classique) est inférieur au temps de réponse (5 unités de temps) de sa quatrième instance. Ceci est dû au fait que $\tau_2$ a été préemptée dans sa quatrième instance et ainsi le coût de la préemption a été rajouté à la durée d'exécution restante à exécuter dans cette instance. ■

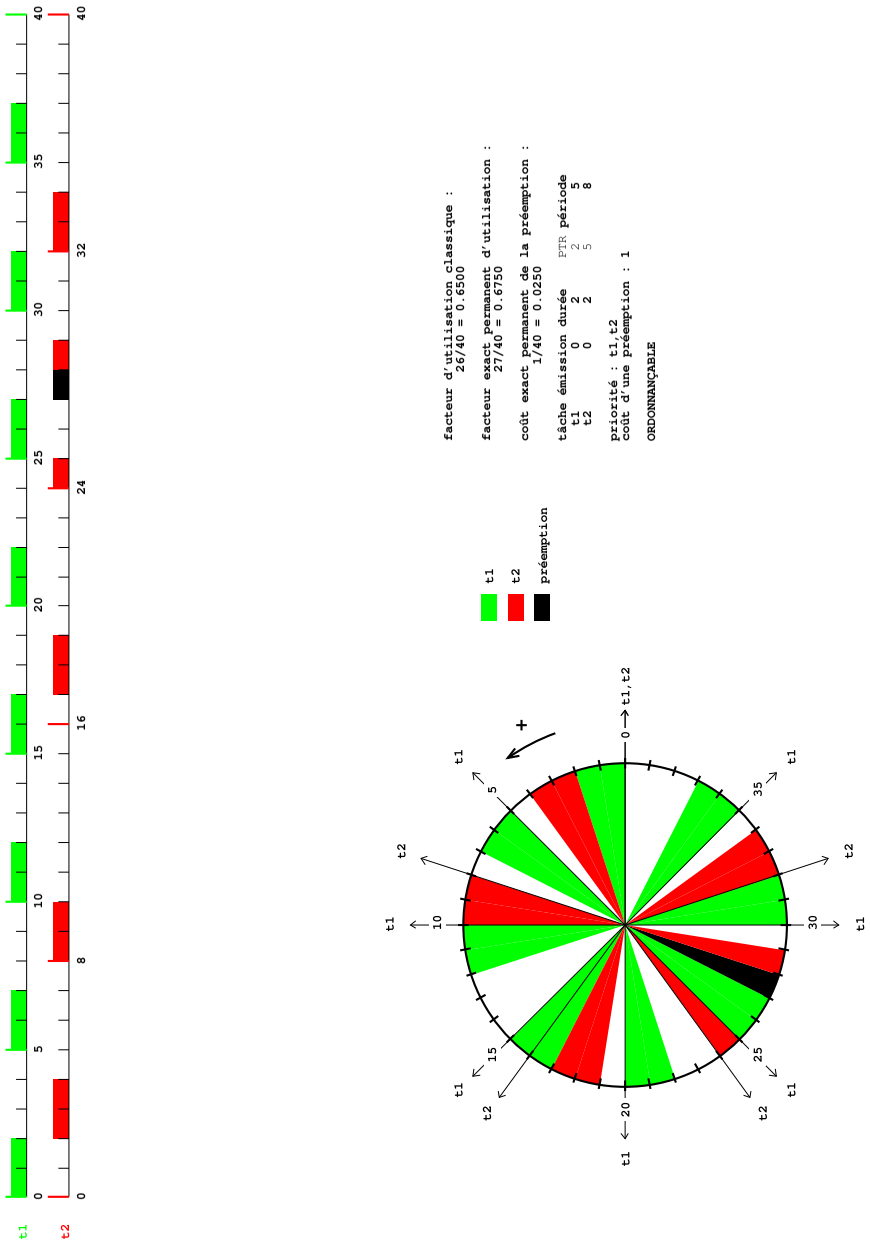


FIG. 4.2 – L'instant critique ne correspond pas forcément au scénario synchrone.

**Théorème 10** *Soit $O\Gamma_n$ un système constitué de $n$ otâches périodiques. Il n'existe pas de suite de dates de premières activations des otâches de $O\Gamma_n$ correspondant à l'*instant critique *de $O\Gamma_n$ lorsque le coût de la préemption de chaque otâche est pris en compte.*

**Preuve** :

La preuve du théorème 10 se fait en deux étapes. Premièrement, si une telle suite de dates de premières activations des otâches correspondant à l'*instant critique* lorsque le coût de la préemption de chaque otâche est pris en compte existait, alors par le théorème 9, elle ne peut pas correspondre à une activation simultanée de toutes les otâches. Deuxièmement, une telle suite de dates de premières activations ne pourrait correspondre à aucun scénario de premières activations *non simultanées* de toutes les otâches. En effet, en considérant le cas pathologique où le coût de chaque préemption vaut zéro pour toutes les otâches, ce scénario de dates de premières activations non simultanées ne correspond clairement pas à l'instant critique [LL73].

■

**Théorème 11** *Soit $\Gamma_n$ un système constitué de $n$ tâches périodiques indépendantes $\tau_i = (r_i^1, C_i, D_i, T_i),\ i = 1, \cdots, n$ et $O\Gamma_n$ le système d'otâches correspondant. La phase transitoire peut avoir un comportement* pire *que celui de la phase permanente du point de vue du pire temps de réponse de chaque tâche (resp. de chaque otâche).*

**Preuve** :

Soit $\Gamma_3$ le système constitué de 3 tâches périodiques indépendantes dont les caractéristiques sont données dans la table 4.2. Il suffit de considérer $\alpha = 2$ unités de temps pour toutes les tâches. Nous supposons que la tâche $\tau_3$ a une priorité supérieure à celle de la tâche $\tau_2$ et $\tau_2$ a une priorité supérieure à celle de la tâche $\tau_1$. Ce système est ordonnançable et nous résumons les résultats dans la figure 4.4, la courbe du temps de réponse de chaque tâche est illustrée par la figure 4.3.

| Tâche | $r_i^1$ | $C_i$ | $D_i$ | $T_i$ |
|---|---|---|---|---|
| $\tau_1$ | 0 | 2 | 8 | 8 |
| $\tau_2$ | 1 | 1 | 6 | 6 |
| $\tau_3$ | 8 | 1 | 4 | 4 |

TAB. 4.2 – Caractéristiques des tâches

Dans la figure 4.4, la phase transitoire correspond à l'intervalle délimité par les dates $t = 0$ et $t = 6$, la phase permanente débute à la date $t = 6$ et est périodique de période $H_3 = 24$. Le temps de réponse (5 unités de temps) de la première instance de la tâche $\tau_1$

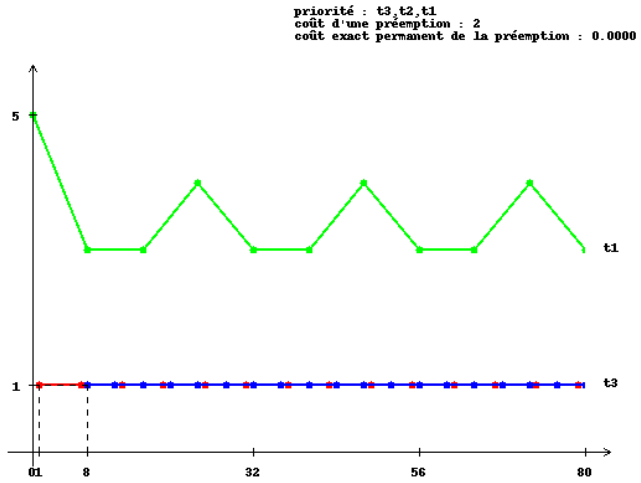


FIG. 4.3 – La phase transitoire est *pire* que la phase permanente.

(appartenant à la phase transitoire) est strictement supérieur à son pire temps de réponse (4 unités de temps) dans la phase permanente (zone mise en évidence). Une fois de plus, ceci est dû au fait que $\tau_1$ a été préemptée dans sa première instance et ainsi le coût de la préemption a été rajouté à la durée d'exécution restante à exécuter dans cette instance. ■

**Théorème 12** *Soit $\Gamma_n$ un système constitué de $n$ tâches périodiques indépendantes $\tau_i = (r_i^1, C_i, D_i, T_i), i = 1, \cdots, n$ et $O\Gamma_n$ le système d'otâches correspondant. Soit $\mathcal{S}$ un choix de priorités des tâches (resp. des otâches) de $\Gamma_n$ (resp. de $O\Gamma_n$). Le résultat de l'analyse d'ordonnançabilité pour la tâche $\tau_i$ (resp. pour la otâche $o\tau_i$) peut changer si deux tâches (resp. deux otâches) qui ont une priorité plus élevée que celle de $\tau_i$ (resp. $o\tau_i$) échangent leurs priorités. En d'autres termes, le résultat de l'analyse d'ordonnançabilité d'une tâche $\tau_i$ (resp. d'une otâche $o\tau_i$) suivant un choix de priorités $\mathcal{S}$ dépend non seulement de l'ensemble des tâches (resp. otâches) ayant une priorité plus élevée que $\tau_i$ (resp. $o\tau_i$) mais aussi de leurs priorités exactes.*

**Preuve** :

Soit $\Gamma_3$ le système constitué de 3 tâches périodiques indépendantes dont les caractéristiques sont données dans la table 4.3. Il suffit de considérer $\alpha = 2$ unités de temps pour toutes les tâches. Nous considérons le choix de priorités $\mathcal{S}$ dans lequel la tâche $\tau_1$ a une priorité supérieure à celle de la tâche $\tau_2$ et la tâche $\tau_2$ a une priorité supérieure à celle de la tâche $\tau_3$. Suivant ce choix de priorités, le système est ordonnançable et donc

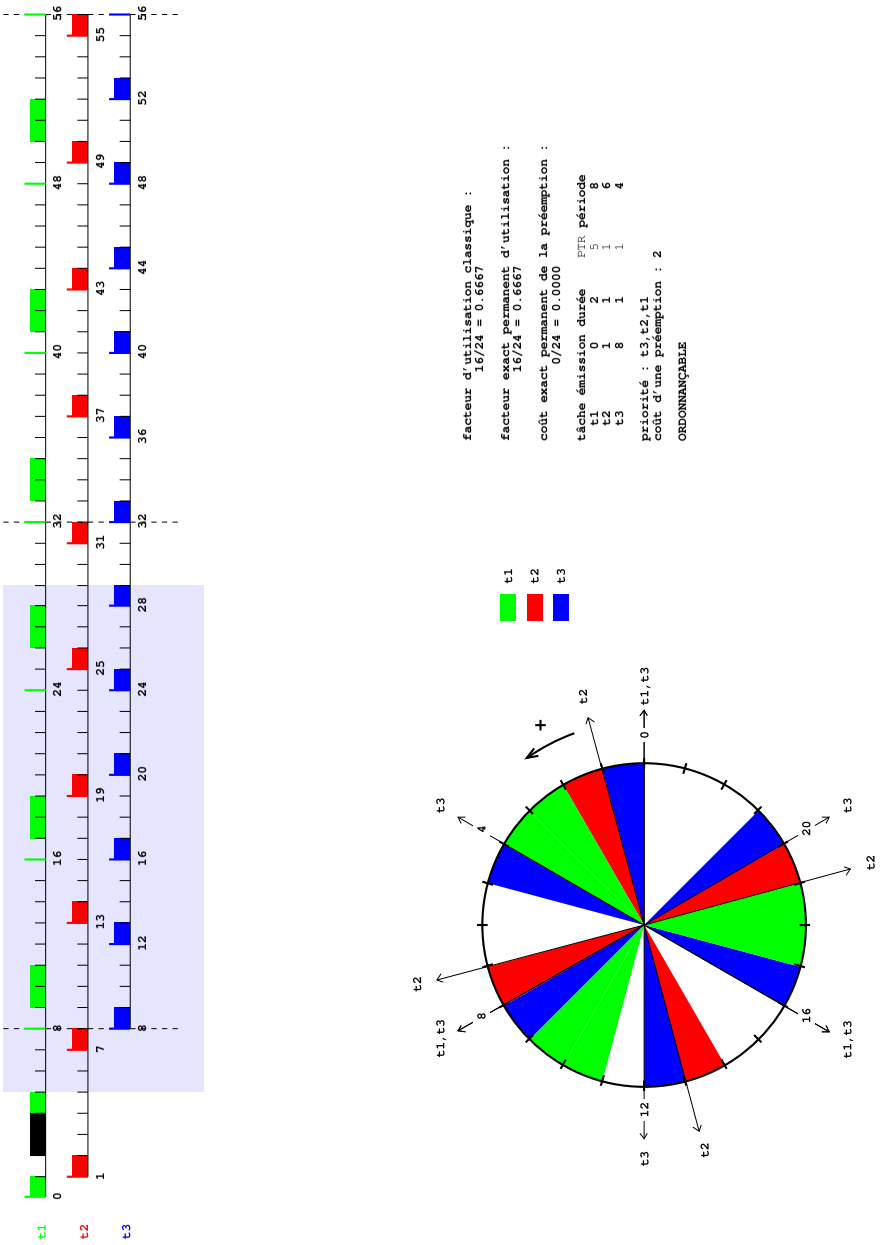


FIG. 4.4 – La phase transitoire est *pire* que la phase permanente.

en particulier la tâche $\tau_3$. Nous résumons les résultats dans la figure 4.5, la courbe du temps de réponse de chaque tâche est illustrée par la figure 4.6.

| Tâche | $r_i^1$ | $C_i$ | $D_i$ | $T_i$ |
|---|---|---|---|---|
| $\tau_1$ | 0 | 3 | 12 | 12 |
| $\tau_2$ | 8 | 2 | 6 | 6 |
| $\tau_3$ | 5 | 2 | 8 | 8 |

TAB. 4.3 – Caractéristiques des tâches

Maintenant nous considérons le choix des priorités dans lequel les tâches $\tau_1$ et $\tau_2$ échangent leurs priorités, c'est-à-dire la tâche $\tau_2$ a une priorité supérieure à celle de la tâche $\tau_1$ et la tâche $\tau_1$ a une priorité supérieure à celle de la tâche $\tau_3$. Suivant ce choix des priorités, le système n'est pas ordonnançable : en particulier à cause de la tâche $\tau_3$. Nous résumons les résultats dans la figure 4.7. Dans cette figure, la seconde échéance de la tâche $\tau_3$ est dépassée à la date $t = 21$. Ce dépassement est dû au fait que la tâche $\tau_2$ a préemptée la tâche $\tau_1$ (par hypothèse la priorité de $\tau_2$ est supérieure à celle de $\tau_1$) à la date $t = 14$ et à cause du coût lié à la préemption, celà a été fatale pour la tâche $\tau_3$ (par hypothèse la priorité de $\tau_2$ est supérieure à celle de $\tau_3$). Par conséquent, le résultat de l'analyse d'ordonnançabilité de la tâche $\tau_3$ dépend non seulement de l'ensemble des tâches ayant une priorité plus élevée que $\tau_3$ mais aussi de leurs priorités exactes.

■

**Théorème 13** *Soit $\Gamma_n$ un système constitué de $n$ tâches périodiques indépendantes $\tau_i = (r_i^1, C_i, D_i, T_i), i = 1, \cdots, n$ et $O\Gamma_n$ le système d'otâches correspondant. Soit $\mathcal{S}$ un choix de priorités des tâches (resp. des otâches) de $\Gamma_n$ (resp. de $O\Gamma_n$). Si la tâche $\tau_i$ (resp. la otâche $o\tau_i$) n'est pas ordonnançable suivant le choix de priorités $\mathcal{S}$ dans lequel la tâche $\tau_j$ (resp. $o\tau_j$) est moins prioritaire que $\tau_i$ (resp. $o\tau_i$), alors la tâche $\tau_i$ (resp. $o\tau_i$) peut être ordonnançable pour le choix de priorités $\mathcal{S}'$ dans lequel $\tau_i$ et $\tau_j$ (resp. $o\tau_i$ et $o\tau_j$) échangent leurs priorités. En d'autres termes, si la tâche $\tau_i$ (resp. la otâche $o\tau_i$) n'est pas ordonnançable, nous pouvons la rendre ordonnançable en abaissant sa priorité.*

**Preuve** :

Soit $\Gamma_3$ le système constitué de 3 tâches périodiques indépendantes dont les caractéristiques sont données dans la table 4.4. Il suffit de considérer $\alpha = 2$ unités de temps pour toutes les tâches. Nous considérons le choix de priorités $\mathcal{S}$ dans lequel la tâche $\tau_1$ a une priorité supérieure à celle de la tâche $\tau_2$ et la tâche $\tau_2$ a une priorité supérieure à celle de la tâche $\tau_3$. Suivant ce choix de priorités, le système n'est pas ordonnançable et en particulier à cause de la tâche $\tau_2$. Nous résumons les résultats dans la figure 4.8.

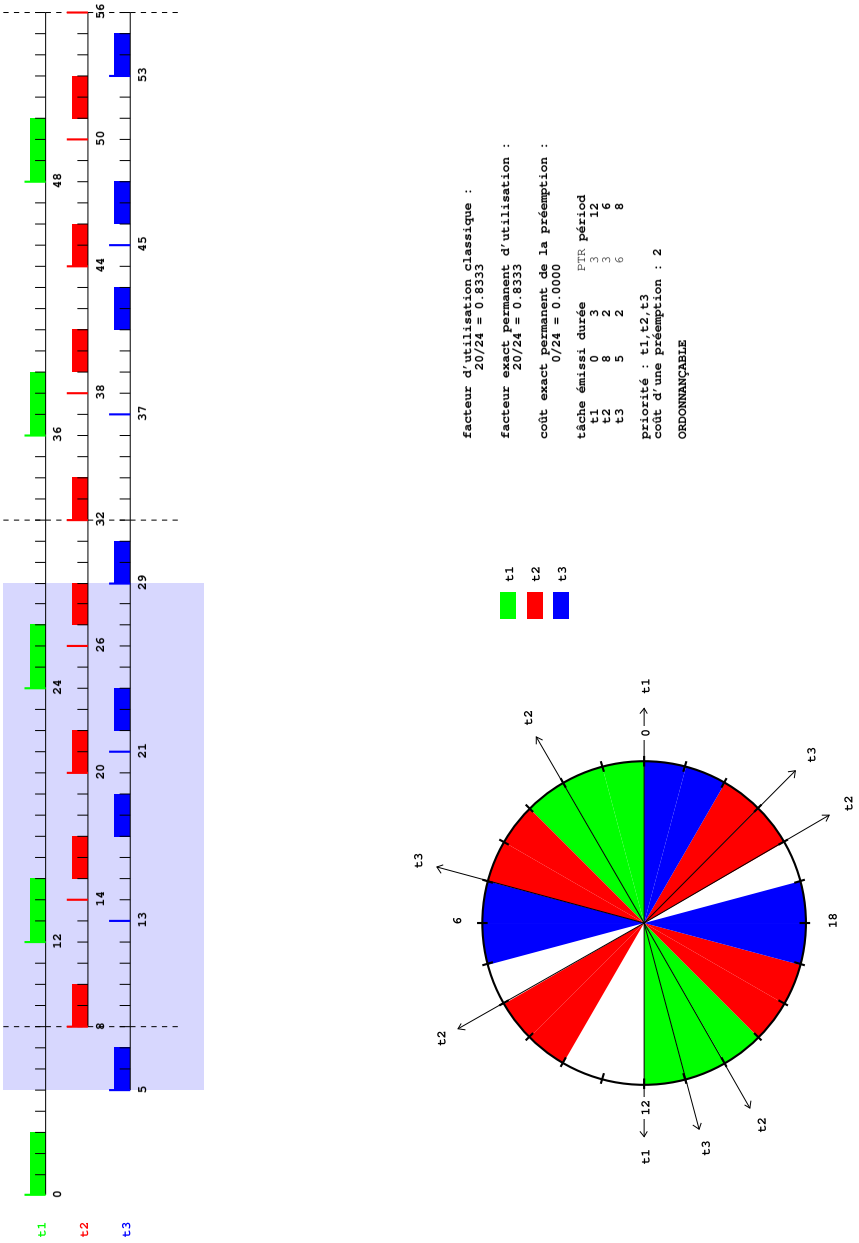


FIG. 4.5 – Suivant le choix de priorités $\mathcal{S}$, la tâche $\tau_3$ est ordonnançable.

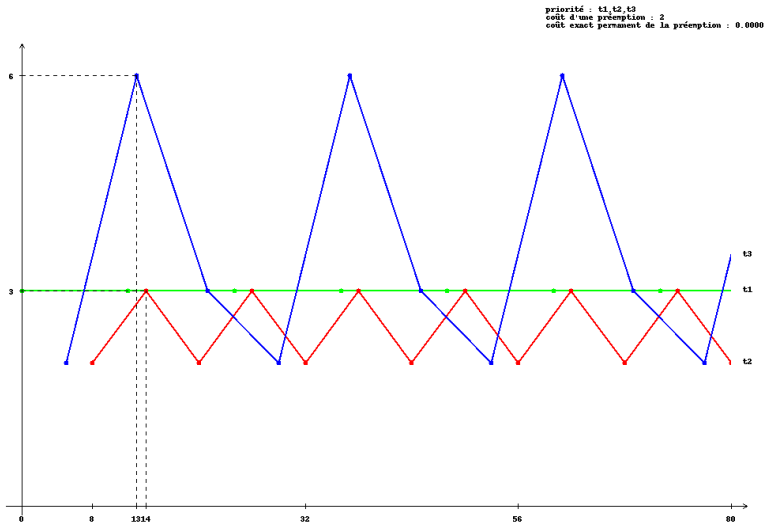


FIG. 4.6 – Courbe du temps de réponse en fonction du temps.

Dans cette figure, la première échéance de la tâche $\tau_2$ est dépassée à la date $t = 5$. Cette situation est due au fait que la tâche $\tau_2$ a été préemptée par la tâche $\tau_1$ à la date $t = 1$ et à cause du coût lié à la préemption, celà a été fatale pour la tâche $\tau_2$.

| Tâche | $r_i^1$ | $C_i$ | $D_i$ | $T_i$ |
|---|---|---|---|---|
| $\tau_1$ | 1 | 1 | 4 | 4 |
| $\tau_2$ | 0 | 3 | 5 | 8 |
| $\tau_3$ | 0 | 1 | 8 | 8 |

TAB. 4.4 – Caractéristiques des tâches

Maintenant nous considérons le choix de priorités $\mathcal{S}'$ dans lequel la tâche $\tau_2$ et la tâche $\tau_3$ échangent leurs priorités, c'est-à-dire la tâche $\tau_1$ a une priorité supérieure à celle de la tâche $\tau_3$ et la tâche $\tau_3$ a une priorité supérieure à celle de la tâche $\tau_2$. Suivant ce choix de priorités, le système est ordonnançable et donc en particulier la tâche $\tau_2$ est ordonnançable. En effet dans ce cas, la préemption que subissait la tâche $\tau_2$ à la date $t = 1$ est maintenant évitée grâce à l'exécution de la tâche $\tau_3$ à la date $t = 0$ (la tâche $\tau_3$ a une priorité supérieure à celle de la tâche $\tau_2$). La tâche $\tau_2$ ne débute son exécution qu'à la date $t = 2$ puisqu'elle a la priorité la plus faible et cette fois, elle ne subit aucune préemption. Par conséquent nous avons rendu la tâche $\tau_2$ ordonnançable en abaissant sa priorité. Nous résumons les résultats dans la figure 4.9, la courbe du temps de réponse de chaque tâche est illustrée par la figure 4.10.

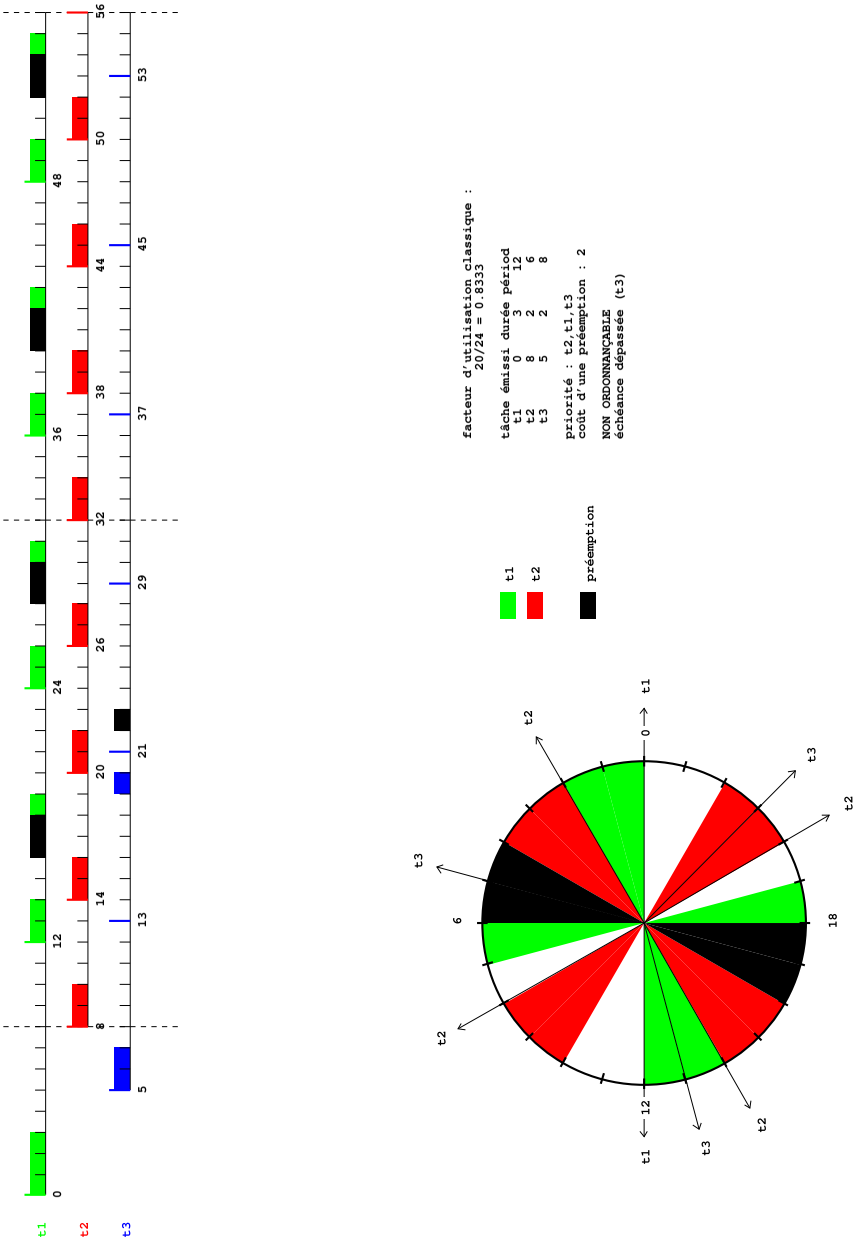


FIG. 4.7 – $\tau_1$ et $\tau_2$ échangent leurs priorités, la tâche $\tau_3$ n'est pas ordonnançable.

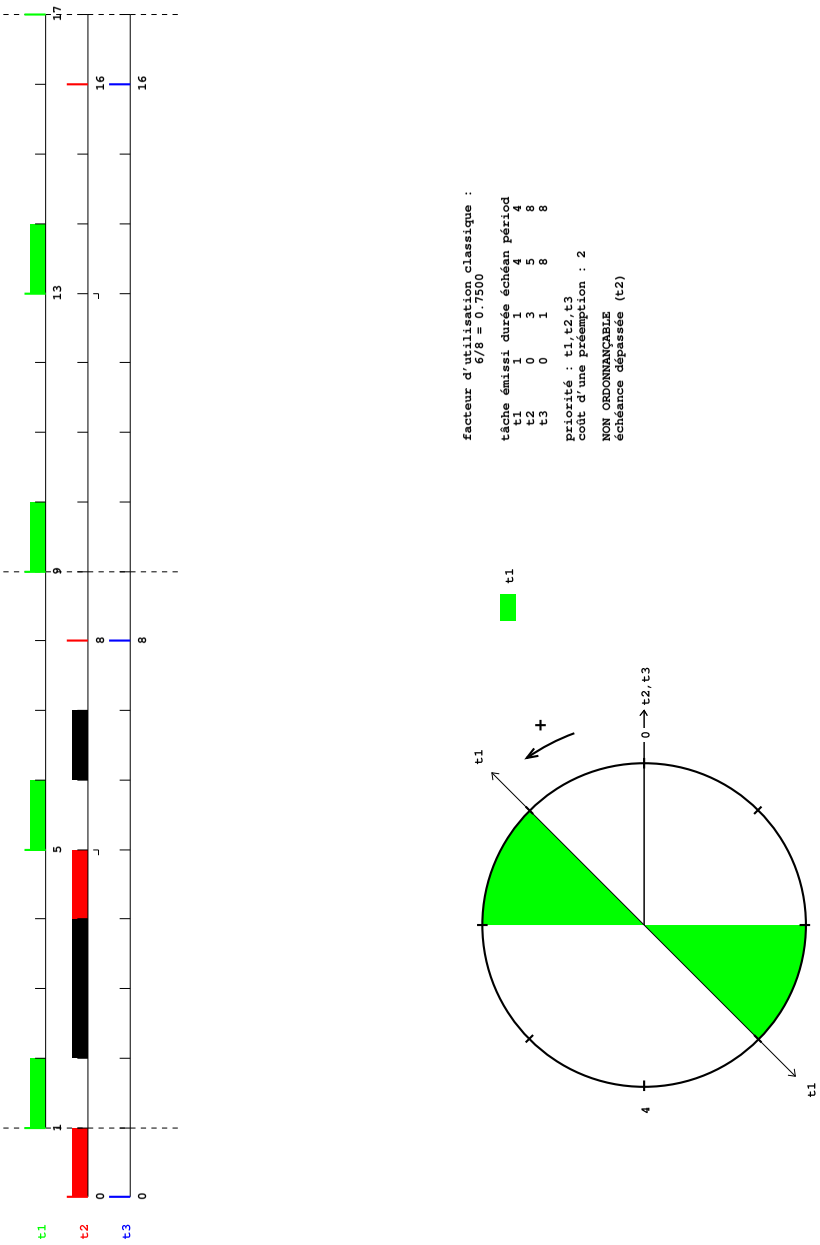


FIG. 4.8 – Suivant le choix de priorités $\mathcal{S}$, la tâche $\tau_2$ n'est pas ordonnançable.

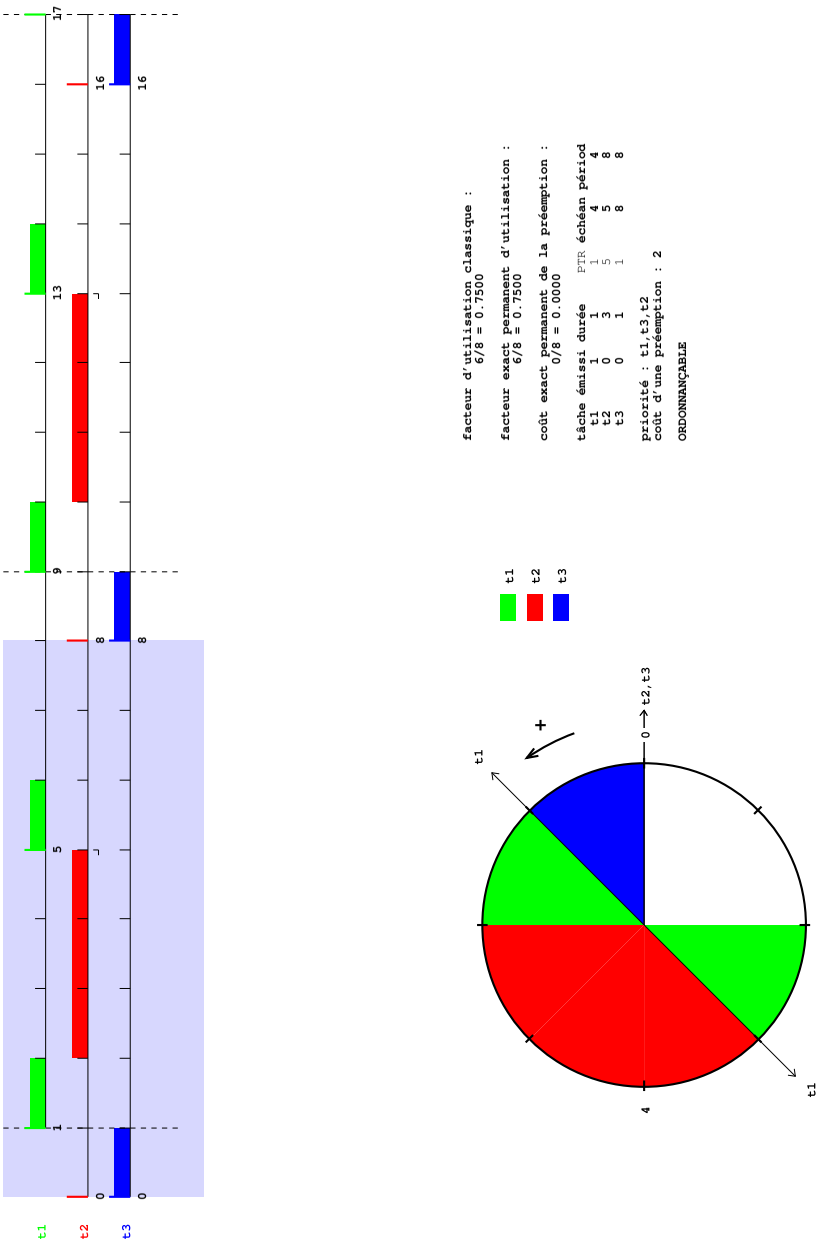


FIG. 4.9 – $\tau_2$ et $\tau_3$ échangent leurs priorités, la tâche $\tau_2$ est ordonnançable.

■

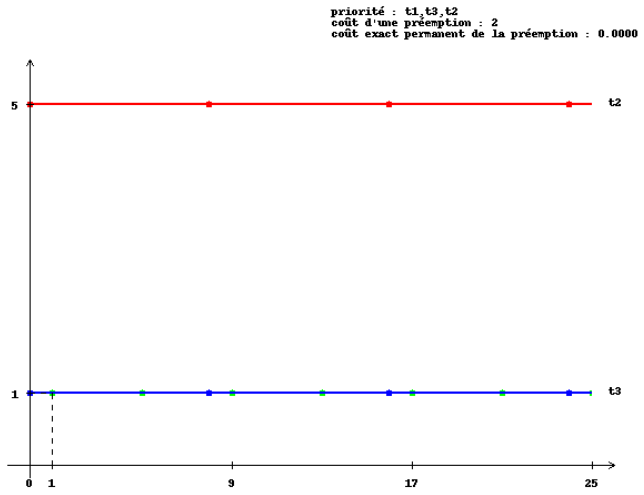


FIG. 4.10 – Courbe du temps de réponse en fonction du temps.

**Théorème 14** *Soit $\Gamma_n$ un système constitué de $n$ tâches périodiques indépendantes $\tau_i = (r_i^1, C_i, D_i, T_i),\ i = 1, \cdots, n$ et $O\Gamma_n$ le système d'otâches correspondant. L'algorithme de choix des priorités des tâches (resp. aux otâches) d'*Audsley *n'est pas optimal dès que le coût de la préemption n'est pas négligeable.*

**Preuve** :

La preuve du théorème 14 découle directement du théorème 12 et du théorème 13. En effet les hypothèses nécessaires pour l'application de l'algorithme d'*Audsley* (voir chapitre 2) ne sont plus vérifiées. ■

Grâce à tous les théorèmes que nous avons présenté jusqu'ici, nous constatons que le coût de la préemption peut avoir un impact important sur l'analyse d'ordonnançabilité et sur l'ordonnancement d'un système donné. De façon évidente, il peut arriver que nous déclarons à la fin de notre analyse d'ordonnançabilité qu'un système est ordonnançable alors qu'en fait, il ne l'est pas et n'a aucune chance de l'être. De plus, ayant montré grâce au théorème 12 et au théorème 13 que nous ne pouvions pas utiliser l'algorithme de choix des priorités d'*Audsley*, il n'existe pas, à notre connaissance, dans la littérature un algorithme ou une technique *optimal* de choix des priorités lorsqu'on prend en compte le coût de la préemption. Nous rappelons que le terme *optimal* fait référence à la capacité à conduire à un ordonnancement valide s'il en existe un. Nous proposerons un algorithme optimal de choix de priorités dans la suite de ce chapitre, section 4.5.

## 4.3 Opération $\oplus$ d'ordonnancement d'un système d'otâches périodiques avec coût exact de la préemption

Grâce au théorème 7, la *phase permanente* d'un système ordonnançable se répète infiniment à partir d'une certaine date $R$. Cette assertion implique que l'intervalle précédent la date $R$ contient nécessairement la *phase transitoire*. Ainsi il suffit de faire l'analyse d'ordonnançabilité dans l'intervalle précédent la date $R$ et dans l'intervalle $[R, R + H_n]$ où $H_n$ correspond au plus petit commun multiple (*ppcm*) des périodes des tâches (resp. des otâches). Puisque le pire temps de réponse de chaque tâche $\tau_j$ (resp. otâche $o\tau_j$) peut survenir indifféremment dans les deux phases (voir figure 4.4), nous devons considérer toutes les instances (resp. tous les $T_j$-*mésoids*) jusqu'à la fin de la première phase permanente pour l'analyse d'ordonnançabilité du système. Parce que notre but est de prendre en compte le coût exact de la préemption, et comme toutes les tâches (resp. otâches) sauf celle ayant la priorité la plus élevée peuvent être préemptées, l'analyse que nous proposons donne une condition d'ordonnançabilité pour chaque tâche (resp. otâche) individuellement relativement à celles ayant une priorité plus élevée. Nous prennons en compte le nombre exact de preemptions dans chaque instance (resp. dans chaque $T_j$-*mésoid*) pour chaque tâche (resp. chaque otâche). S'il paraît clairement que le nombre de préemptions d'une tâche $\tau_j$ (resp. d'une otâche canonique régulière $o\tau_j$) peut varier d'une instance à l'autre (resp. d'un $T_j$-*mésoid* à un autre), alors la durée d'exécution varie aussi d'une instance à l'autre pour la même tâche (resp. d'un $T_j$-*mésoid* à un autre pour la même otâche). Pour cette raison, nous allons introduire la notion de *PET* (*Preempted Execution Time*) d'une tâche $\tau_j$ (resp. d'une otâche canonique régulière $o\tau_j$).

**Définition 36** *Le* PET *(*Preempted Execution Time*) d'une tâche $\tau_j$ (resp. d'une otâche canonique régulière $o\tau_j$) dans une instance donnée (resp. dans un $T_j$-*mésoid*) correspond à la somme de la durée d'exécution de la tâche (resp. otâche) et du coût exact due à la préemption.*

**Notations** :

1. Pour une tâche périodique $\tau_j = (r_j^1, C_j, D_j, T_j)$ dont la otâche correspondante $o\tau_i$ est donnée par la relation 3.8, $C_j^k$ désignera le *PET* de sa $k^{ieme}$ instance (resp. de son $T_j$-*mésoid* de rang $k$).
2. Pour une otâche périodique quelconque $o\tau_i$ illustrée par la figure 3.5, $C_i^{l,k}$ désignera le *PET* correspondant à la sous-durée d'exécution de rang $k \in \{1, \cdots, n_i\}$ dans le $T_i$-*mesoid* de rang $l$.

La figure 4.11 illustre la définition du *PET* d'une tâche (resp. d'une otâche). Il apparaît donc clairement que sa valeur dépend du nombre de préemptions dans chaque instance (resp. dans chaque $T_j$-*mésoid*). Son calcul dans une instance (resp. dans un $T_j$-*mésoid*) est expliqué en détail dans la procédure 5 ci-dessous.

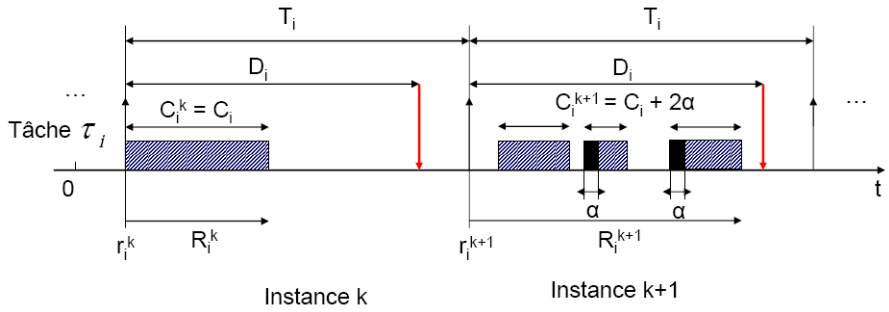


FIG. 4.11 – Illustration du *PET* d'une tâche (resp. d'une otâche canonique régulière).

Puisque chaque otâche canonique régulière $o\tau_j$ ne peut être préemptée que par les otâches ayant une priorité plus élevée, alors l'*hyperpériode de niveau* $j$ correspond à l'entier $H_j$ définit par $H_j = ppcm\{T_l \ : \ o\tau_l \in hp(o\tau_j)\}$ où $T_l$ désigne la période de la otâche $o\tau_l$ et $hp(o\tau_j)$ désigne le sous ensemble des otâches ayant une priorité supérieure à celle de $o\tau_j$ dans le système considéré. Grâce à cette définition, le nombre de $T_j$-*mesoids* permettant de définir la période de la otâche résultat au niveau $j$, et donc la phase permanente, lorsque la otâche $o\tau_j$ est le second opérande de l'opération $\oplus$ est donné par la relation 4.1.

$$\sigma_{perm_j} = \frac{H_j}{T_j} = \frac{ppcm\{T_l \ : \ o\tau_l \in hp(o\tau_j)\}}{T_j} \tag{4.1}$$

Nous rappelons que le *facteur d'utilisation permanent du processeur sans coût de préemption* d'un système $O\Gamma_n$ constitué de $n$ otâches périodiques canoniques quelconques $o\tau_j = \zeta_{(r_j,\beta_j)}\tau^{\infty}_{\beta_j}$ de période $T_j$ est donné par la relation 4.2.

$$U_n = \sum_{j=1}^{n} \frac{C_j}{T_j} \text{ avec } C_j = \sum_{k=1}^{n_j} C^k_{p_j} \tag{4.2}$$

Dans la relation 4.2, $C_j$ et $n_j$ désignent respectivement la somme des sous-durées d'exécution de rang $k \in \{1, \cdots, n_j\}$ et le nombre de séquences du symbole "$e$" du motif de la otâche $o\tau_j$. Nous rappelons que si $U_n > 1$ alors quelque soit l'algorithme utilisé pour choisir les priorités des otâches du système considéré, ce système ne peut pas être ordonnançable. Cette assertion implique qu'une des conditions nécessaires à l'ordonnançabité du système est $U_n \leq 1$. Comme nous l'avons déjà mentionné, toute otâche canonique régulière $o\tau_j$, exceptée la plus prioritaire du système, peut être préemptée et le nombre de préemptions peut varier d'un $T_j$-*mesoid* à un autre lorsqu'on effectue l'opération $\oplus$ au niveau $j$. Celà implique donc qu'il existe $\sigma_{perm_j}$ *PETs* différents qui déterminent la

phase permanente au niveau $j$. En d'autres termes, à chaque sous-durée d'exécution de rang $k \in \{1, \cdots, n_j\}$, on peut définir une fonction $f_k$.

$$\begin{array}{rcl} f_k : \mathbb{N}^+ & \longrightarrow & \mathbb{N}^{+\sigma_{perm_j}} \\ C_{p_j}^k & \longrightarrow & (C_{p_j}^{1,k}, C_{p_j}^{2,k}, \cdots, C_{p_j}^{\sigma_{perm_j},k}) \end{array}$$

où $C_{p_j}^{l,k}$ est le *PET* correspondant à la sous-durée d'exécution $C_{p_j}^k$ dans le $T_j$*-mesoid* de rang $l \in \{1, \cdots, \sigma_{perm_j}\}$ de la phase permanente.

**Définition 37** *Le* facteur exact permanent d'utilisation du processeur $U_j^*$ *d'une otâche périodique* $o\tau_j = \zeta_{(r_j,\beta_j)}\tau_{\beta_j}^{\infty}$ *de période* $T_j$ *ayant* $n_j$ *séquences de symboles "*$e$*" dans sa partie périodique est donné par la relation 4.3.*

$$U_j^* = \frac{C_j^*}{T_j} \ \textit{avec} \ C_j^* = \sum_{k=1}^{n_j} \frac{1}{\sigma_{perm_j}} \sum_{l=1}^{\sigma_{perm_j}} C_{p_j}^{l,k} \tag{4.3}$$

*Dans la relation 4.3,* $\sigma_{perm_j} = \dfrac{H_j}{T_j}$ *avec* $H_j = ppcm\{T_l : \tau_l \in hp(o\tau_j)\}$, $C_j^{l,k}$ *est le* PET *correspondant à la sous-durée d'exécution de rang* $k \in \{1, \cdots, n_j\}$ *dans le* $T_j$-mesoid *de rang* $l \in \{1, \cdots, \sigma_{perm_j}\}$ *de la phase permanente.*

**Définition 38** *Le* facteur exact permanent d'utilisation du processeur *d'un système* $\Gamma_n$ *constitué de* $n$ *otâches périodiques* $\tau_j$ *de période* $T_j$ *est donné par la relation 4.4.*

$$U_n^* = \sum_{j=1}^{n} U_j^* \ \textit{avec} \ U_j^* = \frac{1}{T_j} \cdot \sum_{k=1}^{n_j} \frac{1}{\sigma_{perm_j}} \sum_{l=1}^{\sigma_{perm_j}} C_{p_j}^{l,k} \tag{4.4}$$

*Dans la relation 4.4,* $U_j^*$ *est donnée par la définition 4.3.*

**Remarque 15** *Si* $\alpha = 0$ *dans la procédure 5, alors* $C_{p_j}^{l,k} = C_{p_j}^k, \forall l \in \{1, \cdots, \sigma_j\}$ *et les relations 4.3 et 4.4 se réduisent bien respectivement aux relations 3.6 et 3.7. Il en découle que le* coût exact permanent de la préemption $\epsilon_n$ *d'un système constitué de* $n$ *otâches periodiques est donné par la relation 4.5 lorsque ce système est ordonnançable.*

$$\epsilon_n = U_n^* - U_n = \sum_{j=1}^{n} \left( \frac{1}{T_j} \cdot \sum_{k=1}^{n_j} \frac{1}{\sigma_j} \sum_{l=1}^{\sigma_j} C_{p_j}^{l,k} \right) - \sum_{j=1}^{n} \left( \frac{1}{T_j} \cdot \sum_{k=1}^{n_j} C_{p_j}^k \right) \tag{4.5}$$

Sachant que les préemptions potentielles de la otâche $o\tau_j$ sont répérées par les transitions $(a \rightarrow e)$ dans chaque $T_j$*-mesoid* du premier opérande lorsqu'on effectue l'opération $\oplus$ au niveau $j$, le calcul du *PET* d'une otâche canonique régulière dans un $T_j$*-mesoid*

est résumé dans la procédure 5. Dans cette procédure, $\{\mathbf{expression}\}$ désigne une phase de la procédure, le caractère "%" désigne le début d'un commentaire pour expliquer soit une variable utilisée dans la procédure soit l'idée d'une section de la procédure. Maintenant grâce à toutes les notions que nous avons présentées jusqu'ici, puisqu'il suffit de considérer le cas où le premier opérande de l'opération $\oplus$ est une otâche périodique et le second opérande est une otâche périodique avec une partie initiale qui vaut $\Lambda$, alors le résultat du calcul de l'expression $o\tau_i \oplus o\tau_j$ avec *coût exact de la préemption* est obtenu en effectuant séquentiellement les trois grandes procédures ci-dessous : la procédure 6, la procédure 7 et la procédure 8.

**Procédure 5 :** ***Calcul du PET d'une otâche canonique régulière dans un*** $T_j$**-mesoid.**

1: {***Variables nécessaires***}
2: % $rExecution$ : durée d'exécution restante à ordonnancer
3: % $rPreemption$ : durée restante liée au coût de préemption à ordonnancer
4: % $vPET$ : valeur courante du PET de la otâche dans le $T_j$-mésoïd
5: % $\mathcal{U}$ : univers de la otâche dans le $T_j$-mésoïd
6: % $\phi$ : indexe de parcours des symboles dans le $T_j$-mésoïd
7: % $\alpha$ : coût d'une préemption
8: {***Fonction récursive de calcul du PET d'une otâche dans un*** $T_j$**-mésoïd**}
9: **function** PET($rExecution$, $rPreemption$, $vPET$)
% Il reste à remplacer au moins $rExecution + rPreemption$ symboles "$a$" par des
% symboles "$e$" dans le $T_j$-*mésoid* et jusqu'ici on a remplacé $vPET$ symboles "$a$".
10: **si** $rExecution = 0$ **alors**
11: % la otâche est *ordonnançable* dans ce $T_j$-*mésoid*, le PET vaut $vPET$.
12: return $vPET$
13: **sinon**
14: **si** $vPET \leq |\mathcal{U}|$ **alors**
15: **si** $\phi$ = symbole "$e$" **alors**
16: $rPreemption \leftarrow \alpha$
17: à l'aide de l'indexe $\phi$, sauter la séquence de "$e$" courante
18: **finsi**
19: **si** $rPreemption = 0$ **alors**
20: remplacer le symbole "$a$" par un symbole "$e$" exécutable : lié à la otâche
21: à l'aide de l'indexe $\phi$, passer au symbole suivant
22: PET($rExecution - 1$, $rPreemption$, $vPET + 1$)
23: **sinon**
24: remplacer le symbole "$a$" par un symbole "**ĕ**" : lié au coût de la préemption
25: à l'aide de l'indexe $\phi$, passer au symbole suivant

26: PET($rExecution$, $rPreemption - 1$, $vPET + 1$)
27: **finsi**
28: **sinon**
29: % la otâche est *non ordonnançable* dans ce $T_j$*-mésoid*, l'échéance est dépassée.
30: return "*erreur*"
31: **finsi**
32: **finsi**
33: {***Exécution de l'algorithme***}
34: A l'aide de $\phi$, aller au premier symbole "$a$" dans le $T_j$*-mésoïd*
35: Appeler la fonction PET($C_j$, 0, 0)

□

**Procédure 6 :** ***Procédure 1 modifiée lorsque le premier opérande est une otâche et le second opérande a une partie initiale qui vaut*** $\Lambda$**.**

1: Concaténer les préfixes $\{a\}_{(\epsilon,r_i)}^{|\gamma_{ij}|}$ et $\{a\}_{(\epsilon,r_j^1)}^{|\gamma_{ij}|}$ où $\epsilon = min(r_i, r_j^1)$ respectivement aux otâches $o\tau_i = \zeta_{i(r_i,\beta_i)}(\tau_{0,i})_{\beta_i}^{\infty}$ et $o\tau_j = (\tau_{0,j})_{r_j^1}^{\infty}$. Nous obtenons alors

$$o\tau_i \oplus o\tau_j = \{a\}_{(\epsilon,r_i)}^{|\gamma_{ij}|}\zeta_{i(r_i,\beta_i)}(\tau_{0,i})_{\beta_i}^{\infty} \oplus \{a\}_{(\epsilon,r_j^1)}^{|\gamma_{ij}|}(\tau_{0,j})_{r_j^1}^{\infty} = no\tau_i \oplus no\tau_j \tag{4.6}$$

2: Déterminer les dates $s_i^{'}$ et $s_j^{'}$ permettant de définir la phase transitoire et la phase permanente en utilisant la relation 3.17. Puisque $o\tau_i$ est la otâche ayant la priorité la plus élévée par définition, alors

$$\begin{cases} s_i^{'} = \beta_i \\ \\ s_j^{'} = r_j^1 + \left\lceil \dfrac{(s_i^{'} - r_j^1)^+}{T_j} \right\rceil \cdot T_j \end{cases} \tag{4.7}$$

L'intervalle donnée par $[\epsilon, s_j^{'}]$ définit la phase transitoire et l'intervalle donnée par $[s_j^{'}, s_j^{'} + ppcm(T_i, T_j)]$ définit la phase permanente.

3: Déterminer la valeur de $\sigma_i = \left( \left\lceil \dfrac{s_j^{'} - s_i^{'}}{T_i} \right\rceil - 1 \right) + \dfrac{ppcm(T_i, T_j)}{T_i}$. Grâce à la valeur de $\sigma_i$, à la remarque 7 et au théorème 7, écrire $no\tau_i$ comme la concaténation d'une otâche finie *ph-trans* de cardinal $|$*ph-trans*$| = s_j^{'} - \epsilon$ et d'une partie périodique *ph-perm* dont la date de première activation est $s_j^{'}$ et dont la période est $T_i \vee T_j =$

$ppcm(T_i, T_j)$. Les otâches *ph-trans* et *ph-perm* déterminent respectivement la phase transitoire et la phase permanente.

$$no\tau_i = \textit{ph-trans ph-perm}. \tag{4.8}$$

Puisque le cardinal $|\textit{ph-trans}|$ de la otâche finie *ph-trans* peut aussi s'écrire $|\textit{ph-trans}| = s_j^{'} - \epsilon = (r_j^1 - \epsilon) + (s_j^{'} - r_j^1)$, alors la otâche *ph-trans* peut être écrite comme concaténation de deux autres otâches finies

$$\textit{ph-trans} = (tr_1)_{(\epsilon, r_j^1)}(tr_2)_{(r_j^1, s_j^{'})} \tag{4.9}$$

On obtient alors :

$$no\tau_i = (tr_1)_{(\epsilon, r_j^1)}(tr_2)_{(r_j^1, s_j^{'})}\textit{ph-perm} \tag{4.10}$$

□

**Procédure 7 :** ***Procédure 2 modifiée et mésoïdification du premier opérande*.**

1: Au terme de la procédure 6, nous avons obtenu $o\tau_i \oplus o\tau_j = no\tau_i \oplus no\tau_j$ où $no\tau_i$ est donnée par l'équation 4.10 et $no\tau_j$ est canonique régulière. Grâce à la décomposition d'une otâche, nous rappelons qu'effectuer l'opération $\oplus$ entre deux otâches périodiques normalisées quelconques revient à l'appliquer $n_j$ fois où $n_j$ est le nombre de séquences de symboles "$e$" dans $no\tau_j$. Suite à notre restriction, la durée d'exécution de la otâche régulière $no\tau_j$ vaut $C_j$. Ainsi

$$\mathcal{R}_j = o\tau_i \oplus o\tau_j = no\tau_i \oplus no\tau_j \tag{4.11}$$

2: Mésoïdifier la premier opérande de l'opération $\oplus$, c'est-à-dire la otâche $no\tau_i$. Cette opération consiste à écrire la otâche finie $(tr_2)_{(r_j^1, s_j^{'})}$ et le motif de la otâche *ph-perm* dans l'égalité 4.10 respectivement comme des concaténations de $\sigma_{tr_2} = \left\lceil \dfrac{(s_i^{'} - r_j^1)^+}{T_j} \right\rceil$ et $\sigma_{perm} = \dfrac{ppcm(T_i, T_j)}{T_j}$ otâches finies de cardinaux $T_j$ chacun. On obtient alors

$$no\tau_i = (tr_1)_{(\epsilon, r_j^1)} \left(\mathcal{M}_{tr_2,1} \cdots \mathcal{M}_{tr_2,\sigma_{tr_2}}\right) \left(\mathcal{M}_{perm,1} \cdots \mathcal{M}_{perm,\sigma_{perm}}\right)_{s_j^{'}}^{\infty} \tag{4.12}$$

Dans l'équation 4.12, toutes les otâches $\mathcal{M}_{tr_2,k}$, où $k = 1, \cdots, \sigma_{tr_2}$ et $\mathcal{M}_{perm,l}$, où $l = 1, \cdots, \sigma_{perm}$, sont finies de cardinaux $T_j$ et sont appelées des $T_j$-*mésoid* car elles font intervenir la période du second opérande $no\tau_j$. Chaque $T_j$-*mésoid* correspond à une instance de la tâche $\tau_j$.

□

**Procédure 8 :** ***Procédure 4 modifiée lorsque le coût exact de la préemption est considéré.***

1: Extraire les *domaines d'échéance* : $\mathcal{D}_{tr_2,k}$, où $k = 1, \cdots, \sigma_{tr_2}$ et $\mathcal{D}_{perm,l}$, où $l = 1, \cdots, \sigma_{perm}$ de chaque $T_j$-*mésoid* grâce à l'égalité 4.12 et à l'aide de la définition 16. Cette opération consiste chaque fois à considérer l'extraction du préfixe constitué des $D_j$ premiers symboles de chaque $T_j$-*mésoid*. Pour chaque $T_j$-*mésoid*, cette opération est toujours possible puisque $D_j \leq T_j$ par définition.

2: Extraire de chaque domaine d'échéance obtenu à l'étape précédente l'*univers* associé. Les univers correspondent aux otâches finies $\mathcal{U}_{tr_2,k}$, où $k = 1, \cdots, \sigma_{tr_2}$ et $\mathcal{U}_{perm,l}$, où $l = 1, \cdots, \sigma_{perm}$ constituées uniquement des symboles "$a$" du domaine d'échéance correspondant (c'est-à-dire, les unités de temps disponibles). Chaque univers est obtenu par concaténation de toutes sous otâches constituées uniquement de symbôles "$a$" du domaine d'échéance.

3: Puisque $C_j$ désigne la durée d'exécution de la partie périodique de la otâche $no\tau_j$, alors la otâche $no\tau_j$ est *potentiellement ordonnançable* avec prise en compte du coût exact de la préemption si et seulement si

$$C_j \leq \min_{\substack{k \in \{1, \cdots, \sigma_{tr_2}\}, \\ l \in \{1, \cdots, \sigma_{perm}\}}} (|\mathcal{U}_{tr_2,k}|, |\mathcal{U}_{perm,l}|) \tag{4.13}$$

4: Dans chaque $T_j$-*mésoid*, calculer les différents *PET* $C_j^k$ où $k = 1, \cdots, \sigma_{tr_2}$ et $C_j^l$ où $k = 1, \cdots, \sigma_{perm}$ de la otâche $no\tau_j$ à l'aide de la procédure 5.

5: La otâche $no\tau_j$ est *ordonnançable* avec prise en compte du coût exact de la préemption si et seulement si

$$\begin{cases} C_j^k \leq |\mathcal{U}_{tr_2,k}|, \quad \forall k \in \{1, \cdots, \sigma_{tr_2}\} \\ \\ C_j^l \leq |\mathcal{U}_{perm,l}|, \quad \forall l \in \{1, \cdots, \sigma_{perm}\} \end{cases} \tag{4.14}$$

6: Si l'équation 4.14 est satisfaite, alors la otâche $\mathcal{R}_j = no\tau_i \oplus no\tau_j$ est obtenue à partir de l'égalité 4.12, en remplaçant les $C_j^k, k = 1, \cdots, \sigma_{tr_2}$ (resp. les $C_j^l, l = 1, \cdots, \sigma_{perm}$) premiers symboles "$a$" de chaque $T_j$-*mésoid* $\mathcal{M}_{tr_2,k}$ (resp. $\mathcal{M}_{perm,l}$) par des symboles "$e$". Nous obtenons ainsi $\mathcal{R}_j \neq \Lambda$.
Dans le processus de remplacement des symboles, les *temps de réponse* de la otâche $o\tau_j$ correspondent chaque fois au cardinal du préfixe définit par l'indice du dernier symbole "$a$" remplacé. Le *pire temps de réponse* de la otâche $o\tau_j$ correspond au maximum des temps de réponse.

7: Si l'équation 4.14 n'est pas satisfaite, alors $\mathcal{R}_j = \Lambda$. La otâche $o\tau_j$ et le système d'otâches sont déclarés *non ordonnançables*.

□

## 4.4 Exemple

Dans cette section, nous allons appliquer la définition de l'opération $\oplus$ avec coût exact de la préemption au problème d'ordonnancement d'un système de tâches périodiques. On considère à nouveau le système $\Gamma_3 = \{\tau_1, \tau_2, \tau_3\}$ où $\tau_1$ est la tâche la plus prioritaire et $\tau_3$ la tâche la moins prioritaire et dont les caractéristiques des tâches sont résumées dans la table 3.2 du chapitre 3. Nous considérons que le coût d'une préemption vaut une unité de temps pour toutes les tâches, c'est-à-dire $\alpha_i = \alpha = 1$ unité de temps, $i = 1, 2, 3$. Nous rappelons grâce à la relation 3.8 du chapitre 3 que les otâches $o\tau_1$, $o\tau_2$ et $o\tau_3$ correspondantes respectivement aux tâches $\tau_1$, $\tau_2$ et $\tau_3$ sont données par la relation ci-dessous où $o\tau_1$ est la otâche la plus prioritaire et $\tau_3$ la otâche la moins prioritaire.

$$\begin{cases} o\tau_1 = \{e, e, e, a, a, a, a\llcorner a, a, a, a, a, a, a, a\}_0^\infty \\ o\tau_2 = \{e, e, a, a, a, a\}_5^\infty \\ o\tau_3 = \{e, e, e, e, a, a, a, a, a, a\}_3^\infty \end{cases}$$

Le facteur d'utilisation permanent du processeur sans coût de préemption du système vaut $U_3 = 3/15 + 2/6 + 4/10 = 28/30 = 0.9333$ et le résultat $\mathcal{R}_3$ de l'ordonnancement du système $O\Gamma_3 = \{o\tau_1, o\tau_2, o\tau_3\}$ est obtenu par itération successive de la suite

$$\begin{cases} \mathcal{R}_1 = \Lambda \oplus o\tau_1 = o\tau_1 \\ \mathcal{R}_i = \mathcal{R}_{i-1} \oplus o\tau_i, \quad i = 2, 3 \end{cases}$$

**Première itération :** Calcul de l'opération $\mathcal{R}_2 = o\tau_1 \oplus o\tau_2$

Les otâches normalisées $no\tau_1$ et $no\tau_2$ correspondantes sont respectivement données par $no\tau_1 = \{e, e, e, a, a, a, a\llcorner a, a, a, a, a, a, a, a\}_0^\infty$ pour le premier opérande et $no\tau_2 = \{a, a, a, a, a\}_{(0,5)}\{e, e, a, a, a, a\}_5^\infty$ pour le second opérande. Grâce à l'équation 4.7, nous avons :

$$\begin{cases} s_1' = 0 \\ s_2' = 5 + \left\lceil \dfrac{(0-5)^+}{6} \right\rceil \cdot 6 = 5 \end{cases}$$

Nous avons $H_2 = ppcm(15, 6) = 30$, donc l'intervalle $[0, 5]$ détermine la phase transitoire et l'intervalle $[5, 35]$ détermine la phase permanente.

Le calcul de $\sigma_1$ est donné par $\sigma_1 = \left(\left\lceil \frac{5-0}{15} \right\rceil - 1\right) + \frac{ppcm(15,6)}{15} = 2$. Grâce à la valeur de $\sigma_1$, la remarque 7 et le théorème 7, on peut écrire $no\tau_1$ comme la concaténation de la otâche finie *ph-trans* de cardinal $|$*ph-trans*$| = 5$ et d'une partie périodique *ph-perm* dont la date de première activation est 5 et dont la période est $ppcm(15,6) = 30$. On obtient alors :

$$\begin{aligned} no\tau_1 &= \{e,e,e,a,a,a,a,a,a,a,a,a,a,a,a\}_0^\infty \\ &= \{e,e,e,a,a\}_{(0,5)} \\ &\quad \{a,a,a,a,a,a,a,a,a,a,e,e,e,a,a,a,a,a,a,a,a,a,a,a,a,e,e,e,a,a\}_5^\infty \end{aligned}$$

Par identification par rapport à l'égalité 4.10, la otâche normalisée $no\tau_1$ est bien de la forme $no\tau_1 = (tr_1)_{(\epsilon,r_2^1)}(tr_2)_{(r_2^1,s_2')}$*ph-perm* où les termes impliqués sont respectivement donnés par $(tr_1)_{(\epsilon,r_2^1)} = \{e,e,e,a,a\}_{(0,5)}$, $(tr_2)_{(r_2^1,s_2')} = \Lambda$ puisque $s_2^{'} = r_2^1$ et enfin *ph-perm* $= \{a,a,a,a,a,a,a,a,a,a,e,e,e,a,a,a,a,a,a,a,a,a,a,a,a,e,e,e,a,a\}_5^\infty$.

Pour effectuer la mésoïdification de $no\tau_1$, les calculs de $\sigma_{tr_2}$ et $\sigma_{perm_2}$ donnent respectivement $\sigma_{tr_2} = \left\lceil \frac{(0-5)^+}{6} \right\rceil = 0$ et $\sigma_{perm_2} = \frac{ppcm(15,6)}{6} = 5$. Ainsi nous avons :

$$\begin{aligned} no\tau_1 &= \{e,e,e,a,a\}_{(0,5)} \\ &\quad \{a,a,a,a,a,a,a,a,a,a,e,e,e,a,a,a,a,a,a,a,a,a,a,a,a,e,e,e,a,a\}_5^\infty \\ &= \{e,e,e,a,a\}_{(0,5)}(\{a,a,a,a,a,a\}\{a,a,a,a,e,e\}\{e,a,a,a,a,a\} \\ &\quad \{a,a,a,a,a,a\}\{a,e,e,e,a,a\})_5^\infty \end{aligned}$$

$$= \{e,e,e,a,a\}_{(0,5)}\,(\mathcal{M}_{perm,1}\mathcal{M}_{perm,2}\mathcal{M}_{perm,3}\mathcal{M}_{perm,4}\mathcal{M}_{perm,5})_5^\infty$$

où les 6-*mésoids* $\mathcal{M}_{perm,1}$, $\mathcal{M}_{perm,2}$, $\mathcal{M}_{perm,3}$, $\mathcal{M}_{perm,4}$ et $\mathcal{M}_{perm,5}$ sont donnés par $\mathcal{M}_{perm,1} = \{a,a,a,a,a,a\}$, $\mathcal{M}_{perm,2} = \{a,a,a,a,e,e\}$, $\mathcal{M}_{perm,3} = \{e,a,a,a,a,a\}$, $\mathcal{M}_{perm,4} = \{a,a,a,a,a,a\}$ et $\mathcal{M}_{perm,5} = \{a,e,e,e,a,a\}$. Chaque 6-*mésoid* correspond à une instance de la tâche $\tau_2$. Puisque la tâche $\tau_2$ est à échéance sur activation (c'est-à-dire, $D_2 = T_2$), alors les *domaines d'échéance* sont aussi donnés par $\mathcal{D}_{perm,1} = \{a,a,a,a,a,a\}$, $\mathcal{D}_{perm,2} = \{a,a,a,a,e,e\}$, $\mathcal{D}_{perm,3} = \{e,a,a,a,a,a\}$, $\mathcal{D}_{perm,4} = \{a,a,a,a,a,a\}$ et $\mathcal{D}_{perm,5} = \{a,e,e,e,a,a\}$. Ainsi les univers $\mathcal{U}_{perm,1}, \mathcal{U}_{perm,2}, \mathcal{U}_{perm,3}$, $\mathcal{U}_{perm,4}$ et $\mathcal{U}_{perm,5}$ sont donnés par :

$$\begin{cases} \mathcal{U}_{perm,1} &= \mathcal{D}_{perm,2} = \{a,a,a,a,a,a\} \\ \mathcal{U}_{perm,2} &= \{a,a,a,a\} \\ \mathcal{U}_{perm,3} &= \{a,a,a,a,a\} \\ \mathcal{U}_{perm,4} &= \mathcal{D}_{perm,4} = \{a,a,a,a,a,a\} \\ \mathcal{U}_{perm,5} &= \{a\}\{a,a\} = \{a,a,a\} \end{cases}$$

Grâce aux égalités ci-dessus, $|\mathcal{U}_{perm,1}| = 6$, $|\mathcal{U}_{perm,2}| = 4$, $|\mathcal{U}_{perm,3}| = 5$, $|\mathcal{U}_{perm,4}| = 6$ et $|\mathcal{U}_{perm,5}| = 3$. Comme $no\tau_2 = \{a,a,a,a,a\}_{(0,5)}\{e,e,a,a,a,a\}_5^\infty$, $C_2 = 2$ et par

conséquent la otâche $o\tau_2$ est *potentiellement ordonnançable* puisque la relation 4.13 est satisfaite.

$$C_2 = 2 \leq 3 = \min_{1 \leq j \leq 5}(|\mathcal{U}_{perm,j}|)$$

Grâce à la procédure 5, nous calculons la valeur du *PET* dans chaque 6-*mesoid* et par l'application $f_5$ on obtient $f_5(2) = (2, 2, 2, 2, 3)$. La otâche $\mathcal{R}_2 = no\tau_1 \oplus no\tau_2$ ainsi obtenue est donnée par :

$$\begin{aligned}
\mathcal{R}_2 &= no\tau_1 \oplus no\tau_2 \\
&= \{e,e,e,a,a\}_{(0,5)}(\{a,a,a,a,a,a\}\{a,a,a,a,e,e\}\{e,a,a,a,a,a\} \\
&\qquad \{a,a,a,a,a,a\}\{a,e,e,e,a,a\})_5^{\infty} \oplus \{a,a,a,a,a\}_{(0,5)}\{e,e,a,a,a,a\}_5^{\infty} \\
&= \{e,e,e,a,a\}_{(0,5)}(\{\mathbf{e},\mathbf{e},a,a,a,a\}\{\mathbf{e},\mathbf{e},a,a,e,e\}\{e,\mathbf{e},\mathbf{e},a,a,a\} \\
&\qquad \{\mathbf{e},\mathbf{e},a,a,a,a\}\{\mathbf{e},e,e,e,\check{\mathbf{e}},\mathbf{e}\})_5^{\infty} \\
&= \{e,e,e,a,a\}_{(0,5)} \\
&\qquad \{e,e,a,a,a,a,e,e,a,a,e,e,e,e,e,a,a,a,e,e,a,a,a,a,e,e,e,e,\check{\mathbf{e}},e\}_5^{\infty}
\end{aligned}$$

Du point de vue des temps de réponse, nous avons $R_2^1 = |\{e,e\}| = 2$, $R_2^2 = |\{e,e\}| = 2$, $R_2^3 = |\{e,e,e\}| = 3$, $R_2^4 = |\{e,e\}| = 2$ et $R_2^5 = \{e,e,e,e,\check{\mathbf{e}},e\} = 6$ où $R_2^k$ désigne le temps de réponse de la tâche $\tau_2$ dans sa $k^{ieme}$ instance. Le pire temps de réponse de la tâche $\tau_2$ vaut alors $R_2 = 6$ et il est atteint pour la première fois dans la cinquième instance de $\tau_2$.

**Seconde itération :** Calcul de l'opération $\mathcal{R}_3 = \mathcal{R}_2 \oplus o\tau_3$

Grâce au résultat de l'itération précédente, les otâches normalisées $n\mathcal{R}_2$ et $no\tau_3$ correspondants respectivement aux otâches périodiques $\mathcal{R}_2$ et $o\tau_3$ sont données par $n\mathcal{R}_2 = \{e,e,e,a,a\}_{(0,5)}\{e,e,a,a,a,a,e,e,a,a,e,e,e,e,e,a,a,a,e,e,a,a,a,a,e,e,e,e,\check{\mathbf{e}},e\}_5^{\infty}$ et $no\tau_3 = \{a,a,a\}_{(0,3)}\{e,e,e,e,a,a,a,a,a,a\}_3^{\infty}$. Après un calcul analogue à celui de la première itération, nous pouvons conclure que la otâche $no\tau_3$ est ordonnançable et le

résultat est donné par :

$$
\begin{aligned}
\mathcal{R}_3 &= n\mathcal{R}_2 \oplus no\tau_3 \\
&= \{e,e,e,a,a\}_{(0,5)} \\
&\qquad \{e,e,a,a,a,a,e,e,a,a,e,e,e,e,e,a,a,a,e,e,a,a,a,a,e,e,e,e,\check{\mathbf{e}},e\}_5^{\infty} \\
&\qquad \oplus \{a,a,a\}_{(0,3)}\{e,e,e,e,a,a,a,a,a,a\}_3^{\infty} \\
&= \{e,e,e\}_{(0,3)}\{a,a,e,e,a,a,a,a,e,e\}_{(3,13)} \\
&\qquad \{a,a,e,e,e,e,e,a,a,a,e,e,a,a,a,a,e,e,e,e,\check{\mathbf{e}},e,e,e,a,a,a,a,e,e\}_{13}^{\infty} \\
&\qquad \oplus \{a,a,a\}_{(0,3)}\{e,e,e,e,a,a,a,a,a,a\}_3^{\infty} \\
&= \{e,e,e\}_{(0,3)}\{a,a,e,e,a,a,a,a,e,e\}_{(3,13)} \\
&\qquad (\{a,a,e,e,e,e,e,a,a,a\}\{e,e,a,a,a,a,e,e,e,e\}\{\check{\mathbf{e}},e,e,e,a,a,a,a,e,e\})_{13}^{\infty} \\
&\qquad \oplus \{a,a,a\}_{(0,3)}\{e,e,e,e,a,a,a,a,a,a\}_3^{\infty} \\
&= \{e,e,e\}_{(0,3)}\{\mathbf{e},\mathbf{e},e,e,\check{\mathbf{e}},\mathbf{e},\mathbf{e},a,e,e\}_{(3,13)} \\
&\qquad (\{\mathbf{e},\mathbf{e},e,e,e,e,e,\check{\mathbf{e}},\mathbf{e},\mathbf{e}\}\{e,e,\mathbf{e},\mathbf{e},\mathbf{e},\mathbf{e},e,e,e,e\}\{\check{\mathbf{e}},e,e,e,\mathbf{e},\mathbf{e},\mathbf{e},\mathbf{e},e,e\})_{13}^{\infty} \\
&= \{e,e,e,e,e,e,e,\check{\mathbf{e}},e,e,a,e,e\}_{(0,13)} \\
&\qquad \{e,e,e,e,e,e,e,\check{\mathbf{e}},e,e,e,e,e,e,e,e,e,e,e,e,\check{\mathbf{e}},e,e,e,e,e,e,e,e,e\}_{13}^{\infty} \\
&= \{e,e,e,e,e,e,e,\check{\mathbf{e}},e,e,a\}_{(0,11)} \\
&\qquad \{e,e,e,e,e,e,e,e,e,\check{\mathbf{e}},e,e,e,e,e,e,e,e,e,e,e,\check{\mathbf{e}},e,e,e,e,e,e,e\}_{11}^{\infty}
\end{aligned}
$$

Grâce à la procédure 5, le *PET* dans chaque $10$-*mesoid* vaut $(5)$ pour le $10$-*mesoid* qui débute à la date $3$, puis par l'application $f_{10}$ vaut $f_{10}(4) = (5,4,4)$. Du point de vue des temps de réponse, $R_3^1 = |\{e,e,e,e,\check{\mathbf{e}},e,e\}| = 7$, $R_3^2 = |\{e,e,e,e,e,e,e,\check{\mathbf{e}},e,e\}| = 10$, $R_3^3 = |\{e,e,e,e,e,e\}| = 6$ et $R_3^4 = |\{\check{\mathbf{e}},e,e,e,e,e,e,e\}| = 8$ où $R_3^2 = 10$, $R_3^3 = 6$ et $R_3^4 = 8$ désignent les temps de réponse de la tâche $\tau_3$ dans la phase permanente. Par conséquent, le pire temps de réponse de la tâche $\tau_3$ vaut $R_2 = 10$ et il est atteint pour la première fois dans la deuxième instance de $\tau_3$. La figure 4.13 résume les résultats de cet exemple et la figure 4.12 illustre la courbe du temps de réponse de chaque otâche et donc de chaque tâche en fonction du temps. Dans la figure 4.13 la phase permanente correspond à la zone mise en évidence dans l'ordonnançement et la phase transitoire correspond à l'intervalle précédent cette zone.

Grâce à la définition 38, le *facteur exact permanent d'utilisation du processeur* du système est donné par

$$U_3^* = \frac{3}{15} + \frac{1}{6} \cdot \frac{(2+2+2+2+3)}{5} + \frac{1}{10} \cdot \frac{(5+4+4)}{3} = 1$$

Le *coût exact permanent de la préemption* $\epsilon_3$ est alors donné grâce à la relation 4.5 par

$$\epsilon_3 = U_3^* - U_3 = 1 - \frac{28}{30} = 0.0667 \equiv 6.67\%$$

Après cet exemple relativement simple (le système ne contient que $3$ otâches et le coût de la préemption vaut $\alpha = 1$ pour toutes les otâches) permettant d'expliquer l'application de l'opération $\oplus$, la figure 4.14 illustre le résumé de l'analyse d'un exemple plus

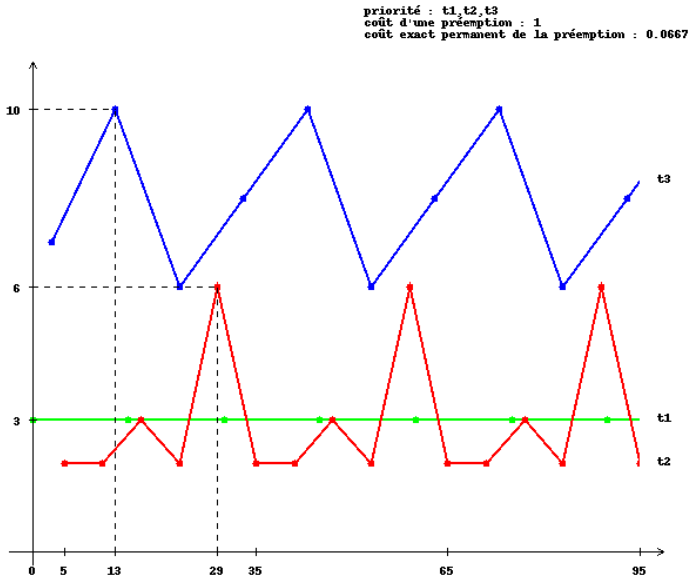


FIG. 4.12 – Courbe du temps de réponse en fonction du temps.

complexe en utilisant notre approche. Le système considéré est constitué de 10 otâches et les caractéristiques et priorités de chaque otâche sont résumés dans le tableau dans la même figure. Le coût d'une préemption varie d'une otâche à l'autre : il vaut par exemple $\alpha_6 = 1$ unité de temps pour la otâche $t_6$ et $\alpha_{10} = 3$ unités de temps pour la otâche $t_{10}$. Le plus petit commun multiple (*ppcm*) des périodes de toutes les otâches vaut $H_{10} = 3600$ unités de temps. La phase transitoire du système débute de la date 0 et se termine à la date 1517. Ainsi la phase permanente débute à la date $t_{initial} = 1517$ et se termine à la date $t_{final} = 5117$ (zone bleue). Le facteur d'utilisation classique du processeur (sans coût de préemption) vaut $U_{10} = 80.08\%$, le facteur exact permanent d'utilisation du processeur vaut $U^*_{10} = 94.83\%$, donc le coût exact permanent de la préemption vaut $\epsilon_{10} = 14.75\%$. La figure 4.15 illustre la courbe du temps de réponse en fonction du temps pour chaque otâche. Grâce à cette figure, nous constatons par exemple que le pire temps de réponse (258 unités de temps) de la otâche $t_{10}$ est atteint pour la première fois à sa quatrième activation tandis que celui de la otâche $t_7$ (108 unités de temps) est atteint pour la première fois à sa dixième activation et celui de la otâche $t_9$ (350 unités de temps) est atteint pour la première fois à sa neuvième activation.

La figure 4.16 illustre le *zoom* d'une fenêtre de l'ordonnancement linéaire fournie par la figure 4.14. Dans cette fenêtre, il apparaît par exemple clairement le temps de réponse de la otâche $t_7$ dont l'activation a lieu à la date 3058. Pour cette activation précise, nous pouvons noter qu'après la première préemption de la otâche à la date 3060, celle-ci ne peut reprendre son exécution effective qu'à la date 3096. Cette situation est dûe d'une

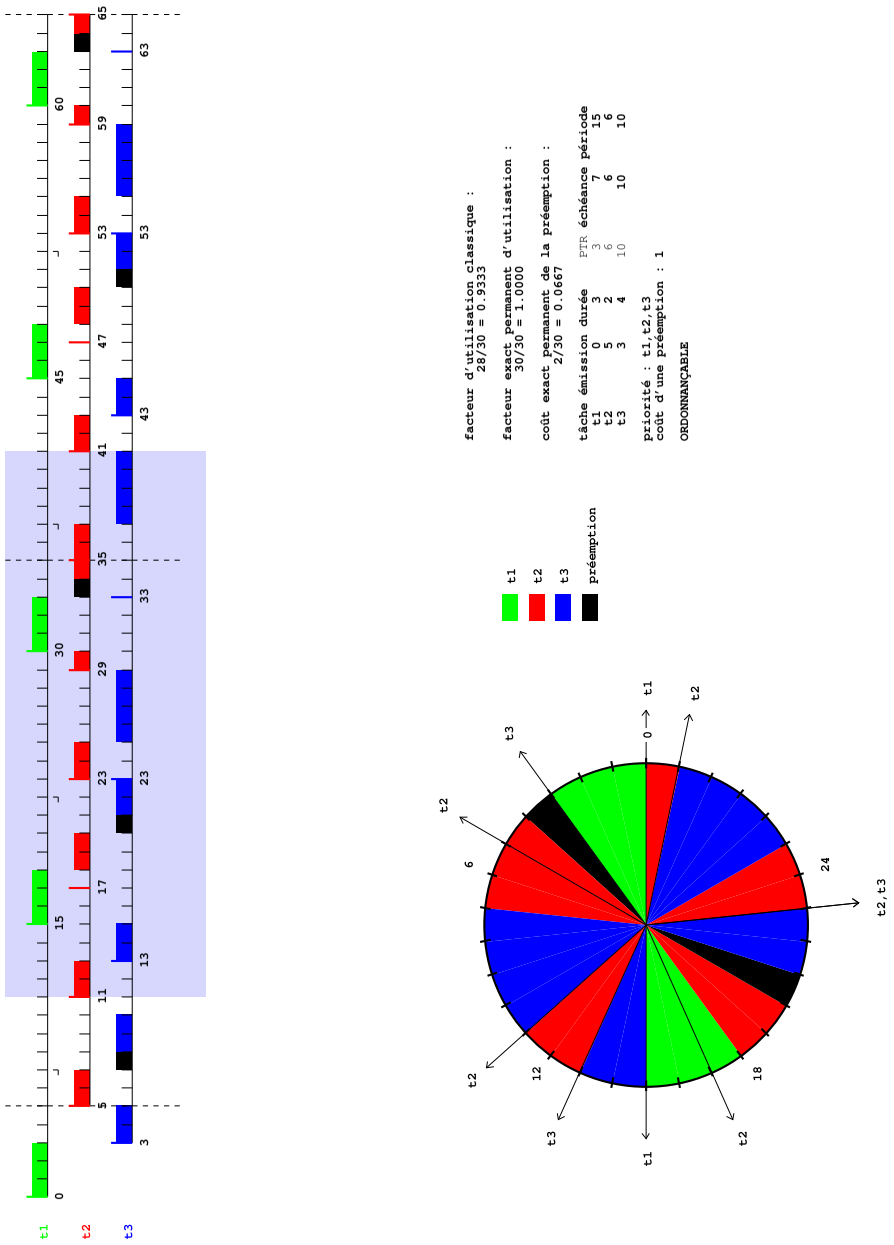


FIG. 4.13 – Résumé des résultats de l'exemple avec coût exact de préemption.

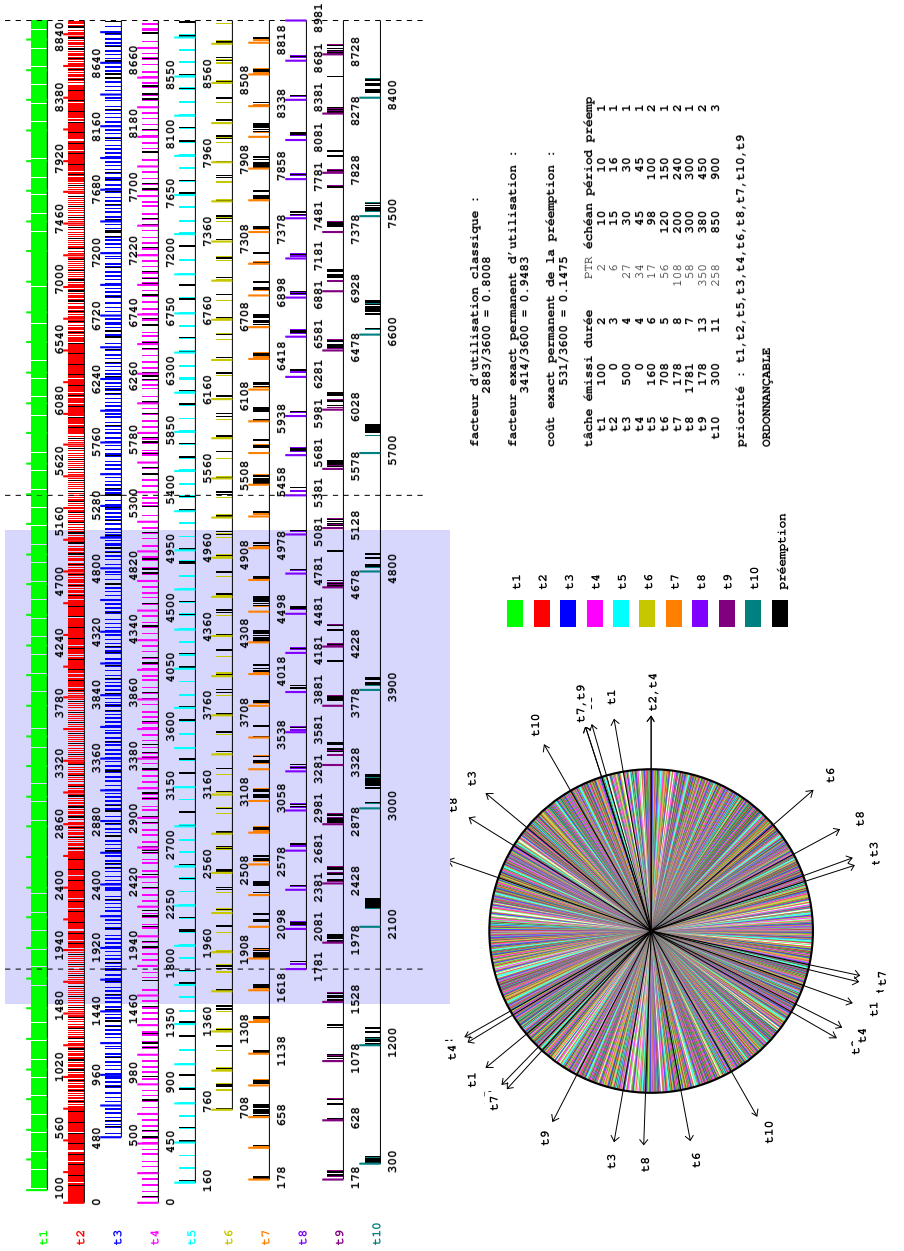


FIG. 4.14 – Ordonnancement d'un système de 10 tâches avec coût exact de préemption.

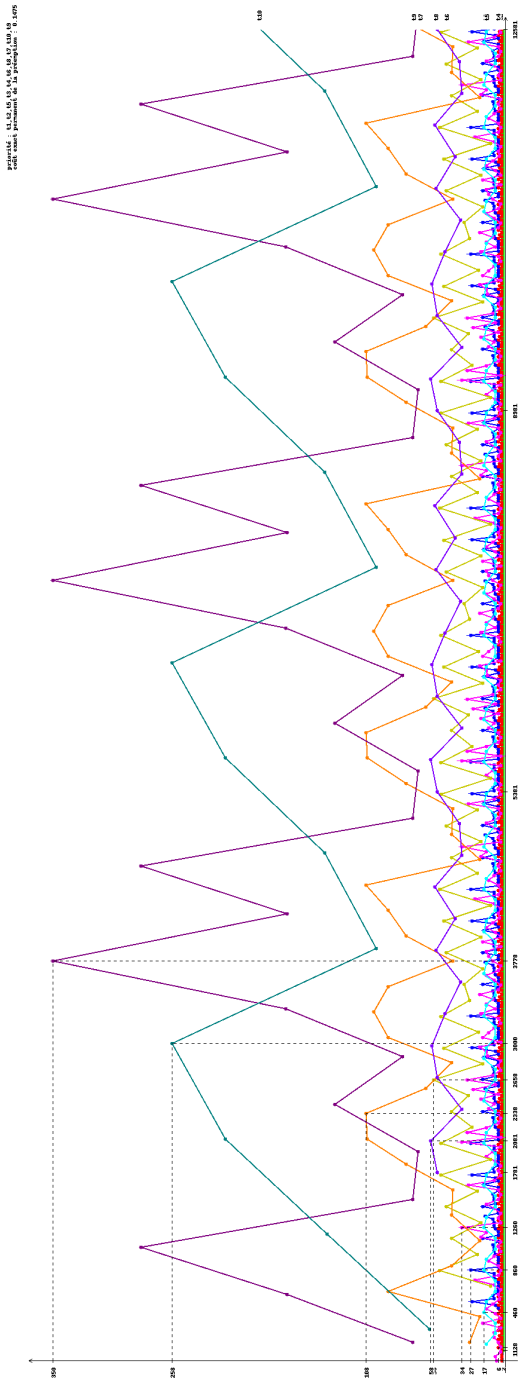

FIG. 4.15 – Courbe du temps de réponse en fontion du temps.

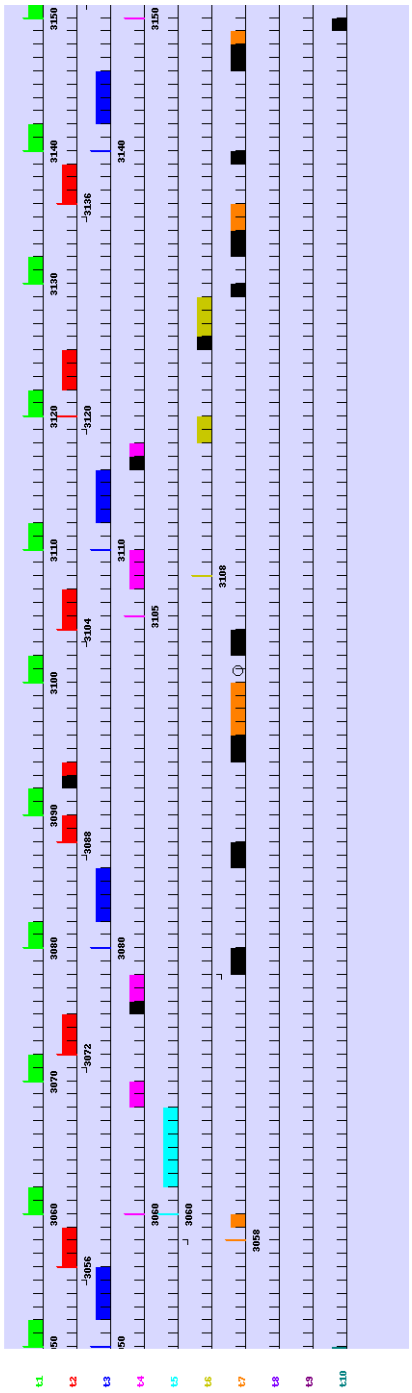

FIG. 4.16 – Zoom d'une fenêtre de l'ordonnancement linéaire.

part aux otâches ayant une priorité plus élevée que celle de $t_7$ et d'autre part au coût de chaque préemption. L'aspect atomique de la restauration du contexte d'une otâche donnée est illustrée par exemple à la date $3130$. En effet le coût d'une préemption de la otâche $t_7$ dans l'exemple vaut $\alpha_7 = 2$ unités de temps et à cette date, la otâche $t_7$ est préemptée par la otâche $t_1$ alors qu'elle est en train de restaurer son contexte. La préemption à lieu et à la date $3132$, la restauration du contexte de la otâche $t_7$ est reprise à zéro.

## 4.5 Choix des priorités des otâches

### 4.5.1 Impact des priorités sur l'ordonnancement

Soit $O\Gamma_n = \{o\tau_1, o\tau_2, \cdots, o\tau_n\}$ un système constitué de $n$ otâches périodiques. Nous avons déjà souligné dans la section 4.2 l'impact des priorités des otâches sur l'analyse d'ordonnançabilité du système. D'une part, nous avons montré que le résultat de l'analyse d'ordonnançabilité d'une otâche suivant un choix de priorités $\mathcal{S}$ dépend non seulement de l'ensemble des otâches ayant une priorité plus élevée mais aussi de leurs priorités exactes. D'autre part, nous avons montré que si une otâche n'est pas ordonnançable, nous pouvons la rendre ordonnançable en abaissant sa priorité.

**Définition 39** *Soit $O\Gamma_n = \{o\tau_1, \cdots, o\tau_n\}$ un système contitué de $n$ otâches périodiques, un* choix de priorités *des otâches de $O\Gamma_n$ est une* permutation *des éléments de $O\Gamma_n$ qui ordonne les éléments de la otâche la plus prioritaire à la otâche la moins prioritaire.*

**Notation :** Pour le système $O\Gamma_n = \{o\tau_1, \cdots, o\tau_n\}$ constitué de $n$ otâches périodiques, nous noterons un choix de priorités par $< o\tau_{g(1)}, o\tau_{g(2)}, \cdots, o\tau_{g(n)} >$ où la priorité de $o\tau_{g(l)}$ est supérieure à celle de $o\tau_{g(r)}$ dès que $l < r$ et $\{g(1), \cdots, g(n)\}$ est l'image de l'ensemble $\{1, \cdots, n\}$ par une permutation $g$.

**Définition 40** *Soit $O\Gamma_n = \{o\tau_1, \cdots, o\tau_n\}$ un système contitué de $n$ otâches périodiques, un* choix de priorités $\mathcal{S} = < o\tau_{g(1)}, o\tau_{g(2)}, \cdots, o\tau_{g(n)} >$ *est dit* ordonnançable *si et seulement si*

$$\mathcal{R}_i = \bigoplus_{j=1}^{i} o\tau_{g(j)} \neq \Lambda, \ \forall\, i \in \{1, \cdots, n\}$$

Puisqu'il peut exister plusieurs choix de priorités différents ordonnançables pour le système $O\Gamma_n$, il reste à définir un critère permettant de choisir un choix de priorités particulier parmi les choix possibles qui satisfont toutes les contraintes pendant l'application

des procédures 6, 7 et 8, s'il en existe au moins un. Pour nous, ce critère consiste à toujours privilégier le choix dont le coût exact permanent de préemption est le moins élevé chaque fois que nous aurons à choisir entre deux choix de priorités ordonnançables. Si deux choix de priorités ordonnançables conduisent au même coût exact permanent de préemption, nous privilégerons celui qui satisfait un autre critère que nous aurons préalablement défini auparavant (par exemple celui pour lequel les temps de réponse sont les moins élevés, etc.). Nous rappelons que l'algorithme de choix des priorités d'*Audsley* [Aud91] n'est plus *optimale* lorsqu'on prend en compte le coût de la préemption et que l'algorithme *naïf* qui consiste à tester toutes les permutations de priorités possibles est très coûteux en terme de complexité puisqu'il requiert $n!$ tests où $n$ est le nombre d'otâches du système considéré. Pour contourner cette difficulté, nous réduisons considérablement le nombre de tests à effectuer l'aide du théorème 15 et du théorème 16.

**Théorème 15** *Soit* $O\Gamma_n = \{o\tau_1, o\tau_2, \cdots, o\tau_n\}$ *un système constitué de* $n$ *otâches périodiques. On considère le choix* $\mathcal{S}$ = <$o\tau_{g(1)}, \cdots, o\tau_{g(i)}, o\tau_{g(i+1)}, \cdots, o\tau_{g(n)}$> *des priorités des otâches de* $O\Gamma_n$ *où* $g$ *est une permutation de* $\{1, 2, \cdots, n\}$. *Si* $\mathcal{S}$ *est* non ordonnançable *à cause de la otâche* $o\tau_{g(i)}$, *c'est-à-dire*

$$\mathcal{R}_i = \bigoplus_{k=1}^{i} o\tau_{g(k)} = \Lambda \text{ et } \forall j < i,\ \mathcal{R}_j \neq \Lambda$$

*alors tout choix de priorités* $\mathcal{S}'$ = <$o\tau_{g(1)}, \cdots, o\tau_{g(i)}, o\tau_{h(i+1)}, \cdots, o\tau_{h(n)}$> *où* $h$ *est une permutation des éléments du sous-ensemble* $\{g(i+1), g(i+2), \cdots, g(n)\}$ *est aussi* non ordonnançable.

**Preuve** : – Par contradiction –

On considère le choix $\mathcal{S}$ = <$o\tau_{g(1)}, \cdots, o\tau_{g(i)}, o\tau_{g(i+1)}, \cdots, o\tau_{g(n)}$> des priorités des otâches de $O\Gamma_n$ où $g$ est une permutation de $\{1, 2, \cdots, n\}$. Supposons que $\mathcal{S}$ est *non ordonnançable* à cause de la otâche $o\tau_{g(i)}$ et qu'il existe au moins un choix de priorités ordonnançable $\mathcal{S}'$ = <$o\tau_{g(1)}, \cdots, o\tau_{g(i)}, o\tau_{h_0(i+1)}, \cdots, o\tau_{h_0(n)}$> où $h_0$ est une permutation des éléments du sous-ensemble $\{g(i+1), g(i+2), \cdots, g(n)\}$. Grâce à la définition 33, nous aurions en particulier

$$\mathcal{R}_i = \bigoplus_{k=1}^{i} o\tau_{g(k)} \neq \Lambda$$

Ce qui contredit l'hypothèse selon laquelle le choix de priorités $\mathcal{S}$ est *non ordonnançable* à cause de la otâche $o\tau_{g(i)}$. ■

**Théorème 16** *Soit* $O\Gamma_n = \{o\tau_1, o\tau_2, \cdots, o\tau_n\}$ *un système constitué de* $n$ *otâches périodiques. On considère le choix* $\mathcal{S}$ = <$o\tau_{g(1)}, \cdots, o\tau_{g(i)}, o\tau_{g(i+1)}, \cdots, o\tau_{g(n)}$> *des priorités*

*des otâches de* $O\Gamma_n$ *où* $g$ *est une permutation de* $\{1, 2, \cdots, n\}$. *Si la otâche* $o\tau_{g(i)}$ *est* ordonnançable *suivant le choix* $\mathcal{S}$, *c'est-à-dire*

$$\mathcal{R}_i = \bigoplus_{k=1}^{i} o\tau_{g(k)} \neq \Lambda$$

*alors tout choix* $\mathcal{S}' = <o\tau_{g(1)}, \cdots, o\tau_{g(i)}, o\tau_{h(i+1)}, \cdots, o\tau_{h(n)}>$ *où* $h$ *est une permutation des éléments du sous-ensemble* $\{g(i+1), g(i+2), \cdots, g(n)\}$ *est égal au choix des priorités* $\mathcal{S}'' = <\mathcal{R}_i, o\tau_{h(i+1)}, \cdots, o\tau_{h(n)}>$.

**Preuve** :

La preuve du théorème 16 découle directement du fait que l'opération $\oplus$ est une opération interne. ■

Le théorème 15 permet de réduire le nombre de tests de permutation de priorités à effectuer lorsque le système est non ordonnançable suivant un choix de priorités particulier et le théorème 16 permet de réduire le nombre de fois qu'on effectue l'opération $\oplus$ en réutilisant un résultat partiel d'un certain choix de priorités pour calculer le résultat d'un autre choix de priorités.

### 4.5.2 Choix optimal des priorités

Dans cette section, nous proposons un algorithme *optimal* de choix des priorités des otâches d'un système. Cet algorithme est optimal au sens où, pour un système d'otâches donné, si il existe un choix de priorités conduisant à un ordonnancement valide alors le choix de priorités produit par notre algorithme conduira également à un ordonnancement valide. Puisque notre proposition de choix de priorités s'appuie sur des remarques faites par rapport à l'algorithme de choix des priorités d'*Audsley*, nous nommerons notre algorithme de choix des priorités $Audsley^{++}$. Cet algorithme intègre d'une part le coût exact de la préemption par application des procédures 6, 7 et 8, et fournit d'autre part toutes les solutions de choix des priorités conduisant à un ordonnancement valide.

L'algorithme $Audsley^{++}$ divise un système d'otâches donné en deux groupes : d'une part les otâches auxquelles ont déjà été attribuées une priorité, et d'autre part les otâches qui n'ont pas de priorités. De même, les priorités sont divisées en deux groupes : d'une part celles qui sont déjà attribuées et d'autre part celles qui sont encore disponibles. L'algorithme attribue toujours la priorité la plus *élevée* disponible à toute otâche qui satisfait ses contraintes lorsque cette priorité lui est attribuée. Grâce au théorème 15 et au théorème 16, l'algorithme peut terminer de deux façons. Soit il attribue des priorités à toutes les otâches, dans ce cas, un choix de priorités ordonnançable a été trouvée : on note cette solution et grâce à un algorithme de *rebroussement* / *backtrack* on recommence le processus afin de rechercher une nouvelle solution s'il en existe une, soit à un certain point

la priorité la plus élevée disponible (par exemple $p_{max}$) ne peut pas être attribuée à l'une des otâches restantes. Dans ce cas, grâce une nouvelle fois à un algorithme de *rebroussement / backtrack*, au théorème 15 et au théorème 16, on recommence le processus afin de rechercher une solution ordonnançable. Si aucune solution n'est trouvée, nous pourrons conclure qu'il n'existe pas de choix de priorités conduisant à un ordonnancement valide (c'est-à-dire, satisfaisant toutes les contraintes).
L'algorithme est donc *optimal*, en ce sens qu'il trouve toujours un choix de priorités ordonnançable, si un tel choix existe grâce au *rebroussement / backtrack*. Notons que le nombre de tests effectué est nettement inférieur par rapport à l'agorithme "*naïf*" qui vérifie toutes les permutations de priorités possibles grâce à l'utilisation du théorème 15 et du théorème 16.

### Exemple

Soit $\Gamma_5 = \{t_1, t_2, t_3, t_4, t_5\}$ un système constitué de cinq tâches temps réel périodiques. Les caractéristiques des tâches sont résumées dans la table 4.5. Nous supposons que les priorités sont attribuées aux tâches suivant l'algorithme $Audsley^{++}$.

| Tâche | $r_i^1$ | $C_i$ | $D_i$ | $T_i$ | $\alpha_i$ |
|---|---|---|---|---|---|
| $t_1$ | 9 | 1 | 6 | 6 | 0 |
| $t_2$ | 13 | 3 | 9 | 12 | 2 |
| $t_3$ | 5 | 2 | 15 | 15 | 2 |
| $t_4$ | 0 | 3 | 21 | 24 | 1 |
| $t_5$ | 15 | 5 | 47 | 60 | 1 |

TAB. 4.5 – Caractéristiques des tâches

Grâce à tout ce que nous avons présenté jusqu'ici, notons que ce système n'est pas ordonnançable ni suivant le choix de priorités correspondant à *Deadline Monotonic / DM* (c'est le choix de priorités selon lequel plus l'échéance relative d'une tâche est faible plus sa priorité est élevée), ni suivant le choix de priorités correspondant à *Rate Monotonic / RM* (c'est le choix de priorités selon lequel plus la période d'une tâche est faible plus sa priorité est élevée). Ce choix de priorités est donnée dans les deux cas par $\mathcal{S}_{DM/RM} = < t_1, t_2, t_3, t_4, t_5 >$ et le résumé des résultats obtenus pour ce choix de priorités est illustré par la figure 4.17. Après application des procédures 6, 7 et 8, $t_1$, $t_2$ et $t_3$ sont ordonnançables mais $t_4$ ne l'est pas. La figure 4.18 illustre le zoom d'une fenêtre de l'ordonnancement linéaire. Dans cette fenêtre, nous pouvons noter le dépassement de l'échéance relative de la tâche $t_4$ à la date $t = 93$. Ce dépassement d'échéance est dû d'une part au coût de la préemption de la tâche $t_4$ elle-même mais aussi d'autre part au

coût de chaque préemption des tâches $t_2$ et $t_3$ qui elles ont une priorité supérieure à celle de la tâche $t_4$.

Grâce à l'application de l'algorithme $Audsley^{++}$ pour le choix des priorités des tâches, quatre choix de priorités conduisent à un ordonnancement valide :

- $\mathcal{S}_1 = < t_2, t_1, t_3, t_4, t_5 >$,
- $\mathcal{S}_2 = < t_2, t_3, t_1, t_4, t_5 >$,
- $\mathcal{S}_3 = < t_3, t_2, t_1, t_4, t_5 >$,
- $\mathcal{S}_4 = < t_4, t_2, t_1, t_5, t_3 >$.

Pour ces quatre choix de priorités ordonnançables, le coût exact permanent de la préemption vaut $\epsilon_{\mathcal{S}_1} = 12.50\%$ pour le choix de priorités $\mathcal{S}_1$, $\epsilon_{\mathcal{S}_2} = 9.17\%$ pour le choix de priorités $\mathcal{S}_2$, $\epsilon_{\mathcal{S}_3} = 11.67\%$ pour le choix de priorités $\mathcal{S}_3$ et seulement $\epsilon_{\mathcal{S}_4} = 5.83\%$ pour le choix de priorités $\mathcal{S}_4$ après application des procédures 6, 7 et 8. La figure 4.19, la figure 4.20, la figure 4.21 et la figure 4.22 résument les résultats obtenus pour chacun de ces quatre le choix. Les figures 4.23, 4.24, 4.25 et 4.26 illustrent les courbes de temps de réponse en fonction du temps de chaque tâche pour ces choix de priorités. La variation du coût de la préemption est due au fait que d'une part le coût d'une préemption varie d'une tâche à l'autre et d'autre part au fait que suivant certains choix de priorités, certaines tâches sont préemptées alors qu'elles ne le sont suivant d'autres choix de priorités.

Suivant le critère de choix de priorités des otâches que nous avons énoncé, c'est-à-dire toujours privilégier le choix dont le coût exact permanent de préemption est le moins élevé, nous privilégeons le choix de priorités $\mathcal{S}_4 = < t_4, t_2, t_1, t_5, t_3 >$. Pour ce choix de priorités, le pire temps de réponse de la tâche $t_1$ vaut $R_1 = 4$ unités de temps et est atteint pour la première fois à sa quatrième activation, le pire temps de réponse de la tâche $t_2$ vaut $R_2 = 5$ unités de temps et est atteint pour la première fois à sa deuxième activation, le pire temps de réponse de la tâche $t_3$ vaut $R_3 = 14$ unités de temps et est atteint pour la première fois à sa septième activation, le pire temps de réponse de la tâche $t_4$ vaut $R_4 = 3$ unités de temps et est atteint pour la première fois dès sa quatrième activation, c'est la tâche la plus prioritaire et enfin le pire temps de réponse de la tâche $t_5$ vaut $R_5 = 16$ unités de temps et est atteint pour la première fois à sa deuxième activation.

Il peut arriver que le coût exact permanent de la préemption d'un système d'otâches ordonnançable soit très élevé suivant un choix de priorités particulier lors de l'application des procédures 6, 7 et 8. Dans ce cas, une façon intuitive d'espérer réduire ce coût pourrait consister à contraindre certaines otâches à être *non-préemptives par niveau de priorité*, c'est-à-dire le processus de remplacement des symboles "$a$" (disponibles) par des symboles "$e$" (exécutables) lors de l'application de l'opération $\oplus$ a lieu seulement lorsque le cardinal de la première séquence de symboles "$a$" à remplacer dans chaque *mesoid* est supérieur à celui des symboles "$e$" exécutables à ce niveau. Cette notion de *non-préemptivité par niveau par priorité* est très différente de la notion de *non-préemptivité* classique car dans ce cas, une fois que les priorités sont attribuées aux

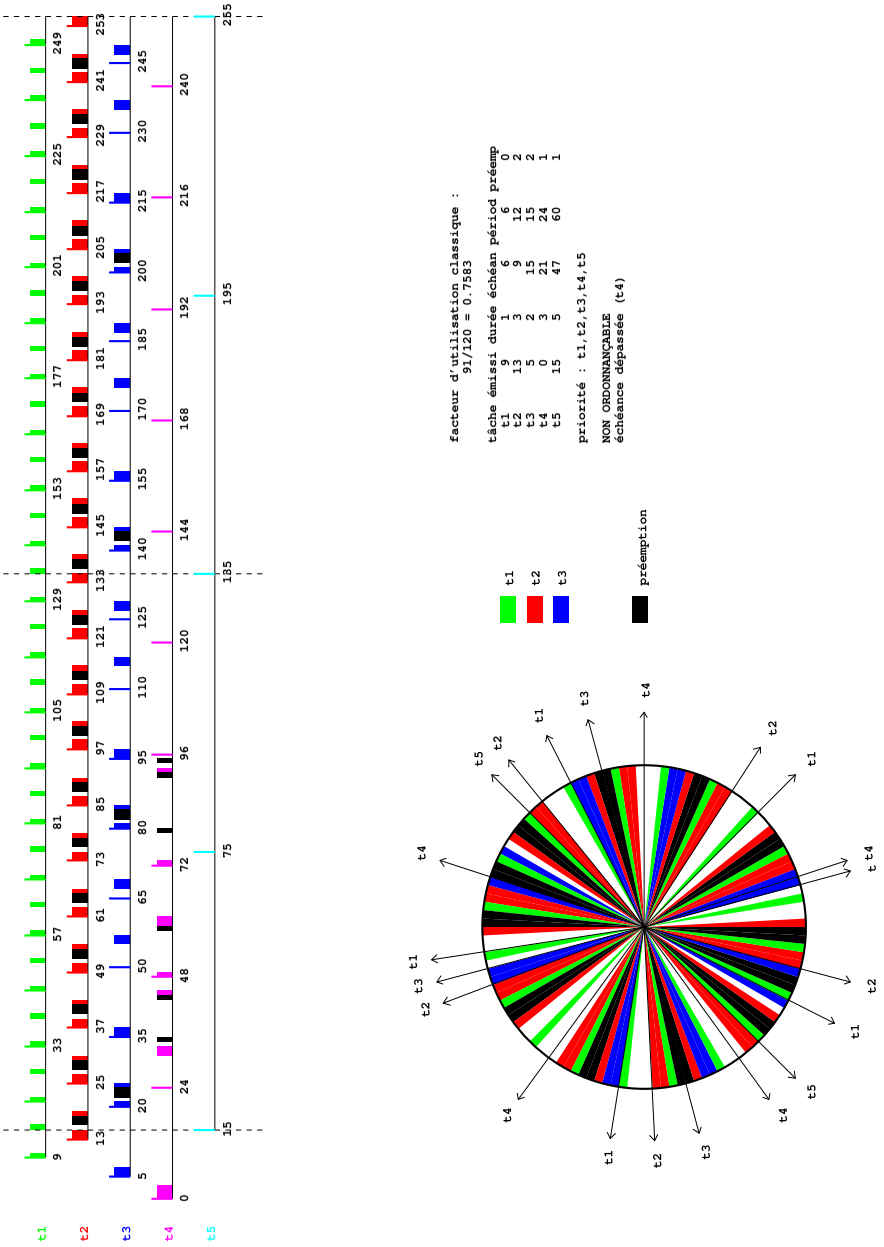


FIG. 4.17 – Résumé des résultats pour le choix $\mathcal{S}_{DM/RM} = < t_1, t_2, t_3, t_4, t_5 >$.

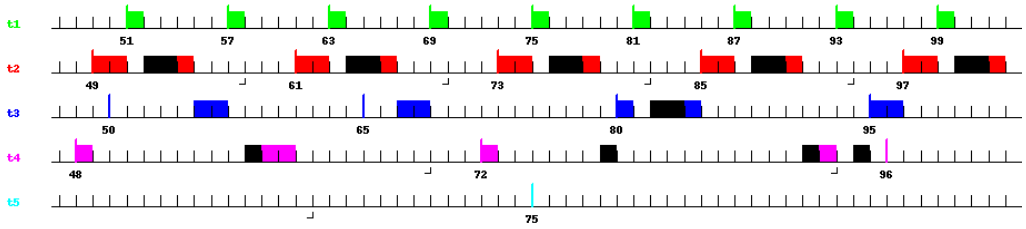


FIG. 4.18 – Zoom d'une fenêtre de l'ordonnancement linéaire.

otâches, une otâche moins prioritaire *ne peut pas retarder* la date de début d'exécution d'une autre otâche plus prioritaire. Nous exploitons alors non seulement les unités de temps disponibles, c'est-à-dire les symboles "$a$", mais aussi nous tenons compte de la longueur de chacune de leurs séquences. Les figures 4.27 et 4.28 illustrent la différence entre les deux notions. Dans cet exemple, la otâche $o\tau_1$ a une priorité supérieure à celle de la otâche $o\tau_2$ et la otâche $o\tau_2$ est non-préemptive.

Dans la figure 4.27, nous pouvons noter que la otâche $o\tau_2$ retarde la date de début d'exécution de la otâche $o\tau_1$, plus prioritaire. En effet à cause de l'activation de la otâche $o\tau_2$ une unité de temps avant celle de la otâche $o\tau_1$, il ne pas y avoir de prémption à la date d'activation de $o\tau_1$ car $o\tau_2$ est non-préemptive. Ainsi, un inconvénient de la non-préemptivité classique d'une otâche est la modification éventuelle du résultat d'ordonnançabilité d'une autre otâche plus prioritaire. Dans ce cas, si le système n'est pas ordonnançable, la otâche fautive du système peut s'avérer difficile à déterminer de façon précise.

Dans la figure 4.28, nous pouvons noter que la date de début d'exécution effective de la otâche $o\tau_2$ a lieu après la date de fin d'exécution de la otâche $o\tau_1$, plus prioritaire. En effet même si l'activation de la otâche $o\tau_2$ a lieu une unité de temps avant celle de la otâche $o\tau_1$, celle-ci ne débute pas son exécution à cette date parce qu'elle est d'une part non-préemptive et d'autre part moins prioritaire, il n'y a pas assez d'unités de temps libres pour l'exécuter avant la prochaine activation de $o\tau_1$. Par conséquent, un avantage de la non-préemptivité par niveau de priorité d'une otâche est le fait qu'elle n'a pas d'impact sur le résultat d'ordonnançabilité d'une autre otâche plus prioritaire. Avec cette approche, certaines préemptions très coûteuses peuvent être évitées, diminuant ainsi le coût exact permanent de la préemption pour un choix de priorités donné. La figure 4.29 illustre cette assertion pour l'exemple du système constitué de $5$ otâches précédent. Dans cette figure, le coût exact permanent de la préemption passe de $5.83\%$ à $2.50\%$ suivant le choix de priorités $\mathcal{S}_4$. La figure 4.30 illustre la courbe du temps de réponse en fonction du temps de chaque otâche. Notons au passage une légère amélioration du pire temps de réponse de la otâche $t_3$, il passe de $14$ unités de temps à $13$ unités de temps.

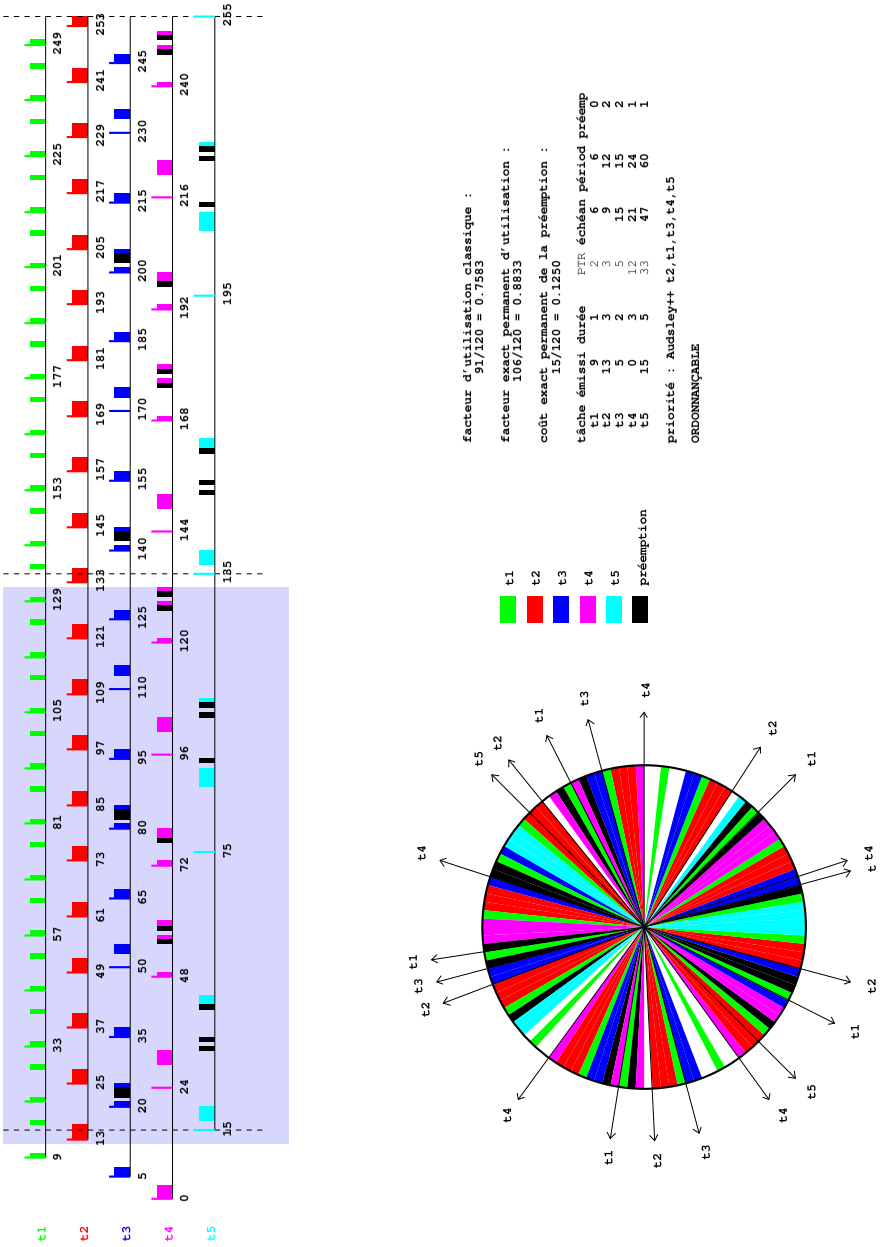


FIG. 4.19 – Résumé des résultats pour le choix $\mathcal{S}_1 = < t_2, t_1, t_3, t_4, t_5 >$.

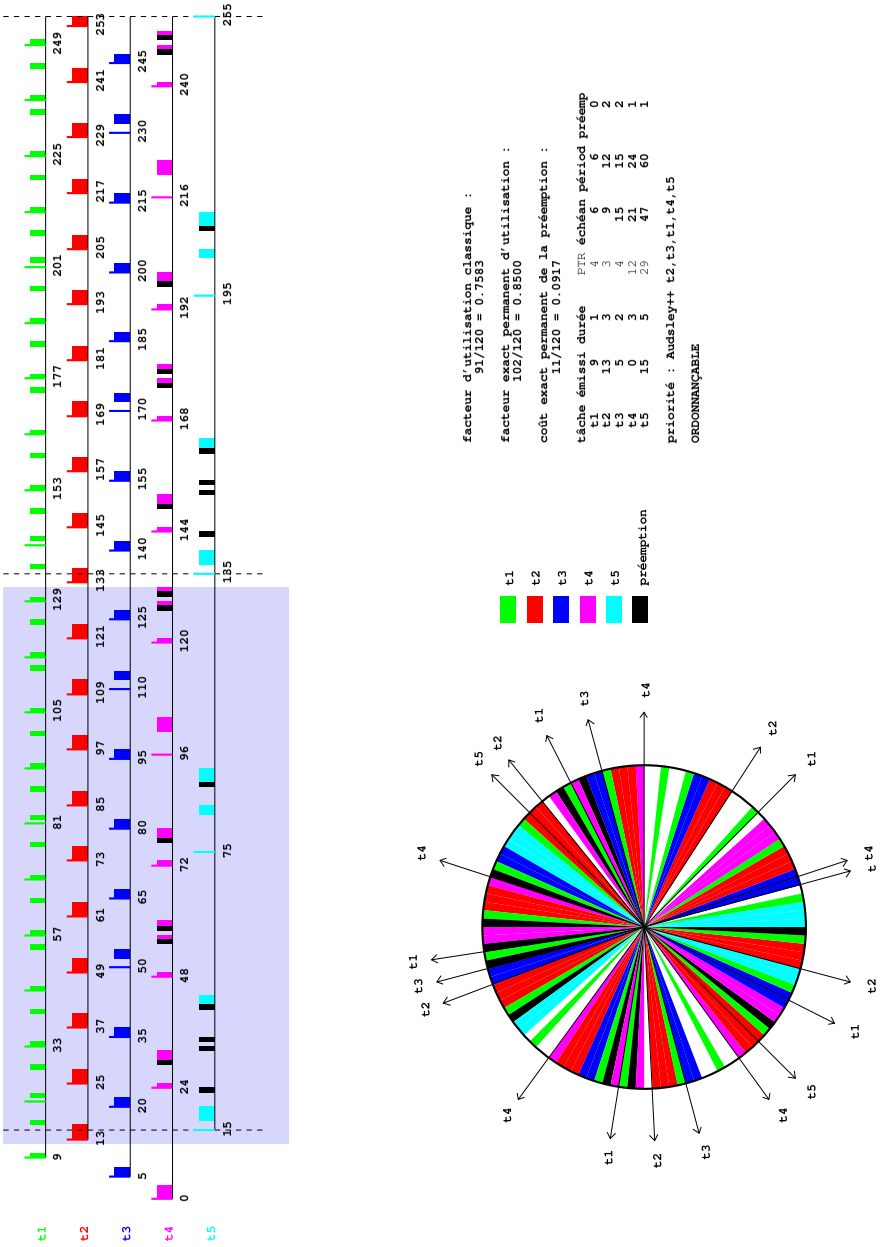


FIG. 4.20 – Résumé des résultats pour le choix $\mathcal{S}_2 = < t_2, t_3, t_1, t_4, t_5 >$.

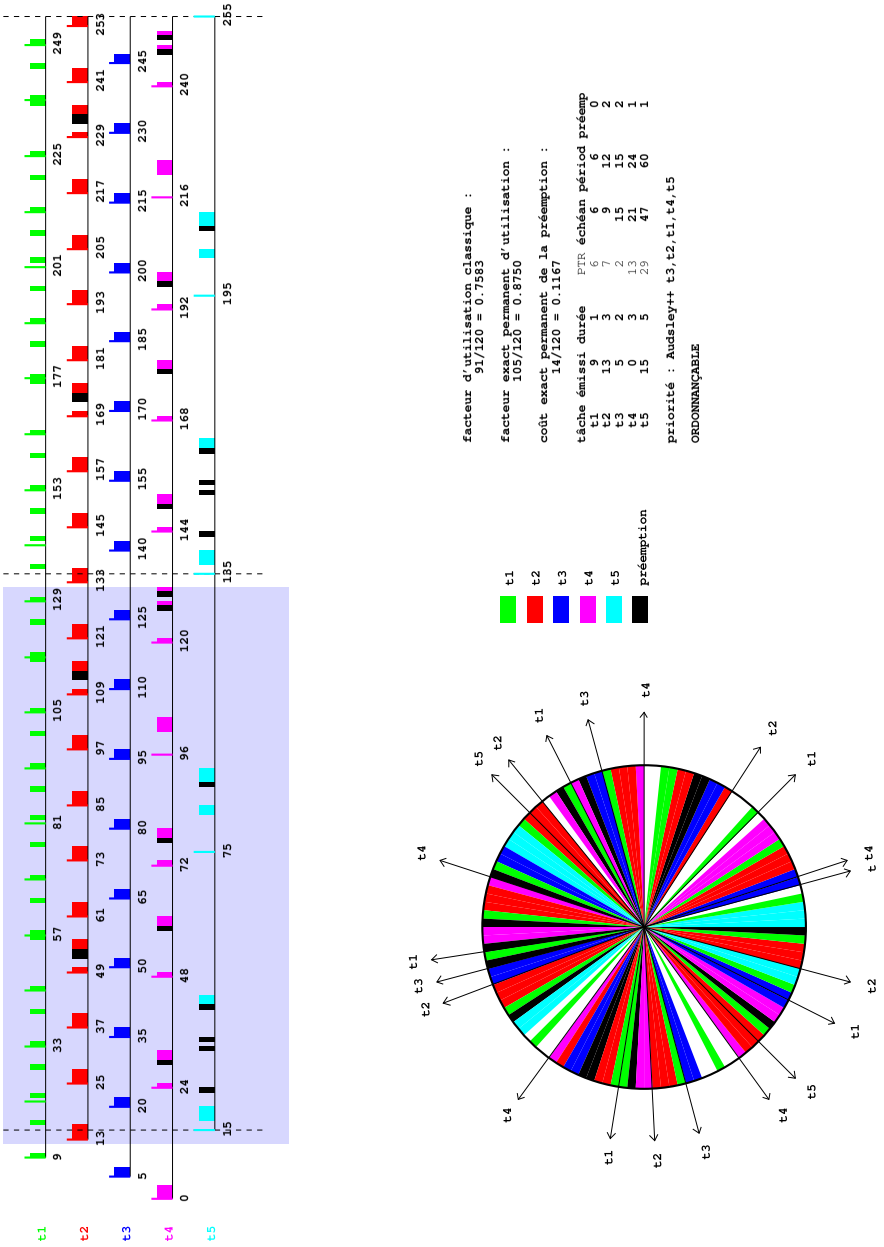


FIG. 4.21 – Résumé des résultats pour le choix $\mathcal{S}_3 = < t_3, t_2, t_1, t_4, t_5 >$.

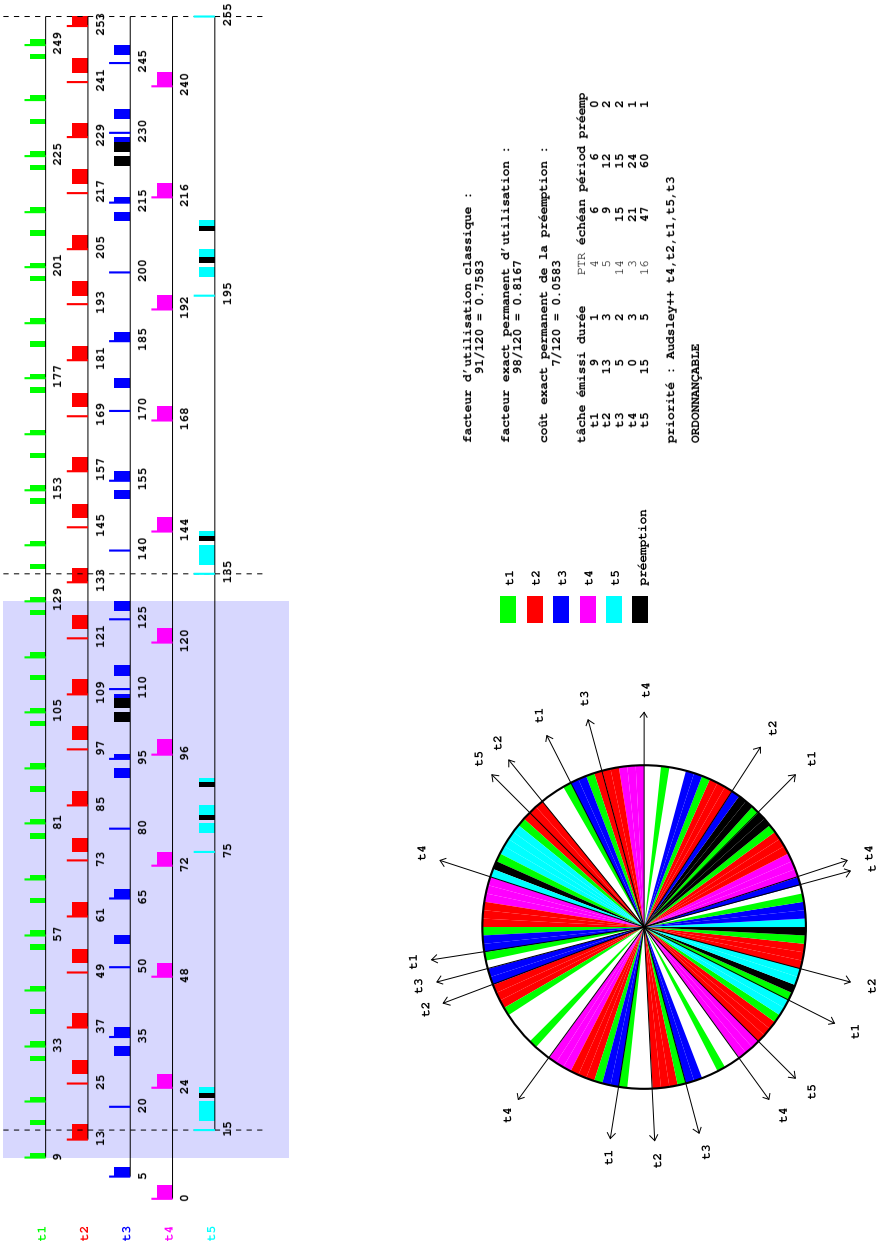


FIG. 4.22 – Résumé des résultats pour le choix $\mathcal{S}_4 = < t_4, t_2, t_1, t_5, t_3 >$.

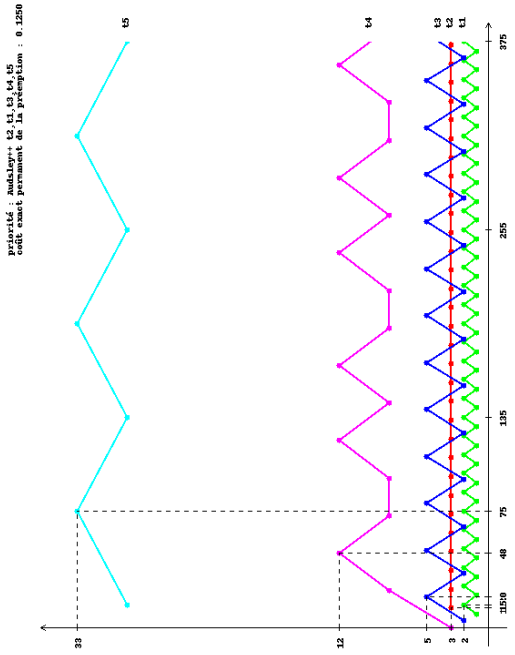


FIG. 4.23 – $\mathcal{S}_1 = < t_1, t_2, t_3, t_4, t_5 >$.

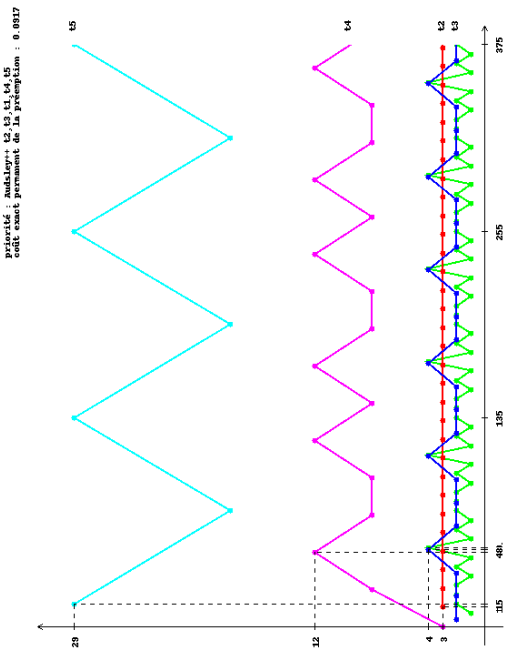


FIG. 4.24 – $\mathcal{S}_2 = < t_2, t_3, t_1, t_4, t_5 >$.

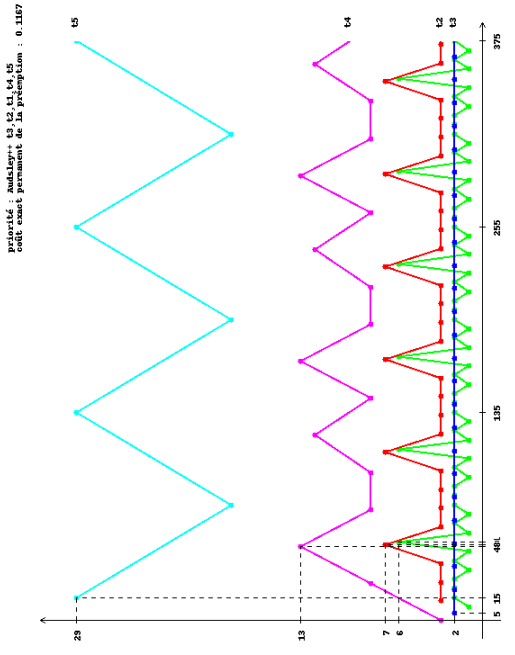


FIG. 4.25 – $\mathcal{S}_3 = < t_3, t_2, t_1, t_4, t_5 >$.

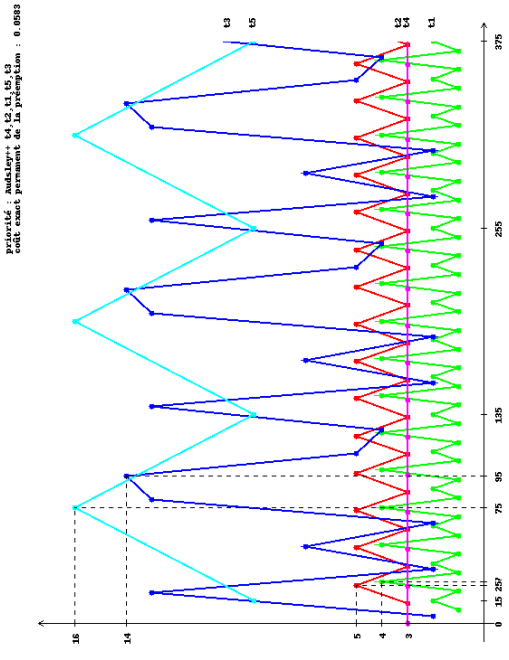


FIG. 4.26 – $\mathcal{S}_4 = < t_4, t_2, t_1, t_5, t_3 >$.

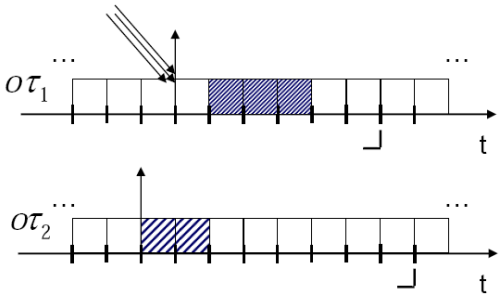


FIG. 4.27 – Non-préemptif classique.

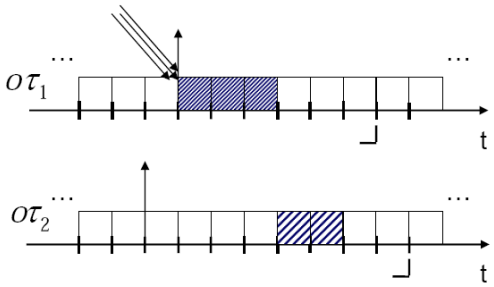


FIG. 4.28 – Non-préemptif par niveau.

Si la contrainte de non-préemptivité par niveau de priorité peut d'une part permettre de réduire le coût exact permanent de la préemption d'un système pour un choix de priorités fixée, il peut d'autre part au contraire, le rendre plus important pour un autre choix de priorités pour le même système. Ceci est dû au fait que la non-préemptivité par niveau de priorité a un impact direct sur le coût de la préemption à un niveau donné mais seulement indirect sur le coût global de la préemption. La figure 4.31 illustre cette assertion pour l'exemple du système constitué de $5$ otâches précédent. Le coût exact permanent de la préemption passe de $2.50\%$ suivant le choix de priorités $\mathcal{S}_4$ à $21.67\%$ suivant le choix de priorités donné par $\mathcal{S}_5 = < t_1, t_2, t_4, t_3, t_5 >$. La figure 4.32 illustre la courbe du temps de réponse en fonction du temps de chaque otâche suivant ce choix de priorités. Notons au passage que le pire temps de réponse de la otâche $t_5$ vaut $46$ unités de temps au lieu de $16$ unités de temps suivant le choix de priorités $\mathcal{S}_4$. Ce paragraphe montre un avantage supplémentaire de l'algorithme $Audsley^{++}$ car celui-ci produit toutes les solutions ordonnançables, permettant ainsi de comparer le coût de chaque solution par rapport aux autres.

## 4.6 Conclusion

Dans ce chapitre, nous avons commencé par illustrer l'impact de la prise en compte du coût de la préemption sur l'ordonnançabilité d'un système donné. Ensuite basé sur le modèle d'otâches introduit dans le chapitre 3, nous avons étendu la définition de l'opération $\oplus$ au cas du problème d'ordonnancement d'un système d'otâches périodiques tout en tenant compte du coût exact de la préemption lorsque la priorité de chaque otâche est imposée. Puisqu'à notre connaissance il n'existe pas, dans la littérature, d'algorithme optimal de choix de priorités lorsque le coût de la préemption est pris en compte, nous avons montré l'impact du choix des priorités sur l'analyse d'ordonnançabilité et sur l'ordonnancement obtenu pour un système. Enfin, nous avons proposé un algorithme optimal de choix de priorités des otâches conduisant à des conditionsd'ordonnançabilité

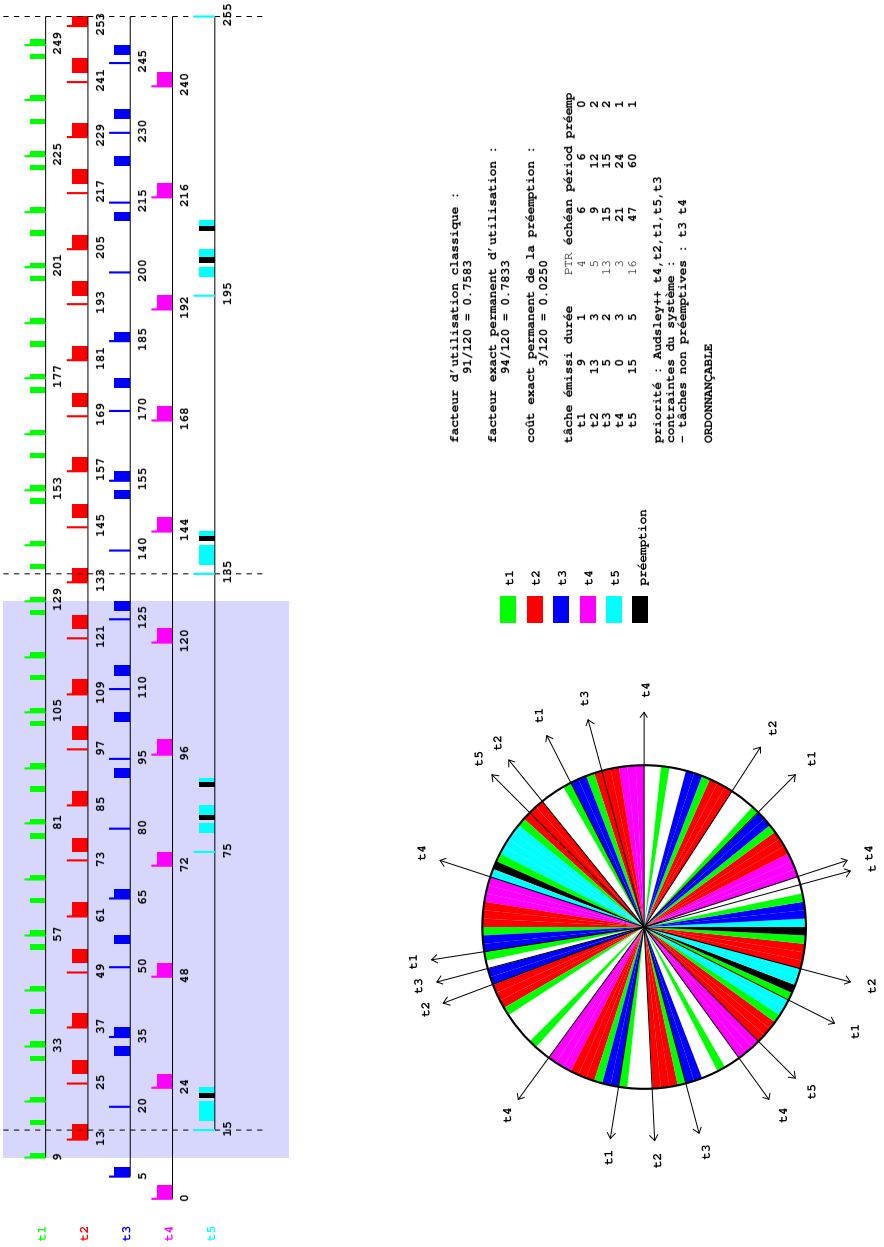


FIG. 4.29 – Résumé des résultats pour le choix $\mathcal{S}_4 = < t_4, t_2, t_1, t_5, t_3 >$.

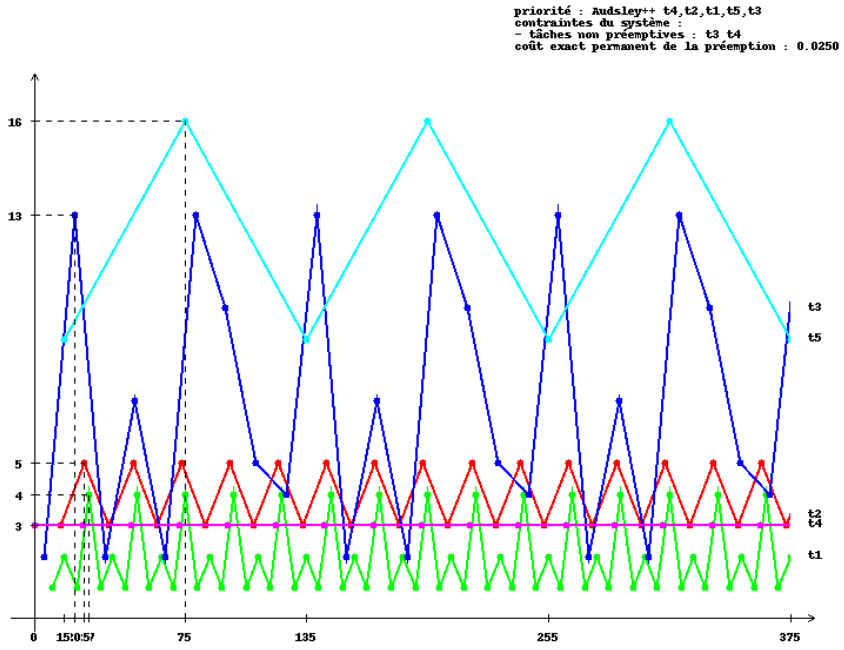


FIG. 4.30 – Courbe du temps de réponse en fonction du temps.

plus solides que celles existant dans la littérature. Ici  signifie que s'il existe un choix de priorités des otâches conduisant à un ordonnancement valide alors le choix de priorités proposé conduit lui aussi à un ordonnancement valide. Ces nouvelles conditions garantiront toujours un bon fonctionnement du système à l'exécution et élimineront tout gaspillage de la ressource (CPU).

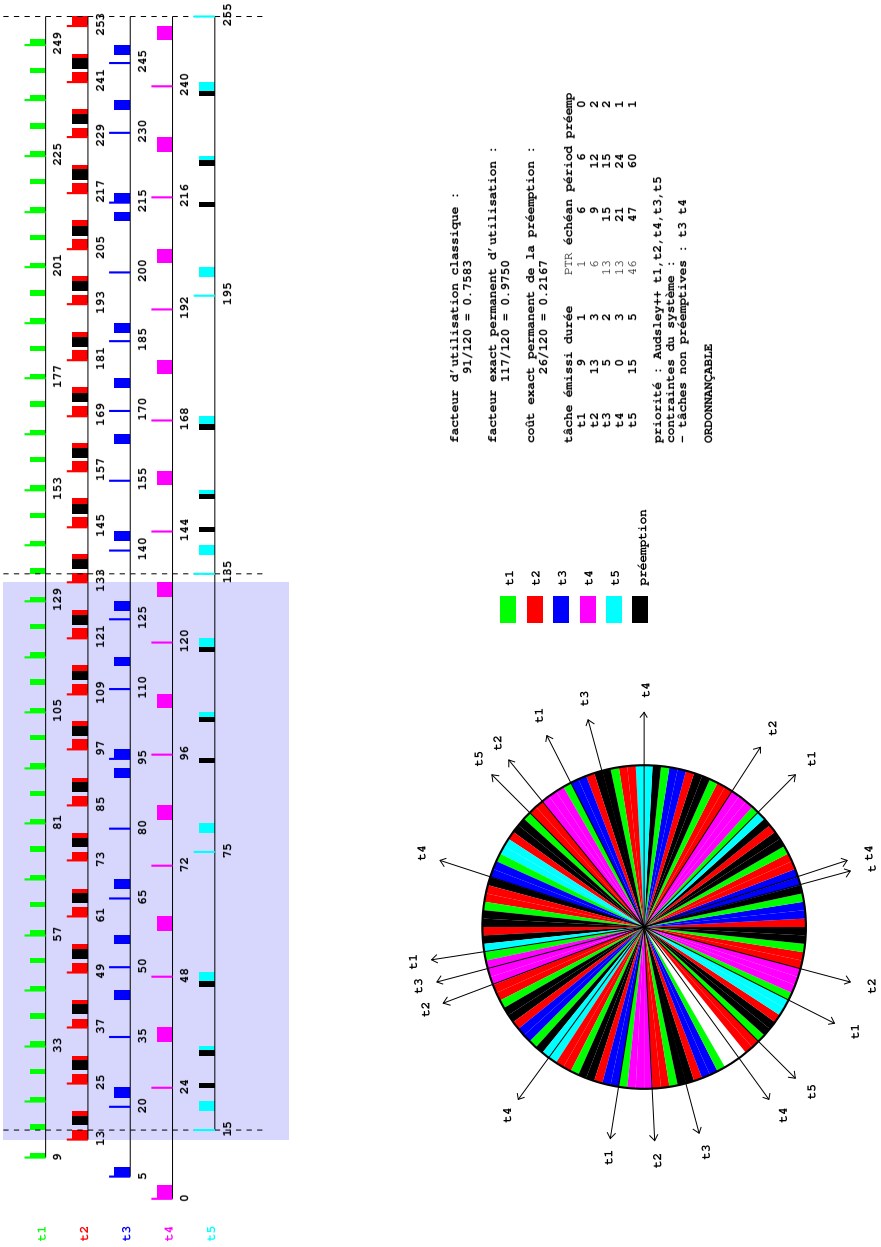


FIG. 4.31 – Résumé des résultats pour le choix $\mathcal{S}_5 = < t_1, t_2, t_4, t_3, t_5 >$.

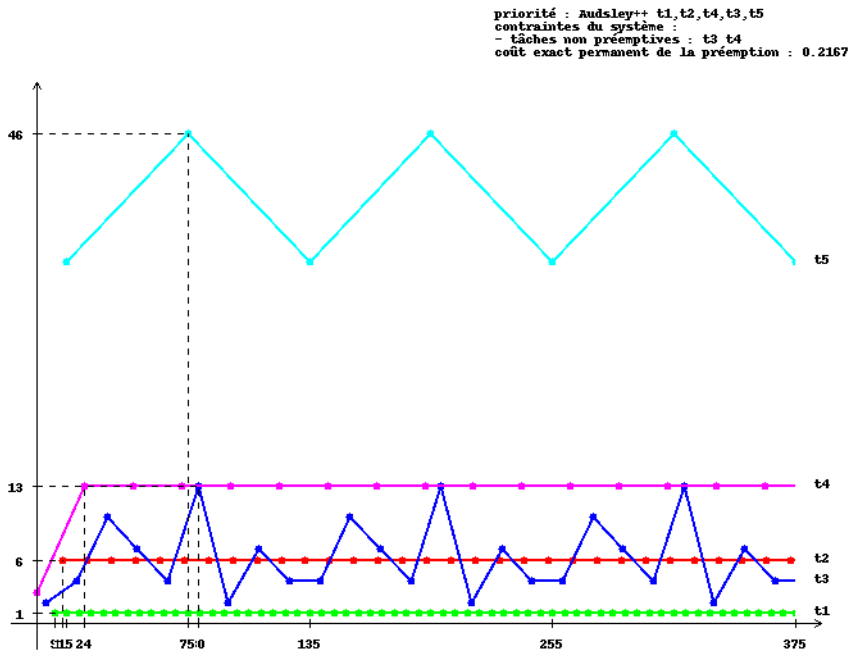


FIG. 4.32 – Courbe du temps de réponse en fonction du temps.

# Chapitre 5

# Ordonnancement temps réel avec contraintes multiples

*L'obstacle est matière*
*à action.*
**Marc Aurèle**

## 5.1 Introduction

Les systèmes temps réel durs auxquels s'applique la théorie de l'ordonnancement sont généralement soumis à des contraintes multiples : les tâches peuvent par exemple être soumises à des contraintes de précédence, de périodicité stricte, de latence, ou encore de gigue. Pendant l'analyse d'ordonnançabilité et la validation d'un tel système, il faut absolument satisfaire ces contraintes afin d'éviter des conséquences désastreuses lors de l'exécution effective du système. Jusqu'ici, pour chaque système que nous avons considéré, nous avons fait l'hypothèse dans le modèle que toutes les tâches (resp. otâches) étaient indépendantes et que la seule contrainte à satisfaire était l'échéance relative de chaque tâche (resp. chaque otâche). Cette hypothèse est limitative puisqu'elle ne permet par exemple pas de représenter les communications entre les tâches. Dans ce chapitre,nous allons présenter la troisième partie de notre contribution. Nous allons lever cette restriction. Nous allons nous intéresser à l'étude de systèmes ayant les contraintes multiples mentionnées ci-dessus (précédence, périodicité stricte, latence, gigue) lorsque la préemption est autorisée entre les tâches (resp. otâches) avec un coût non négligeable. Puisque nous avons montré qu'une tâche temps réel périodique correspond à une otâche périodique particulière, cette étude sera faite à l'aide de systèmes d'otâches périodiques. Dans toute la suite, nous supposons toujours que les otâches périodiques que nous considérons sont canoniques et que la partie initiale de chacune d'elles vaut $\Lambda$. Au début de chaque section, nous donnerons alors l'équivalent de la définition de la contrainte considérée pour les otâches.

## 5.2 Contrainte de précédence

Dans certains systèmes, certaines tâches (resp. otâches) périodiques doivent satisfaire une contrainte de précédence [Law73, PK89, Law71, Ram95, CMT90, MBFSV06]. Cette assertion se traduit par le fait que certaines tâches (resp. otâches) ne peuvent pas débuter leur exécution avant la fin d'une autre tâche (resp. otâches) de même rang. Les exemples de telles tâches (resp. otâches) concernent souvent les communications dans ces systèmes.

**Définition 41** *Soit $O\Gamma_n$ un système constitué de $n$ otâches périodiques,* la contrainte de précédence *sur un sous ensemble $K \subseteq O\Gamma_n$ est donnée par un ordre partiel d'exécution des otâches de $K$. Si $x$ et $y$ sont deux otâches de $K$ ayant une contrainte de précédence, alors on note $x \prec y$ pour signifier que la otâche $y$ ne peut pas débuter avant la fin de la otâche $x$ de même rang. On dit alors que $x$ précède $y$.*

Pour le système $O\Gamma_n$, nous noterons $prec(x) = \{z \in O\Gamma_n : z \prec x\}$ l'ensemble de toutes les otâches de $O\Gamma_n$ qui précèdent la otâche $x$. Ainsi, nous avons $y \prec x$ si et seulement si $y \in prec(x)$. L'ordre partiel défini par la contrainte de précédence entre certaines otâches du système $O\Gamma_n$ définit aussi des *priorités* entre ces otâches. Cela signifie, pour chaque otâche $x$ de $O\Gamma_n$, qu'elle a une priorité qui est inférieure à celle de toute otâche appartenant à $prec(x)$. Graphiquement, la contrainte de précédence est représentée par un *Graphe Acyclique Orienté* / *Directed Acyclic Graph* (*DAG*) dans lequel les sommets sont des otâches et les arcs représentent les précédences. Dans ce *DAG*, $y \prec x$ est illustré par l'existence d'un chemin partant de la otâche $y$ à la otâche $x$ et deux otâches n'ayant pas de contrainte de précédence sont *indépendantes*. Maintenant, si $y \prec x$ et $\nexists z \in O\Gamma_n$ tel que $y \prec z \prec x$, alors on dit que $y$ est un *prédécesseur direct* de $x$. La figure 5.1 illustre la contrainte de précédence entre des otâches périodiques d'un système donné. Dans cette figure, $K_1$, $K_2$ et $K_3$ sont trois sous ensembles du système $O\Gamma_n$. Nous avons $prec(A) = prec(G) = prec(H) = prec(J) = \emptyset$, $prec(B) = prec(D) = \{A\}$, $prec(C) = prec(E) = \{A, B, D\}$, $prec(F) = \{A, B, D, C, E\}$ et enfin $prec(I) = \{A, D\}$. La otâche $A$ est un prédécesseur direct de la otâche $B$ et nous avons $A \prec F$ car il existe le chemin $< A, B, C, F >$ qui part de $A$ à $F$. Les otâches $A$ et $H$ sont indépendantes car il n'existe aucun chemin entre ces deux otâches : $A \notin prec(H)$ et $H \notin prec(A)$. Les otâches $G$ et $J$ sont indépendantes entre elles et en plus, elles sont indépendantes avec toutes les otâches du sous ensemble $K_1 \cup K_2 \cup K_3 = \{A, B, C, D, E, F, H, I\}$.

Du point de vue des priorités, si la contrainte de précédence nous indique par exemple que la otâche $A$ a une priorité supérieure à celle de la otâche $E$ ($A \in prec(E)$), nous ne pouvons pour l'instant rien dire des priorités des otâches $A$ et $G$ puisque nous pouvons bien avoir la priorité de $A$ supérieure à celle de $G$ ou l'inverse tout en satisfaisant toutes les contraintes de précédence. Cette assertion implique la nécessité de définir la notion de

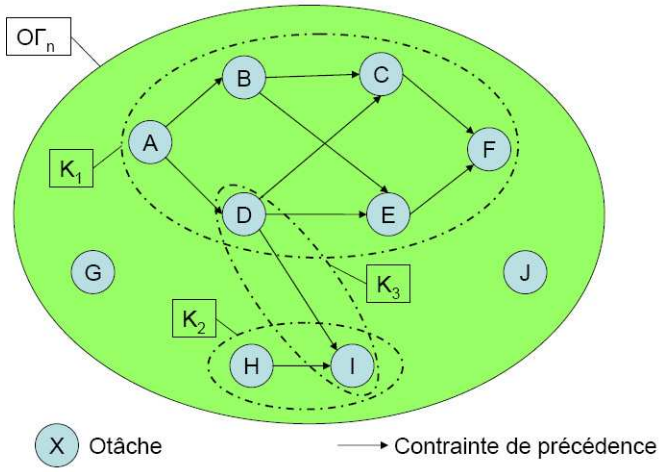


FIG. 5.1 – Illustration de la contrainte de précédence entre des otâches périodiques

*choix admissible des priorités* pour les otâches afin d'éviter toute ambiguïté sur l'ordre d'exécution des otâches.

**Définition 42** *Soit* $O\Gamma_n = \{o\tau_1, \cdots, o\tau_n\}$ *un système constitué de* $n$ *otâches périodiques dont certaines sont soumises à des contraintes de précédence, un* choix admissible des priorités *des otâches de* $O\Gamma_n$ *est une* permutation *des éléments de* $O\Gamma_n$ *qui satisfait toutes les contraintes de précédence.*

**Notation :** Pour le système $O\Gamma_n$, nous noterons un choix admissible des priorités grâce à la relation < $o\tau_{g(1)}, o\tau_{g(2)}, \cdots, o\tau_{g(n)}$ > où la priorité de la otâche $o\tau_{g(l)}$ est supérieure à celle de la otâche $o\tau_{g(r)}$ dès que $l < r$ et $\{g(1), \cdots, g(n)\}$ est l'image de l'ensemble $\{1, \cdots, n\}$ par une permutation $g$.
Dans l'exemple illustré par la figure 5.1, le choix < $G, A, B, C, J, D, E, F, H, I$ > et le choix < $J, A, D, B, H, C, E, F, G, I$ > sont deux choix admissibles de priorités tandis que le choix de priorités < $J, \mathbf{B}, D, C, \mathbf{A}, H, E, F, J, I$ > n'est pas admissible puisque nous avons par exemple $A \in prec(B)$ et il apparaît dans le choix de priorités que $B$ a une priorité supérieure à celle de $A$. Malgré les priorités déjà partiellement imposées par les contraintes de précédence et parce qu'il peut exister plus d'un choix admissible de priorités pour un système donné, il reste à définir un critère permettant de privilegier un choix particulier de priorités parmi tous les choix admissibles de priorités qui satistisfont toutes les contraintes pendant l'application des procédures 6, 7 et 8 du chapitre 4.

Nous rappelons que la contrainte de *précédence* impose qu'une otâche ne puisse pas démarrer son exécution avant la fin d'une autre otâche de même rang qui la précède. Si la première activation de la otâche $o\tau_j$ a lieu à la date $r_j^1$ alors ses activations suivantes

ont lieu aux dates $r_j^k = r_j^1 + (k-1)T_j, \ \ k \geq 1$. Maintenant, si la otâche $o\tau_i$ précède la otâche $o\tau_j$ alors nous devons forcément avoir $r_i^k \leq r_j^k \ \ \forall\, k \geq 1$ pour que la otâche $\tau_j$ satisfasse la contrainte de précédence. Cette assertion additionnée à la première implique que la contrainte de précédence modifie la date de première activation de la otâche $o\tau_j$. Ainsi, lorsque nous avons $A \prec B$, la date de première activation de la otâche $B$ est modifiée grâce à la relation 5.1 [Bla76, CMT90]. Si nous appelons $B^\star$ la otâche obtenue après la modification, alors la date de première activation est le seul paramètre qui change éventuellement de valeur entre les paramètres des otâches $B$ et $B^\star$. Cette modification est illustrée par la figure 5.2. Dans cette figure, la quantité $R_A$ désigne le temps de réponse de la otâche $A$ après sa première activation.

$$r_B{}^\star = max\left(r_B, \max_{A \in prec(B)}(r_A + R_A)\right) \tag{5.1}$$

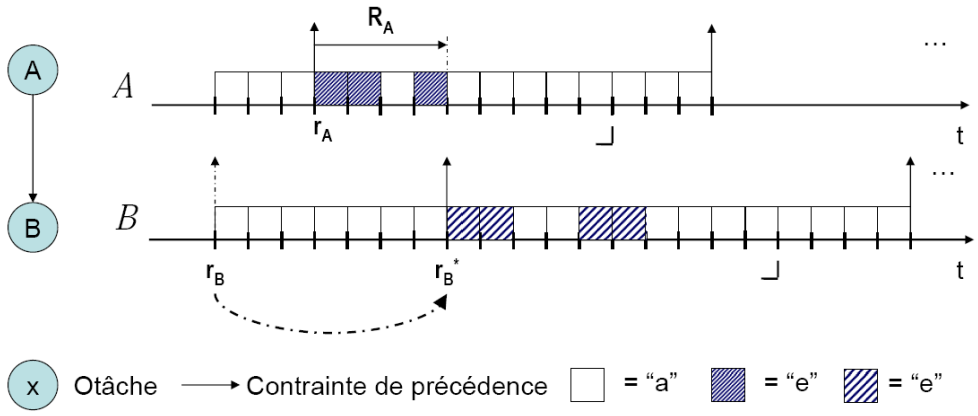


FIG. 5.2 – Illustration de la modification de la date de première activation

Etant donné deux tâches temps réel $\tau_i$ et $\tau_j$ de périodes respectives $T_i$ et $T_j$, *L. Cucu* a démontré dans sa thèse [Cuc04] que la contrainte de précédence entre $\tau_i$ et $\tau_j$ n'a un sens que si la période de $\tau_j$ est supérieure à celle de $\tau_i$. Par conséquent, nous avons la propriété suivante :

$$\tau_i \prec \tau_j \Longrightarrow T_i \leq T_j$$

Puisque cette propriété n'implique que les périodes des tâches, nous pouvons l'étendre aux otâches, ainsi nous obtenons

$$o\tau_i \prec o\tau_j \Longrightarrow T_i \leq T_j \tag{5.2}$$

Dans l'équation 5.2, $o\tau_i$ et $o\tau_j$ désignent deux otâches périodiques de périodes respectives $T_i$ et $T_j$. Donc, l'application de l'opération d'ordonnancement $\oplus$ à des otâches

impliquées dans une contrainte de précédence doit nécessairement se faire suivant les périodes croissantes de ces otâches. En définitive, nous obtenons le théorème 17.

**Théorème 17** *Soit* $O\Gamma_n = \{o\tau_1, \cdots, o\tau_n\}$ *un système constitué de* $n$ *otâches périodiques dont certaines otâches sont soumises à des contraintes de précédence. Soit* $\mathcal{S} = < o\tau_{g(1)}, o\tau_{g(2)}, \cdots, o\tau_{g(n)} >$ *un choix admissible de priorités des otâches de* $O\Gamma_n$*. Soit* $r_{g(j)}, j = 1, \cdots, n$ *la date de première activation de la otâche* $o\tau_{g(j)}$*. Le système* $O\Gamma_n$ *est* ordonnançable *avec respect des contraintes de précédence pour le choix de priorités* $\mathcal{S}$ *si et seulement si le choix de priorités* $\mathcal{S}^\star = < o\tau^\star_{g(1)}, o\tau^\star_{g(2)}, \cdots, o\tau^\star_{g(n)} >$ *est ordonnançable, c'est-à-dire*

$$\mathcal{R}_n = \bigoplus_{i=1}^{n} o\tau^\star_{g(i)} \neq \Lambda$$

*avec*

$$r_{g(i)}{}^\star = max\left(r_{g(i)}, \max_{o\tau_{g(j)} \in prec(o\tau_{g(i)})}(r_{g(j)} + R_{o\tau_{g(j)}})\right)$$

**Preuve** : La preuve du théorème 17 découle directement de la relation 5.1, de la définition 33 et de l'application des procédures 6, 7, 8 du chapitre 4. ■

**Exemple**

Soit $\Gamma_4 = \{t_1, t_2, t_3, t_4\}$ un système constitué de quatre tâches temps réel périodiques. Les caractéristiques des tâches sont résumées dans la table 5.1. Les contraintes de précédence sont données par les relations $t_1 \prec t_2$, $t_1 \prec t_4$ et $t_3 \prec t_4$ et sont illustrées grâce au graphe de la figure 5.3.

| Tâche | $r_i^1$ | $C_i$ | $D_i$ | $T_i$ | $\alpha_i$ |
|---|---|---|---|---|---|
| $t_1$ | 9 | 1 | 6 | 6 | 1 |
| $t_2$ | 13 | 2 | 9 | 12 | 1 |
| $t_3$ | 5 | 3 | 10 | 10 | 2 |
| $t_4$ | 0 | 5 | 29 | 30 | 1 |

TAB. 5.1 – Caractéristiques des tâches

Grâce à la donnée des contraintes de précédence du système (voir figure 5.3), quatre choix de priorités sont admissibles : $\mathcal{S}_1 = < t_1, t_2, t_3, t_4 >$, $\mathcal{S}_2 = < t_1, t_3, t_2, t_4 >$, $\mathcal{S}_3 = < t_3, t_1, t_2, t_4 >$ et enfin $\mathcal{S}_4 = < t_3, t_1, t_4, t_2 >$. Parmi ces choix de priorités, seuls deux sont ordonnançables ($\mathcal{S}_2$ et $\mathcal{S}_3$). La figure 5.4 et la figure 5.5 résument les résultats obtenus pour chacun de ces deux choix admissibles. La figure 5.6 illustre les courbes de temps de

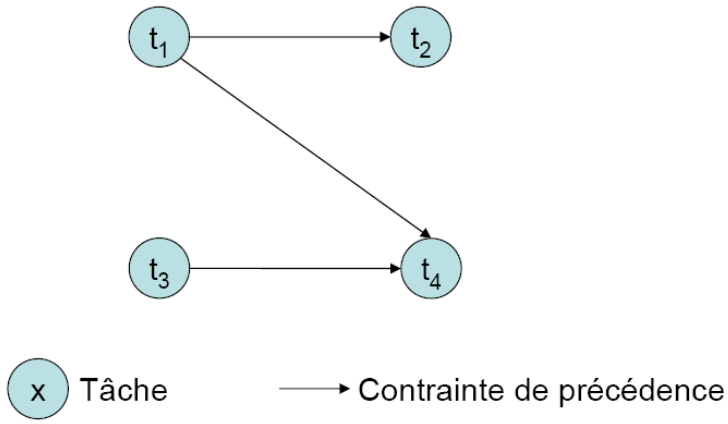


FIG. 5.3 – Illustration du graphe de précédence

réponse en fonction du temps de chaque tâche pour ces choix. Le coût exact permanent de la préemption vaut $13.33\%$ pour le choix de priorités $\mathcal{S}_2$ et seulement $6.67\%$ pour le choix de priorités $\mathcal{S}_3$. Cette différence de coût de la préemption est due au fait que le coût d'une préemption de la tâche $t_3$ vaut $\alpha_3 = 2$ unités de temps et au fait que $t_3$ est préemptée dans l'ordonnancement produit pour le choix de priorités $\mathcal{S}_2$ et pas dans celui produit pour le choix de priorités $\mathcal{S}_3$.

## 5.3 Contrainte de périodicité stricte

Dans certains systèmes, certaines tâches (resp. otâches) périodiques doivent satisfaire une contrainte de périodicité stricte [DCG00, CKS02, CS03]. Plusieurs exemples de telles tâches (resp. otâches) concernent souvent des entrées/sorties de tels systèmes : le maintien de la stabilité d'un système de pilotage d'un procédé passe par une commande stable (au sens période de restitution) des actionneurs, la manipulation de matières dangereuses par des robots nécessite une bonne transmission de l'information entre le procédé et le système temps réel.

**Définition 43** *Soit $x$ une otâche périodique, on dit que $x$ est soumise à la* contrainte de périodicité stricte *lorsque toutes les sous-durées d'exécution de $x$ doivent toujours débuter exactement à leurs dates de sous-activation.*

La contrainte de *périodicité stricte* impose à chaque otâche périodique $o\tau_j$ soumise à cette contrainte, que toutes ses sous-durées d'exécution débutent exactement à leurs dates de sous-activation. Sans nuire à la généralité et grâce à la décomposition des

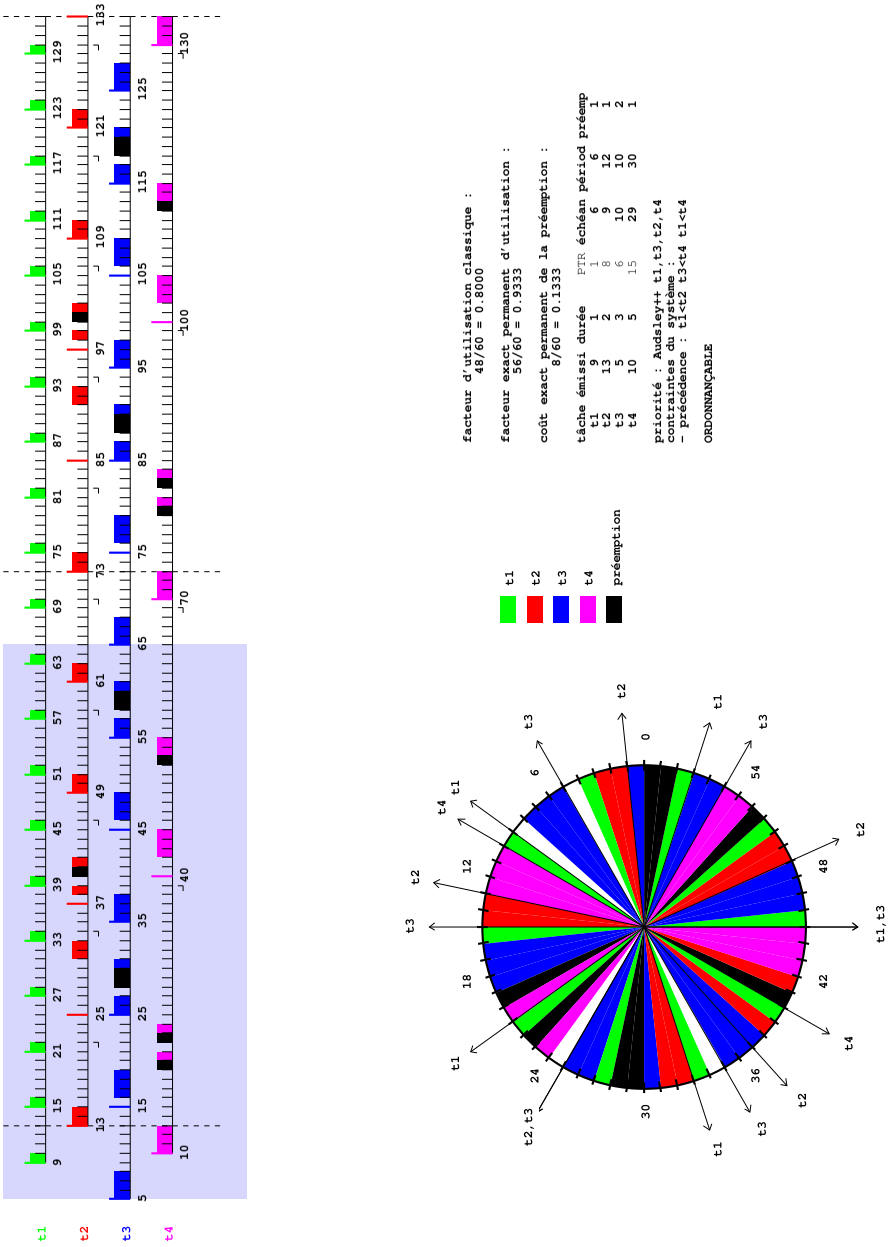


FIG. 5.4 – Résumé des résultats pour le choix admissible <$t_1$, $t_3$, $t_2$, $t_4$>

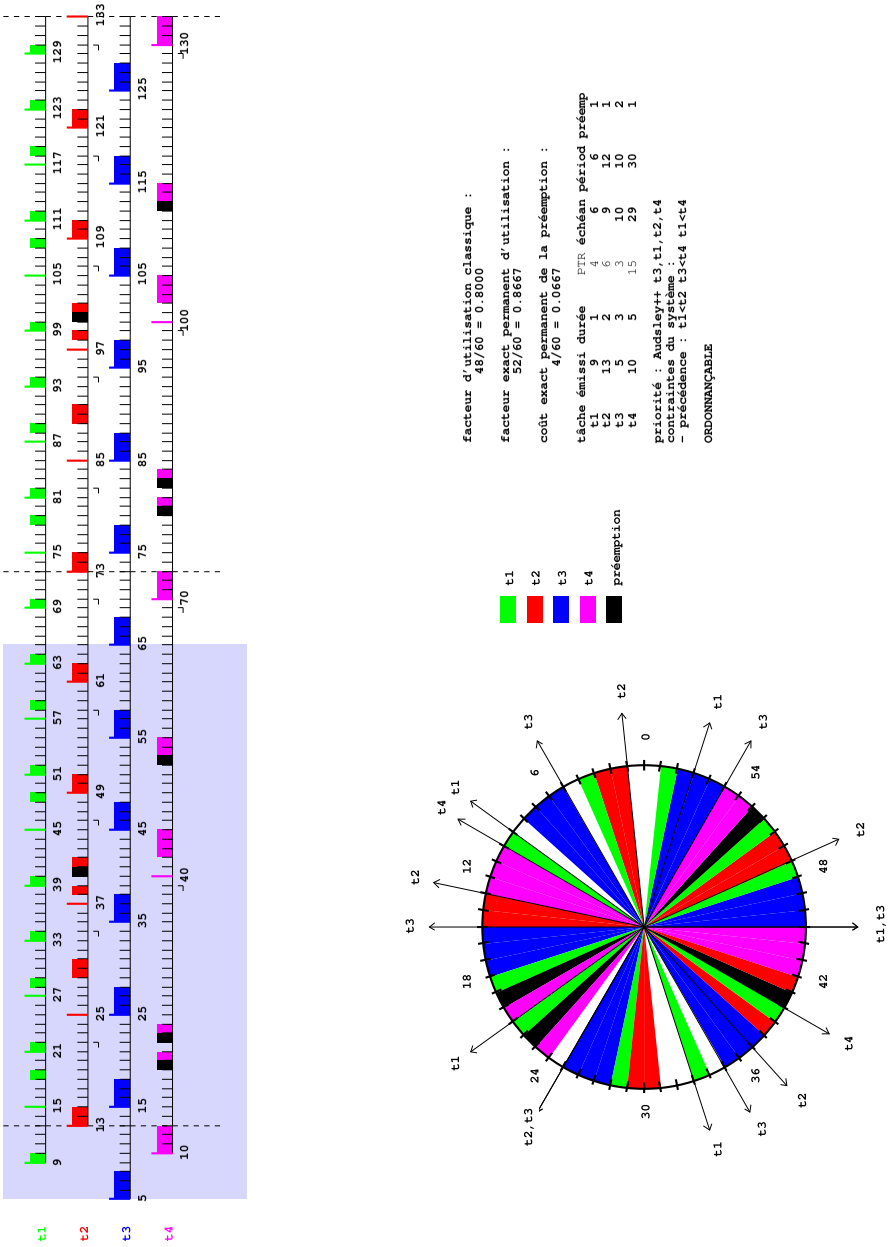


FIG. 5.5 – Résumé des résultats pour le choix admissible <$t_3$, $t_1$, $t_2$, $t_4$>

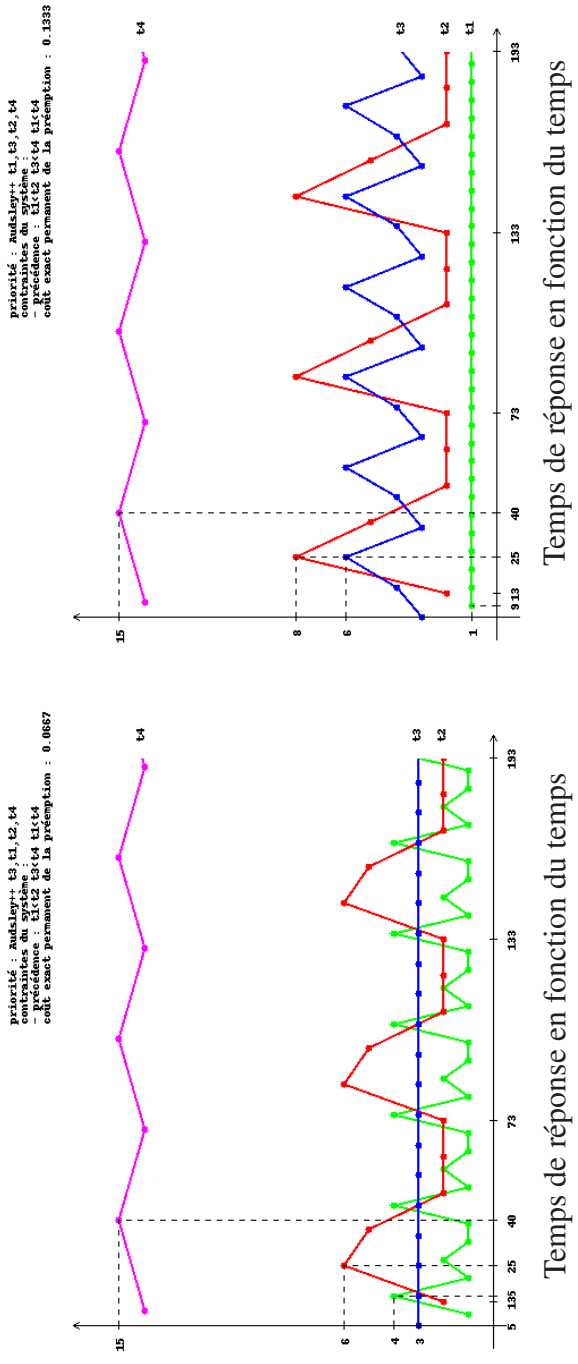


FIG. 5.6 – ***Haut)*** Choix <$t_1$, $t_3$, $t_2$, $t_4$>. ***Bas)*** Choix <$t_3$, $t_1$, $t_2$, $t_4$>.

otâches du chapitre 3, il suffit de raisonner en considérant que la otâche $o\tau_j$ est canonique régulière (voir figure 3.8) afin de garantir cette contrainte pour cette otâche. Si la date de première activation de $o\tau_j$ est $r_j^1$ alors ses activations suivantes ont lieu aux dates $r_j^k = r_j^1 + (k-1)T_j, \ \ k \geq 1$. Maintenant, si $s_j^k$ désigne la date de début d'exécution effective de la otâche $o\tau_j$ après sa $k^{ieme}$ activation alors nous devons forcément avoir $s_j^k = r_j^k$ et $s_j^{k+1} - s_j^k = T_j, \ \ \forall\, k \geq 1$ pour que la otâche $\tau_j$ satisfasse la contrainte de périodicité stricte. Par conséquent, nous utilisons le modèle de tâches sans dates de réveils introduit au chapitre 1. Comme nous sommes dans le cadre de l'étude de problèmes d'ordonnancement monoprocesseurs, les deux assertions suivantes doivent toujours être vérifiées :

1. deux dates d'activation de deux otâches soumises chacune à la contrainte de périodicité stricte *ne doivent jamais* être égales.
2. une otâche soumise à la contrainte de périodicité stricte *ne doit jamais* être activée lorsque le processeur est en train d'exécuter une autre otâche ayant une priorité plus élevée.

Puisque nous avons montré la correspondance entre une tâche temps réel périodique $\tau_j$ et la otâche périodique $o\tau_j$ associée (voir relation 3.8), les deux assertions ci-dessus valent aussi pour des tâches temps réel soumises à cette contrainte. Le point 1 est garanti pour un système donné grâce au théorème 18 et au théorème 19 et le point 2 grâce au théorème 20.

La preuve des théorèmes 18 et 19 utilise la propriété de périodicité stricte des otâches et le *théorème de Bézout* [1]. Le *théorème de Bézout* affirme que l'équation $a \cdot x + b \cdot y = 1$ admet des solutions si et seulement si les entiers relatifs $a$ et $b$ sont premiers entre eux, c'est-à-dire, $pgcd(a,b) = 1$ ou $pgcd(a,b)$ désigne le plus grand commun diviseur de $a$ et $b$. La première démonstration actuellement connue de ce théorème est due à *Claude-Gaspard Bachet de Méziriac* [2] ; elle figure dans la seconde édition de son ouvrage "*Problèmes plaisants et délectables qui se font par les nombres*", parue en $1624$.

**Théorème 18** *Soit $O\Gamma_n$ un système constitué de $n \geq 2$ otâches périodiques canoniques régulières. Soient $o\tau_i = (\tau_1)_{r_i^1}^{\infty}$, $o\tau_j = (\tau_2)_{r_j^1}^{\infty}$ deux otâches de $O\Gamma_n$ de périodes respectives $T_i = |\tau_1|$ et $T_j = |\tau_2|$ soumises à la contrainte de périodicité stricte. Si $o\tau_i$ et $o\tau_j$ démarrent leurs premières exécutions respectivement aux dates $s_i^1$ et $s_j^1$ et sont telles que*

$$pgcd(T_i, T_j) = 1 \tag{5.3}$$

*Alors le système $O\Gamma_n$ est non ordonnançable. De plus, toute autre hypothèse supplémentaire sur le système est inutile lorsque l'égalité 5.3 est satisfaite.*

[1] Étienne Bézout, né à Nemours le 31 mars 1730 et mort à Avon le 27 septembre 1783, est un mathématicien français.

[2] Claude-Gaspard Bachet dit de Méziriac, né le 9 octobre 1581 à Bourg-en-Bresse, où il est mort le 26 février 1638, est un mathématicien, poète et traducteur français.

**Preuve** : Soient $o\tau_i = (\tau_1)_{r_i^1}^{\infty}$, $o\tau_j = (\tau_2)_{r_j^1}^{\infty}$ deux otâches de $O\Gamma_n$ de périodes respectives $T_i = |\tau_1|$ et $T_j = |\tau_2|$ soumises à la contrainte de périodicité stricte. Nous supposons que $o\tau_i$ et $o\tau_j$ démarrent leurs premières exécutions respectivement aux dates $s_i^1$ et $s_j^1$ et sont telles que $pgcd(T_i, T_j) = 1$. Sans nuire à la généralité, supposons $s_j^1 \geq s_i^1$ et posons $\gamma = s_j^1 - s_i^1$.
Si $\gamma = 0$ alors le résultat est immédiat puisque nous avons alors $s_j^1 = s_i^1$. Ceci n'est pas possible car nous sommes dans un cadre monoprocesseur.
Supposons maintenant que $\gamma \neq 0$, alors

$$\exists (k_1, k_2) \in \mathbb{Z}^2 \text{ tel que } k_1 T_i = k_2 T_j + \gamma. \tag{5.4}$$

En effet, grâce au *théorème de Bézout*, puisque $pgcd(T_i, T_j) = 1$, alors $\exists (m_1, m_2) \in \mathbb{Z}^2$ tel que $m_1 T_i - m_2 T_j = 1$, c'est-à-dire $\exists (m_1, m_2) \in \mathbb{Z}^2$ tel que $(m_1\gamma) \cdot T_i - (m_2\gamma) \cdot T_j = \gamma$. Il découle de la dernière égalité que le couple $(m_1\gamma, m_2\gamma) \in \mathbb{Z}^2$ est une solution particulière de l'équation 5.4 et ainsi,

$$\begin{cases} (k_1 - m_1\gamma) \cdot T_i - (k_2 - m_2\gamma) \cdot T_j = 0 & (*) \\ (m_1\gamma) \cdot T_i - (m_2\gamma) \cdot T_j = \gamma & (**) \end{cases}$$

L'égalité $(*)$ donne $(k_1 - m_1\gamma) \cdot T_i = (k_2 - m_2\gamma) \cdot T_j$. Par conséquent, nous avons $T_i \mid (k_2 - m_2\gamma) \cdot T_j$ et $pgcd(T_i, T_j) = 1$, donc $T_i \mid (k_2 - m_2\gamma)$, c'est-à-dire $\exists k \in \mathbb{Z}$ tel que $k_2 - m_2\gamma = k \cdot T_i$. En reportant cette égalité dans $(*)$, on obtient $k_1 - m_1\gamma = kT_j$. D'où l'ensemble $\{(m_1\gamma + kT_j, m_2\gamma + k \cdot T_i) \in \mathbb{Z}^2 : k \in \mathbb{Z}\}$ représente toutes les solutions de l'équation 5.4. Par suite pour tout entier $k$ fixé, nous avons une activation simultanée des otâches $o\tau_i$ et $o\tau_j$ à la date

$$\begin{aligned} t = s_i^1 + (m_1\gamma + kT_j) \cdot T_i &= s_i^1 + (m_1(s_j^1 - s_i^1) + kT_j) \cdot T_i \\ = s_j^1 + (m_2\gamma + kT_i) \cdot T_j &= s_j^1 + (m_2(s_j^1 - s_i^1) + kT_i) \cdot T_j \end{aligned}$$

Ce qui viole la contrainte de périodicité stricte et rend le système non ordonnançable. Puisqu'il n'est pas possible d'éviter cette situation, toute autre hypothèse supplémentaire sur le système est inutile lorsque l'égalité 5.3 est satisfaite. ■

**Remarque 16** *Dans l'expression 5.4, le choix des entiers $k_1$ et $k_2$ peut se faire sans problème dans l'ensemble des entiers naturels $\mathbb{N}$ afin de conserver l'idée du temps positif. En effet, si l'expression 5.4 est satisfaite avec $k_1$ ou $k_2$ négatif, alors il suffit de considérer un entier $h_0 \in \mathbb{N}$ assez grand tel que $k_1' = k_1 + h_0 T_j \in \mathbb{N}$ et $k_2' = k_2 + h_0 T_i \in \mathbb{N}$. Ainsi l'expression 5.4 est satisfaite si et seulement si $k_1' T_i - k_2' T_j = \gamma$. Dans toute la suite, nous ne ferons plus attention à ce détail car il est facile à résoudre.*

**Exemple**

Cet exemple illustre le théorème 18. Soit $O\Gamma_2$ un système d'otâches périodiques et $\{o\tau_1, o\tau_2\} \subseteq O\Gamma_2$ tels que

$$\begin{cases} o\tau_1 = \{e, a, a, a\}_0^\infty \\ o\tau_2 = \{e, a, a, a, a\}_3^\infty \end{cases}$$

Les périodes des otâches $o\tau_1$ et $o\tau_2$ sont respectivement données par $T_1 = 4$ et $T_2 = 5$, donc $pgcd(T_1, T_2) = 1$. Grâce au théorème 18, nous pouvons conclure que ce système est non ordonnançable. En effet, à la date $t = 8$ dans la figure 5.7 les deux otâches $o\tau_1$ et $o\tau_2$ sont activées simultanément et par conséquent nous ne pouvons pas les ordonnancer sans violer la contrainte de périodicité stricte de l'une des deux otâches. En d'autres termes, le résultat de l'application de l'opération d'ordonnancement $\oplus$ dont $o\tau_1$ et $o\tau_2$ seraient à un moment donné un des opérandes vaut $\Lambda$.

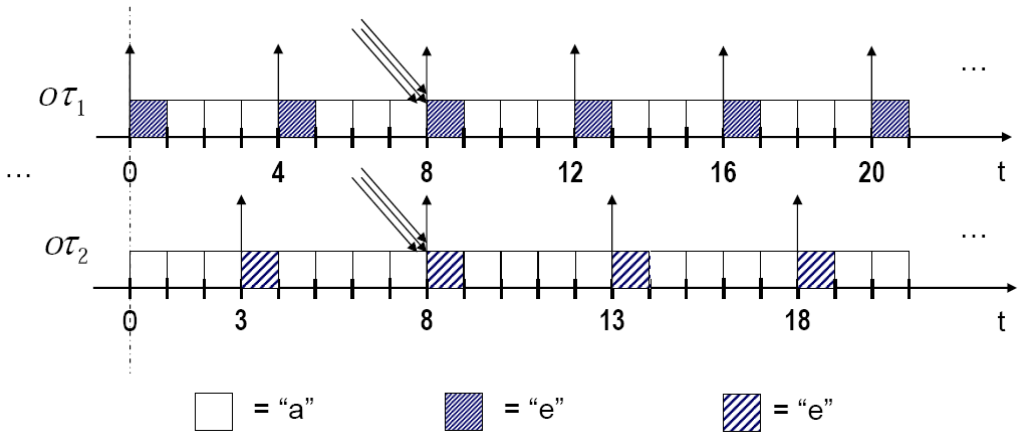


FIG. 5.7 – Echec de la contrainte de périodicité stricte : $pgcd(T_1, T_2) = 1$

**Théorème 19** *Soit $O\Gamma_n$ un système constitué de $n \geq 2$ otâches périodiques canoniques régulières. Soient $o\tau_i = (\tau_1)_{r_i^1}^\infty$, $o\tau_j = (\tau_2)_{r_j^1}^\infty$ deux otâches de $O\Gamma_n$ de périodes respectives $T_i = |\tau_1|$ et $T_j = |\tau_2|$ soumises à la contrainte de périodicité stricte. Si $o\tau_i$ et $o\tau_j$ démarrent leurs premières exécutions respectivement aux dates $s_i^1$ et $s_j^1$ et sont telles que*

$$pgcd(T_i, T_j) \mid (s_j^1 - s_i^1) \tag{5.5}$$

*Alors le système $O\Gamma_n$ est non ordonnançable. De plus, toute autre hypothèse supplémentaire sur le système est inutile lorsque l'égalité 5.3 est satisfaite.*

**Preuve** : Soient $o\tau_i = (\tau_1)_{r_i^1}^{\infty}$, $o\tau_j = (\tau_2)_{r_j^1}^{\infty}$ deux otâches de $O\Gamma_n$ de périodes respectives $T_i = |\tau_1|$ et $T_j = |\tau_2|$ soumises à la contrainte de périodicité stricte. Nous supposons que $s_i^1$ et $s_j^1$ désignent respectivement les dates de première exécution de $o\tau_i$ et $o\tau_j$. De plus, nous supposons que $o\tau_i$ et $o\tau_j$ sont telles que $pgcd(T_i, T_j) \mid (s_j^1 - s_i^1)$. Sans nuire à la généralité, nous choisissons une fois de plus $s_j^1 \geq s_i^1$ et posons $\gamma = s_j^1 - s_i^1$.
Si nous considérons directement le cas où $\gamma \neq 0$ (*Cf. la preuve du théorème 19*), alors

$$\exists (k_1, k_2) \in \mathbb{Z}^2 \text{ tel que } k_1 T_i = k_2 T_j + \gamma. \tag{5.6}$$

En effet, en supposant que $pgcd(T_i, T_j) = \lambda \in \mathbb{N}^*$, il existe $T_i^1$ et $T_j^2 \in \mathbb{N}^*$ tels que $T_i = \lambda T_i^1$, $T_j = \lambda T_j^2$ et $pgcd(T_i^1, T_j^2) = 1$. Grâce au *théorème de Bézout*, puisque $pgcd(T_i^1, T_j^2) = 1$, alors $\exists (m_1, m_2) \in \mathbb{Z}^2$ tel que $m_1 T_i^1 - m_2 T_j^2 = 1$, c'est-à-dire $\exists (m_1, m_2) \in \mathbb{Z}^2$ tel que $m_1 \cdot (\lambda T_i^1) - m_2 \cdot (\lambda T_j^2) = \lambda$. Il découle de la dernière égalité que

$$(k_1 - m_1) T_i - (k_2 - m_2) T_j = \gamma - \lambda$$

Par hypothèse, nous avons $\lambda \mid \gamma$, c'est-à-dire $\exists p \in \mathbb{N}^*$ tel que $\gamma = p\lambda$. Finalement, nous avons $(k_1 - m_1) T_i - (k_2 - m_2) T_j = (p-1)\lambda$, c'est-à-dire $(k_1 - m_1) T_i^1 - (k_2 - m_2) T_j^2 = (p-1)$. Ainsi,

$$\begin{cases} m_1 (p-1) T_i^1 - m_2 (p-1) T_j^2 = (p-1) & (*) \\ (k_1 - m_1) T_i^1 - (k_2 - m_2) T_j^2 = (p-1) & (**) \end{cases}$$

Donc

$$\exists (m_1, m_2) \in \mathbb{Z}^2 \text{ tel que } (k_1 - m_1 p) T_i^1 = (k_2 - m_2 p) T_j^2$$

Par conséquent, nous avons $T_i^1 \mid (k_2 - m_2 p) \cdot T_j^2$ et $pgcd(T_i^1, T_j^2) = 1$, donc inévitablement, $T_i^1 \mid (k_2 - m_2 p)$, c'est-à-dire $\exists k \in \mathbb{N}^*$ tel que $k_2 - m_2 p = k T_i^1$. En reportant cette égalité, on obtient aussi $k_1 - m_1 p = k T_j^2$. D'où l'ensemble

$$\left\{ \left( m_1 \frac{s_j^1 - s_i^1}{pgcd(T_i, T_j)} + k \frac{T_j}{pgcd(T_i, T_j)}, m_2 \frac{s_j^1 - s_i^1}{pgcd(T_i, T_j)} + k \frac{T_i}{pgcd(T_i, T_j)} \right) \in \mathbb{Z}^2 : k \in \mathbb{Z} \right\}$$

représente toutes les solutions de l'équation 5.6. Par suite, pour tout entier $k$ fixé, nous avons une activation simultanée des otâches $o\tau_i$ et $o\tau_j$ à la date

$$\begin{aligned} t &= s_i^1 + \left[ (m_1 (s_j^1 - s_i^1) + k T_j) \right] \cdot \frac{T_i}{pgcd(T_i, T_j)} \\ &= s_j^1 + \left[ (m_2 (s_j^1 - s_i^1) + k T_i) \right] \cdot \frac{T_j}{pgcd(T_i, T_j)} \end{aligned}$$

Ce qui viole la contrainte de périodicité stricte et rend le système non ordonnançable. Puisqu'il n'est pas possible d'éviter cette situation, toute autre hypothèse supplémentaire sur le système est inutile lorsque l'égalité 5.5 est satisfaite. ■

**Exemple**

Cet exemple illustre le théorème 19. Soit $O\Gamma_2$ un système d'otâches périodiques et $\{o\tau_1, o\tau_2\} \subseteq O\Gamma_2$ tels que

$$\begin{cases} o\tau_1 = \{e, a, a, a\}_0^\infty \\ \\ o\tau_2 = \{e, a, a, a, a, a\}_6^\infty \end{cases}$$

Les périodes des otâches $o\tau_1$ et $o\tau_2$ sont respectivement données par $T_1 = 4$ et $T_2 = 6$, donc $pgcd(T_1, T_2) = 2 \mid (s_2^1 - s_1^1) = 6$. Grâce au théorème 19, nous pouvons conclure que ce système est non ordonnançable. En effet, à la date $t = 12$ dans la figure 5.8 les deux otâches $o\tau_1$ et $o\tau_2$ sont activées simultanément et par conséquent nous ne pouvons pas les ordonnancer sans violer la contrainte de périodicité stricte de l'une des deux otâches. En d'autres termes, le résultat de l'application de l'opération $\oplus$ dont $o\tau_1$ et $o\tau_2$ seraient à un moment donné un des opérandes vaut $\Lambda$.

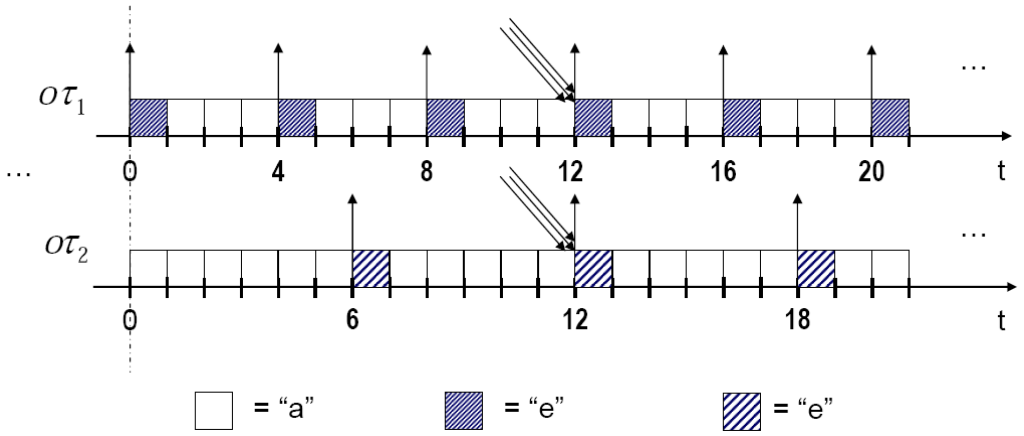


FIG. 5.8 – Echec de la contrainte de périodicité stricte : $pgcd(T_1, T_2) \mid (s_2^1 - s_1^1)$

Le théorème 18 et le théorème 19 donnent des conditions de non ordonnançabilité d'un système d'otâches périodiques en exploitant juste des relations sur les périodes des otâches. Ces conditions sont données indépendamment des priorités attribuées aux otâches (c'est-à-dire, de l'ordre dans lequel on applique l'opération $\oplus$). Lorsque des priorités sont attribuées aux otâches, la contrainte de périodicité stricte signifie entre autre pour la otâche périodique $o\tau_j$ qu'aucun $T_j$*-mesoids* ne doit débuter par une séquence de symboles "$e$" avant l'application de l'opération $\oplus$ au niveau $j$. Si le *chevauchement* désigne cette situation, alors le point 2 est assuré par le théorème 20.

**Théorème 20** *Soit $O\Gamma_n = \{o\tau_1, \cdots, o\tau_n\}$ un système constitué de $n \geq 2$ otâches périodiques canoniques régulières rangées suivant les priorités décroissantes. Soit $o\tau_j = (\tau_2)_{r_j^1}^{\infty} \in O\Gamma_n$ de période $T_j = |\tau_2|$ soumise à la contrainte de périodicité stricte. S'il existe un $T_j$-*mesoid *qui débute par une séquence de symboles "$e$", alors le système $O\Gamma_n$ est non ordonnançable. De plus, toute autre hypothèse supplémentaire sur le système est inutile lorsque l'égalité 5.3 est satisfaite.*

**Preuve** : La preuve de ce théorème découle directement du fait qu'une otâche donnée ne peut être préemptée que par celles ayant une priorité supérieure. ■

**Exemple**

Cet exemple illustre le théorème 20. Soit $O\Gamma_2 = \{o\tau_1, o\tau_2\}$ un système constitué de deux otâches périodiques tels que

$$\begin{cases} o\tau_1 = \{e, e, e, a, a, a, a, a, a\}_0^{\infty} \\ o\tau_2 = \{e, e, a, a, a, a\}_6^{\infty} \end{cases}$$

Grâce au théorème 20, nous pouvons conclure que ce système est non ordonnançable. En effet, si nous supposons que la otâche $o\tau_2$ a la priorité la plus élevée alors à la date $t = 10$ dans la figure 5.9, la otâche $o\tau_1$ est activée pendant que le processeur est occupé et par conséquent nous ne pouvons pas ordonnancer $o\tau_1$ sans violer sa contrainte de périodicité stricte, c'est-à-dire le résultat de l'application de l'opération $\oplus$ à ce niveau vaut $\Lambda$. De même, si nous supposons que la otâche $o\tau_1$ a la priorité la plus élevée alors à la date $t = 21$ dans la figure 5.9 la otâche $o\tau_2$ est activée pendant que le processeur est occupé et par conséquent nous ne pouvons pas ordonnancer $o\tau_2$ sans violer sa contrainte de périodicité stricte, c'est-à-dire le résultat de l'application de l'opération $\oplus$ à ce niveau vaut $\Lambda$.

**Remarque 17** *En définitive, un système d'otâches périodiques dans lequel certaines otâches sont soumises à la contrainte de périodicité stricte est* potentiellement ordonnançable *si et seulement si nous ne nous retrouvons pas dans l'une des situations décrites soit par le théorème 18, le théorème 19 ou le théorème 20.*

Maintenant que nous avons défini le cadre dans lequel un système avec la contrainte de périodicité stricte est *potentiellement ordonnançable*, nous allons nous intéresser à une autre contrainte temps réel tout aussi importante que la précédente.

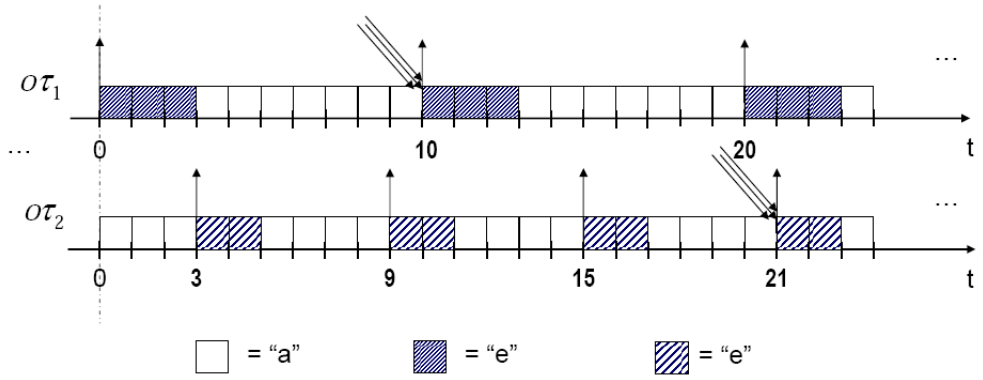


FIG. 5.9 – Echec de la contrainte de périodicité stricte : *chevauchement*

## 5.4 Contrainte de latence

Pour un système temps réel dur, il est important de garantir le traitement de certaines données dans un temps borné. Si par exemple une donnée critique produite à la suite d'une activation précise d'une tâche périodique $x$ doit avoir été pris en compte à la fin de l'exécution d'une activation précise d'une tâche périodique $y$, alors il peut s'avérer important et même indispensable de garantir que le temps effectif qui s'écoule entre la date de début de la production de la donnée et la date de fin de sa prise en compte n'excède jamais une certaine durée. Une telle contrainte est une *contrainte de latence* [God00, Cuc04, CPS07].

**Définition 44** *Soit $O\Gamma_n$ un système constitué de $n$ otâches périodiques et soient $x$ et $y$ deux otâches de $O\Gamma_n$. La* latence *entre la $i^{ieme}$ activation de la otâche $x$ et la $j^{ieme}$ activation de la otâche $y$ est un nombre entier $L_{x,y}^{i,j}$ correspondant au nombre d'unités de temps effectif qui s'écoulent entre la date de début d'exécution de la $i^{ieme}$ activation de $x$ et la date de fin de la $j^{ieme}$ activation de $y$.*

**Définition 45** *Une* contrainte de latence *est une contrainte qui impose qu'une certaine latence $L_{x,y}^{i,j}$ soit inférieure à une certaine valeur entière $L$.*

**Convention :** Soient $x$ et $y$ deux otâches périodiques d'un système donné. Par souci de clarté, nous spécifierons la contrainte de latence entre la $i^{ieme}$ activation de la otâche $x$ et la $j^{ieme}$ activation de la otâche $y$ en utilisant la syntaxe illustrée par la formule 5.7.

$$x(i) \rightarrow y(j) = L_{x,y}^{i,j} < L \tag{5.7}$$

Pour un système $O\Gamma_n = \{o\tau_1, \cdots, o\tau_n\}$ constitué de $n$ otâches périodiques, nous avons montré dans le chapitre 3, grâce à la période de chaque otâche, que la phase transitoire et la phase permanente suffisent pour faire une étude complète de l'ordonnançabilité du système. En effet ces deux phases garantissent de façon déterministe le comportement globale du système lors de son exécution, c'est-à-dire si toutes les contraintes du système sont satisfaites dans ces deux phases, alors celui-ci est déclaré *ordonnançable*. Puisque la phase permanente se répète indéfiniment, nous noterons $P_q$ ($q \geq 1$ la $q^{ieme}$ *répétition* de la phase permanente. La figure 5.10 illustre la phase transitoire et les répétitions de la phase permanente d'un système donné. Nous rappelons que la longueur de la phase permanente vaut $H_n = ppcm\{T_i : i = 1, \cdots, n\}$ où $T_i$ désigne la période de la otâche $o\tau_i$. Dans cette phase, la otâche $o\tau_i$ est activée exactement $n_i = \dfrac{H_n}{T_i}$ fois. La distance entre la première activation et la dernière activation de $o\tau_i$ est exactement égale à $(n_i - 1)T_i$. Par conséquent, la distance entre la dernière activation de $o\tau_i$ dans la répétition $P_q$, $q \geq 1$ et sa première activation dans la répétition suivante ($P_{q+1}$) est égale à $H_n - (n_i - 1)T_i$. Cette dernière quantité vaut exactement $T_i$ puisque nous avons $n_i = \dfrac{H_n}{T_i}$.

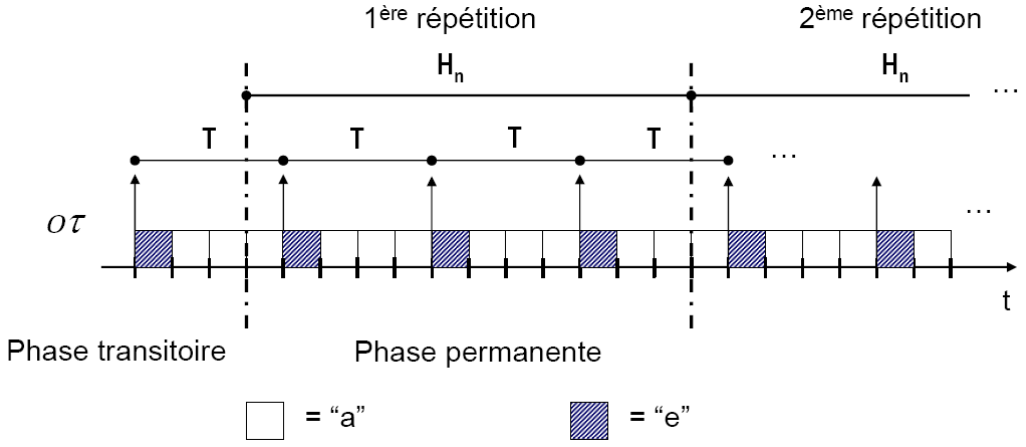


FIG. 5.10 – Illustration de la phase transitoire et des répétitions de la phase permanente.

Considérons la $k^{ieme}$ activation de la otâche $o\tau_i$ et la $l^{ieme}$ activation de la otâche $o\tau_j$ avec $i \neq j$ et une contrainte de latence entre ces deux activations. Soient $s_i^k$ et $f_i^k$ (resp. $s_j^l$ et $f_j^l$) les dates de début et de fin effective d'exécution de la $k^{ieme}$ activation de la otâche $o\tau_i$ (resp. les dates de début et de fin effective d'exécution de la $l^{ieme}$ activation de la otâche $o\tau_j$). Par hypothèse, nous avons toujours $s_i^k < f_i^k$ et $s_j^l < f_j^l$. De plus, la relation 5.8 est toujours satisfaite puisque nous sommes dans un cadre monoprocesseur. Le séquencement des activations d'une otâche nous permet d'exclure de la relation 5.8

le cas où $i = j$.

$$f_i^k \leq s_j^l \quad \text{ou} \quad f_i^k > s_j^l \tag{5.8}$$

Afin d'illustrer la relation 5.8, nous avons par exemple dans la figure 5.11 la relation $f_1^2 \leq s_3^1$ entre la deuxième activation de la otâche $o\tau_1$ et la première activation de la otâche $o\tau_3$, puis $f_1^4 > s_2^1$ entre la quatrième activation de la otâche $o\tau_1$ et la première activation de la otâche $o\tau_2$. Une contrainte de latence entre deux otâches dépend donc non seulement de la donnée produite elle-même, mais aussi de la otâche receptrice de cette donnée. Notons qu'il est nécessaire que la donnée produite ne soit pas transmise avant la fin de l'exécution de la otâche productrice. Par conséquent, pour le système $O\Gamma_n$ constitué de $n \geq 2$ otâches périodiques, si le temps effectif écoulé entre la $k^{ieme}$ activation de la otâche $o\tau_i$ et la fin de l'exécution de la $l^{ieme}$ activation de la otâche $o\tau_j$ ne doit jamais excéder une quantité $L$, alors la latence entre ces deux activations ($L_{i,j}^{k,l}$) doit satisfaire la contrainte spécifiée par la relation 5.9.

$$f_j^l - s_i^k = L_{i,j}^{k,l} \leq L \tag{5.9}$$

**Convention :** Lorsqu'une contrainte de latence est satisfaite, elle est graphiquement représentée à l'aide d'une couleur *noire*, sinon elle est représentée à l'aide d'une couleur *rouge*.

Si le premier membre de l'inégalité 5.9 est positif alors la contrainte est définie sans ambiguïté. La figure 5.11 illustre la latence entre la première activation de la otâche $o\tau_1$ et la première activation de la otâche $o\tau_2$ : $L_{1,2}^{1,1}$. Dans ce cas, le problème du respect de la contrainte de latence se restreint à trouver un choix de priorités des otâches permettant de la satisfaire. Par contre, si le premier membre de l'inégalité 5.9 est négatif alors deux situations sont possibles :

1. la otâche productrice appartient à une répétition de la phase permanente et la otâche receptrice appartient à la phase transitoire,
2. la otâche productrice et la otâche receptrice appartiennent à deux répétitions (éventuellement identiques) de la phase permanente mais $f_j^l < s_i^k$.

Dans le premier cas, la contrainte de latence n'est pas bien définie. La donnée produite ne pourra jamais être reçue par la otâche receptrice et donc sera perdue. En effet, la phase trainstoire ne se répète pas et par conséquent nous ne pourrons pas récupérer la donnée émise. Lorsque nous aurons une telle situation, la contrainte de latence sera tout simplement *ignorée*.

Dans le second cas, la situation est différente puisque grâce aux répétitions de la phase permanente, nous pouvons reporter la otâche receptrice jusqu'à la première acti-

vation appartenant à la répétition succédant celle de la otâche productrice. Afin d'illustrer cette dernière assertion, la latence entre la quatrième activation de la otâche $o\tau_1$ et la première activation de la otâche $o\tau_2$ dans la figure 5.11 correspond à la latence entre la quatrième activation de la otâche $o\tau_1$ et la première activation de la otâche $o\tau_2$ dans la seconde répétition (troisième activation de $o\tau_2$). Dans ce report de la otâche receptrice, si la otâche productrice appartient à la répétition $P_{q_1}$ et la otâche receptrice appartient initialement à la répétition $P_{q_2}$, alors la date d'activation de la nouvelle otâche receptrice est donnée par la relation 5.10.

$$r_j^{l'} = \begin{cases} r_j^l + |q_2 - q_1| \cdot H_n & \text{si } r_i^k \bmod [H_n] \leq r_j^l \bmod [H_n] \\ r_j^l + (|q_2 - q_1| + 1) \cdot H_n & \text{si } r_i^k \bmod [H_n] > r_j^l \bmod [H_n] \end{cases} \tag{5.10}$$

Dans la relation 5.10, $r_j^l$ correspond à la date de la $l^{ieme}$ activation de la otâche $o\tau_j$. Nous rappelons que "$x$ mod $y$" est un nombre entier correspondant au reste de la division euclidienne de $x$ par $y$. Par exemple, $8$ mod $5$ = $3$ puisque $3$ correspond au reste de la division euclidienne de $8$ par $5$ : $8 = 5 \cdot 1 + 3$.

**Exemple**

Soit $\Gamma_4 = \{t_1, t_2, t_3, t_4\}$ un système constitué de quatre tâches temps réel périodiques. Les caractéristiques des tâches sont résumées dans la table 5.2. Nous supposons que les tâches de ce système sont soumis à des contraintes de précédence, de périodicité stricte et de latence. Les contraintes de précédence entre les tâches sont données par : $t_1 \prec t_2$, $t_1 \prec t_4$ et $t_3 \prec t_4$, elles sont illustrées grâce au graphe de la figure 5.3. Une contrainte de périodicité stricte est imposée à la tâche $t_3$. Les contraintes de latence sont données par :

$$\begin{cases} t_1(1) \rightarrow t_3(6) < 70 \\ t_4(2) \rightarrow t_2(3) < 55 \end{cases}$$

c'est-à-dire, la latence entre la première activation de la tâche $t_1$ et la sixième activation de la tâche $t_3$ ne doit pas excéder $70$ unités de temps d'une part et la latence entre la deuxième activation de la tâche $t_4$ et la troisième activation de la tâche $t_2$ ne doit pas excéder $55$ unités de temps.

Grâce à notre algorithme optimal de choix de priorités des otâches ($Audsley^{++}$), ce système est ordonnançable suivant le choix de priorités $\mathcal{S} = <t_3, t_1, t_2, t_4>$. De plus, ce choix de priorités est le seul conduisant à un ordonnancement valide.

La figure 5.13 et la figure 5.15 résument les résultats d'ordonnançabilité obtenus. Chacune de ces figures fournit grâce à l'utilisation de l'opération $\oplus$ et de toutes les notions que nous avons introduites jusqu'ici

- le facteur d'utilisation classique du processeur,

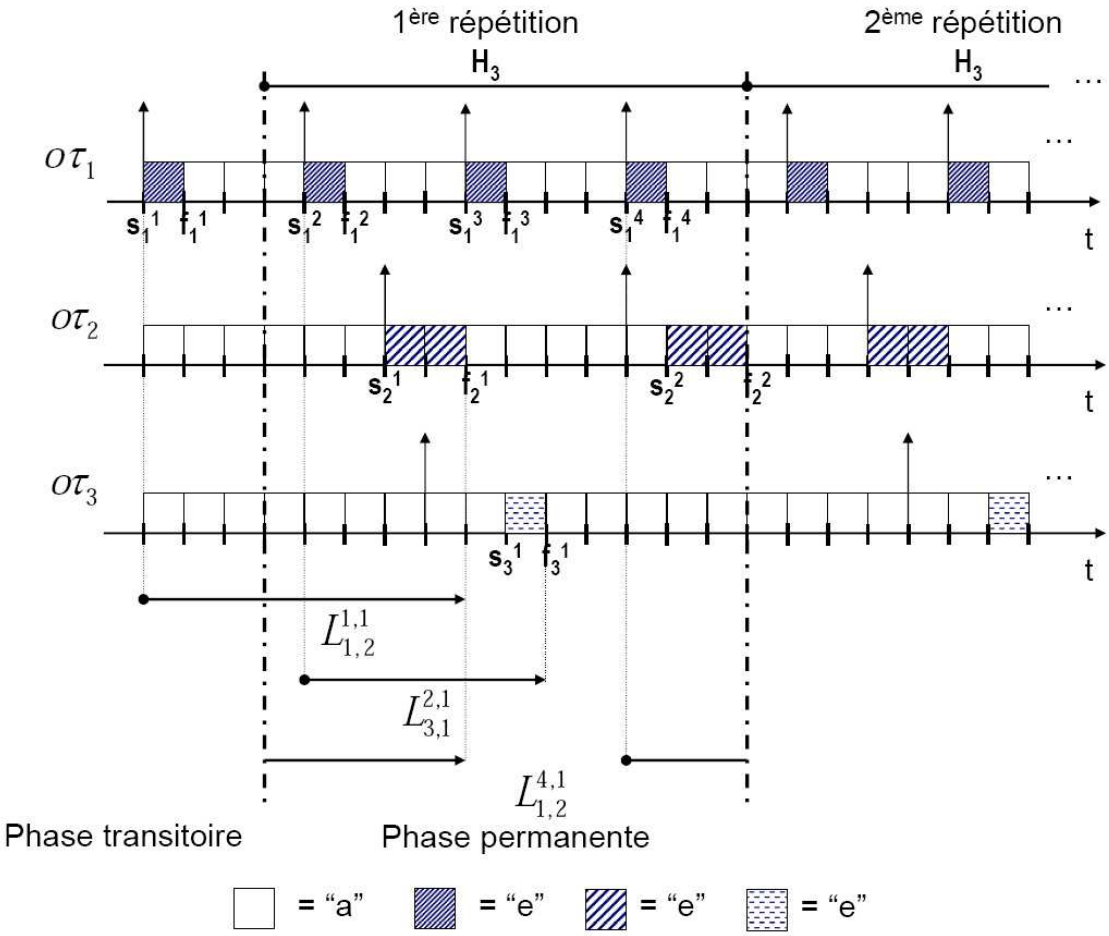


FIG. 5.11 – Illustration des contraintes de latence.

| Tâche | $r_i^1$ | $C_i$ | $D_i$ | $T_i$ | $\alpha_i$ |
|---|---|---|---|---|---|
| $t_1$ | 9 | 1 | 6 | 6 | 1 |
| $t_2$ | 13 | 2 | 9 | 12 | 1 |
| $t_3$ | 5 | 3 | 10 | 10 | 2 |
| $t_4$ | 18 | 5 | 29 | 30 | 1 |

TAB. 5.2 – Caractéristiques des tâches

- le facteur exact permanent d'utilisation du processeur,
- le coût exact permanent de la préemption,
- le pire temps de réponse de chaque tâche

- le respect des contraintes auxquelles est soumis le système.

La figure 5.13 illustre la contrainte de latence imposée entre la première activation de la tâche $t_1$ (appartenant à la phase transitoire) et la sixième activation de la tâche $t_3$ (appartenant à la première répétition de la phase permanente). La valeur de cette latence est de $L_{1,3}^{1,6} = 49$ unités de temps, qui est bien inférieure à la valeur imposée, $70$ unités de temps.

De même, la figure 5.15 illustre la contrainte de latence imposée entre la deuxième activation de la tâche $t_4$ (appartenant à la première répétition de la phase permanente) et la troisième activation de la tâche $t_2$ (appartenant aussi à la première répétition de la phase permanente). Pour cette seconde contrainte de latence, nous avons $f_2^3 = 42 < 52 = s_4^2$. Grâce aux répétitions de la phase permanente, nous pouvons reporter la tâche receptrice jusqu'à la première activation appartenant à la répétition succédant celle de la tâche productrice. Ainsi, la latence $L_{4,2}^{2,3}$ correspond à la latence entre la deuxième activation de la tâche $\tau_4$ et la huitième activation de la otâche $\tau_2$. En effet, comme l'hyperpériode vaut $H_4 = ppcm\{6, 12, 10, 30\} = 60$ unités de temps, alors grâce à la relation 5.10, la date d'activation de la nouvelle tâche receptrice est donnée par $r_2^{3'} = r_2^3 + (|1 - 1| + 1) \cdot 60 = 37 + 60 = 97 = r_2^8$ puisque nous avons la relation $r_4^2 \bmod [H_4] = 48 > 42 = r_2^3 \bmod [H_4]$. Par conséquent, suivant le choix de priorités des tâches, nous avons $L_{4,2}^{2,3} = L_{4,2}^{2,8} = 50$ unités de temps, qui est bien inférieure à la contrainte de latence imposée, $55$ unités de temps. Comme les deux activations (la deuxième activation de $t_4$ et la troisième activation de $t_2$) appartiennent à la phase permanente, nous retrouvons le même résultat en utilisant la représentation circulaire de l'ordonnancement (le *Dameid*) sans avoir à déterminer la date d'activation de la nouvelle tâche receptrice. En effet, puisque la première activation de la tâche $t_4$ a lieu à la date $r_4^1 = 18$ alors sa seconde activation a lieu à la date $r_4^2 = 18 + 30 = 48$. De même, comme la première activation de la tâche $t_2$ a lieu à la date $r_2^1 = 13$ alors sa troisième activation a lieu à la date $r_2^3 = 13 + 2 \cdot 12 = 37$. Par conséquent, la latence entre la deuxième activation de la tâche $t_4$ et la troisième activation de la tâche $t_2$ est représentée par l'arc orienté (dans le sens trigonométrique) séparant les instants $t = 52$ et $t = 42$, soit $L_{4,2}^{2,3} = 50$ unités de temps. La figure 5.12 illustre la courbe du temps de réponse en fonction du temps pour chaque tâche.

Dans la figure 5.12, nous pouvons noter que le pire temps de réponse de la tâche $t_1$ vaut $R_1 = 4$ unités de temps et est atteint pour la première fois à sa seconde activation, le pire temps de réponse de la tâche $t_2$ vaut $R_2 = 6$ unités de temps et est atteint pour la première fois à sa seconde activation, le pire temps de réponse de la tâche $t_3$ vaut $R_3 = 3$ unités de temps et est atteint pour la première fois dès sa première activation, et enfin, le pire temps de réponse de la tâche $t_4$ vaut $R_4 = 24$ unités de temps et est atteint pour la première fois à sa seconde activation. Le facteur d'utilisation classique du processeur vaut $80\%$ et le facteur exact permanent d'utilisation du processeur vaut $91.67\%$. Il en découle que le coût exact permanent de la préemption est de $\epsilon_4 = 11.67\%$.

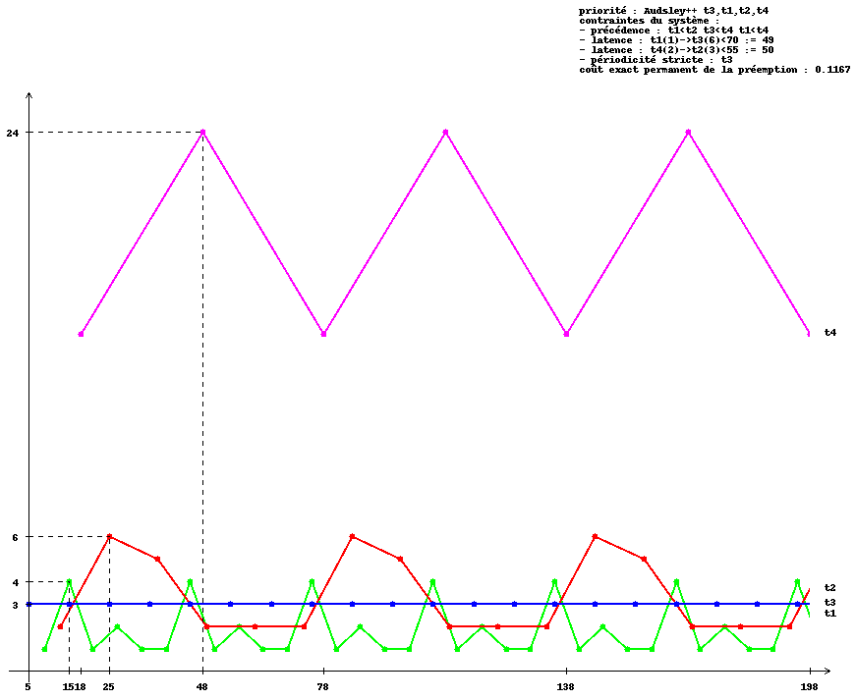


FIG. 5.12 – Courbe du temps de réponse en fonction du temps.

Maintenant que nous avons présenté les techniques permettant de prendre en compte les contraintes de latence dans un système, nous allons nous intéresser à une autre contrainte temps réel non moins importante, c'est la contrainte de gigue d'activation.

## 5.5 Contrainte de gigue

Les algorithmes classiques de choix de priorités tels que *Rate Monotonic*, *Deadline Monotonic*, *Audsley* ou encore l'algorithme de choix de priorités $Audsley^{++}$ ne permettent pas de prendre en compte le phénomène de *gigue d'activation*. La gigue d'activation correspond à l'incertitude sur la date d'occurrence de chaque activation des tâches (resp. otâches) périodiques appartenant à un système donné. Cependant, il est important de prendre en compte ce phénomène ou au moins comprendre l'impact que ce phénomène a ou pourrait avoir sur les résultats de l'analyse d'ordonnançabilité d'un système temps réel. En effet, cette gigue s'avère particulièrement néfaste lorsque les tâches (resp. otâches) ont une période qui doit être strictement respectée (par exemple pour l'acquisition de données, la commande d'un actionneur, etc.). Nous nous proposons dans cette section d'apporter quelques éléments de réponse permettant de mieux comprendre l'im-

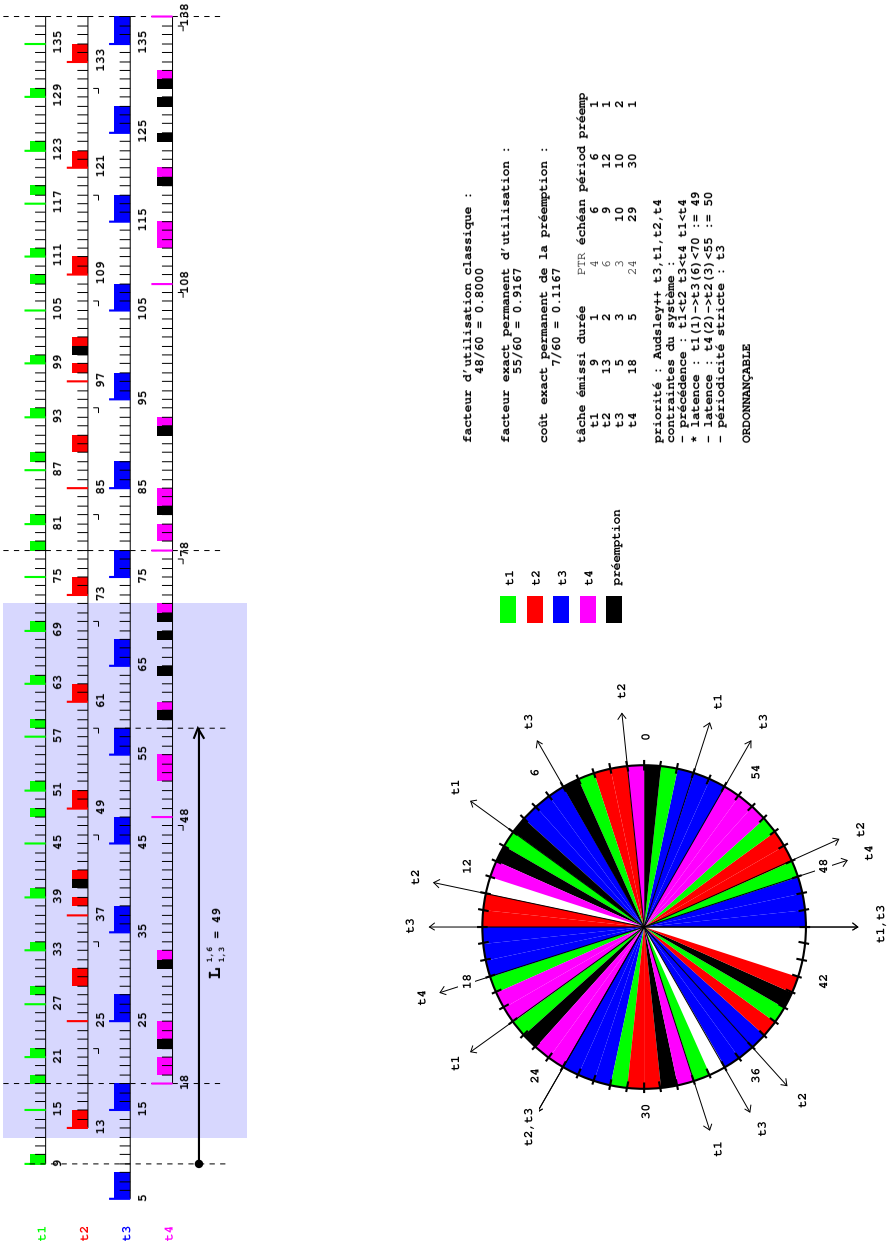


FIG. 5.13 – Résultats pour $\mathcal{S} = <t_3, t_1, t_2, t_4>$ et illustration de la latence $L^{2,3}_{4,2}$.

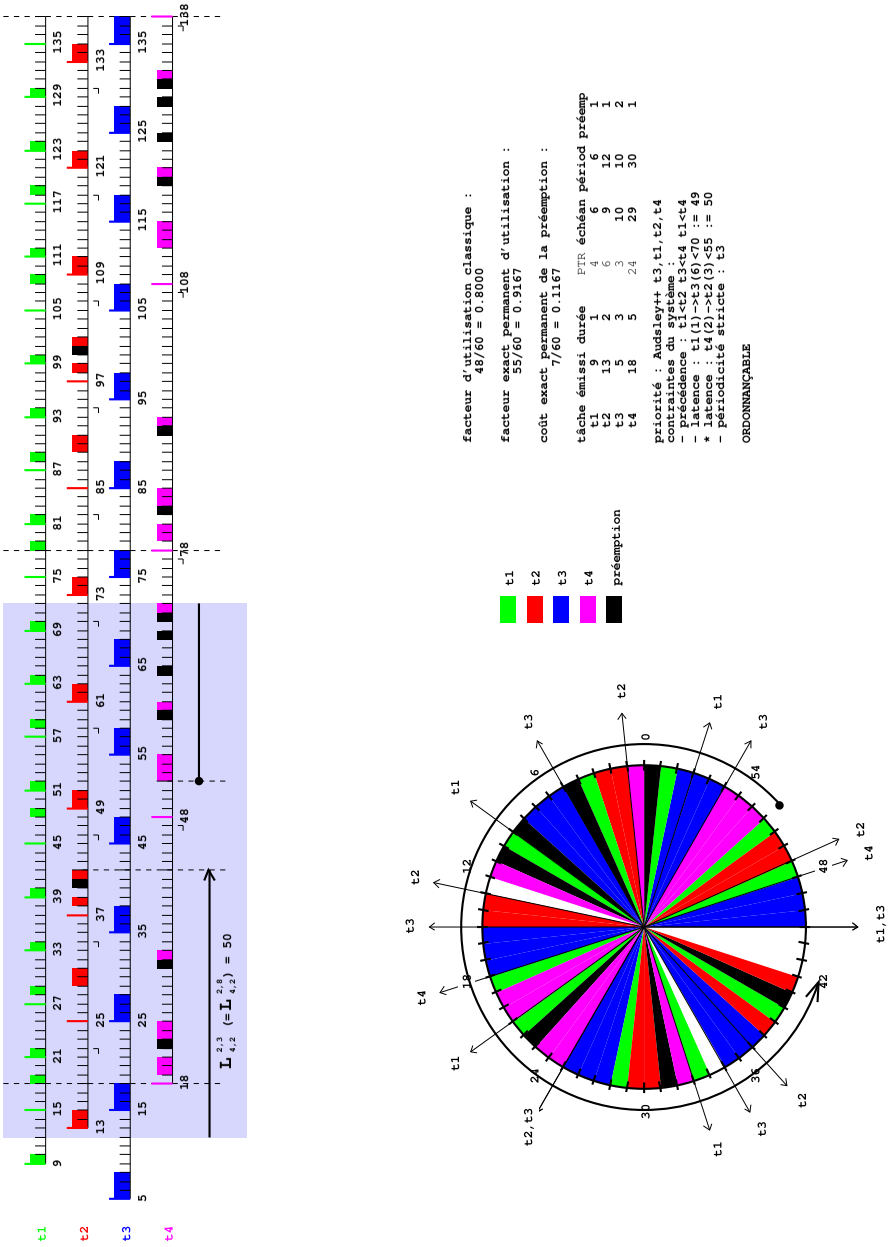


FIG. 5.14 – Résultats pour $\mathcal{S} = <t_3, t_1, t_2, t_4>$ et illustration de la latence $L_{1,3}^{1,6}$.

pact de la gigue d'activation afin de savoir comment la prendre en compte pour des tâches (resp. otâches) périodiques d'un système lorsque le coût de la préemption n'est pas négligeable.

L'analyse d'ordonnançabilité d'un système de tâches (resp. d'otâches) périodiques pour lequel aucune tâche (resp. d'otâche) n'est soumise à la contrainte de gigue d'activation et dont le coût de la préemption des tâches (resp. des otâches) n'est pas pris en compte est déjà un problème très complexe en soi, mais assez bien maîtrisé [LL73, LM80, BHR90]. Par conséquent, un modèle permettant de prendre en compte la gigue d'activation lorsque le coût de la préemption n'est pas négligeable ne peut présenter que des avantages pour les systèmes temps réel. Ce modèle intègrerait des caractéristiques importantes que nous retrouvons généralement dans un processus qui se répète périodiquement.

Dans la pratique, les dates d'occurrence des activations des tâches (resp. des otâches) périodiques ne sont pas aussi prévisibles que celles des modèles que nous avons présentés jusqu'ici. Plutôt que d'être en mesure de prévoir, au moment de la conception du système, la date exacte de chaque activation pour une tâche (resp. une otâche) périodique, le mieux que nous puissions généralement faire est de spécifier un intervalle de temps pendant lequel l'occurrence de l'activation est garantie – cette incertitude sur la date d'activation correspond à la gigue d'activation. Afin d'intégrer la gigue dans les modèles que nous avons présentés, nous proposons d'ajouter deux autres paramètres supplémentaires à la définition des tâches (resp. des otâches) périodiques. Telle qu'elle a été présentée, la gigue d'activation a un impact direct sur la période de chaque otâche périodique $o\tau_i$. Par conséquent, en plus des caractéristiques habituelles (date de première activation $r_i^1$ et période $T_i$), chaque otâche périodique $o\tau_i$ sera enrichie des paramètres :

- $\eta_i$ : la gigue inférieure,
- $\nu_i$ : la gigue supérieure.

Dans ce modèle d'otâches plus général, la $k^{ieme}$ activation de la otâche périodique $o\tau_i$ peut avoir lieu à tout moment dans l'intervalle $[r_i^1 + (k-1)T_i - \eta_i, r_i^1 + (k-1)T_i + \nu_i]$, avec $k \geq 1$. A cause de cette incertitude sur la date effective d'occurrence de chaque activation, deux modèles de gigues d'activation s'opposent pour l'interprétation des échéances de chaque otâche :

1. le modèle de *gigue non-cascadé* : dans ce modèle, l'échéance absolue de la $k^{ieme}$ activation de $o\tau_i$ a lieu à la date $r_i^1 + (k-1)T_i + D_i$ quelque soit sa date d'occurrence.

2. le modèle de *gigue cascadé* : dans ce modèle, quelque soit la date d'occurrence de la $k^{ieme}$ activation de $o\tau_i$, $k \geq 1$, l'échéance à respecter pour cette activation est exactement $D_i$ unités de temps après son occurrence.

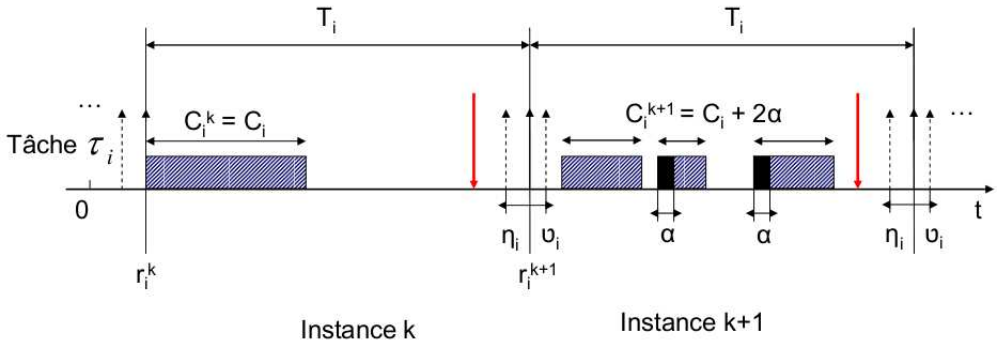


FIG. 5.15 – Illustration de la gigue d'activation.

Nous pouvons faire les deux remarques ci-dessous concernant la gigue inférieure relativement à ces deux modèles de gigue d'activation :

1. Dans le modèle de gigue non-cascadé, la $k^{ieme}$ activation de la otâche $o\tau_i$ peut avoir lieu à tout moment dans l'intervalle $[r_i^1 + (k-1)T_i - \eta_i, r_i^1 + (k-1)T_i + \nu_i]$, avec $k \geq 1$ et l'échéance absolue de la $k^{ieme}$ activation de cette otâche a lieu à la date $r_i^1 + (k-1)T_i + D_i$ quelque sa date d'occurrence. Si la $k^{ieme}$ activation a lieu avant sa date prévue (c'est-à-dire, avant la date $r_i^1 + (k-1)T_i$), alors elle peut toujours être sauvegardée et ignorée par l'ordonnanceur jusqu'à sa date d'activation prévue, d'où le paramètre correspondant à la gigue inférieure $\eta_i$ est sans intérêt dans ce modèle de gigue.
2. Dans le modèle de gigue cascadé, la $k^{ieme}$ activation de la otâche $o\tau_i$ peut avoir lieu à tout moment dans l'intervalle $[r_i^1 + (k-1)T_i - \eta_i, r_i^1 + (k-1)T_i + \nu_i]$, avec $k \geq 1$ et l'échéance absolue de la $k^{ieme}$ activation a lieu $D_i$ unités de temps après sa date d'occurrence, d'où le paramètre correspondant à la gigue inférieure $\eta_i$ est a un intérêt certain dans ce modèle de gigue.

Relativement à ces deux modèles, bien que plusieurs travaux aient été réalisés dans la littérature pour la maîtrise de la gigue d'activation des tâches temps réel périodiques, la plupart de ces travaux a porté sur l'élaboration de stratégies visant à minimiser ou "annuler" cette gigue dans le cas où le coût lié à la préemption est toujours négligé ou approximé. Par exemple, Lin et Herkert [LH96] ont étudié un modèle de tâches avec contrainte de distance dans lequel la différence entre les dates de fin de deux exécutions consécutives de la même tâche doit être limitée par une valeur précise. Cette approche tente de contourner le problème de la gigue en réduisant au minimum la gigue provoquée par l'ordonnanceur sur les dates de fin d'exécution.

Les chercheurs en réseaux ont également étudié le phénomène de la gigue d'activation. Ils ont examiné le problème de la garantie du temps de bout en bout dans les réseaux

où la transmission de données traverse plusieurs noeuds, subissant à chaque noeud une gigue d'activation. Une solution permettant de réduire au minimum la gigue de bout en bout dans de tels systèmes a été proposée par Ferrari [Fer90], puis Verma, Zhang, et Ferrari [VZF91] dans le cas où le coût de la préemption n'est pas pris en compte. De leur côté, Audsley, Burns, Richardson, Tindell et Wellings ont également proposé des techniques pour prendre en compte la gigue d'activation dans un cadre d'ordonnancement à priorités fixes [ABR$^+$93, TBW94], mais toujours dans le cas où le coût de la préemption n'est pas pris en compte. La contribution de cette section est différente des travaux ci-dessus en ceci que d'une part nous ne négligeons pas le coût lié à la préemption et d'autre part nous ne considérons pas la question de minimisation de la gigue d'activation ; mais plutôt, nous nous intéressons au problème de sa prise en compte dans le cas où nous avons peu ou pas de contrôle sur son apparition.

**Théorème 21** *Soit $O\Gamma_n = \{o\tau_1, \cdots, o\tau_n\}$ un système constitué de $n \geq 2$ otâches périodiques canoniques régulières. Soit $o\tau_j = (\tau_2)_{r_j^1}^{\infty}$ une otâche de $O\Gamma_n$ de période $T_j = |\tau_2|$. Si la otâche $o\tau_j$ n'est pas la plus prioritaire et est soumise à la contrainte de gigue avec les paramètres $\eta_j$ et $\nu_j$, si le coût d'une préemption de $o\tau_j$ vaut $\alpha_j \neq 0$ unités de temps, alors la $k^{ieme}$ activation de $o\tau_j$, $k \geq 1$, peut être ordonnançable lorsque son occurrence a lieu à la date $r_j^1 + (k-1)T_j + \nu_j$ et pas ordonnançable lorsque son occurrence a lieu à la date $r_j^1 + (k-1)T_j$ quelque soit le modèle de gigue considéré.*

**Preuve** :

La preuve du théorème 21 découle directement du fait que la otâche $o\tau_j$ n'est pas la otâche la plus prioritaire du système, et par conséquent peut subir une préemption dont le coût est non nul par hypothèse. Dans ce cas, quelque soit le modèle de gigue considéré (non-cascadé ou cascadé), nous pouvons toujours construire une situation où la $k^{ieme}$ activation, $k \geq 1$, de la otâche $o\tau_j$ subit une préemption lorsque son activation a lieu à la date $r_j^1 + (k-1)T_j$, causant par la suite un dépassement d'échéance et donc la non ordonnançabilité alors qu'elle est ordonnançable lorsque l'activation a lieu à la date $r_j^1 + (k-1)T_j + \nu_j$.

■

**Théorème 22** *Soit $O\Gamma_n = \{o\tau_1, \cdots, o\tau_n\}$ un système constitué de $n \geq 2$ otâches périodiques canoniques régulières. Soit $o\tau_j = (\tau_2)_{r_j^1}^{\infty}$ une otâche de $O\Gamma_n$ de période $T_j = |\tau_2|$. Si $o\tau_j$ n'est pas la otâche la plus prioritaire et est soumise à la contrainte de gigue avec les paramètres $\eta_j$ et $\nu_j$, si le coût d'une préemption de $o\tau_j$ vaut $\alpha_j \neq 0$ unités de temps, alors la $k^{ieme}$ activation de $o\tau_j$, $k \geq 1$, peut être ordonnançable lorsque son occurrence a lieu à la date $r_j^1 + (k-1)T_j - \eta_j$ et pas ordonnançable lorsque son occurrence a lieu à la date $r_j^1 + (k-1)T_j$ quelque soit le modèle de gigue considéré.*

**Preuve** :

La preuve du théorème 22 est similaire à celle du théorème 21. Elle découle directement du fait que la otâche $o\tau_j$ n'est pas la otâche la plus prioritaire du système, et par conséquent peut subir une préemption dont le coût est non nul par hypothèse ou être suffisamment retardée par les otâches plus prioritaires qu'elle dépasse son échéance. Dans ce cas, quelque soit le modèle de gigue considéré (non-cascadé ou cascadé), nous pouvons toujours construire une situation où la $k^{ieme}$ activation, $k \geq 1$, de la otâche $o\tau_j$ ne subit pas de préemption lorsque son activation a lieu à la date $r_j^1 + (k-1)T_j - \eta_j$, la rendant ordonnançable alors qu'elle est non ordonnançable lorsque l'activation a lieu à la date $r_j^1 + (k-1)T_j$.

■

Le théorème 21 et le théorème 22 ont une conséquence majeure sur l'analyse d'ordonnançabilité de tout système qui est soumis à la contrainte de gigue d'activation lorsque le coût de la préemption est pris en compte. Cette conséquence est formellement citée dans le corollaire ci-dessous.

**Corollaire 2** *Soit $O\Gamma_n = \{o\tau_1, \cdots, o\tau_n\}$ un système constitué de $n \geq 2$ otâches périodiques canoniques régulières dont certaines sont soumises à la contrainte de gigue. Lorsque le coût de la préemption est pris en compte et est non nul pour au moins une otâche n'ayant pas la priorité la plus élevée du système, alors l'analyse d'ordonnançabilité de $O\Gamma_n$ ne peut pas être réduit à celle d'un autre système sans contrainte de gigue sur les otâches.*

## 5.6 Conclusion

Dans ce chapitre, nous avons principalement présenté des techniques permettant de prendre en compte le coût exact de la préemption pour un système soumis à des contraintes temps réels multiples telles que la précédence, la périodicité stricte, la latence ou encore la gigue d'activation. Nous avons commencé par introduire chacune des contraintes ci-dessus citées individuellement en étendant la définition de chacune d'elles des tâches périodiques aux otâches périodiques afin de souligner leurs impacts sur les conditions d'ordonnançabilité. Ensuite, nous avons considéré les contraintes conjointement. Nous avons fourni les conditions d'ordonnançabilité pour un système dont les priorités fixes des otâches avaient été attribuées suivant une politique de choix de priorités particulier (par exemple *Rate Monotonic*, *Deadline Monotonic*, etc.). Puis, pour un système dont les priorités des otâches n'avaient pas été attribuées, nous avons appliqué l'algorithme optimal $Audsley^{++}$ pour fournir un ordonnancement valide du système si un tel ordonnancement existe. Enfin, nous avons souligné une fois de plus l'impact du coût de la préemption sur l'analyse d'ordonnançabilité notamment en présence de la

contrainte de gigue d'activation des otâches. Dans ce cas, la conclusion à laquelle nous sommes parvenue est que l'analyse d'ordonnançabilité d'un système ne peut pas être réduite à celle d'un autre système sans contrainte de gigue sur les otâches.

# Chapitre 6

# Logiciel d'analyse d'ordonnançabilité avec coût exact de la préemption

*Quand les mystères sont très malins,*
*ils se cachent dans la lumière.*
**Jean Giono**

## 6.1 Introduction

Il nous a paru indispensable de développer un logiciel pour mettre à la disposition d'utilisateurs les résultats que nous avons obtenus. Ce logiciel permet de montrer, grâce à des exemples simples et concrets, la faisabilité de notre étude. Il a pour nom *SAS* pour *Simulation Analysis of Scheduling* et est *multilingue*. Actuellement, il est disponible en trois langues (*Français*, *Anglais* et *Allemand*). Ce logiciel a été conçu et développé dans l'unité de recherche *INRIA* Paris-Rocquencourt en France. Le but de ce chapitre est la présentation du logiciel et des fonctionnalités que nous y avons implémentées. *SAS* s'inscrit dans le cadre du développement logiciel de l'équipe-projet *AOSTE* de l'*INRIA* Paris-Rocquencourt. Il présente principalement les résultats théoriques de cette thèse. L'activité majeure de l'équipe-projet *AOSTE* consiste à proposer des modèles et méthodes pour l'analyse et l'optimisation des systèmes temps réel embarqués [LSSS91, LS92, KS98, GLS99, BBB$^+$01, DNSS01, PGBS03, Sor04, Sor05]. Le fruit des multiples recherches menées dans cette équipe-projet est basé sur la méthodologie "*Adéquation Algorithme-Architecture*" (*AAA*) [LSSS91, GS03, Sor94, DLAS98, GS02, Rau06]. Cette méthodologie est utilisée pour le prototypage rapide et l'optimisation de la mise en oeuvre des systèmes temps réel distribués soumis à des contraintes telles que la précédence, la périodicité stricte et la latence sur des architectures "hétérogènes". Cependant, elle s'appuie principalement sur des algorithmes d'ordonnancement *non-preemptifs*. Le logiciel incarné ici par *SAS* permet la préemption et maîtrise son coût exact et donc le coût lié au système d'exploitation dans les conditions d'ordonnançabilité de systèmes temps réel durs soumis à des contraintes multiples telles que la précédence, la périodicité stricte et la latence. Il fournit les résultats de l'analyse d'ordonnançabilité, l'attribution

optimale des priorités, l'ordonnancement linéaire et l'ordonnancement circulaire correspondant à la phase permanente d'un système de tâches (resp. d'otâches) périodiques donnée.

## 6.2 Saisie d'un système d'otâches périodiques à analyser

Au lancement du logiciel *SAS*, l'interface graphique dont une copie d'écran est illustrée par la figure 6.1 apparaît

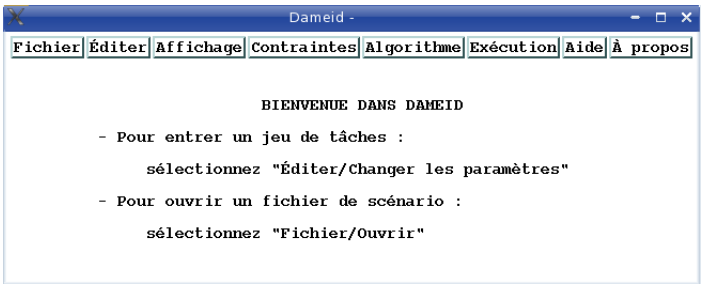


FIG. 6.1 – Illustration de l'interface de démarrage du logiciel SAS.

Grâce à cette l'interface graphique, le logiciel *SAS* permet à l'utilisateur soit de saisir les paramètres de chaque otâche d'un système à analyser, soit d'utiliser les paramètres d'un système d'otâches dans un fichier, nommé *nom_fichier.dm*, précédemment sauvegardé. À l'aide du menu *Editer/Changer les paramètres*, une autre interface graphique dont une copie d'écran est illustrée par la figure 6.2 apparaît

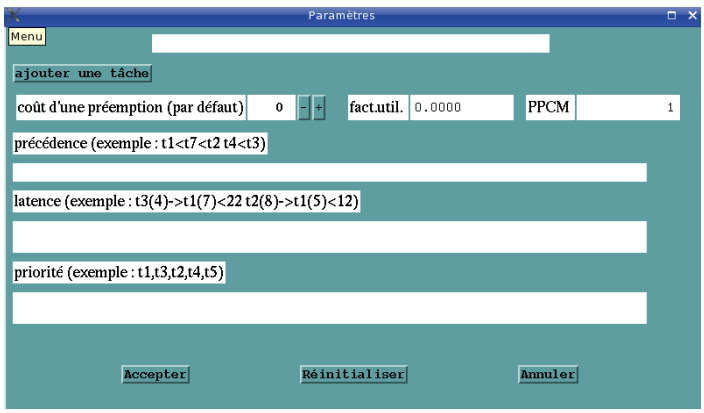


FIG. 6.2 – Illustration de l'action *Editer/Changer les paramètres*.

En utilisant simplement le bouton "ajouter une tâche" de l'interface de la figure 6.2, l'utilisateur peut rentrer les paramètres d'une nouvelle tâche (resp. une nouvelle otâche), la figure 6.3 illustre l'exemple de la saisie d'un système constitué de $4$ otâches. Nous faisons toujours l'hypothèse que la partie initiale de toutes les otâches vaut $\Lambda$.

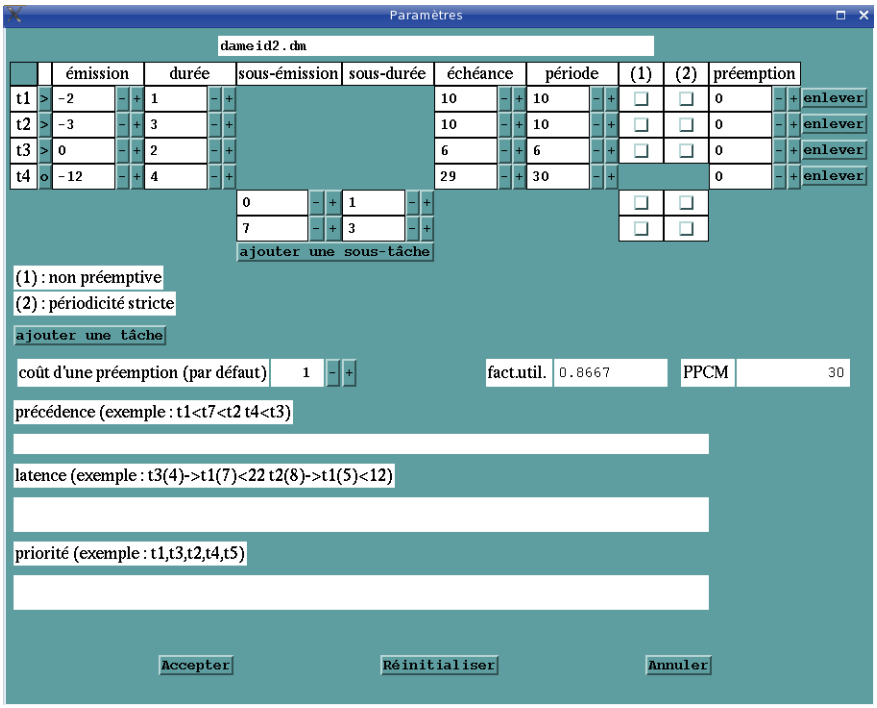


FIG. 6.3 – Illustration de l'interface graphique de SAS.

Grâce à cette interface, l'utilisateur peut spécifier la date de première activation de chaque otâche (*émission*), sa durée d'exécution (*durée*) à chaque activation sans aucune approximation du coût de la préemption, son échéance relative (*échéance*), sa période (*période*) et le coût exact d'une préemption. Si le coût d'une préemption n'est pas spécifiée (*valeur "0" pour la otâche dans la colonne préemption*) alors c'est le coût par défaut d'une préemption (*coût d'une préemption (par défaut)*) qui est considéré. Au fur et à mesure qu'on rajoute de nouvelles otâches grâce au bouton "ajouter une tâche", le *plus petit commun multiple (ppcm)* des périodes de toutes les otâches du système est calculé. A titre d'exemple, le *ppcm* des périodes des otâches du système saisi vaut $30$, les paramètres de la otâche $t_1$ sont : sa date de première activation vaut "$-2$", sa durée d'exécution à chaque activation sans aucune approximation du coût de la préemption vaut "$3$" unités de temps, son échéance relative vaut "$10$" et sa période vaut "$10$". Lorsque la otâche n'est pas canonique régulière comme c'est par exemple le cas pour la otâche $t_4$ de la figure 6.3, les dates de sous-activation (*sous-activation*) et les sous-durées d'exécution (*sous-durée*)

sont précisées. Il est possible de rajouter différents types de contraintes en respectant la syntaxe prévue à cet effet dans l'interface : *précédence*, *périodicité stricte*, *latence*, *non-préemptivité par niveau de priorité* ou même imposer les priorités des otâches en donnant un ordre total des otâches séparées par des virgures dans l'espace reservé à cet effet dans l'interface de la figure 6.3. Par exemple, l'ordre $t_1$, $t_3$, $t_2, \cdots$ dans le champ *priorité* de la figure 6.3 signifie que la otâche $t_1$ est plus prioritaire que la otâche $t_3$, la otâche $t_3$ est plus prioritaire que la otâche $t_2$ et ainsi de suite.

Une fois les paramètres du système saisis, on clique sur le bouton *Accepter* et on obtient l'interface graphique dont une copie d'écran est illustrée par la figure 6.4 qui prépare l'analyse d'ordonnançabilité et l'ordonnancement du système. Cette interface contient 3 parties : *la représentation linéaire* de l'ordonnancement / *GantChart*, *la représentation circulaire* de l'ordonnancement / le *Dameid* et enfin une *zone texte* illustrant les informations sur le système d'otâches saisi. C'est dans cette dernière partie que le résultat d'ordonnançabilité du système sera fourni.

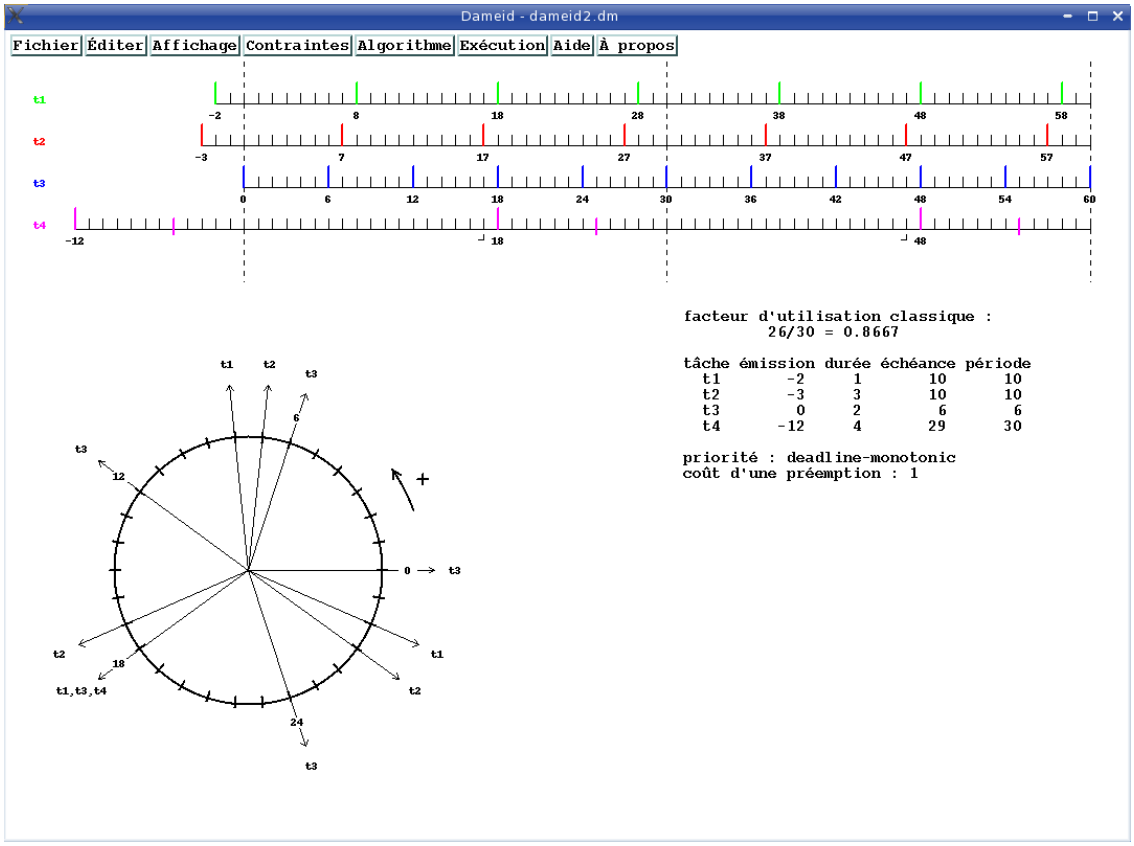


FIG. 6.4 – Illustration de l'interface graphique après validation des paramètres.

Dans cette interface graphique, chaque otâche est représentée par une couleur différente. Pour l'exemple de la figure 6.4, l'exécution de la otâche $t_1$ sera représentée grâce à la couleur *verte*, celle de la otâche $t_2$ grâce à la couleur *rouge*, celle de la otâche $t_3$ grâce à la couleur *bleue* et enfin celle de la otâche $t_4$ grâce à la couleur *rose*. Dans les différents menus, l'utilisateur peut spécifier l'algorithme d'ordonnancement à utiliser dans le menu *Algorithme* (*Rate Monotonic*, *Deadline Monotonic*, $Audsley^{++}$ ou attribuer les priorités à sa convenance : *utilisateur*).

Une fois que les contraintes et l'algorithme de choix des priorités sont spécifiés, il est possible de procéder à l'analyse d'ordonnançabilité et à l'ordonnancement du système considéré soit étape par étape, c'est à dire niveau de priorité par niveau de priorité des otâches, soit pour l'ensemble des otâches, en utilisant le menu *Exécution*). Selon les contraintes qui ont été spécifiées, il est possible de revenir en arrière, de modifier les paramètres des otâches grâce au menu *Editer*, puis reprendre l'analyse à nouveau, et ainsi de suite. Une représentation circulaire de l'ordonnancement, un disque nommé "*dameid*", montre l'occupation du processeur dans la phase permanente lorsque le système est ordonnançable et une fois de plus chaque otâche est représentée par une couleur différente. Un avantage du *dameid* sur la représentation linéaire (*GantChart*) de l'ordonnancement est le fait que la position et la longueur des unités disponibles (*idle slots*) à chaque étape de l'ordonnancement du système.

Pour les besoins éventuelles d'impression, il est possible de représenter l'exécution de chaque otâche grâce à la couleur *grise* dans la représentation linéaire et grâce à un motif différent en *noir et blanc* dans le *dameid*. Pour les systèmes pour lesquels le plus petit commun multiple (*ppcm*) des périodes des otâches est un grand nombre (c'est-à-dire un nombre tel qu'il n'est pas possible de visualiser clairement chaque unité de temps) pour la fenêtre d'analyse, il est possible de *zoomer* des parties de la représentation linéaire de l'ordonnancement et de la représentation circulaire de l'ordonnancement en faisant les deux étapes suivantes :

1. déplacer le pointeur de la souris sur la zone à zoomer,
2. taper la touche '$z$' sur votre clavier.

## 6.3 Analyse d'ordonnançabilité et ordonnancement

Dans notre méthode d'analyse, nous fournissons des résultats d'ordonnançabilité plus sûr que ceux proposés par tous les logiciels académiques ou commerciaux pour les systèmes temps réel durs du même type que *SAS* parce que nos résultats sont fondés sur la prise en compte du coût exact dû à la préemption et donc du système d'exploitation, ce qui n'est pas le cas des logiciels tels que *Cheddar*, *MAST*, *SymtA/S Overview*, etc. Une fois les paramètres du système d'otâches saisis, de façon textuelle à l'aide d'un

fichier nommé *nom_fichier.dm* ou à l'aide de l'interface graphique et une fois l'algorithme de choix des priorités spécifié, l'utilisateur peut exécuter *SAS* à l'aide du menu *Exécution* de la figure 6.4. Le résultat de l'analyse d'ordonnançabilité est alors affiché, ainsi que :

1. le facteur d'utilisation classique du processeur,
2. le facteur exact permanent d'utilisation du processeur,
3. le coût exact permanent dû à la préemption,
4. le pire temps de réponse (*PTR*) de chaque otâche.

Une copie d'écran des résultats produits par le logiciel *SAS* est illustrée par la figure 6.5.

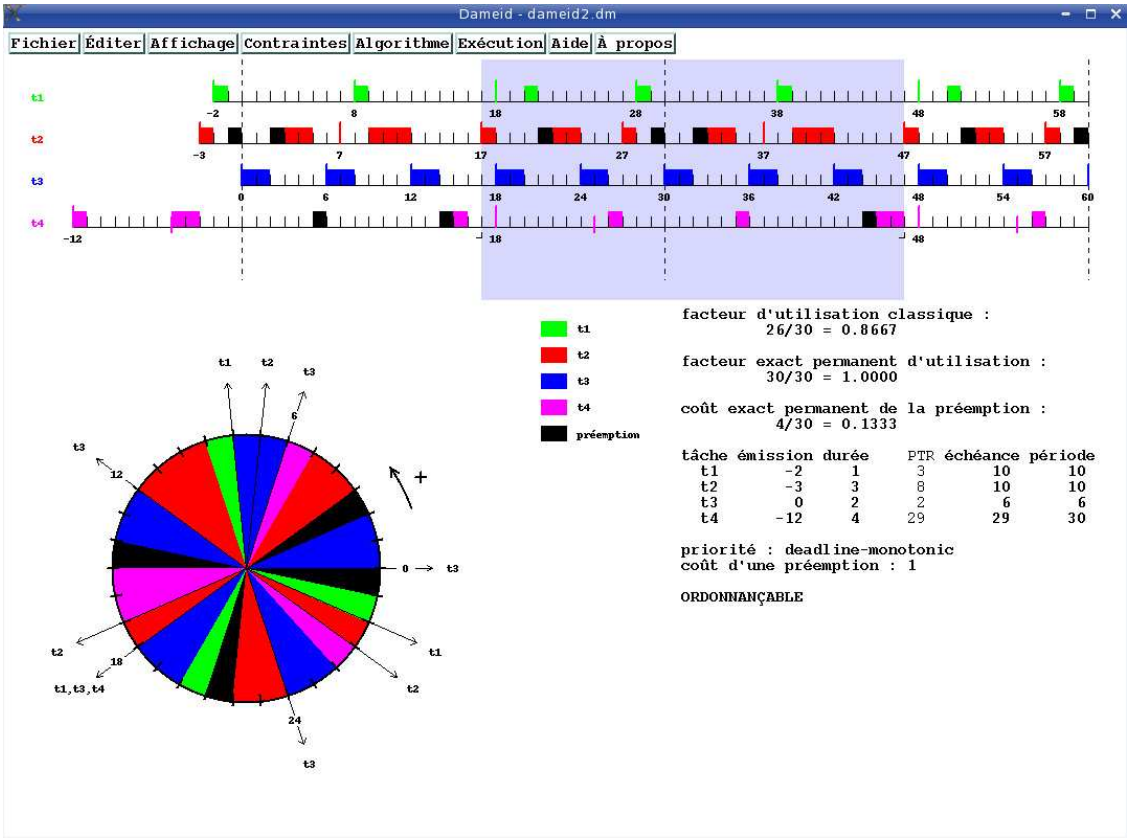


FIG. 6.5 – Illustration des résultats produits par *SAS*.

Lorsque le système est ordonnançable suivant l'algorithme de choix de priorités spécifié dans la figure 6.5, la *phase permanente* correspond à la zone en "*bleue*" de la représentation linéaire de l'ordonnancement et la *phase transitoire* correspond à l'intervalle

précédent cette zone. Une colonne supplémentaire est créée automatiquement dans la zone "texte" de la figure 6.5 donnant le Pire Temps de Réponse (*PTR*) de chaque otâche. La otâche finale de l'ordonnancement par application successive de l'opération $\oplus$ est également obtenu grâce au menu *Affichage/Mesoid*, une copie d'écran est illustrée par la figure 6.6.

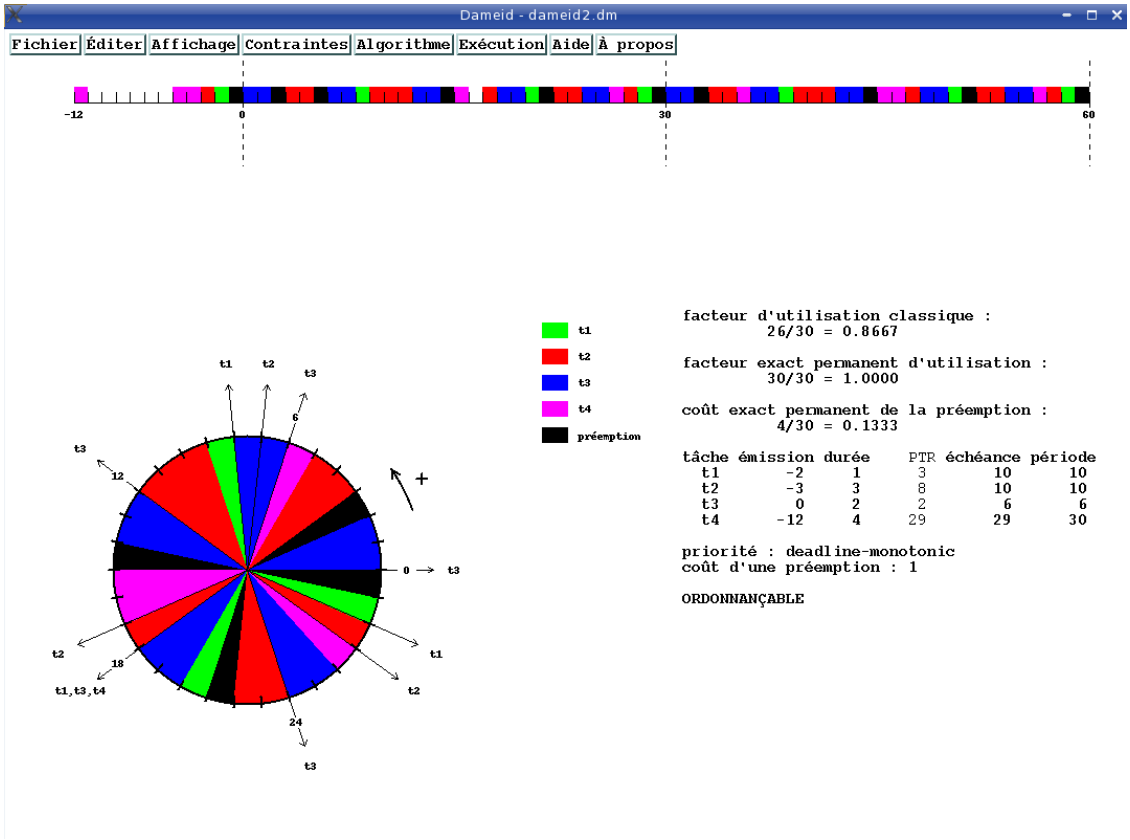


FIG. 6.6 – Illustration de la otâche finale de l'ordonnancement.

Une fenêtre supplémentaire fournissant la *courbe du temps de réponse en fonction du temps* de chaque otâche est obtenue grâce au menu *Exécution* de la figure 6.5. Une copie d'écran de cette fenêtre pour notre exemple est illustrée par la figure 6.7. La courbe du temps de réponse en fonction du temps de chaque otâche est représentée à l'aide de la même couleur que celle de son ordonnancement. Les points reliés sur chaque courbe par les segments de droite représentent ses dates d'activations. Suivant l'algorithme *DM*, nous pouvons par exemple noter que le pire temps de réponse (*PTR*) de la otâche $t_1$ vaut $3$ unités de temps et celui-ci est atteint pour la première fois à sa troisième activation, celui de la otâche $t_2$ vaut $8$ unités de temps et est atteint pour la première fois à sa première

activation, celui de la otâche $t_3$ vaut 2 unités de temps et cette valeur est invariante quelque soit son activation, enfin celui de la otâche $t_4$ vaut 29 unités de temps et est atteint pour la première fois à sa deuxième activation.

Toutes les figures illustrées jusqu'ici peuvent être converties au format *PostScript* à l'aide du menu *Fichier* de la figure 6.5 pour l'impression.

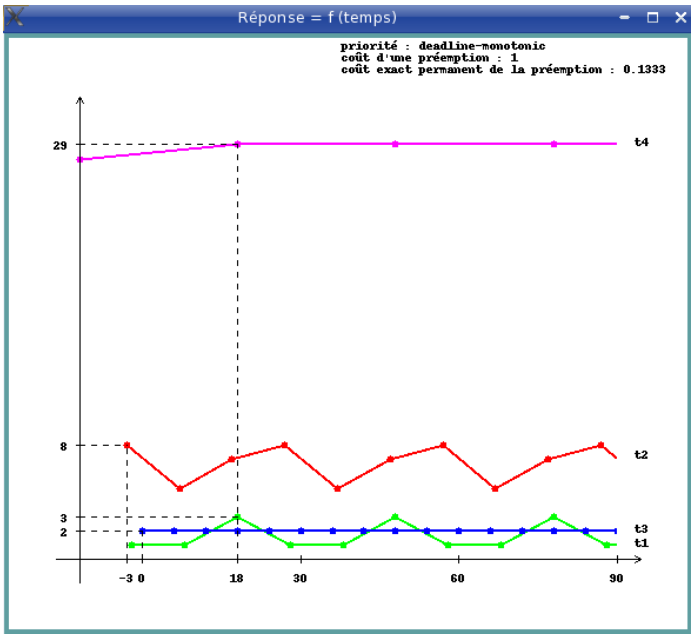


FIG. 6.7 – Courbe du temps de réponse en fonction du temps de chaque otâche.

## 6.4 Résultats obtenus pour le choix optimal des priorités

Le choix des priorités attribuées aux otâches a un grand impact sur le résultat d'ordonnançabilité du système considéré lorsqu'on prend en compte le coût de la préemption. Dès les premiers chapitres de cette thèse, nous avons montré que le choix de priorités suivant *Rate-Monotonic* et *Deadline-Monotonic* n'est pas *optimal*. De même, nous avons montré que le choix de priorités suivant l'algorithme d'*Audsley* n'est plus applicable et donc plus optimal lorsqu'on prend en compte le coût de la préemption. Ici, un algorithme de choix de priorités est dit *optimal* lorsque s'il existe un algorithme de choix de priorités qui conduit à un ordonnancement valide alors le choix de priorités proposé par cet algorithme conduit lui-aussi à un ordonnancement valide.

Dans le logiciel *SAS*, l'utilisateur peut spécifier à sa guise un algorithme de choix de priorités parmi les algorithmes d'ordonnancement à priorités fixes ci-dessous grâce au menu *Algorithme* de la figure 6.5. Il a le choix entre :

1. *Deadline-Monotonic* : les priorités sont attribuées aux otâches relativement à leurs échéances relatives, plus l'échéance relative d'une otâche est petite plus sa priorité est élévée. Lorsque deux otâches ont la même échéance relative, la plus prioritaire est choisie arbitrairement.
2. *Rate-Monotonic* : les priorités sont attribuées aux otâches relativement à leurs périodes, plus la période d'une otâche est petite plus sa priorité est élevée. Lorsque deux otâches ont la même période, la plus prioritaire est choisie arbitrairement.
3. *Priorités utilisateur* : les priorités sont attribuées aux otâches relativement à un choix prédéfini par l'utilisateur, les otâches sont rangées de la plus prioritaire à la moins prioritaire.
4. $Audsley^{++}$ : les priorités sont attribuées aux otâches relativement à la procédure décrire dans sous-section 4.5.2 du chapitre 4, ce choix fourni toutes les solutions d'ordonnancements possibles.

Suivant les trois premières politiques de choix des priorités des otâches, une seule solution est fournie par le logiciel *SAS* puisque les priorités sont fixées à priori. Suivant la politique de choix des priorités $Audsley^{++}$, en plus du fait qu'elle est *optimale* au sens que s'il existe un choix de priorités conduisant à un ordonnancement valide alors le choix de priorités proposé par cet algorithme conduira lui-aussi à un ordonnancement valide, tous les choix de priorités conduisant à un ordonnancement valide sont fournies une par une relativement au nombre total de choix de priorités possibles : $n!$ choix sont possibles pour un système constitué de $n$ otâches. Les résultats d'ordonnançabilité sont fournis pour chaque choix de priorités conduisant à un ordonnancement valide ainsi que la courbe du temps de réponse en fonction du temps de chaque otâche. Une fenêtre supplémentaire fournie le rang de chaque choix de priorités conduisant à un ordonnancement valide trouvé ainsi que le nombre total de solutions. La figure 6.8 fournie une copie d'écran de cette fenêtre pour notre exemple constitué de quatre otâches. Nous sommes à la solutions numéro $5$ sur $4! = 24$ choix de priorités possibles. Le curseur permet de marquer le pourcentage de choix de priorités déjà parcouru. Ainsi nous pouvons avoir une idée du pourcentage de choix de priorités restant à parcourir. L'intérêt du curseur est qu'il peut permettre d'arrêter la recherche de nouvelles solutions si nous avons déjà obtenu une solution satisfaisante suivant un critère fixé.

Lorsque l'algorithme de choix des priorités $Audsley^{++}$ est choisi pour notre exemple, les résultats d'ordonnançabilité (*coût exact permanent de la préemption*, *PTR* de chaque otâche, etc.) pour chaque choix de priorités conduisant à un ordonnancement valide sont fournis par la figure 6.9, la figure 6.10, la figure 6.11, la figure 6.12 et la figure 6.13. Ces figures illustrent aussi les fichiers enrégistrés au format *PostScript* obtenus. Le coût exact

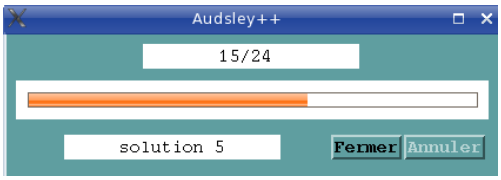


FIG. 6.8 – Illustration du rang et du nombre total de solutions trouvées.

permanent de la préemption pour ce système d'otâches varie entre $3.33\%$ pour le choix de priorités <$t_3, t_2, t_1, t_4$> et $13.33\%$ pour le choix de priorités <$t_1, t_3, t_2, t_4$> et pour le choix de priorités <$t_3, t_1, t_2, t_4$>. Ce coût vaut $6.67\%$ pour les deux autres choix de priorités. Nous pouvons noter que le système n'est ordonnançable suivant aucun algorithme de choix de priorités pour lequel $t_4$ a la priorité la plus élevée.

La figure 6.14, la figure 6.15, la figure 6.16, la figure 6.17 et la figure 6.18 illustrent la courbe du temps de réponse de chaque otâche en fonction du temps suivant le choix des priorités spécifiées dans le bord supérieur droit de la fenêtre. La figure 6.18 illustre la courbe conduisant au coût exact permanent de la préemption le plus bas, soit $3.33\%$. Dans chaque fenêtre, il est également indiqué le coût exact permanent de la préemption et les contraintes du système le cas échéant. La synthaxe utilisée pour prendre en compte chaque contrainte (précédence, périodicité stricte, latence, non-préemptivité par niveau de priorité, etc.) est celle précisée dans l'interface graphique illustrée par la figure 6.3. Chaque contrainte est à remplir dans le champs reservé à cet effet.

## 6.5 Developpement du logiciel SAS

Le logiciel *SAS* a été écrit en *OCaml* (Objective Caml), en utilisant le préprocesseur syntaxique *Camlp5* et la toolkit graphique *Olibrt* écrit en *OCaml* sous X-Window, par *Daniel de Rauglaudre*[1].

*Caml* est un langage de programmation généraliste, conçu pour garantir la sûreté et la fiabilité des programmes. Il est très expressif et néanmoins facile d'apprentissage et d'emploi. le language *Caml* se prête à la programmation dans un style fonctionnel, impératif ou orienté objets. Il est développé et distribué par l'*INRIA* depuis 1985. Le système *Objective Caml* est la principale implémentation du langage *Caml*. Il offre un puissant système de modules ainsi qu'une couche orientée objets. Il est livré avec un compilateur produisant du code natif pour de nombreuses architectures, pour une haute performance ; un compilateur produisant du code-octets (« bytecode »), pour une portabilité accrue ;

[1]Daniel de Rauglaudre, né le 19 juillet 1955 à Pélissanne, Bouches-du-Rhône, est ingénieur de recherche à l'INRIA.

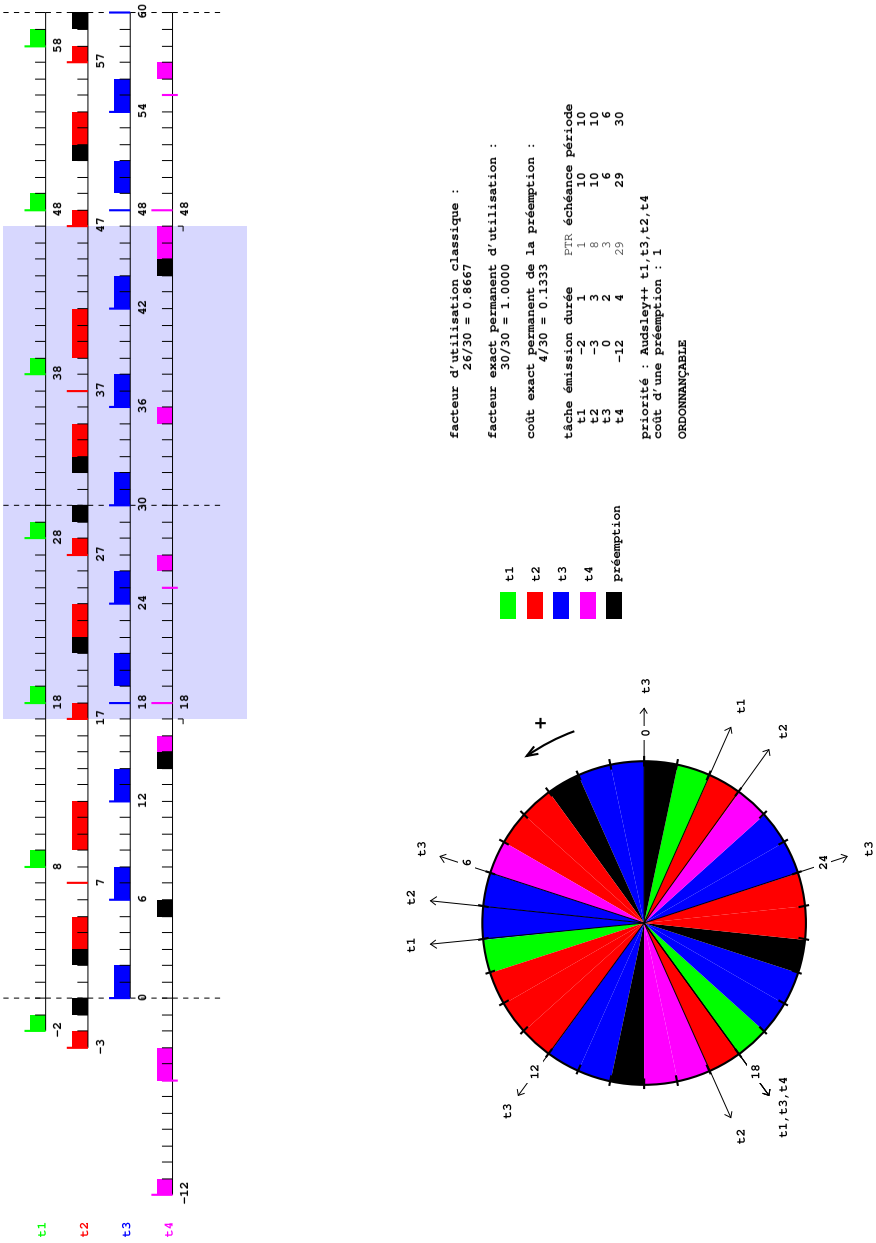


FIG. 6.9 – Solution 1 : choix des priorités <$t_1, t_3, t_2, t_4$>.

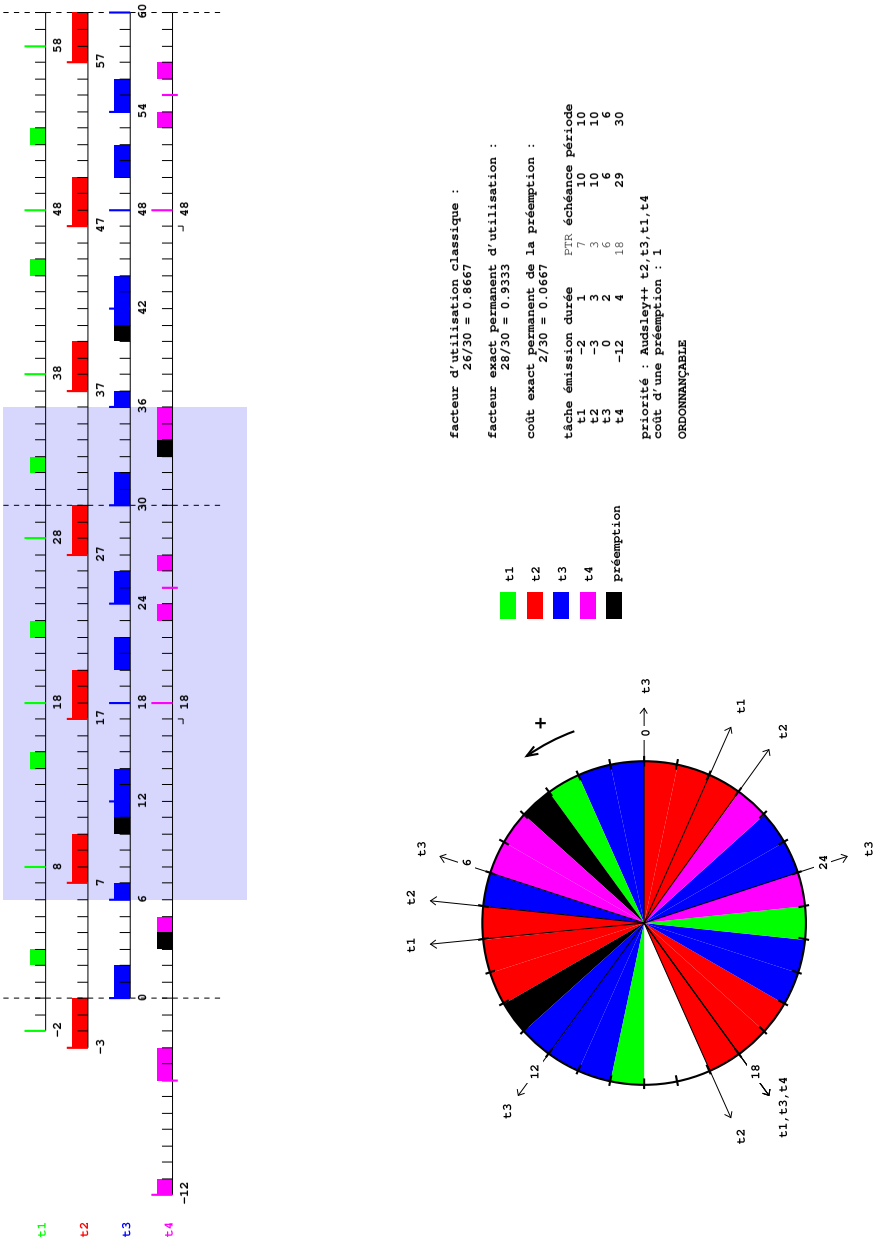


FIG. 6.10 – Solution 2 : choix des priorités <$t_2, t_3, t_1, t_4$>.

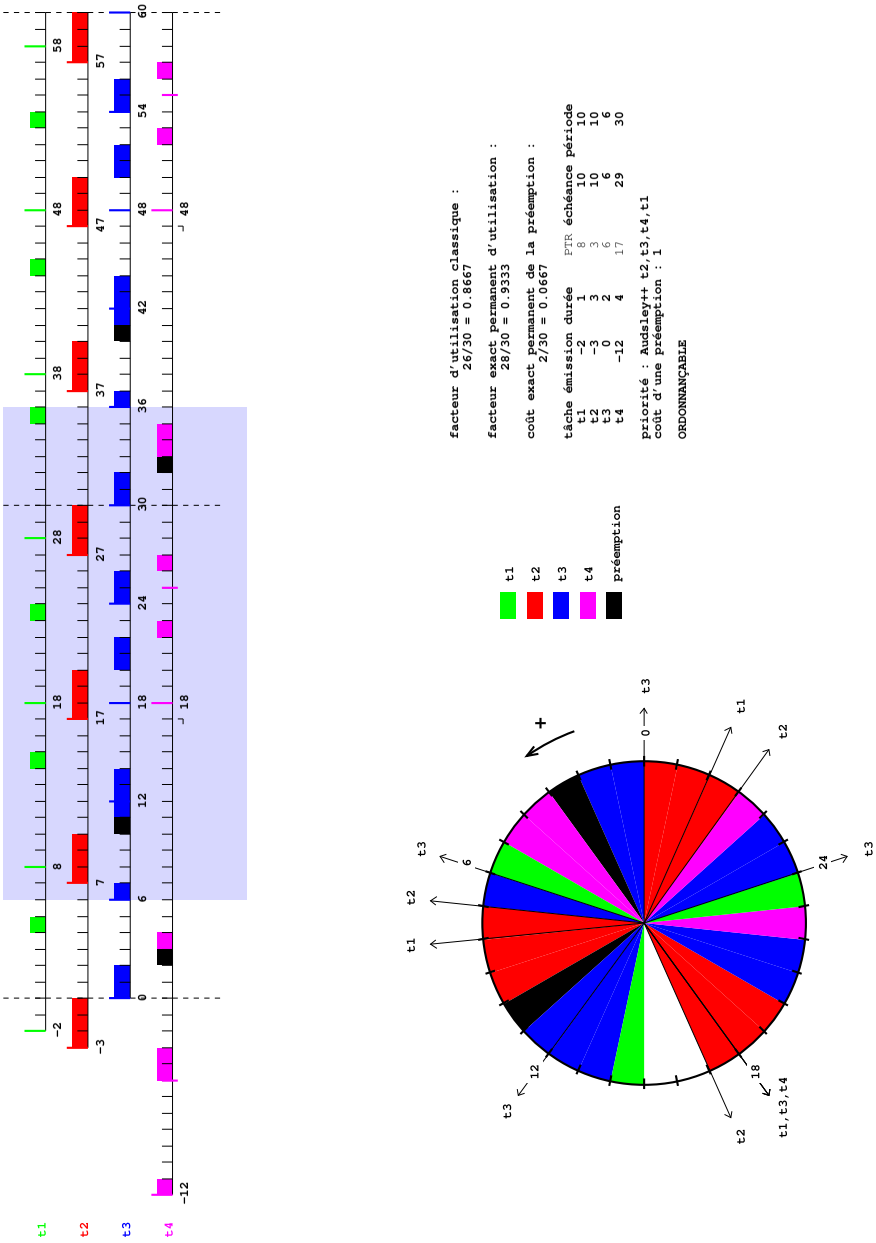


FIG. 6.11 – Solution 3 : choix des priorités <$t_1, t_3, t_4, t_1$>.

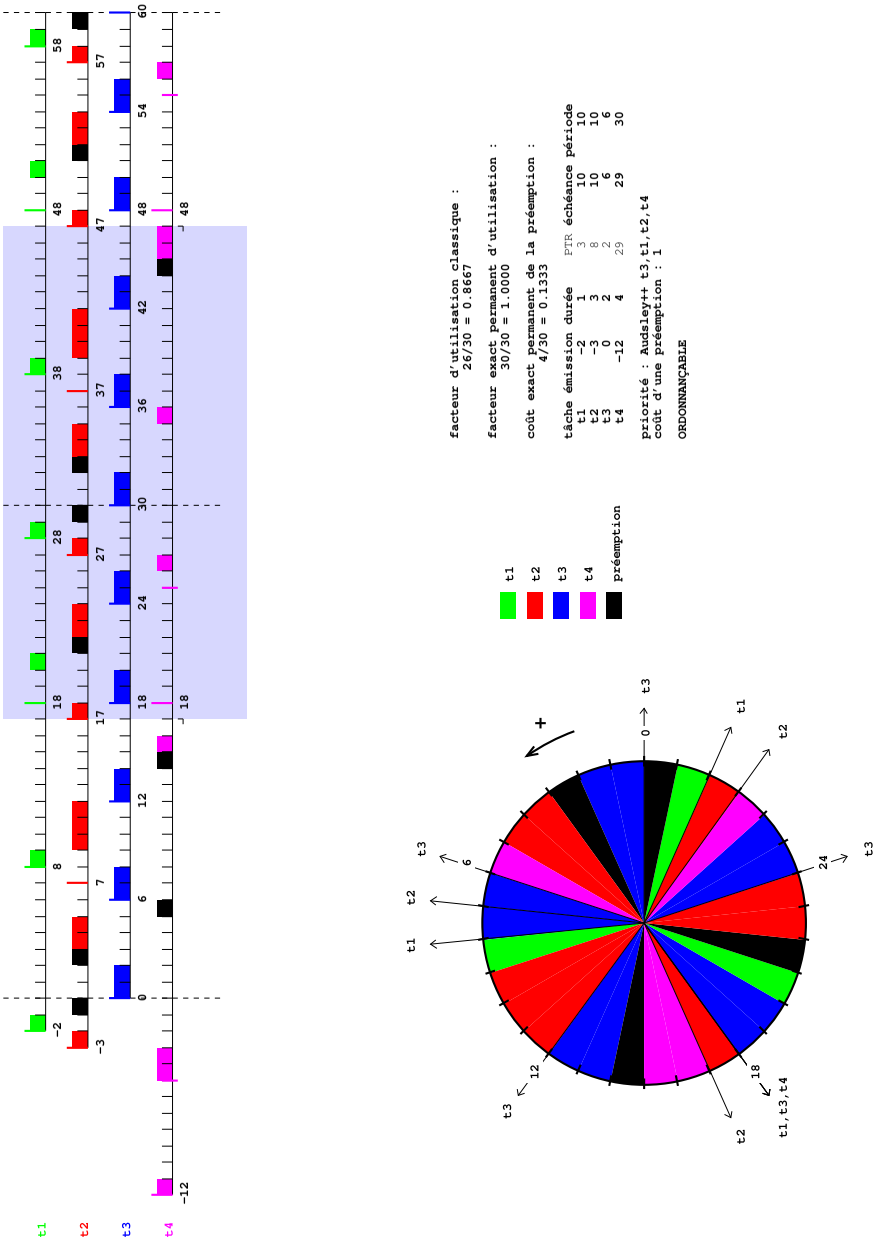


FIG. 6.12 – Solution 4 : choix des priorités <$t_3, t_1, t_2, t_4$>.

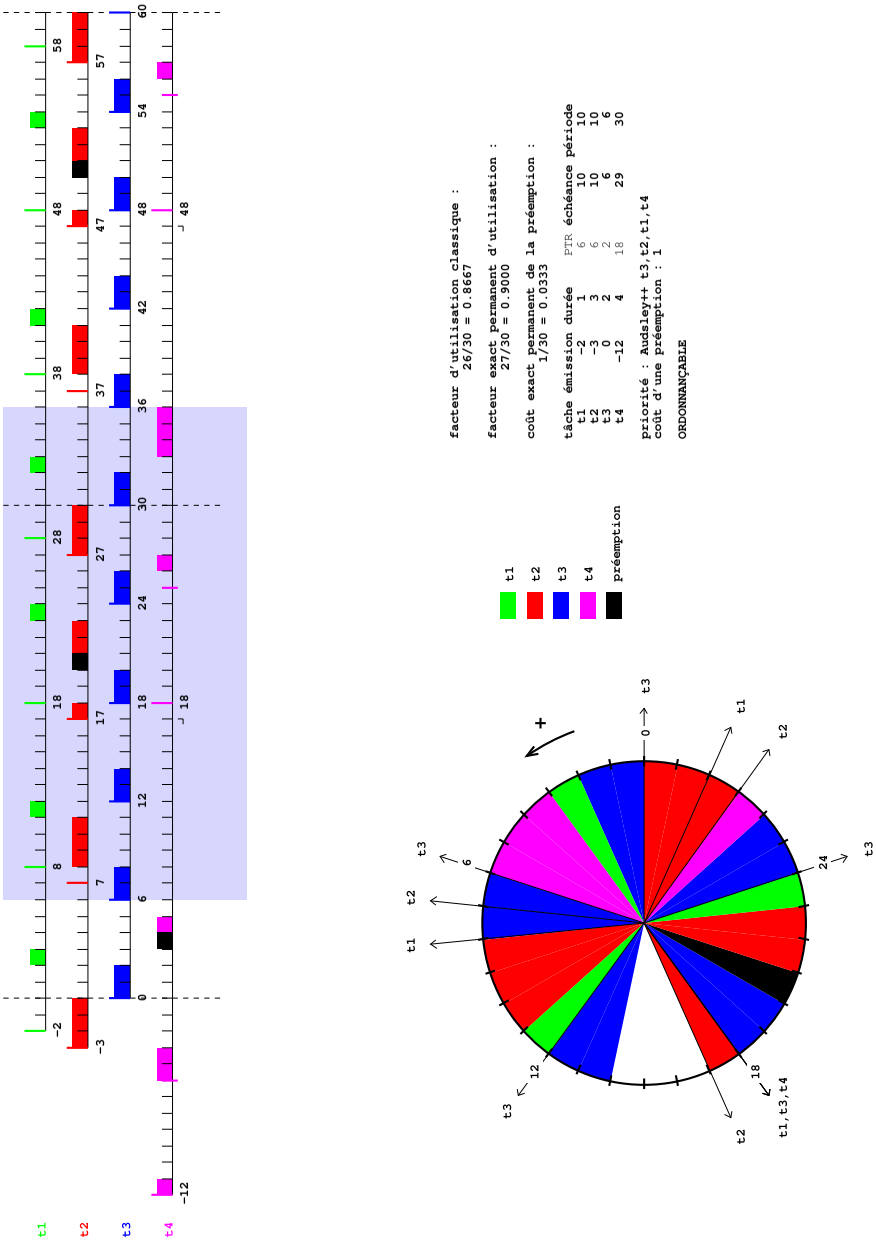


FIG. 6.13 – Solution 5 : choix des priorités <$t_3, t_2, t_1, t_4$>.

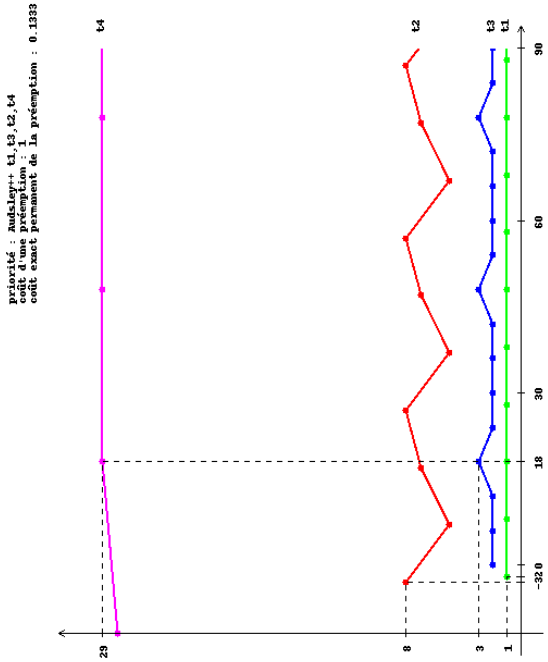


FIG. 6.14 – $\mathcal{S}_1 = < t_1, t_3, t_2, t_4 >$.

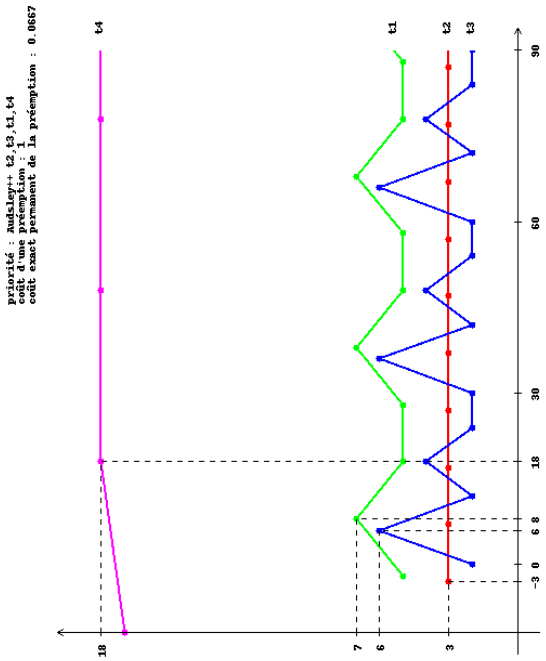


FIG. 6.15 – $\mathcal{S}_2 = < t_2, t_3, t_1, t_4 >$.

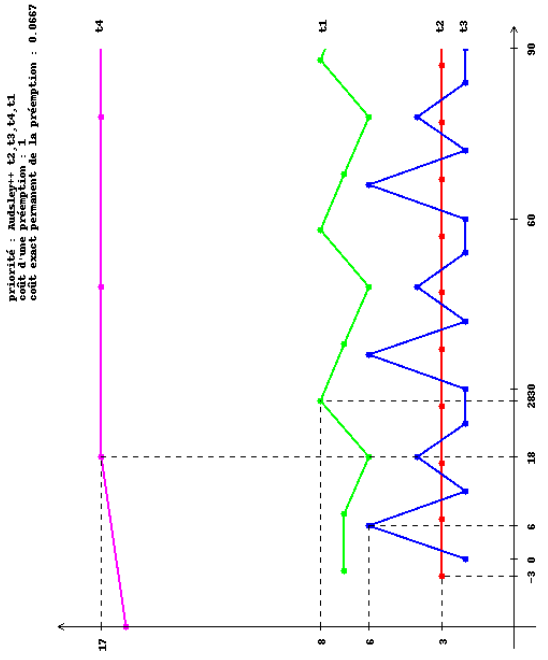


FIG. 6.16 – $\mathcal{S}_3 = < t_2, t_3, t_4, t_1 >$.

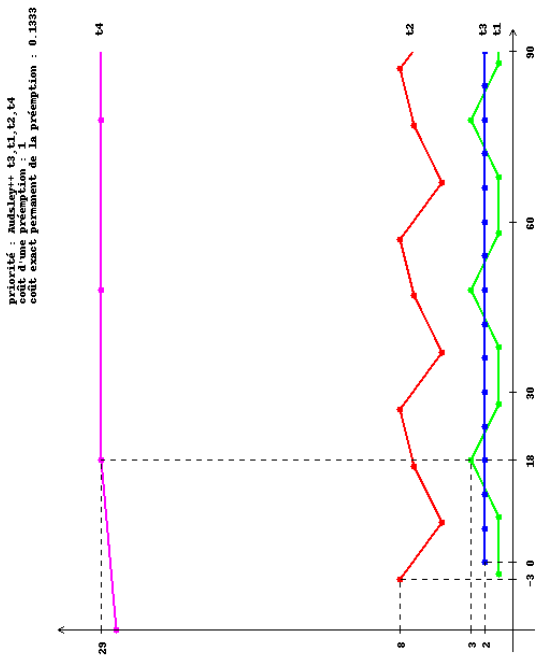


FIG. 6.17 – $\mathcal{S}_4 = < t_3, t_1, t_2, t_4 >$.

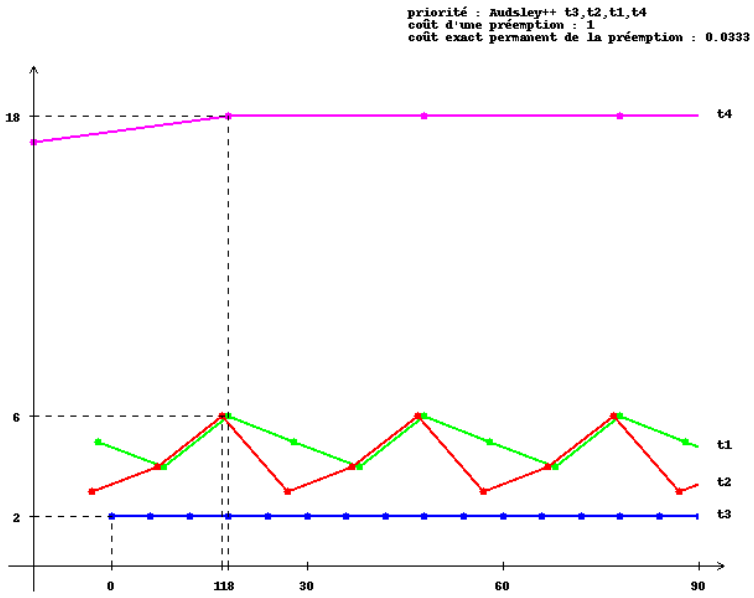


FIG. 6.18 – Temps de réponse en fonction du temps : $\mathcal{S}_5 = < t_3, t_2, t_1, t_4 >$.

et une boucle interactive, permettant l'expérimentation et un développement rapide.

*Camlp5* est un préprocesseur syntaxique d'*OCaml* qui est compatible avec les versions d'*OCaml* 3.08.0 à 3.11.0 inclus.

## 6.6 Conclusion

Dans ce chapitre, nous avons présenté le logiciel *SAS*. Ce logiciel simule, analyse et ordonnance des systèmes d'otâches périodiques en prennant en compte le coût exact de la préemption. Nous avons commencé par situer le contexte du développement de *SAS* puis nous avons présenté les étapes à suivre pour saisir les paramètres d'un système d'otâches périodiques donné dans *SAS*. Ensuite, nous avons présenté les résultats de l'analyse d'ordonnançabilité suivant une politique de choix de priorités des otâches fixée par l'utilisateur et nous avons expliqué comment obtenir la courbe du temps de réponse en fonction du temps de chaque otâche. Enfin, nous avons présenté les outils utilisés pour la réalisation de ce logiciel.

# Conclusion et perspectives

*La vérité scientifique a pour signe*
*la cohérence et l'efficacité.*
*La vérité poétique a pour signe la beauté.*
**Aimé Cesaire**

Dans cette thèse, nous avons considéré le problème d'ordonnancement temps réel monoprocesseur dans les systèmes temps réel durs. La plupart des problèmes rencontrés proviennent du fait que ces systèmes sont constitués de tâches périodiques qui nécessitent un respect absolu de toutes contraintes qui les caractérisent. Nous avons vu dans une première partie qu'il existait des algorithmes d'ordonnancement à priorités fixes optimaux permettant une validation analytique des systèmes de tâches périodiques. Lorsque le coût du système d'exploitation (*OS*) pour lequel le coût de la préemption est la partie variable est pris en compte, ce n'est plus le cas et le problème d'ordonnancement devient nettement plus difficile à cause de la prise en compte du coût de chaque préemption dans les conditions d'ordonnançabilité. Quelques travaux ont été proposés dans la littérature pour résoudre ce problème mais les résultats conduisaient à la prise en compte soit d'un nombre minimal soit d'un nombre maximal de préemptions. Par conséquent, ce n'était pas toujours possible de garantir l'ordonnançabilité d'un système de tâches très contraints.

Parce que les modèles classiques n'étaient pas adapter pour résoudre le problème, nous avons introduit un nouveau modèle pour résoudre le problème général de l'ordonnancement de systèmes temps réel durs avec des contraintes multiples telles que la précédence, la périodicité stricte, la latence et la gigue, tout en tenant compte du coût exact de la préemption pour tout scénario de première activation de toutes les tâches (simultané ou non simultané). Ce nouveau modèle permet donc de résoudre des problèmes rarement proposés dans les autres modèles tels que le modèle de *Liu & Layland* ou le modèle multiframe de *Mok & Chen* par exemple. L'analyse d'ordonnançabilité que nous avons proposée est basée sur la définition d'une opération binaire d'ordonnancement $\oplus$ dont les opérandes sont appelés *otâches*. Nous avons montré la correspondance entre une otâche et une tâche périodique. Puisque nous sommes dans un cadre d'ordonnancement à priorités fixes, comme une convention dans la définition de l'opération $\oplus$, la priorité la plus élevée est toujours attribuée à l'opérande de gauche et la plus basse à l'opérande de droite, donc l'opération $\oplus$ n'est pas *commutative*. Chaque otâche est un multiensemble ordonné composé d'une séquence finie ou infinie de symboles appartenant à un ensemble

fini appelé *générateur*. Pour le générateur constitué de deux symboles, $\{a, e\}$, le symbole "$a$" correspond toujours à une *unité de temps disponible* et le symbole "$e$" correspond à une *unité de temps déjà exécutée* quand il appartient à la otâche de gauche et à une *unité de temps exécutable* quand il appartient à la otâche de droite de l'opération $\oplus$. Pour un ensemble d'otâches, l'opération $\oplus$ est utilisée autant de fois qu'il y a d'otâches dans le système grâce à son *associativité*. Nous avons examiné deux approches. Premièrement, nous avons considéré le cas où la priorité de chaque otâche est connue, conduisant ainsi à un ordre décroissant des priorités des otâches. Deuxièmement, nous avons considéré le cas où la priorité de chaque otâche n'est pas connue. Dans ce cas, nous avons proposé un algorithme *optimal* de choix de priorités des otâches nommé $Audsley^{++}$. Nous avons utilisé notre modèle pour résoudre complètement le problème d'ordonnancement de systèmes temps réel avec des contraintes de précédence, de périodicité stricte et de latence. Nous avons étudié de près l'impact du scénario de premières activations de toutes les otâches sur l'analyse d'ordonnançabilité. Nous avons prouvé d'une part que le scénario, où les premières activations de toutes les otâches sont simultanées, ne correspond pas au pire cas, et d'autre part qu'il n'existe pas un tel scénario lorsque le coût de la préemption est pris en compte. Ensuite, nous avons prouvé que le choix des priorités des otâches fondé sur la politique d'*Audsley* n'est plus applicable, ni optimal. Ici optimal signifie que s'il existe une politique de choix des priorités qui conduit à un ordonnancement valide alors le choix des priorités proposé permettra également d'obtenir un ordonnancement valide. Nous avons simulé plusieurs ensembles de tâches avec *RTAI* sous Linux afin d'évaluer son impact sur l'analyse d'ordonnançabilité. Enfin, nous avons présenté l'impact de la contrainte de gigue sur notre modèle, les résultats préliminaires que nous avons obtenus ont concernés l'impossibilité de transformer un système soumis à cette contrainte en un autre système qui ne l'est pas lorsqu'on prend en compte le coût de la préemption. Nous avons montré l'impact de cette contrainte sur les conditions d'ordonnançabilité mais ceci demande encore d'être approfondie.

Nous avons développé un logiciel appelé *SAS* pour *Simulation Analysis of Scheduling* pour montrer l'applicabilité de notre approche et mettre à la disposition d'utilisateurs les résultats obtenus. *SAS* est un logiciel écrit en *Objective Caml (OCaml)* et est pour le moment supporté par les noyaux Linux et Windows. Il effectue l'analyse d'ordonnançabilité de systèmes d'otâches périodiques dans le cas monoprocesseur. Sa principale contribution, par rapport à d'autres outils commerciaux et académiques du même type, est que *SAS* prend en compte le coût exact de préemption au cours de l'analyse d'ordonnançabilité.

Les perspectives des travaux présentés dans cette thèse sont nombreuses. L'une des perspectives les plus prometteuses à l'utilisation de notre approche semble être une modélisation formelle du cas où les priorités sont attribuées aux otâches périodiques suivant un choix de priorités fixe, chaque otâche étant totalement préemptive et où il y a autorisation d'insertion de temps creux. Une autre piste d'amélioration intéressante consiste

à considérer le problème d'ordonnancement où les priorités sont attribuées aux otâches de façon dynamique lors de la formulation du problème d'ordonnancement, par exemple suivant les algorithmes *Earliest Deadline First (EDF) , Least Laxity First (LLF), etc*. Une autre piste non moins intéressante consisterait à étendre le travail de cette thèse à l'analyse d'ordonnançabilité de systèmes d'otâches sporadiques et apériodiques. Par ailleurs, il serait aussi intéressant d'étendre ces travaux à l'analyse de systèmes multi-processeurs tant pour les algorithmes de choix de priorités partionnés que globales en prennant en compte les mécanismes de synchronisation entre les différents processeurs car la dérive des processeurs dans une machine parallèle peut être importante.

# Références


[AARW91] N.C. Audsley, Burns A., M.F. Richardson, and A.J. Wellings.
Hard real-time scheduling : The deadline-monotonic approach.
*Proceedings 8th IEEE Workshop on Real-Time Operating Systems and Software*, 1991.

[ABR+93] N.C. Audsley, A. Burns, M.F. Richardson, Tindell K., and A.J. Wellings.
Applying new scheduling theory to static priority pre-emptive scheduling.
*Software Engineering Journal*, 1993.

[ABW93] N.C. Audsley, A. Burns, and A.J. Wellings.
Deadline monotonic scheduling theory and application.
*J. Control Engineering Practice*, 1993.

[AKA95] Burns A., Tindell K., and Wellings A.
Effective analysis for engineering real-time fixed priority schedulers.
*IEEE Trans. Software Eng.*, 1995.

[ATB93] N.C. Audsley, K. W. Tindell, and A. Burns.
The end of the line for the static cyclic scheduling ?
*In proceeding of the 5th Euromicro Workshop on Real-Time Systems*, 1993.

[Aud91] N. C. Audsley.
Optimal priority assignment and feasibility of static priority tasks with arbitrary start times.
*Dept. Comp. Science Report YCS 164, University of York*, 1991.

[Aud93] N.C. Audsley.
flexible scheduling of hard real-time systems.
*Ph.D. thesis, Departement of Computer science, University of York*, 1993.

[BB02] E. Bini and G. Buttazzo.
The space of rate monotonic schedulability.
*In Proc. of the 23rd IEEE Real-Time Systems Symposium, pp. 169-178*, 2002.

[BBB+01] E. Belhaire, E. Bourennane, G. Bouvier, D. Demigny, P. Garda, L. Kessal, L. Lacassagne, F. Lohier, M. Paindavoine, Y. Sorel, L. Torres, and S. Weber.
*Méthodes et architectures pour le traitement du signal et des images en*

*temps réel*, chapter 5 : Y. Sorel, Méthodologie AAA et logiciel SynDEx, pages 79–108.
IC2. Hermes, 2001.

[BBC+01] J. Bennett, K. Benson, A. Charny, et al.
Delay jitter bounds and packet scale rate guarantee for Expedited Forwarding.
*INFOCOM'01*, Anchorage, USA, April 2001.

[BBGL99] Sanjoy Baruah, Giorgio Buttazzo, Sergey Gorinsky, and Giuseppe Lipari.
Scheduling periodic task systems to minimize output jitter.
*Proceedings of the IEEE International Conference on Real-Time Computing Systems and Applications, pp 62-69, Hong Kong*, 1999.

[BCM97] Sanjoy Baruah, Deji Chen, and Aloysius Mok.
Jitter concerns in periodic task systems.
*Proceedings of the Real-Time Systems Symposium, pp 68-77, IEEE Computer Society Press*, 1997.

[Ber03] G. Bernat.
Response time analysis of asynchronous real-time systems.
*Journal of Real-Time Systems*, 2003.

[BGK97] S. Baruah, Goddard, and Jeffay K.
Feasibility concerns in pgm graphs with bounded buffers.
*IEEE*, 1997.

[BHR90] S.K. Baruah, R.R. Howell, and L.E. Rosier.
Algorithms and complexity concerning the preemptive scheduling of periodic real-time tasks on one processor.
*Journal of Real-Time Systems*, 1990.

[Bim07] F. Bimbard.
*Dimensionnement Temporel de Systèmes Embarqués : Application à OSEK*.
PhD thesis, Université Pierre et Marie Curie, Paris 6, 2007.

[Bla76] J. Blazewicz.
Scheduling dependent tasks with different arrival times to meet deadlines.
*Modelling and Performance Evaluation of Computer Systems*, 1976.

[Bli91] W.D. Blizard.
*The Development of Multiset Theory*.
Modern Logic, 1 :319-352, 1991.

[BMR90] S.K. Baruah, A.K. Mok, and L.E. Rosier.
Preemptively scheduling hard realtime sporadic tasks on one processor.
*In proc. 11th IEEE Real-Time Systems Symposium*, 1990.

[BP05] F. Bodin and I. Puaut.

A WCET-oriented static branch prediction scheme for real-time systems.
In *Proc. of the 17th Euromicro Conference on Real-Time Systems*, Palma de Mallorca, Spain, July 2005.

[But97] G. Buttazzo.
Hard real-time computing systems : Predictable scheduling algorithms and applications.
*Kluwer, Academic Publisher, Boston*, 1997.

[BWBF93] A. Burns, A.J. Wellings, C. Bailey, and E. Fyfe.
The olympus attitude and orbital control system, a case study in hard real-time system design and implementation.
*Springer Verlag, pp. 240-248*, 1993.

[CC86] H. Chetto and M. Chetto.
On the acceptation of non-periodic time critical tasks in distributed systems.
*Proc. 7th IFAC Workshop Distributed Computer Control Systems (DCCP-86)*, 1986.

[CC89] H. Chetto and M. Chetto.
Some results on the earliest deadline scheduling algorithm.
*IEEE Transactions on Software Engineering*, 1989.

[CKS02] L. Cucu, R. Kocik, and Y. Sorel.
Real-time scheduling for systems with precedence, periodicity and latency constraints.
*Real-time and Embedded Systems*, 2002.

[CMK99] D. Chen, A. K. Mok, and T. Kuo.
Utilization bound revisited.
*In Proceedings of the Sixth International Conference on Real-Time Computing and Applications, pp. 295-302*, 1999.

[CMT90] H. Chetto, Silly M., and Bouchentouf T.
Dynamic scheduling of real-time tasks under precedence constraints.
*The Journal of Real-Time Systems*, 1990.

[CP00] A. Colin and I. Puaut.
Worst case execution time analysis for a processor with branch prediction.
*Real-Time Systems, Special issue on worst-case execution time analysis*, 18(2) :249–274, April 2000.

[CPS07] L. Cucu, N Pernet, and Y. Sorel.
Periodic real-time scheduling : from deadline-based model to latency-based model.
*Annals of Operational Research*, 2007.

[CS03] L. Cucu and Y. Sorel.
Schedulability condition for systems with precedence and periodicity constraints without preemption.
*Real-time and Embedded Systems*, 2003.

[Cuc04] L. Cucu.
*Ordonnancement non préemptif et condition d'ordonnançabilité pour système embarqués à contraintes temps réel*.
PhD thesis, Université de Paris Sud, Spécialité Électronique, 28/05/2004.

[DCG00] L. David, F. Cottet, and E. Grolleau.
Gigue temporelle et ordonnancement par échéance dans les applications temps réel.
*IEEE Conf. Inter. Francophone d'Automatique (CIFA)*, 2000.

[DG00] R. Devilliers and J. Goossens.
*Liu and Layland's Schedulability Test Revisited*.
PhD thesis, Information Processing Letters, 73(5-6) :157-161, 2000.

[DLAS98] A. Dias, C. Lavarenne, M. Akil, and Y. Sorel.
Optimized implementation of real-time image processing algorithms on field programmable gate arrays.
In *Proceedings of Fourth International Conference on Signal Processing, ICSP'98*, Beijing, China, October 1998.

[DNSS01] R. Djenidi, R. Nikoukhah, S. Steer, and Y. Sorel.
Interface scicos-syndex.
Rapport de recherche 4250, INRIA, septembre 2001.

[DRM97] Kang D.I., Gerber R., and Saksena M.
Performance-based design of distributed real-time systems.
*In Proceedings of IEEE Real-time Technology and Applications, 2-13*, 1997.

[ERC95] J. Echagüe, I. Ripoll, and A. Crespo.
Hard real-time preemptively scheduling with high context switch cost.
*In Proceedings of the 7th Euromicro Workshop on Real-Time Systems*, 1995.

[Fer90] D. Ferrari.
Client requirements for real-time communications services.
*IEEE Communications, 28(11)*, 1990.

[GLS99] T. Grandpierre, C. Lavarenne, and Y. Sorel.
Optimized rapid prototyping for real-time embedded heterogeneous multiprocessors.
In *Proceedings of 7th International Workshop on Hardware/Software Co-Design, CODES'99*, Rome, Italy, May 1999.

[God00] S. Goddard.
*Constraints on data-flow*.
PhD thesis, University of North Carolina, 2000.

[Goo98] J. Goossens.
*Scheduling of Hard Real-Time Periodic Systems with Various Kinds of Deadline and Offset Constraints*.
PhD thesis, Université Libre de Bruxelles, 1998.

[GPM97] R. Gerber, W. Pugh, and Saksena M.
Parametric dispatching of hard real-time tasks.
*IEEE Transactions on Computers, 44(3), 471-479*, 1997.

[Gra69] R. Graham.
*Bounds on multiprocessing timing anomalies*.
SIAM Journal of Applied Mathematics 17(2), 1969.

[Gro99] E. Grolleau.
*Ordonnancement Temps Réel Hors-ligne Optimal à l'aide de Réseaux de Pétri en Environnement Monoprocesseur et Multiprocesseur*.
PhD thesis, Ecole Nationale Supérieure de Mécanique et d'Aérotechnique, 1999.

[GS02] T. Grandpierre and Y. Sorel.
Un nouveau modèle générique d'architecture hététogène pour la méthodologie aaa.
In *Actes des Journées Francophones sur l'Adéquation Algorithme Architecture, JFAAA'02*, Monastir, Tunisia, December 2002.

[GS03] T. Grandpierre and Y. Sorel.
From algorithm and architecture specification to automatic generation of distributed real-time executives : a seamless flow of graphs transformations.
In *Proceedings of First ACM and IEEE International Conference on Formal Methods and Models for Codesign, MEMOCODE'03*, Mont Saint-Michel, France, June 2003.

[Han98] C.-C Han.
A better polynomial-time schedulability test for real-time multiframe tasks.
*In Proceeding of the 19th IEEE Real-Time Systems Symposium, pp 104-113*, 1998.

[HD03] P. Hladik and A. Deplanche.
Ordonnancement préemptif à priorités fixes : comparaison des méthodes analytiques de calcul de pire temps de réponse.
*In Proc. Embedded Systems (RTS'03), Paris*, 2003.

[HP08] D. Hardy and I. Puaut.
Predictable code and data paging for real-time systems.
In *Proc. of the 20th Euromicro Conference on Real-Time Systems*, Prague, Czech Republic, July 2008.

[HT97] C.-C Han and H.-Y. Tyan.
A better polynomial-time schedulability test for real-time fixed-priority scheduling algorithms.
*In Proceeding of the 18th IEEE Real-Time Systems Symposium, pp 38-45*, 1997.

[Jos85] M. Joseph.
On a problem in real-time computing.
*Information Processing Letters, vol. 20, 4 : pp 173-177*, 1985.

[JP86a] M. Joseph and P. Pandya.
Finding response times in a real-time system.
*BCS Comp. Jour.*, 29(5), pp. 390-395,, 1986.

[JP86b] M. Joseph and P. Pandya.
Finding response times in real-time system.
*BCS Computer Journal*, 1986.

[JS93] K. Jeffay and D.L. Stone.
Accounting for interrupt handling costs in dynamic priority tasks systems.
*Proc. IEEE 14th Real-Time Systems Symp.*, 1993.

[KAL96] J.H.M. Korst, E.H.L. Aarts, and J.K. Lenstra.
Scheduling periodic tasks.
*INFORMS Journal on Computing 8*, 1996.

[KAL97] J.H.M. Korst, E.H.L. Aarts, and J.K. Lenstra.
Scheduling periodic tasks with slack.
*INFORMS Journal on Computing*, 1997.

[KAS93] D. Katcher, H. Arakawa, and J. Strosnider.
Engineering and analysis of fixed priority schedulers.
*IEEE Transactions on Software Engineering, vol. 19 : pp 920-934*, 1993.

[Knu98] Donald E. Knuth.
*The art of computer programming*.
Vol 2 : Seminumerical algorithms, 3rd edition, Addison Wesley, 694, 1998.

[KRP+93] M. Klein, T. Ralya, B. Pollack, R. Obenza, and M. Gonzales-H.
A practitioner's handbook for real-time system analysis.
*Kluwer Academic Publishers*, 1993.

[KS98] R. Kocik and Y. Sorel.

A methodology to design and prototype optimized embedded robotic systems.
In *Proceedings of 2nd IMACS International Multiconference, CESA'98*, Hammamet, Tunisia, April 1998.

[Law71] E.L. Lawler.
Optimal sequencing of a single machine subject to precedence constraints.
Technical report, University of California, 1971.

[Law73] E.L. Lawler.
Optimal sequencing of a single machine subject to precedence constraints.
*Management Science*, 1973.

[Law78] E.L. Lawler.
Sequencing problems with series parrallel precedence constraints.
*Proc. Confer. on Combinatorial Optimization*, 1978.

[Leh90] J.P. Lehoczky.
Fixed priority scheduling of periodic task sets with arbitrary deadlines.
*Proceedings 11th IEEE Real-Time Systems Symposium*, pp 201-209, Dec. Lake Buena Vista, FL, USA, 1990.

[LH96] K. Lin and A. Herkert.
Jitter control in time-triggered systems.
*In proceedings of the 29th Hawaï International Conference on System Sciences*, 1996.

[Liu00] J. Liu.
*Real-time systems*.
Prentice Hall, 2000.

[LL73] C.L. Liu and J.W. Layland.
Scheduling algorithms for multiprogramming in a hard-real-time environment.
*Journal of the ACM*, 1973.

[LM80] J. Y. T. Leung and M.L. Merrill.
A note on premptive scheduling of periodic, Real Time Tasks.
*Information Processing Letters*, Vol 11, num 3, Nov. 19980.

[LMM98] S. Lauzac, R. Melhem, and D. Mosse.
An efficient rms admission control and its application to multiprocessor scheduling.
*In Parallel Processing Symposium, pp. 511-518*, 1998.

[LS92] C. Lavarenne and Y. Sorel.

Specification, performance optimization and executive generation for real-time embedded multiprocessor applications with SynDEx.
In *Proceedings of Real-Time Embedded Processing for Space Applications*, Les Saintes Maries de la Mer, France, November 1992. CNES International Symposium.

[LSD89] J.P. Lehoczky, L. Sha, and Y Ding.
The rate monotonic sheduling algorithm : exact characterization and average case bahavior.
*Proceedings of the IEEE Real-Time Systems Symposium*, 1989.

[LSSS91] C. Lavarenne, O. Seghrouchni, Y. Sorel, and M. Sorine.
The SynDEx software environment for real-time distributed systems, design and implementation.
In *Proceedings of European Control Conference, ECC'91*, Grenoble, France, July 1991.

[LW82] J. Leung and J. Whitehead.
On the complexity of fixed-priority scheduling of periodic real-time tasks.
*Performance Evaluation(4)*, 1982.

[MA98] Y. Manabe and S. Aoyagi.
A feasibility decision algorithm for rate monotonic and deadline monotonic.
*Real-Time Systems, 14(2) :171-181*, 1998.

[MBFSV06] L. Mangeruca, M. Baleani, A. Ferrari, and A. L. Sangiovanni-Vincentelli.
Uniprocessor scheduling under precedence constraints.
In *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, San Jose, California, United States, April 2006.

[Mok83] A. K. Mok.
Fundamental design problems for the hard real-time environments.
*MIT Ph.D. Dissertation*, May 1983.

[MW85] Z. Manna and R. Waldinger.
*The Logical Basis for Computer Programming*.
volume 1 : Deductive Reasoning. Addison-Wesley, Reading, Massachusetts, 1985.

[OCSF97] J. Orozco, R. Cayssials, J. Santos, and E. Ferro.
Precedence constraints in hard real-time distributed systems.
*3rd International Conference on Engineering of Complex Computer Systems*, 1997.

[PGBS03] Q. Pan, T. Gautier, L. Besnard, and Y. Sorel.

Signal to syndex : Translation between synchronous formalisms. internal report, 2003.
Internal report, INRIA, Rocquencourt, France, 2003.

[PK89] D. Peng and Shin K.G.
Static allocation of periodic tasks with precedence constraints in distributed real-time systems.
*9th International Conference on Distributed Computing Systems*, 1989.

[PNKK95] D.W. Park, S. Natarajan, A. Kanevsky, and M. J. Kim.
A generalized utililization bound test for fixed-priority real-time scheduling.
*In Proceedings of the 2nd International Workshop on Real-Time Computing Systems and Applications, pp.73-77*, 1995.

[Ram95] K. Ramamritham.
Allocation and scheduling of precedence-related periodic tasks.
*IEEE Transactions on Parallel and Distributed Systems*, 1995.

[Rau06] M. Raulet.
*Optimisation mémoire dans la méthodologie AAA pour code embarqué sur architectures parallèles*.
PhD thesis, Institut des Sciences Appliquées de Rennes, Spécialité Electronique et Traitement du Signal, 18/05/2006.

[RT02] O. Redell and M. Törgren.
Calculating exact worst-case response times for static priority scheduled tasks with offsets and jitter.
*In Proceedings of Real-Time Systems and Applications Symposium (RTAS'02)*, 2002.

[SCGM99] Baruah S., D. Chen, S. Gorinsky, and A. Mok.
Generalized multiframe tasks.
*Real-Time Systems, Volume 17, Number 1*, 1999.

[Ser72] O. Serlin.
Scheduling of time critical processes.
*In Proceedings AFIPS Spring Computing Conference*, 1972.

[SKB91] L. Sha, Mark H. Klein, and Goodenough John B.
Rate monotonic analysis for real-time systems.
*Technical Report CMU/SEI-91-TR-6 ESD-91-TR-6*, 1991.

[Sor94] Y. Sorel.
Massively parallel systems with real time constraints, the algorithm architecture adequation methodology.
In *Proceedings of Conference on Massively Parallel Computing Systems, MPCS'94*, Ischia, Italy, May 1994.

[Sor04] Y. Sorel.
Syndex : System-level cad software for optimizing distributed real-time embedded systems.
*Journal ERCIM News*, 59 :68–69, October 2004.

[Sor05] Y. Sorel.
From modeling/simulation with scilab/scicos to optimized distributed embedded real-time implementation with syndex.
In *Proceedings of the International Workshop On Scilab and Open Source Software Engineering, SOSSE'05*, Wuhan, China, October 2005.

[SRL93] Baruah S.K., Howell R.R., and Rosier L.E.
Feasibility problems for recurring tasks on one processor.
*Theoretical Computer Science 118(1)*, 1993.

[SS94] M. Spuri and J.A. Stankovic.
How to integrate precedence constraints and shared resources in real-time scheduling.
*IEEE Transactions on Computers (43)*, 1994.

[Sta88] J.A. Stankovic.
A serious problem for next-generation systems.
*IEEE Computer Society, 21(10) :10-19*, 1988.

[SW98] Klaus Schild and Jörg Würtz.
Off-line scheduling of a real-time system.
*SAC '98 : Proceedings of the 1998 ACM symposium on Applied Computing*, 1998.

[TBW94] K. Tindell, A. Burns, and A. J. Wellings.
An extendible Approach For Analysing Fixed Priority Hard Real-Time Tasks.
*Real-Time Systems 6(2)v*, 1994.

[Tin93] K. Tindell.
Holistic scheduling analysis for distributed hard real-time systems.
*No. YCS 197, Departement of Computer Science, University of York*, 1993.

[Tov02] A. Tovey.
tutorial on computational complexity.
*Interface 32 (3)*, 2002.

[VZF91] D. Verma, H. Zhang, and D. Ferrari.
Delay jitter control for real-time communication in a packet switching network.

*Proceedings of the IEEE Conference on Communications Software : Communications for Distributed Applications and Systems*, 1991.

[WEE+08] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström.
The Determination of Worst-Case Execution Times—Overview of the Methods and Survey of Tools.
7(3), 2008.
ACM Transactions on Embedded Computing Systems (TECS).

[Yag86] Ronald R. Yager.
*On the theory of bags*.
Int. J. General Systems, 13 :23-37, 1986.

[ZDE+93] L. Zhang, S. Deering, S. Estrin, et al.
RSVP : A new Resource ReServation Protocol.
*IEEE Network*, Vol. 7, No. 5, pp. 8-18, September 1993.

# Index