Modèles de temps et leur intérêt à la vérification formelle des systèmes temps-réel

Nabil Belala

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mentouri de Constantine
Faculté des Sciences de l'Ingénieur
Département d'Informatique

Année : 2010
N° d'ordre : 95/TS/2010
N° de série : 10/INF/2010

THESE

pour l'obtention du diplôme de Docteur en Sciences
Spécialité : Informatique

Modèles de temps et leur intérêt à la vérification formelle des systèmes temps-réel

Nabil Belala

Soutenue le 20 octobre 2010 devant le jury composé de :

Dr Allaoua Chaoui, Président, Université Mentouri de Constantine
Dr Djamel-Eddine Saïdouni, Rapporteur, Université Mentouri de Constantine
Dr Rachid Boudour, Examinateur, Université Badji Mokhtar de Annaba
Dr Mohamed Taher Kimour, Examinateur, Université Badji Mokhtar de Annaba
Dr Salah Merniz, Examinateur, Université Mentouri de Constantine

Thèse préparée au Laboratoire MISC, Université Mentouri de Constantine

Modèles de Temps et leur Intérêt à la Vérification Formelle des Systèmes Temps-Réel

Nabil Belala

Equipe CFSC, Laboratoire MISC
Université de Mentouri, 25000 Constantine, Algérie

A mes défunts parents ...

Remerciements

Je tiens à remercier en premier lieu Dr Allaoua Chaoui pour avoir accepter la lourde tâche de présider le jury de la soutenance de ma thèse de Doctorat en Sciences.

Je remercie également docteurs Rachid Boudour, maitre de conférences à l'Université Badji Mokhtar de Annaba, Mohamed Taher Kimour, maitre de conférences à l'Université Badji Mokhtar de Annaba, et Salah Merniz, maitre de conférences à l'Université Mentouri de Constantine, pour le temps qu'il ont accordé pour examiner ce travail.

Je remercie tout particulièrement Dr Djamel-Eddine Saïdouni qui m'a encadré durant cette thèse. Il a été toujours disponible pour répondre aux questions que je lui posais, que ce soit théoriques, techniques ou LaTeXniques. Son enthousiasme et sa patience ont beaucoup facilité et agrémenté mon travail.

Bien entendu, je n'oublie pas Dr Mohamed Khireddine Kholladi, qui m'a chaleureusement accueilli dans le Laboratoire MISC (ça sent MISC !).

Je remercie aussi tous les membres de ma famille qui se sont occupés du pot de thèse.

J'adresse également mes remerciements aux chercheurs, enseignants, ingénieurs et thésards que j'y ai côtoyés durant toutes ces années.

Résumé

Actuellement, les méthodes formelles sont de plus en plus utilisées dans le but d'analyser le comportement les systèmes dits temps-réel. Ces méthodes utilisent des modèles formels de spécification dotés de sémantiques bien définies et des techniques de vérification formelle.

D'autre part, les sémantiques de vrai parallélisme, comme la sémantique de maximalité, peuvent être utilisées à bon escient si on veut échapper à l'hypothèse de l'atomicité temporelle et structurelle des actions. Des travaux antérieurs ont montré l'aptitude de modèles basés sur la maximalité, à l'image des systèmes de transitions étiquetées maximales, à supporter l'expression du parallélisme d'exécution d'actions.

Notre travail qui s'inscrit dans le cadre de la spécification et la vérification formelle des systèmes temps-réel consiste à tirer profit des avantages offerts par la sémantique de maximalité en l'appliquant à quelques modèles temporels et/ou temporisés. Ainsi, nous proposons trois méthodes de génération de structures basés sur la maximalité (systèmes de transitions étiquetées maximales, DATA et DATA*) à partir de modèles de haut niveau à savoir le langages Basic LOTOS avec durées d'actions, la langage D-LOTOS et les réseaux de Petri place/transition.

Mots-clés Systèmes temps-réel, sémantique de maximalité, réseaux de Petri, LOTOS, spécification formelle, vérification formelle.

Table des matières

Table des figures

Liste des tableaux

Liste d'algorithmes

Chapitre 1

Introduction

Aujourd'hui, les systèmes informatiques deviennent de plus en plus complexes et le risque de les mal concevoir est un problème croissant. Par conséquent, la présence d'un comportement indésirable dans un module ou un logiciel critique peut mener à des conséquences dramatiques ; il existe de nombreux exemples de notre vie courante de mal fonctionnement qui avaient entraîné des pertes humaines ou des dégâts matériels.

Les premières étapes dans le processus de développement d'un système ont pour but de garantir que le celui-ci respecte les exigences de fiabilité et de performances spécifiées au départ. Or, le cahier des charges peut contenir un grand nombre d'exigences, et ceci peut être source d'erreurs ou d'ambigüités. Les méthodes formelles de développement ont vu le jour pour contourner ces ambigüités en proposant des approches plus ou moins rigoureuses.

Pour certains systèmes, les exigences du cahier des charges comprennent les contraintes d'exécution que le système ou une de ses modules doit respecter. Un système est dit *temps-réel* si son bon fonctionnement est liée au respect de ces contraintes d'exécution. Ainsi, des modèles et des méthodes formelles *temporelles* et/ou *temporisées* ont été développées, et les techniques de spécification et de vérification ont été étendues au systèmes temps-réel.

1.1 Problématique

Une technique de vérification formelle particulièrement attractive du point de vue coût/performances est celle basée sur l'évaluation de modèles (*model checking*). Dans cette approche, le système à analyser est d'abord spécifié dans un modèle ou langage de spécification de haut niveau. Cette spécification est ensuite traduite vers un modèle (graphe d'états ou automate) sur lequel les propriétés attendues du système, exprimées

dans un formalisme approprié tel que les logiques temporelles, sont vérifiées à l'aide de *model checkers*. Bien que restreinte à des systèmes ayant un nombre fini d'états, cette approche a l'avantage d'être totalement automatisable, elle tire également profit de l'abstraction et de la modularité du formalisme des logiques temporelles.

Cependant, les modèles sur lesquels la vérification est faite sont souvent générés à partir de spécifications en considérant la sémantique d'entrelacement. Pourtant, cette dernière n'exprime pas naturellement le vrai parallélisme, en plus elle ne permet pas de représenter correctement le comportement des systèmes concurrents dès que les actions ne sont plus atomiques (indivisibles et/ou instantanées). Pour remédier à ce problème, on peut utiliser les modèles du vrai parallélisme qui permettent d'éviter ces difficultés et de conserver la nature parallèle des programmes concurrents.

En profitant de cet avantage, une sémantique permettant d'exprimer le vrai parallélisme, la *sémantique de maximalité*, a été définie [CS95, Saï96] et utilisée dans la littérature [SB03a, SB05, SB04, SDAB04]. La sémantique de maximalité a été prouvée nécessaire et suffisante pour le raffinement d'actions et pour les durées d'actions [Saï96].

Dans le but de continuer d'exploiter ces résultats, nous proposons dans ce travail d'étendre l'adoption de la sémantique de maximalité sur les modèles utilisés pour modéliser les systèmes temps-réel. Ceci nous mène à la proposition de modèles sémantiques de maximalité pour deux extensions de l'algèbre de processus LOTOS [ISO88], à savoir Basic LOTOS avec durées d'actions et D-LOTOS [SC03]. Les structures générées seront appelées DATA et DATA* (pour *Durational Action Timed Automata*(*)).

Dans la perspective d'étendre la maximalité aux autres modèles de spécification, nous proposons de donner une sémantique de maximalité pour les réseaux de Petri place/transition en terme de structures implémentables basées sur la maximalité. Ce dernier modèle s'appelle les *systèmes de transitions étiquetées maximales*.

1.2 Contributions

Notre travail s'inscrit dans le contexte de la spécification des systèmes temps-réel. Nous proposons l'étude des points suivants :

Étude de modèles de temps Nous passons en revue une variété de modèles conçues spécialement pour spécifier les systèmes temps-réel. Ces modèles varient des extensions temporelles et temporisées des algèbres de processus (langage D-LOTOS [SC03]) aux réseaux de Petri, en passant par les automates temporisés. Chacun des modèles cités est étudié dans le but d'analyser son aptitude à exprimer les

comportements des systèmes temps-réel, et principalement son aptitude à exprimer les durées des actions, les contraintes temporelles et les contraintes d'urgence.

Sémantique de vrai parallélisme pour LOTOS et D-LOTOS Nous proposons la génération de structures sémantiques pour l'algèbre de processus Basic LOTOS [ISO88] et son extension D-LOTOS. Les structures ainsi définies s'appellent les *Durational Action Timed Automata* (DATA et DATA*). Les DATA forment un modèle de bas niveau permettant l'expression des comportements sans avoir à éclater les actions en évènements représentant le début et la fin de leur exécution. Leur objectif est de rendre la spécification aussi naturelle que possible. D'autre part, les DATA* sont une extension des DATA permettant l'expression des contraintes temporelles et des contraintes d'urgence. Ces dernières sont supportées par le langage D-LOTOS, ainsi un passage d'expressions D-LOTOS vers les DATA* est proposé.

Sémantique de vrai parallélisme pour les réseaux de Petri Dans une autre partie de ce travail, nous donnerons une sémantique au modèle des réseaux de Petri place/transition en terme de systèmes de transitions étiquetées maximales. L'avantage de générer un système de transitions étiquetées maximales qui correspond à chaque réseau de Petri place/transition est double. D'une part, le modèle des systèmes de transitions étiquetées maximales est un modèle implémentable. D'autre part, ce dernier préserve l'élégance de la spécification du parallélisme inhérente aux réseaux de Petri en exprimant le parallélisme de façon naturelle.

Méthodes de réduction pour la génération Dans la dernière partie, nous profitons de la sémantique de maximalité pour explorer quelques techniques de réduction lors de la génération de systèmes de transitions étiquetées maximales à partir de réseaux de Petri. La première étant l'agrégation de transitions équivalentes vis-à-vis de la relation de bisimulation de maximalité qui sera introduite. La deuxième technique a comme but la génération, à-la-volée (*on-the-fly*), de systèmes de transitions étiquetées maximales réduits modulo la relation de la α-équivalence.

1.3 Plan du document

Ce mémoire est organisé de la manière suivante :

- Le Chapitre 2 introduit les systèmes temps-réel et des notions générales sur spécification et la vérification formelle de ces derniers. Une deuxième partie de ce

chapitre donne l'intuition de la sémantique de maximalité qui constitue le noyau sémantique de nos travaux.

- Le Chapitre 3 résume les modèles sur lesquels nous nous basons dans cette thèse, à savoir le modèle de spécification Basic LOTOS et l'une de ses extensions temporelles et temporisées D-LOTOS, le modèle des réseaux de Petri place/transition, le modèle des systèmes de transitions étiquetées maximales ainsi que les automates temporisés. Une présentation intuitive suivie d'une formalisation sont données pour chacun des modèles cités.

- Le Chapitre 4 présente et motive la technique de vérification formelle appelée model checking. Une adaptation de l'algorithme de Clarke et Emerson [CES86, Eme90] est donnée pour les systèmes de transitions étiquetées maximales. Nous montrons dans ce chapitre comment vérifier plusieurs propriétés intéressantes de sûreté et vivacité en utilisant une approche basée sur la sémantique de maximalité.

- Dans le Chapitre 5, nous définissons deux modèles sémantiques de bas niveau destinés respectivement aux systèmes avec durées explicites d'actions et aux systèmes avec durées explicites, contraintes temporelles et contraintes d'urgence. Ces modèles sont appelés DATA et DATA* (pour *Durational Action Timed Automata*(*)).

- Dans le chapitre suivant, le Chapitre 6, nous donnons une sémantique de maximalité en terme de DATA, DATA* et systèmes de transitions étiquetées maximales respectivement aux expressions Basic LOTOS avec durées explicites d'actions, aux expressions D-LOTOS, et aux réseaux de Petri place/transition.

- Le Chapitre 7 clôture le mémoire avec une conclusion et perspectives.

Chapitre 2

Introduction générale

Ce chapitre constitue une introduction générale au domaine de la spécification et de la vérification formelle des systèmes temps-réels. Nous commençons par donner des notions des systèmes temps-réel tout en motivant l'emploi de formalismes, de modèles ou de méthodes dites *formelles* afin de les étudier.

2.1 Notions préliminaires

Aujourd'hui, l'efficacité de plusieurs systèmes et composants informatiques matériels ou logiciels ne dépend pas seulement des résultats ou effets qu'ils produisent mais aussi du temps dans lequel ses résultats sont produits. Ces systèmes à temps contraint peuvent varier du système d'assistance au freinage ABS dans une voiture aux moniteurs vitaux dans les hôpitaux, et des contrôleurs utilisés dans la robotique au système de vol automatique dans un avion. Par exemple, quand un conducteur d'une voiture applique un freinage, le contrôleur ABS (*Antiblockiersystem*) analyse l'environnement dans lequel il est embarqué (vitesse du véhicule, la surface de la route et la direction) et active le freinage avec la fréquence appropriée. Le résultat (activation du freinage) et le temps dans lequel le résultat est produit sont tous les deux importants pour s'assurer de la sûreté du véhicule, du conducteur et des passagers.

Récemment, le matériel et les logiciels informatiques sont de plus en plus introduits dans la plupart de ces systèmes temps-réel afin de surveiller et contrôler leurs opérations. Ces systèmes informatiques sont appelés systèmes *embarqués*, systèmes informatiques temps-réel ou tout simplement systèmes *temps-réel*. Contrairement aux systèmes informatiques conventionnels, les systèmes temps-réel sont généralement couplés avec leur environnement et doivent répondre à des contraintes temporelles strictes en plus de la satisfaction des besoins fonctionnels.

Afin d'analyser ce type de systèmes et s'assurer de leur bon fonctionnement et qu'ils prennent effectivement en compte leurs contraintes temporelles en respectant les besoins fonctionnels pour lesquels ils ont été conçus, plusieurs approches d'analyses, généralement complémentaires, s'imposent. Ces approches varient de la simulation à la surveillance (monitoring) en passant par le test et la vérification.

La simulation consiste en la construction et l'étude d'un modèle d'un système existant ou un système à construire. Le modèle (ou le simulateur) peut être une entité physique ou bien une représentation informatique. Un modèle informatique est en général moins coûteux si on le compare avec un modèle physique, et peut représenter aussi bien un système physique qu'un système informatique ou un programme. Un modèle informatique peut représenter aussi un système rassemblant à la fois des composants logiciels et matériels comme par exemple une voiture ayant un système embarqué contrôlant la transmission et le freinage. L'un des inconvénients de la simulation comme technique d'analyse et vérification des systèmes temps-réel et d'autres systèmes est la difficulté de simuler toutes les séquences d'évènements observables possibles si le nombre de ces dernières est infini. Même si ce nombre est fini, le nombre d'évènements peut être très large au point que les systèmes informatiques les plus performants ne peuvent les tracer.

De l'autre côté, le test, qui est peut-être l'une des techniques les plus anciennes pour la détection des erreurs ou problèmes dans les implémentations des systèmes logiciels ou matériels, consiste à exécuter le système à tester en utilisant un ensemble fini de valeurs d'entrée tout en vérifiant si le comportement résultant est correct ou non par rapport à la spécification de départ. Le test peut donner de bon résultats pour révéler les erreurs dans le système testé mais ne garantit pas, en général, que le système satisfait tous les besoins, comme c'est le cas du test exhaustif pour les systèmes très larges.

D'autres techniques d'étude des systèmes temps-réel consistent à les vérifier *formellement*. Ceci consiste à spécifier les besoins ou les propriétés du système à implémenter ainsi que le système lui-même en utilisant un langage de spécification non ambigu (ou *formel*). Une fois que ces spécifications sont écrites, on peut vérifier si le système satisfait ou non les besoins spécifiés ou propriétés attendues en utilisant un certain nombre de *méthodes* et *outils formels*.

En dernier lieu vient la surveillance ou le monitoring (on parle parfois du *runtime monitoring*) qui peut jouer un rôle important pour détecter éventuellement des comportements inattendus du système temps-réel au moment de son mise en œuvre. Ceci peut être dû à la non considération de certains évènements ou actions au moment de la modélisation du système ou bien c'est le résultat de poser de simples hypothèses au début. Même si la système répond aux différentes exigences de sûreté et aux contraintes,

le monitoring peut fournir davantage d'information et donner des idées pouvant accroitre la performance et la fiabilité du système surveillé.

Les techniques citées précédemment sont complémentaires du fait que chacune des sus-citées intervient dans telle ou telle étape de développement du système temps-réel. Nous nous intéressons dans notre contexte de la technique de vérification. Pour se faire, la vérification formelle est précédée par l'étape de spécification à la fois du système à vérifier et les propriétés que l'on attend de ce dernier.

2.2 Spécification et vérification formelle

Comme nous l'avons déjà cité, la vérification formelle, qui est une étape très importante dans une approche de conception formelle, nécessite la présence de plusieurs ingrédients :

1. La spécification des besoins que l'on attend de notre futur système temps-réel. On exige en général de notre système qu'il vérifie certaines *propriétés* qui peuvent être classées en propriétés de sûreté et des propriétés de vivacité. La sûreté veut dire que quelque chose de mauvais ne doit jamais arriver tandis que les propriétés de vivacité stipulent que quelques chose de bon doit toujours avoir lieu. Des propriétés de vivacité on peut dégager la classe des propriétés d'équité qui ajoutent la notion d'égalité de chances entre les différents agents ou processus du futur système ; on peut citer l'exemple de l'accès à une ressource partagée où nous devons imposer une équité entre plusieurs processus.

   Il existe dans la littérature plusieurs formalismes pour exprimer les propriétés attendues d'un système, allant des logiques temporelles [CE81, CES86, EL86, Eme90] aux automates et systèmes de transitions [AD90, AD94, Arn92]. L'avantage des logiques temporelles par rapport à d'autres formalismes réside dans l'utilisation d'opérateurs ayant un sens très proche de la perception humaine comme *toujours*, *parfois*, *jusqu'à*, etc.

2. La spécification de notre futur système qu'on veut réaliser. Cela consiste à modéliser son comportement global en utilisant ce qu'on appelle un *modèle formel de spécification*. Nous voulons simplement dire par modèle *formel* tout modèle ayant une syntaxe et une sémantique bien définie. Nous pouvons citer, parmi les modèles qui existent dans la littérature :

   - Les algèbres de processus, ou les FDT (*Formal Description Techniques*) qui

sont des langages de spécification de haut niveau pour les systèmes réactifs en général, c'est-à-dire les systèmes qui opèrent au milieu d'un environnement et leurs actions effectuées sont choisies en fonction du comportement de leur environnement. On peut citer parmi les algèbres de processus CSP (*Communicating Sequential Processes*) [Hoa85], CCS (*Calculus of Communicating Systems*) [Mil89], ACP (*Algebra of Communicating Processes*) [BK85] et LOTOS (*Language Of Temporal Ordering Specifications*) [BB87, ISO88]. Nous pouvons citer aussi quelques unes de leurs extensions temporelles et temporisées utilisées principalement dans le cadre de la spécification des contraintes temporelles (comme le cas dans systèmes temps-réel) : TCCS (*Timed CCS*) [MT90], RT-LOTOS [CdS93b, CSLO00], D-LOTOS [SC03], etc.

- Les réseaux de Petri [Pet62] avec leurs variantes et extensions comme les réseaux de Petri temporels (t-temporels [Mer74], p-temporels [Kha97]) et les réseaux de Petri temporisés [Ram74].

  Malgré que le modèle de base (réseaux de Petri places/transitions) est prouvé par exemple moins expressif que le langage LOTOS (voir [BB93]), les réseaux de Petri sont adoptés par plusieurs concepteurs à cause de leur représentation graphique élégante, claire et concise des systèmes modélisés, ainsi que leur prise en considération intrinsèque du vrai parallélisme, c'est-à-dire la possibilité de spécifier plusieurs actions s'exécutant simultanément.

- Les automates temporisés [AD90, AD94] et leur large ensemble de sous-classes et d'extensions. Ce modèle est très utilisé dans la littérature pour la spécification et la vérification formelle des systèmes temps-réel ou, plus précisément, tout système ayant un ensemble de contraintes temporelles. Un automate temporisé est un automate classique sur lequel on ajoute un certain nombre de variables réelles appelées *horloges* en plus d'un ensemble de contraintes écrites en fonction de ces horloges. Dans certaines définitions, on ajoute un ensemble d'*invariants* définies sur les états d'un automate temporisé et qui doivent être satisfaites pour que le système puisse résider dans un certain état.

  Il faut citer que les automates temporisés ainsi que leurs extensions sont actuellement largement utilisés afin de spécifier et vérifier des systèmes temps-réel, que ce soit pour des exemples académiques ou de systèmes industriels. On peut citer plusieurs outils et logiciels basés sur le formalisme

des automates temporisés : Uppaal[1] [LPY97], Kronos[2] [Yov97] et HyTech[3] [HHWT97].

Nous pouvons ajouter à ces modèles sus-cités, la logique de réécriture [Mes92], ainsi que le formalisme du Duration Calculus [CHR91, Cha98].

3. A partir de la spécification du système, on génère totalement ou en partie un modèle de plus bas niveau appelé *modèle sémantique* exprimant la sémantique (du parallélisme) du modèle de spécification. C'est sur ce dernier que la vérification formelle s'effectue principalement. Le modèle sémantique doit comporter toutes les traces (ou une partie) du comportement du système, incluant les exécutions séquentielles et parallèles possibles. Il est important de citer qu'il existe différentes approches adoptées dans la littérature pour exprimer la potentialité d'avoir des processus ou des actions s'exécutant en parallèle. L'approche la plus simple est dans laquelle on exprime l'exécution parallèle de deux actions par leurs exécutions alternées ou *entrelacées*, d'où la définition de modèles sémantique dits *modèles d'entrelacement* (*interleaving models*). Une deuxième approche a la vocation d'exprimer ce parallélisme de manière naturelle ; ceci est réalisé en ne représentant pas forcément les exécutions parallèles d'actions comme séquentielles mais plutôt d'intégrer directement les informations sur le parallélisme dans le modèle sémantique. Par conséquent, nous aurons ce qu'on appelle *modèles de non entrelacement*.

   Le plus connu des modèles sémantiques d'entrelacement est sans doute le modèle des *systèmes de transitions étiquetées* (ou STE) [Arn92]. Ces derniers seront formalisés par la Définition 3.11. On peut citer aussi dans cette catégorie de modèles sémantiques les *arbres de synchronisation* [Mil80].

   Il faut noter que les modèles d'entrelacement ont été critiqués pour diverses raisons. Nous pouvons citer entre autres le fait qu'il est indispensable de faire l'hypothèse d'atomicité temporelle et structurelle des actions, c'est-à-dire que les actions doivent être de durée nulle et indivisible. Tout ceci est pour utiliser ces modèles à bon escient. Par ailleurs, dans de nombreuses applications, l'atomicité des actions pourra être une abstraction acceptable, ce qui rend la simplicité de ces modèles très attirante.

   Au contraire des modèles d'entrelacement, les modèles de non entrelacement permettent de décrire le parallélisme entre les occurrences des actions. Leur intérêt

[1] `http://www.uppaal.com/`

[2] `http://www-verimag.imag.fr/TEMPORISE/kronos`

[3] `http://www-cad.eecs.berkeley.edu/~tah/HyTech/`

majeur est la considération d'actions non atomiques. L'utilisation de ces modèles en tant que domaines sémantiques pour les modèles de spécification permet la conception des systèmes par raffinement successif d'actions. Ça consiste à remplacer une action par un processus ou à la réécriture de spécifications. Les modèles de non entrelacement peuvent être classés en deux catégories : les modèles d'ordre partiel, et les modèles du vrai parallélisme.

On peut citer comme modèles d'ordre partiel, les arbres causaux [DD89] et les systèmes de transitions étiquetées causaux [Coe93]. Pour les modèles du vrai parallélisme, nous pouvons citer le modèle des réseaux de Petri, qui peut être considéré à fois un modèle de spécification et un modèle sémantique, les structures d'évènements [Win87] et leurs extensions, les arbres de synchronisation [Win84] et les arbres maximaux [Saï96]. Une version à états finis et implémentable des arbres maximaux (les systèmes de transitions étiquetées maximales) sera introduite dans la Section 3.3. Ce dernier modèle est basé sur le principe de la sémantique de maximalité ; cette dernière est introduite dans la Section 2.3.

On note que dans [dV89], il a été montré que les sémantiques d'ordre partiel ne sont ni nécessaires ni suffisantes pour la levée de l'hypothèse d'atomicité d'actions.

4. Le dernier point étant la technique ou la méthodes de vérification. Il existe plusieurs classes divisées selon le moyen utilisé pour exprimer les propriétés attendues du système. On distingue deux grandes classes : celle basée sur l'approche comportementale et celle basée sur l'approche logique. Dans l'approche comportementale, la description du système (son graphe de comportement) ainsi que sa spécification sont tous les deux exprimés par des comportements. La procédure de décision revient alors à décider de l'équivalence entre deux graphes. De nombreuses relations d'équivalence ont été proposées pour l'analyse des systèmes concurrents, allant de l'équivalence langage à l'équivalence observationnelle, en passant par les équivalences de test.

   Dans cette approche, les propriétés attendues du système sont exprimées par des assertions dans une logique, par exemple la logique temporelle. Dans ce contexte on distingue deux approches : une basée sur la preuve de théorèmes (*theorem proving*) et une autre basée sur le contrôle de modèle (*model checking*).

2.3 Sémantique de maximalité

L'étude d'un système de manière formelle nécessite généralement la donnée d'une sémantique précise aux langages et modèles de spécification utilisés. Dans le besoin de préserver la nature des systèmes à spécifier, nous nous intéressons aux sémantiques dites de *vrai parallélisme*. Ces sémantiques nous offrent la possibilité de spécifier, dans un contexte de haut niveau, le comportement des systèmes à étudier tout en s'intéressant à la concurrence.

Dans cette section, nous présentons une sémantique de vrai parallélisme des modèles de spécification appelée *sémantique de maximalité*.

Le choix de cette sémantique est poussé par le fait qu'elle est prouvée nécessaire et suffisante pour le raffinement d'actions et les durées d'actions [Saï96], c'est-à-dire que deux systèmes de transitions étiquetées maximales sur la maximalité équivalents (vis-à-vis de la relation de bisimulation maximale qui sera définie par la Définition 3.14) restent équivalents même après le raffinement d'une ou plusieurs actions en processus ou même après le changement des durées correspondantes aux actions.

Avant d'introduire la maximalité, nous citons la sémantique d'entrelacement qui a séduit les chercheurs pour plusieurs raisons. La première est la facilité qu'elle offre pour exprimer des comportements concurrents même en considérant des actions structurellement et temporellement *atomiques*. Cela est réalisé soit en éclatant les actions qui durent dans le temps en deux évènements atomiques qui correspondent au début et à la fin d'exécution des actions, soit en faisant totalement abstraction des durées non nulles en les considérant comme des actions atomiques.

La deuxième raison derrière l'adoption de la sémantique d'entrelacement par une grande partie de la communauté scientifique est le nombre croissant de systèmes ayant des milliers, voire des millions d'états, qui ont été vérifiés formellement par les différentes approches de vérification formelle dans des temps assez réduits. Nous pouvons citer l'approche de vérification par BDDs (binary decision diagrams) [McM92] où on arrivait à vérifier des systèmes de tailles pouvant dépasser 10^{20} états [BCM+92].

Pour utiliser à bon escient la sémantique d'entrelacement, l'hypothèse que les actions sont structurellement et temporellement atomiques est admise. Cette hypothèse nous semble forte et contraignante du fait que dans la pratique, les systèmes ont des actions ayant généralement des durées pas nécessairement nulles et peuvent être raffinées en plusieurs instructions indivisibles. Cela peut être perçu dans les méthodologies de conception formelle basées sur le raffinement d'actions.

Afin de clarifier ces idées, prenons quelques exemples pour lesquels la sémantique

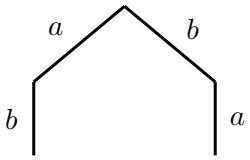

FIGURE 2.1 – Arbre de dérivations associé aux deux expressions E et F

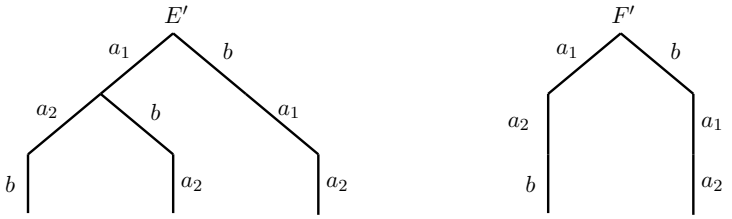

FIGURE 2.2 – Arbres de dérivations associés aux deux expressions E' et F'

d'entrelacement n'est pas satisfaisante.

Soient les comportements E et F de deux systèmes. Le premier système se compose en deux sous-systèmes concurrents offrant respectivement les actions a et b, tandis que le deuxième système exécute en séquence a puis b ou bien b puis a et s'arrête. E et F peuvent être représentés respectivement par les deux expressions de comportement définies par $E = a; stop \,|||\, b; stop$ et $F = a; b; stop \,[]\, b; a; stop$.

Dans la sémantique d'entrelacement, E et F sont fortement bisimilaires, c'est-à-dire qu'elles sont équivalentes en vue de la relation de bisimulation forte. En effet, ces deux expressions ont le comportement représenté par la Figure 2.1.

Si maintenant l'action a est supposée un processus qui exécute en séquence deux actions a_1 et a_2, les expressions de comportement E et F sont respectivement équivalentes aux expressions $E' = a_1; a_2; stop \,|||\, b; stop$ et $F' = a_1; a_2; b; stop \,[]\, b; a_1; a_2; stop$.

Les comportements E' et F' sont représentés par la Figure 2.2.

En examinant ces comportements, nous pouvons constater que la dérivation suivante est possible à partir de E' :

$$E' \xrightarrow{a_1}_i E" \equiv a_2; stop \,|||\, b; stop$$

alors que la seule dérivation possible de l'action a_1 à partir de l'expression de comportement F' est

$$F' \xrightarrow{a_1}_i F" \equiv a_2; b; stop$$

Il est clair qu'à partir de $E"$, l'occurrence de l'action b est toujours possible, ce qui n'est pas le cas à partir de $F"$, cela montre que les expressions de comportement E' et F'

ne sont pas fortement bisimilaires. Or, l'opération de raffinement n'a fait que montrer la non atomicité de l'action a, ce qui implique que les expressions de comportement E et F ne devraient pas être identifiées comme étant fortement bisimilaires ; par conséquent, la sémantique d'entrelacement a identifié des processus qui en réalité ne le sont pas, ce qui veut dire que l'entrelacement ne peut pas être utilisé comme sémantique des systèmes pour lesquels les actions ne sont pas atomiques.

Une autre façon de distinguer les expressions de comportement E et F est de supposer que les actions ont des durées non nulles. Soient $\tau(a) > 0$ et $\tau(b) > 0$ les durées respectives des actions a et b. Nous pouvons voir que dans E l'exécution des actions a et b peut être faite dans un temps égal à $max\{\tau(a), \tau(b)\}$, alors que le temps minimal pour exécuter les deux actions a et b dans F est de $\tau(a) + \tau(b)$.

Pour s'échapper à l'hypothèse d'atomicité des actions imposée par la sémantique d'entrelacement, de nouvelles sémantiques de parallélisme ont été définies dans la littérature. Nous pouvons citer les sémantiques d'ordre partiel dont le principe consiste à considérer les relations de dépendances entre les occurrences d'actions. Cependant, dans [vG89], il a été montré que ces sémantiques ne sont ni nécessaires ni suffisantes pour la levée de l'hypothèse d'atomicité des actions.

Comme alternative aux sémantiques d'ordre partiel, nous trouvons la ST-sémantique[4] qui consiste à diviser chaque action en deux actions atomiques s'exécutant en séquence et qui représentent respectivement le début et la fin de l'action [vV87].

La sémantique de maximalité peut être considérée comme étant la réconciliation des sémantiques d'ordre partiel et des ST-sémantiques [Vog91, Dev92a]. Le principe de cette sémantique consiste à utiliser les relations de dépendance entre les occurrences d'actions pour associer à chaque état du système les actions qui sont potentiellement (éventuellement) en cours d'exécution. Dans ce qui suit, nous donnons une présentation intuitive de la sémantique de maximalité.

Principe de la sémantique de maximalité

La sémantique d'un système concurrent peut être caractérisée par l'ensemble des états possibles du système et les transitions par lesquelles le système change d'un état à un autre. Les différentes sémantiques diffèrent par le sens associé aux états et aux transitions. Par exemple, dans le cas de la sémantique d'entrelacement, les transitions sont des évènements qui correspondent à l'exécution entière des actions associées aux transitions.

[4]ST signifie *Place et Transition*

Dans l'approche basée sur la maximalité, les transitions sont des évènements qui ne représentent que le début de l'exécution des actions. Par conséquent, l'exécution concurrente de plusieurs actions devient possible, c'est-à-dire qu'intuitivement on peut distinguer exécutions séquentielles et exécutions parallèles d'actions. Or, étant donné que plusieurs actions qui ont le même nom peuvent s'exécuter en parallèle (*auto-concurrence*), nous associons, pour distinguer les exécutions de chacune de ces actions, un identificateur à chaque début d'exécution d'action.

Dans un état, un évènement est dit *maximal* s'il correspond au début de l'exécution d'une action qui peut éventuellement être toujours en train de s'exécuter dans cet état.

Associer des noms d'évènements maximaux aux états nous conduit à la notion de configuration qui sera formalisée par la Définition 3.1.

Pour illustrer la maximalité et cette notion de configuration, considérons l'expression de comportement $G = a; c; d; stop \;||[c]||\; b; c; e; stop$ exprimant deux sous-systèmes s'exécutant en concurrence et se synchronisant sur une action c.

La configuration initiale de G est notée ${}_{\emptyset}[G]$, ce qui exprime le fait que dans l'état initial, l'ensemble des évènements maximaux est vide puisque aucune action n'a encore commencé son exécution.

Dans cette configuration, l'action a est sensibilisée ; soit x le nom de l'évènement associé au début de l'exécution de a, ce qui implique la dérivation suivante :

$$ {}_{\emptyset}[G] \xrightarrow{{}^{\emptyset}a_x}_m \quad {}_{\{x\}}[c; d; stop] \;\; ||[c]|| \;\; {}_{\emptyset}[b; c; e; stop] $$

Le triplet ${}_{M}a_x$, où M est un ensemble de noms d'évè nements, est appelé un *atome*.

La configuration dérivée représente un état dans lequel l'évolution future de la composante à gauche de l'opérateur $||[c]||$ dépend de la terminaison de l'action identifiée par x (dans ce cas, c'est l'action a), alors que la composante à droite peut commencer son exécution de manière indépendante. Comme l'action c ne peut commencer son exécution que si les deux composantes se synchronisent sur cette action c, la seule dérivation possible à partir de cet état là est donc :

$$ {}_{\{x\}}[c; d; stop] \;\; ||[c]|| \;\; {}_{\emptyset}[b; c; e; stop] \xrightarrow{{}^{\emptyset}b_y}_m \;\; {}_{\{x\}}[c; d; stop] \;\; ||[c]|| \;\; {}_{\{y\}}[c; e; stop] $$

Dans cette dérivation, le début de l'action b est identifié par y. Dans la configuration résultante, x et y identifient deux évènements maximaux correspondant respectivement au début des actions a et b.

A partir de cet état, la synchronisation sur l'action c est possible dès que les deux

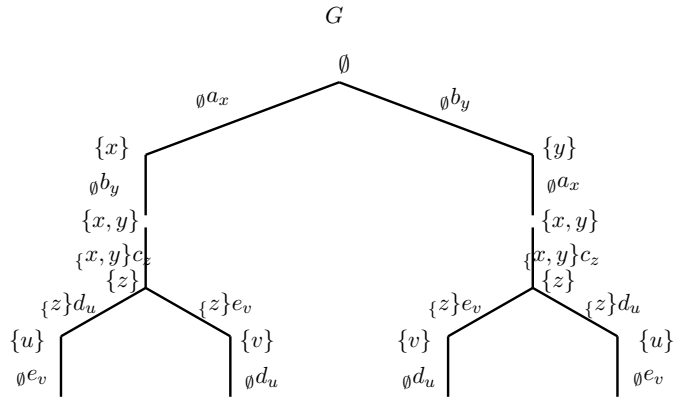

FIGURE 2.3 – Arbre de dérivations basé sur la maximalité

actions a et b ont terminé leur exécution. En supposant que le début de l'exécution de l'action c est identifié par le nom d'évènement z, nous obtenons la dérivation suivante :

$$_{\{x\}}\,[c; d; stop] \;\; |[c]| \;\; _{\{y\}}\,[c; e; stop] \xrightarrow{\{x,y\}c_z}_m \;\; _{\{z\}}\,[d; stop] \;\; |[c]| \;\; _{\{z\}}\,[e; stop]$$

Dans la configuration résultante, il est clair que z est le nom du seul évènement maximal ; la dépendance causale entre l'exécution de l'action c et l'exécution des actions a et b est exprimée par $_{\{x,y\}}c_z$.

A partir de cet état, les actions d et e peuvent commencer leur exécution indépendamment aussitôt que l'action c est terminée. Une fois que l'une de ces actions commence son exécution, l'évènement identifié par z ne reste plus maximal. En supposant que le début de l'exécution de l'action d est identifié par u, la transition suivante devient possible :

$$_{\{z\}}\,[d; stop] \;\; |[c]| \;\; _{\{z\}}\,[e; stop] \xrightarrow{\{z\}d_u}_m \;\; _{\{u\}}\,[stop] \;\; |[c]| \;\; _{\emptyset}\,[e; stop]$$

En appliquant cette approche de maximalité, l'arbre de toutes les dérivations possibles associé à l'expression de comportement G est donné par la Figure 2.3.

Pour voir maintenant comment la sémantique de maximalité distingue des comportements identifiés par la sémantique d'entrelacement, nous reprenons les deux expressions de comportement $E = a; stop \,|||\, b; stop$ et $F = a; b; stop \,[]\, b; a; stop$.

En appliquant le principe de la sémantique de maximalité, les transitions suivantes sont possibles :

$$_{\emptyset}\,[E] \xrightarrow{\emptyset a_x}_m \;\; _{\{x\}}\,[stop] \;\; ||| \;\; _{\emptyset}\,[b; stop] \xrightarrow{\emptyset b_y}_m \;\; _{\{x\}}\,[stop] \;\; ||| \;\; _{\{y\}}\,[stop]$$

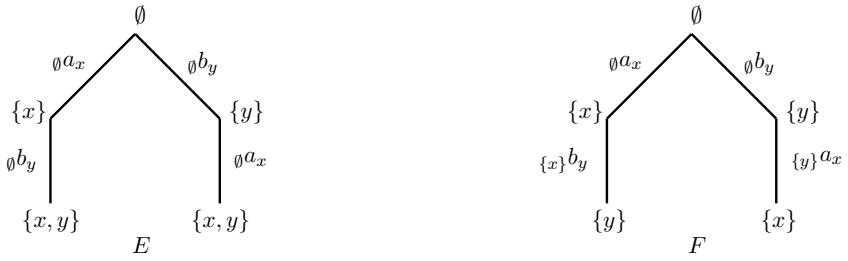

FIGURE 2.4 – Arbres de dérivations de E et F

x (respectivement y) est le nom de l'évènement identifiant le début de l'action a (respectivement b). Puisque rien ne peut être conclu à propos de la terminaison des deux actions a et b dans la configuration ${}_{\{x\}}[stop] \;|||\; {}_{\{y\}}[stop]$, x et y sont alors tous les deux maximaux dans cette configuration. Notons que x est également maximal dans l'état intermédiaire représenté par la configuration ${}_{\{x\}}[stop] \;|||\; {}_{\emptyset}[b; stop]$.

Pour la configuration initiale, associée à l'expression de comportement F, la transition suivante est possible :

$$ {}_{\emptyset}[F] \xrightarrow{\emptyset a_x}_m \; {}_{\{x\}}[b; stop] $$

Comme précédemment, x identifie le début de l'action a et il est le nom du seul évènement maximal dans la configuration ${}_{\{x\}}[b; stop]$. Il est clair que suivant la sémantique de l'opérateur de préfixage $\ll ; \gg$, le début de l'exécution de l'action b n'est possible que si l'action a termine son exécution. Par conséquent, x ne reste plus maximal lorsque l'action b commence son exécution ; l'unique évènement maximal dans la configuration résultante est celui identifié par y qui correspond au début de l'exécution de l'action b. L'ensemble des noms des évènements maximaux a donc été modifié par la suppression de x et l'ajout de y, ce qui justifie la dérivation

$$ {}_{\{x\}}[b; stop] \xrightarrow{\{x\} b_y}_m \; {}_{\{y\}}[stop] $$

La configuration ${}_{\{y\}}[stop]$ est différente de la configuration ${}_{\{x\}}[stop] \;|||\; {}_{\{y\}}[stop]$, car la première configuration ne possède qu'un seul évè nement maximal identifié par y, alors qu'il y en a deux dans la deuxième configuration (identifiés par x et y).

Les arbres de dérivation des expressions de comportement E et F obtenus par l'application de la maximalité sont représentés par la Figure 2.4.

Nous notons qu'une sémantique de maximalité est attribuée pour les langages de spécification Basic LOTOS (Définition 3.5), D-LOTOS (voir Section 3.1.2) et pour les

réseaux de Petri place/transition (voir Section 6.3.2).

2.4 Conclusion

Ce chapitre introduisait quelques notions de base sur la spécification et la vérification formelle de systèmes réactifs en général et temps-réel en particulier.

Dans la deuxième partie, nous avons donné, informellement, le principe de la sémantique de maximalité. C'est cette sémantique qui sera adoptée par la suite tout au long de notre travail vu les avantages qu'elles offre. Ceci se résume principalement dans l'expression naturelle de comportement comportant des exécutions parallèles d'actions concurrentes. Et si on note que plusieurs, si on ne dit pas la plupart, des systèmes temps-réel sont constitués de modules ou composants d'exécutant en concurrence et se synchronisant sur des points de rendez-vous, nous pouvons rapidement percevoir l'intérêt d'utiliser ce type de sémantique.

Première partie

Outils formels

Chapitre 3

Modèles

Dans ce chapitre, nous introduisons, avec plus ou moins de détails, quelques modèles ayant une relation de près ou de loin avec les résultats de notre travail de thèse, à savoir les algèbres de processus Basic LOTOS et D-LOTOS, les réseaux de Petri place/transition, les systèmes de transitions étiquetées maximales et les automates temporisés.

3.1 Algèbres de processus

Une *algèbre de processus* est un langage concis pour la description des étapes d'exécution de processus. Ce langage est constitué d'un ensemble d'opérateurs et de règles syntaxiques pour spécifier un processus utilisant de composants simples ou atomiques. Les algèbres de processus utilisent la notion d'équivalence pour montrer que deux processus ont le même comportement. Plusieurs algèbres de processus ont été utilisées pour spécifier et analyser des processus concurrents qui communiquent entre eux. Nous pouvons citer CSP (*Communicating Sequential Processes*) de Hoare [Hoa85], CCS (*Calculus of Communicating Systems*) de Milner [Mil89], ACP (*Algebra of Communicating Processes*) de Bergstra et Klop [BK85] et LOTOS (*Language Of Temporal Ordering Specifications*) [BB87, ISO88].

Ces algèbres de processus citées précédemment sont des algèbres atemporelles du fait qu'elles ne permettent que de raisonner sur l'ordre d'exécution des évènements et étapes d'exécution. Il existe plusieurs algèbres de processus conçues pour exprimer le temps d'une manière ou d'une autre, nous passons en revue par exemple tout au long de ce chapitre quelques extensions temporelles et/ou temporisées de LOTOS. Nous donnerons la syntaxe et la sémantique de maximalité de l'extension D-LOTOS [SC03], un langage de haut niveau supportant à la fois les contraintes temporelles, les durées

d'actions et les contraintes d'urgence.

Généralement, pour utiliser une approche basée sur les algèbres de processus afin de spécifier et analyser un système, nous traduisons en premier lieu la spécification des besoins d'un système en un processus abstrait P_1 de haut niveau, et la description de conception en un processus plus détaillé P_2. Nous montrons alors que ces deux processus P_1 et P_2 sont *équivalents*, c'est-à-dire que la description de la conception est *correcte* par rapport à la spécification des besoins.

Une algèbre de processus a quatre éléments de base :

1. un langage concis pour spécifier un système comme un processus ou un ensemble de processus,

2. une sémantique non ambigüe offrant un sens précis au comportement des processus spécifiés et montrant les étapes d'exécution possibles de ces processus,

3. une relation de pré-ordre ou d'équivalence pour comparer les comportements de ces processus, et

4. un ensemble de règles de manipulation des spécifications de processus.

Dans le but de spécifier et vérifier les systèmes temps-réel, les algèbres de processus non temporisées ont été étendues par la notion de temps par l'ajout d'opérateurs temporels à l'ensemble d'opérateurs conventionnels. Par conséquent, plusieurs algèbres de processus temporisées ont vu le jour comme résultat de cette extension.

Dans ce qui suit, nous commençons par donner la syntaxe formelle, la sémantique d'entrelacement et la sémantique de maximalité de l'algèbre de processus LOTOS. Ce langage a été standardisé par ISO [ISO88].

3.1.1 Le langage Basic LOTOS

LOTOS (*Language of Temporal Ordering Specification*) a été promu au rang de norme ISO en 1988 [ISO88]. C'est une algèbre de processus permettant d'exprimer la structure logique et temporelle de comportements. Il s'appuie sur le langage CCS de Milner (étendu par un mécanisme de synchronisation multiple hérité de CSP de Hoare) pour la spécification de la partie comportementale ; la partie description des structures de données est inspirée de Act-One [EM85], un formalisme de description des types de données abstraits algébriques. LOTOS est un langage asynchrone, avec synchronisation par rendez-vous multiples. Les processus offrent des synchronisations à leur environnement au travers de *portes* de communication (ou *actions*).

Nous appelons *Basic LOTOS* le sous-ensemble de LOTOS où les processus interagissent entre eux par synchronisation pure, sans échange de valeurs. En Basic LOTOS, les actions sont identiques aux portes de synchronisation des processus. Les principaux opérateurs de Basic LOTOS sont listés dans le Tableau 3.1.

Syntaxe formelle de Basic LOTOS

Soit PN l'ensemble des variables de processus parcouru par X et soit $\mathcal{G}$ l'ensemble des noms de portes définies par l'utilisateur (ensemble des actions observables) parcouru par g. Une porte observable particulière $\delta \notin \mathcal{G}$ est utilisée pour notifier la terminaison avec succès des processus. L dénote tout sous-ensemble de $\mathcal{G}$, l'action interne est désignée par i. $\mathcal{B}$ parcouru par E, F, ... dénote l'ensemble des expressions de comportement dont la syntaxe est :

$$E ::= \texttt{stop} \mid \texttt{exit} \mid X[L] \mid g;E \mid i;E \mid E[]E \mid E|[L]|E \mid hide\ L\ in\ E \mid E \gg E \mid E\ [>\ E$$

Etant donné un processus dont le nom est P et dont le comportement est E, la définition de P est exprimée par $P := E$. L'ensemble de toutes les actions est désigné par Act ($Act = \mathcal{G} \cup \{i, \delta\}$).

Sémantique d'entrelacement de Basic LOTOS

Nous donnons dans ce qui suit la sémantique opérationnelle structurée (SOS) d'entrelacement de Basic LOTOS à la Plotkin [Plo81].

Définition 3.1 *La sémantique opérationnelle structurée d'entrelacement du langage Basic LOTOS est donnée par l'ensemble des règles d'inférence suivantes :*

1. $\dfrac{}{exit \xrightarrow{\delta} stop}$
2. $\dfrac{}{a;E \xrightarrow{a} E}$
3. (a) $\dfrac{E \xrightarrow{a} E'}{E \ [] \ F \xrightarrow{a} E'}$
   (b) $\dfrac{E \xrightarrow{a} E'}{F \ [] \ E \xrightarrow{a} E'}$
4. (a) i. $\dfrac{E \xrightarrow{a} E' \quad a \notin L \cup \{\delta\}}{E \ |[L]| \ F \xrightarrow{a} E' \ |[L]| \ F}$
   ii. $\dfrac{E \xrightarrow{a} E' \quad a \notin L \cup \{\delta\}}{F \ |[L]| \ E \xrightarrow{a} F \ |[L]| \ E'}$
   (b) $\dfrac{E \xrightarrow{a} E' \quad F \xrightarrow{a} F' \quad a \in L \cup \{\delta\}}{F \ |[L]| \ E \xrightarrow{a} F' \ |[L]| \ E'}$

Opérateur		**Notation**	**Description informelle**
Inaction		`stop`	Processus de base n'interagissant pas avec son environnement
Terminaison avec succès		`exit`	Processus qui se termine (action δ) et se transforme en `stop`
Préfixage par une action	non observable	$i; P$	Processus qui réalise l'action i ou g, puis se transforme en P
	observable	$g; P$	
Choix non-déterministe		$P_1 [] P_2$	Processus qui se transforme en P_1 ou en P_2 suivant l'environnement
Composition parallèle	cas général	$P_1 \|[g_1, \ldots, g_n]\| P_2$	P_1 et P_2 s'exécutent en parallèle et se synchronisent sur les portes $g_1, ...g_n$ et δ
	asynchrone	$P_1 \|\|\| P_2$	P_1 et P_2 s'exécutent en parallèle sans se synchroniser (sauf sur δ)
	synchrone	$P_1 \|\| P_2$	P_1 et P_2 s'exécutent en parallèle et se synchronisent sur chaque porte visible
Intériorisation		*hide* $g_1, ...g_n$ *in* P	Les actions $g_1, ...g_n$ sont cachées à l'environnement de P et deviennent des actions internes
Composition séquentielle		$P_1 \gg P_2$	P_2 est activé dès que P_1 se termine.
Préemption (interruption)		$P_1 [> P$	P_2 peut interrompre P_1 tant que P_1 ne s'est pas terminé

TABLE 3.1 – *Principaux opérateurs de Basic LOTOS*

5. (a) $\frac{E \xrightarrow{a} E' \quad a \notin L}{hide\ L\ in\ E \xrightarrow{a} hide\ L\ in\ E'}$

   (b) $\frac{E \xrightarrow{a} E' \quad a \in L}{hide\ L\ in\ E \xrightarrow{i} hide\ L\ in\ E'}$

6. (a) $\frac{E \xrightarrow{a} E' \quad a \neq \delta}{E \gg F \xrightarrow{a} E' \gg F}$

   (b) $\frac{E \xrightarrow{\delta} E'}{E \gg F \xrightarrow{i} F}$

7. (a) $\frac{E \xrightarrow{a} E' \quad a \neq \delta}{E\ [>\ F \xrightarrow{a} E'\ [>\ F}$

   (b) $\frac{E \xrightarrow{\delta} E'}{E\ [>\ F \xrightarrow{\delta} E'}$

   (c) $\frac{F \xrightarrow{a} F'}{E\ [>\ F \xrightarrow{a} F'}$

8. (a) $\frac{E \xrightarrow{a} E' \quad a \notin \{a_1,...,a_n\}}{E[b_1/a_1,...,b_n/a_n] \xrightarrow{a} E'[b_1/a_1,...,b_n/a_n]}$

   (b) $\frac{E \xrightarrow{a} E' \quad a = a_i\ (1 \leq i \leq n)}{E[b_1/a_1,...,b_n/a_n] \xrightarrow{b_i} E'[b_1/a_1,...,b_n/a_n]}$

9. $\frac{P := E \quad E \xrightarrow{a} E'}{P \xrightarrow{a} E'}$

Sémantique de maximalité de Basic LOTOS

Afin de donner une sémantique de maximalité aux expressions du langage Basic LOTOS, nous avons besoin d'introduire la notion de configurations.

Définition 3.2 *L'ensemble des noms des évènements est un ensemble dénombrable noté $\mathcal{M}$. $\mathcal{M}$ est parcouru par $x, y, \ldots$. $M, N, \ldots$ dénotent des sous-ensembles finis de $\mathcal{M}$. L'ensemble des atomes de support Act est $Atm = 2^{\mathcal{M}}_{fn} \times Act \times \mathcal{M}$, $2^{\mathcal{M}}_{fn}$ étant l'ensemble des parties finies de $\mathcal{M}$. Pour $M \in 2^{\mathcal{M}}_{fn}$, $x \in \mathcal{M}$ et $a \in Act$, l'atome (M, a, x) sera noté ${}_M a_x$. Le choix d'un nom d'évènement peut se faire de manière déterministe par l'utilisation de toute fonction get : $2^{\mathcal{M}} - \{\emptyset\} \to \mathcal{M}$ satisfaisant $get(M) \in M$ pour tout $M \in 2^{\mathcal{M}} - \{\emptyset\}$.*

Définition 3.1 *L'ensemble $\mathcal{C}$ des configurations des expressions de comportement de Basic LOTOS est le plus petit ensemble défini par induction comme suit :*

- $\forall E \in \mathcal{B}, \forall M \in 2^{\mathcal{M}}_{fn} : {}_M[E] \in \mathcal{C}$
- $\forall P \in PN, \forall M \in 2^{\mathcal{M}}_{fn} : {}_M[P] \in \mathcal{C}$
- *si $\mathcal{E} \in \mathcal{C}$ alors hide L in $\mathcal{E} \in \mathcal{C}$*
- *si $\mathcal{E} \in \mathcal{C}$ et $F \in \mathcal{B}$ alors $\mathcal{E} \gg F \in \mathcal{C}$*

- *si* $\mathcal{E}, \mathcal{F} \in \mathcal{C}$ *alors* $\mathcal{E}$ *op* $\mathcal{F} \in \mathcal{C}$ $\qquad op \in \{ \ [] \ , \ ||| \ , \ || \ , \ |[L]| \ , \ [> \ \}$
- *si* $\mathcal{E} \in \mathcal{C}$ *et* $\{a_1, ..., a_n\}$, $\{b_1, ..., b_n\} \in 2^{\mathcal{G}}_{fn}$ *alors* $\mathcal{E}\,[b_1/a_1, ..., b_n/a_n] \in \mathcal{C}$

Etant donné un ensemble $M \in 2^{\mathcal{M}}_{fn}$, ${}_M[\cdots]$ est appelée opération d'*encapsulation*. Cette opération est distributive par rapport aux opérations $[], |[L]|$, *hide*, $[>$ et le renommage des portes. Nous admettons aussi que ${}_M[E \gg F] \equiv {}_M[E] \gg F$. Une configuration est dite canonique si elle ne peut plus être réduite par la distribution de l'opération d'encapsulation sur les autres opérateurs. Par la suite, nous supposons que toutes les configurations sont canoniques.

Proposition 3.1 [Saï96] *Toute configuration canonique est sous l'une des formes suivantes* ($\mathcal{E}$ *et* $\mathcal{F}$ *étant des configurations canoniques) :*

${}_M[stop]$	${}_M[exit]$	${}_M[a; E]$	${}_M[P]$	$\mathcal{E} \ [] \ \mathcal{F}$
$\mathcal{E} \ \|[L]\| \ \mathcal{F}$	$hide \ L \ in \ \mathcal{E}$	$\mathcal{E} \gg F$	$\mathcal{E} \ [> \ \mathcal{F}$	$\mathcal{E}\,[b_1/a_1, ..., b_n/a_n]$

Définition 3.2 *La fonction* $\psi : C \to 2^{\mathcal{M}}_{fn}$, *qui détermine l'ensemble des noms des événements dans une configuration, est définie récursivement par :*

$$\psi({}_M[E]) = M \qquad \psi(\mathcal{E} \ [] \ \mathcal{F}) = \psi(\mathcal{E}) \cup \psi(\mathcal{F}) \qquad \psi(\mathcal{E} \ |[L]| \ \mathcal{F}) = \psi(\mathcal{E}) \cup \psi(\mathcal{F})$$
$$\psi(\mathcal{E} \gg F) = \psi(\mathcal{E}) \qquad \psi(hide \ L \ in \ \mathcal{E}) = \psi(\mathcal{E}) \qquad \psi(\mathcal{E} \ [> \mathcal{F}) = \psi(\mathcal{E}) \cup \psi(\mathcal{F})$$
$$\psi(\mathcal{E}\,[b_1/a_1, ..., b_n/a_n]) = \psi(\mathcal{E})$$

Définition 3.3 *Soit* $\mathcal{E}$ *une configuration ;* $\mathcal{E}\backslash N$ *dénote la configuration obtenue par la suppression de l'ensemble des noms des événements* N *de la configuration* $\mathcal{E}$. $\mathcal{E}\backslash N$ *est définie récursivement sur la configuration* $\mathcal{E}$ *comme suit :*

$$({}_M[E])\backslash N = {}_{M-N}[E] \qquad (\mathcal{E} \ [] \ \mathcal{F})\backslash N = \mathcal{E}\backslash N \ [] \ \mathcal{F}\backslash N$$
$$(\mathcal{E} \ |[L]| \ \mathcal{F})\backslash N = \mathcal{E}\backslash N \ |[L]| \ \mathcal{F}\backslash N \qquad (hide \ L \ in \ \mathcal{E})\backslash N = hide \ L \ in \ \mathcal{E}\backslash N$$
$$(\mathcal{E} \gg F)\backslash N = \mathcal{E}\backslash N \gg F \qquad (\mathcal{E} \ [> \mathcal{F})\backslash N = \mathcal{E}\backslash N \ [> \mathcal{F}\backslash N$$
$$(\mathcal{E}\,[b_1/a_1, ..., b_n/a_n])\backslash N = \mathcal{E}\backslash N\,[b_1/a_1, ..., b_n/a_n]$$

Définition 3.4 *L'ensemble des fonctions de substitution des noms des événements est Subs (i.e.* $Subs = \mathcal{M} \to 2^{\mathcal{M}}_{fn}$*) ;* $\sigma, \sigma_1, \sigma_2, ...$ *désignent des éléments de Subs. Etant donnés* $x, y, z \in \mathcal{M}$ *et* $M \in 2^{\mathcal{M}}_{fn}$, *alors*

- *L'application de* σ *à* x *sera écrite* σx *;*
- *La substitution identité* ι *est définie par* $\iota x = \{x\}$ *;*
- $M\sigma = \cup_{x \in M} \ \sigma x$ *;*

- $\sigma\,[y/z]$ *est définie par :* $\sigma\,[y/z]\,x = \begin{cases} \{y\} & si\ z = x \\ \sigma x & sinon \end{cases}$

Soit σ une fonction de substitution, la substitution simultanée de toutes les occurrences de x dans $\mathcal{E}$ par σx, est définie récursivement sur la configuration $\mathcal{E}$ comme suit :

$(_M\,[E])\,\sigma = {}_{M\sigma}\,[E]$ $\qquad$ $(\mathcal{E}\ []\ \mathcal{F})\,\sigma = \mathcal{E}\sigma\ []\ \mathcal{F}\sigma$

$(\mathcal{E}\ |[L]|\ \mathcal{F})\,\sigma = \mathcal{E}\sigma\ |[L]|\ \mathcal{F}\sigma$ $\qquad$ $(hide\ L\ in\ \mathcal{E})\,\sigma = hide\ L\ in\ \mathcal{E}\sigma$

$(\mathcal{E} \gg F)\,\sigma = \mathcal{E}\sigma \gg F$ $\qquad$ $(\mathcal{E}\ [>\ \mathcal{F})\,\sigma = \mathcal{E}\sigma\ [>\ \mathcal{F}\sigma$

$(\mathcal{E}\,[b_1/a_1, ..., b_n/a_n])\,\sigma = \mathcal{E}\sigma\,[b_1/a_1, ..., b_n/a_n]$

Définition 3.5 *La relation de transition de maximalité $\longrightarrow \subseteq \mathcal{C} \times Atm \times \mathcal{C}$ est définie comme étant la plus petite relation satisfaisant les règles suivantes :*

1. $\dfrac{}{{}_M[exit] \xrightarrow{M^{\delta}_{x}} {}_{\{x\}}[stop]}\ x{=}get(\mathcal{M})$

2. $\dfrac{}{{}_M[a;E] \xrightarrow{M^{a}_{x}} {}_{\{x\}}[E]}\ x{=}get(\mathcal{M})$

3. (a) $\dfrac{\mathcal{E} \xrightarrow{M^{a}_{x}} \mathcal{E}'}{\mathcal{F}\ []\ \mathcal{E} \xrightarrow{M^{a}_{x}} \mathcal{E}' \qquad \mathcal{E}\ []\ \mathcal{F} \xrightarrow{M^{a}_{x}} \mathcal{E}'}$

4. (a) i. $\dfrac{\mathcal{E} \xrightarrow{M^{a}_{x}} \mathcal{E}' \qquad a \notin L \cup \{\delta\}}{\mathcal{E}\ |[L]|\ \mathcal{F} \xrightarrow{M^{a}_{y}} \mathcal{E}'[y/x]\ |[L]|\ \mathcal{F}\backslash M}\ y{=}get(\mathcal{M}{-}((\psi(\mathcal{E}){\cup}\psi(\mathcal{F})){-}M))$

   ii. $\dfrac{\mathcal{E} \xrightarrow{M^{a}_{x}} \mathcal{E}' \qquad a \notin L \cup \{\delta\}}{\mathcal{F}\ |[L]|\ \mathcal{E} \xrightarrow{M^{a}_{y}} \mathcal{F}\backslash M\ |[L]|\ \mathcal{E}'[y/x]}\ y{=}get(\mathcal{M}{-}((\psi(\mathcal{E}){\cup}\psi(\mathcal{F})){-}M))$

   (b) $\dfrac{\mathcal{E} \xrightarrow{M^{a}_{x}} \mathcal{E}' \qquad \mathcal{F} \xrightarrow{M^{a}_{y}} \mathcal{F}' \qquad a \in L \cup \{\delta\}}{\mathcal{E}\ |[L]|\ \mathcal{F} \xrightarrow{M \cup N^{a}_{z}} \mathcal{E}'[z/x]\backslash N\ |[L]|\ \mathcal{F}'[z/y]\backslash M}\ z{=}get(\mathcal{M}{-}((\psi(\mathcal{E}){\cup}\psi(\mathcal{F})){-}(M{\cup}N)))$

5. (a) $\dfrac{\mathcal{E} \xrightarrow{M^{a}_{x}} \mathcal{E}' \qquad a \notin L}{hide\ L\ in\ \mathcal{E} \xrightarrow{M^{a}_{x}} hide\ L\ in\ \mathcal{E}'}$

   (b) $\dfrac{\mathcal{E} \xrightarrow{M^{a}_{x}} {}_m\mathcal{E}' \qquad a \in L}{hide\ L\ in\ \mathcal{E} \xrightarrow{M^{i}_{x}} hide\ L\ in\ \mathcal{E}'}$

6. (a) $\dfrac{\mathcal{E} \xrightarrow{M^{a}_{x}} {}_m\mathcal{E}' \qquad a \neq \delta}{\mathcal{E} \gg F \xrightarrow{M^{a}_{x}} \mathcal{E}' \gg F}$

   (b) $\dfrac{\mathcal{E} \xrightarrow{M^{\delta}_{x}} \mathcal{E}'}{\mathcal{E} \gg F \xrightarrow{M^{i}_{x}} {}_{\{x\}}[F]}$

7. (a) $\dfrac{\mathcal{E} \xrightarrow{M^{a}_{x}} \mathcal{E}' \qquad a \neq \delta}{\mathcal{E}\ [>\ \mathcal{F} \xrightarrow{M^{a}_{y}} \mathcal{E}'[y/x]\ [>\ \mathcal{F}\backslash M}\ y{=}get(\mathcal{M}{-}(\psi(\mathcal{E}){\cup}\psi(\mathcal{F}){-}M))$

   (b) $\dfrac{\mathcal{E} \xrightarrow{M^{\delta}_{x}} \mathcal{E}'}{\mathcal{E}\ [>\ \mathcal{F} \xrightarrow{M^{\delta}_{y}} \mathcal{E}'[y/x]\ [>\ {}_{\psi(\mathcal{F})-M}[stop]}\ y{=}get(\mathcal{M}{-}(\psi(\mathcal{E}){\cup}\psi(\mathcal{F}){-}M))$

   (c) $\dfrac{\mathcal{F} \xrightarrow{M^{a}_{x}} \mathcal{F}'}{\mathcal{E}\ [>\ \mathcal{F} \xrightarrow{M^{a}_{y}} {}_{\psi(\mathcal{E})-M}[stop]\ [>\ \mathcal{F}'[y/x]}\ y{=}get(\mathcal{M}{-}(\psi(\mathcal{E}){\cup}\psi(\mathcal{F}){-}M))$

8. (a) $\dfrac{\mathcal{E} \overset{M^a_x}{\longrightarrow} \mathcal{E}' \qquad a \notin \{a_1,...,a_n\}}{\mathcal{E}[b_1/a_1,...,b_n/a_n] \overset{M^a_x}{\longrightarrow} \mathcal{E}'[b_1/a_1,...,b_n/a_n]}$

   (b) $\dfrac{\mathcal{E} \overset{M^a_x}{\longrightarrow} \mathcal{E}' \qquad a=a_i \ (1 \leq i \leq n)}{\mathcal{E}[b_1/a_1,...,b_n/a_n] \overset{M^{bi}_x}{\longrightarrow} \mathcal{E}'[b_1/a_1,...,b_n/a_n]}$

9. $\dfrac{P:=E \qquad {}_M[E] \overset{M^a_x}{\longrightarrow} \mathcal{F}}{{}_M[P] \overset{M^a_x}{\longrightarrow} \mathcal{F}}$

Principales extensions temporelles à LOTOS

La problématique de l'expression explicite du temps dans LOTOS a engendré une série d'approches différentes. Nous listons les principaux modèles ci-dessous :

- TIC-LOTOS (Quemada 1987, UPM - Espagne)
- LOTOS-T (Miguel 1992, UPM - Espagne)
- T-LOTOS et U-LOTOS (Bolognesi 1991, CNUCE - Italie)
- TLOTOS (Leduc 1992, ULG - Belgique)
- Time LOTOS et ET-LOTOS (Leduc, Leonard 1993-94, ULG - Belgique)
- TE-LOTOS (Leduc, Leonard, Quemada, Miguel et al., 1995)
- LOTOS/T (Nakata 1993, ES-Osaka - Japon)
- RT-LOTOS (Courtiat, de Camargo, Saïdouni 1993, LAAS - France)
- E-LOTOS (ISO/IEC 15437 :2001)
- D-LOTOS (Saïdouni, Courtiat 2003, LIRE - Algérie et LAAS - France)

Pour plus de détails sur chacune des extensions citées, le lecteur peut se référer à [Loh02].

Certaines de ces approches ont servi de prémisses et de base de réflexion aux approches qui ont suivi. Pour différencier ces approches, les aspects suivants sont considérés :

Choix du domaine temporel

1. Soit *discret* : les grandeurs temporelles sont définies sur les entiers naturels ($\mathbb{Z}$), une progression d'une unité de temps peut alors être transcrite par l'occurrence d'une action spécifique (fréquemment notée `tic`). Ceci facilite la définition formelle, car le

modèle rapproche du modèle non temporisé, mais occasionne un fort risque d'explosion combinatoire.

2. Soit *dense* et *dénombrable* : les grandeurs temporelles sont définies sur les rationnels positifs $\mathbb{Q}_{\geq 0}$. Cela semble parfaitement convenir à la grande majorité des cas pratiques d'utilisation d'un langage de spécification formelle de systèmes temps-réel. Le modèle mathématique sous-jacent est, par contre, plus complexe à définir et à mettre en œuvre dans le cadre de la vérification.

3. Soit *réel* : quelques rares modèles évoquent la possibilité de définir les grandeurs temporelles dans $\mathbb{R}_{\geq 0}$. Notons qu'alors, presque aucune technique de validation n'a été proposée.

Temporisation des actions Cette temporisation se fait soit par l'ajout d'un opérateur temporel dédié ($\ll$*opérateur*$\langle \ldots \rangle P \gg$), soit par une extension de l'opérateur de préfixe (préfixe d'un processus par une action : $\ll g\langle \ldots \rangle; P \gg$), soit les deux.

Hypothèses d'urgence des actions Une action est dite *urgente* lorsqu'elle doit se réaliser immédiatement, sans progression possible du temps, dès qu'elle est sensibilisée. Certains modèles présupposent implicitement que toutes les actions sont urgentes. La plupart associent l'urgence aux actions internes (l'action i ou une action intériorisée par hide). Certains modèles introduisent un mot-clé spécifique pour déclarer urgentes des actions internes ou observables. D'autres, enfin, proposent des mécanismes pour relâcher l'urgence des actions internes sous certaines conditions.

Opérateurs additionnels Certains langages proposent des facilités d'écriture pour spécifier des comportements réputés classiques par le biais d'opérateurs de haut niveau, qui toutefois n'introduisent pas véritablement de fonctionnalités nouvelles dans le modèle (en d'autres termes, les comportements spécifiés par ces opérateurs de haut niveau peuvent également être spécifiés au moyen d'opérateurs du modèle de base, mais de manière plus lourde).

Le Tableau 3.2 reprend quelques points de comparaison entre les différentes extensions temporelles à LOTOS.

3.1.2 Le langage D-LOTOS

Dans ce qui suit, nous introduisons le langage D-LOTOS comme il a été défini dans [SC03]. C'est un langage de spécification de haut niveau pour des systèmes temps-réel avec des durées d'actions pas nécessairement nulles.

	Domaine temporel	**Temporisation**	**Urgence**
TIC-LOTOS	Discret	Préfixe et opérateur	Toute action
LOTOS-T	Discret ou dense	Préfixe	Actions cachées
T-LOTOS	Discret	Opérateur	Mot-clé spécifique
U-LOTOS	Discret	Opérateur	Mot-clé spécifique
TLOTOS	Discret	Opérateur	Relâchée
Time LOTOS	Dense	Opérateur	Actions cachées
ET-LOTOS	Dense	Préfixe et opérateur	Actions cachées
TE-LOTOS	Dense	Préfixe et opérateur	Toute action
LOTOS/T	Discret	Préfixe	Mot-clé spécifique
RT-LOTOS	Dense	Préfixe et opérateur	Actions cachées
E-LOTOS	Dense	Préfixe et opérateur	Actions cachées
D-LOTOS	Dense	Préfixe et opérateur	Actions cachées

TABLE 3.2 – *Comparaison des extensions temporelles à LOTOS*

Soit $\mathcal{D}$ un ensemble dénombrable, les éléments de $\mathcal{D}$ désignent des valeurs temporelles. Soit $\mathfrak{T}$ l'ensemble de toutes les fonctions $\tau : Act \to \mathcal{D}$ telles que $\tau(i) = \tau(\delta) = 0$. τ_0 est la fonction constante définie par $\tau_0(a) = 0$ pour tout $a \in Act$.

La fonction de durée τ étant fixée, considérons l'expression de comportement $G = a; b; stop$. Dans l'état initial, aucune action n'est en cours d'exécution et la configuration associée est donc $_{\emptyset}[a; b; stop]$; à partir de cet état, la transition $_{\emptyset}[a; b; stop] \xrightarrow{\emptyset a_x} {}_{\{x\}}[b; stop]$ est possible. L'état résultant interprète le fait que l'action a est potentiellement en cours d'exécution. Selon la sémantique de maximalité, nous n'avons pas le moyen de déterminer si l'action a a terminé ou pas son exécution, sauf dans le cas où l'action b a débuté son exécution (le début de b dépend de la fin de a) ; ainsi, si b a débuté son exécution, nous pouvons déduire que a a terminé de s'exécuter. Nous pouvons ainsi constater que les durées d'action sont présentes de manière intrinsèque mais implicite dans l'approche de maximalité ; leur prise en compte de manière explicite va nous permettre de raisonner sur des propriétés quantitatives du comportement d'un système.

En prenant en compte la durée de l'action a, nous pouvons accepter la transition $_{\emptyset}[a; b; stop] \xrightarrow{\emptyset a_x} {}_{\{x:a:\tau(a)\}}[b; stop]$. La configuration résultante montre que l'action b ne peut débuter son exécution que si une durée égale à $\tau(a)$ s'est écoulée, cette durée ne représentant rien d'autre que le temps nécessaire à l'exécution de l'action a. Nous pou-

vons bien sûr également considérer les états intermédiaires représentant l'écoulement d'un laps de temps $t \leq \tau(a)$ par ${}_{\{x:a:\tau(a)\}}[b; stop] \xrightarrow{t} {}_{\{x:a:\tau(a)-t\}}[b; stop]$; de telles configurations seront appelées par la suite *configurations temporelles*, ce qui nous amène à constater qu'une configuration générée par la sémantique de maximalité représente en fait une classe de configurations temporelles.

La prise en compte explicite des durées d'action dans les algèbres de processus ne permet pas cependant à elle seule de spécifier des systèmes temps-réel [GRS95]. Pour combler ce manque, nous considérons des opérateurs classiques de délai similaires à ceux introduits dans des extensions temporelles de LOTOS, telles que ET-LOTOS [LL97] ou RT-LOTOS [CdS93a, CSLO00], la sémantique de ces opérateurs étant bien entendu exprimée dans notre contexte de maximalité. Du fait que les actions ne sont pas atomiques, les contraintes temporelles concernent dans ce contexte le début d'exécution des actions et non pas l'exécution complète des actions. Le langage ainsi défini est appelé *D-LOTOS*, pour LOTOS avec durées d'action.

La syntaxe de D-LOTOS est définie comme suit :

$$\begin{aligned} E ::= \quad & stop \mid exit\{d\} \mid \Delta^d E \mid X[L] \mid g@t[SP]; E \mid i@t\{d\}; E \mid \\ & E[]E \mid E|[L]|E \mid \ hide\, L\, in\, E \mid E \gg E \mid E\,[>\, E \end{aligned}$$

Soient a une action (observable ou interne), E une expression de comportement et $d \in \mathcal{D}$ une valeur dans le domaine temporel. Intuitivement, $a\{d\}$ signifie que l'action a doit commencer son exécution dans l'intervalle temporel $[0, d]$. $\Delta^d E$ signifie qu'aucune évolution de E n'est permise avant l'écoulement d'un délai égal à d. Dans $g@t[SP]; E$ (resp. $i@t\{d\}; E$), t est une variable temporelle mémorisant le temps écoulé depuis la sensibilisation de l'action g (resp. i) et qui sera substituée par zéro lorsque cette action termine son exécution.

Définition 3.3 [1] *L'ensemble $\mathcal{C}_t$ des configurations temporelles est donné par :*

- $\forall E \in \mathcal{B}$, $\forall M \in 2^{\mathcal{M}\times Act\times\mathcal{D}}_{fn}$: ${}_M[E] \in \mathcal{C}_t$
- $\forall P \in PN$, $\forall M \in 2^{\mathcal{M}\times Act\times\mathcal{D}}_{fn}$: ${}_M[P] \in \mathcal{C}_t$
- *si* $\mathcal{E} \in \mathcal{C}_t$ *alors hide* L *in* $\mathcal{E} \in \mathcal{C}_t$
- *si* $\mathcal{E} \in \mathcal{C}_t$ *et* $F \in \mathcal{B}$ *alors* $\mathcal{E} \gg F \in \mathcal{C}_t$
- *si* $\mathcal{E}, \mathcal{F} \in \mathcal{C}_t$ *alors* $\mathcal{E}$ *op* $\mathcal{F} \in \mathcal{C}_t$ $\qquad op \in \{\ []\ ,\ |||\ ,\ ||\ ,\ |[L]|\ ,\ [>\ \}$

[1] Pour alléger l'exposé, nous continuons à utiliser les mêmes symboles désignant les expressions de comportements, les configurations temporelles, ... Le discours étant clarifié par le contexte.

- *si* $\mathcal{E} \in \mathcal{C}_t$ *et* $\{a_1, ..., a_n\}$, $\{b_1, ..., b_n\} \in 2^{\mathcal{G}}_{fn}$ *alors* $\mathcal{E}\,[b_1/a_1, ..., b_n/a_n] \in \mathcal{C}_t$

Les opérations d'encapsulation, de formes canoniques et de calcul d'ensembles maximaux définies précédemment sur les configurations s'étendent de manière naturelle aux configurations temporelles. Une précision reste à faire pour les fonctions de substitution, désormais $Subs = \mathcal{M} \times Act \times \mathcal{D} \longrightarrow 2^{\mathcal{M}\times Act\times\mathcal{D}}_{fn}$. Par exemple $[y : a : 3/x : a : 4]x : a : 4 = y : a : 3$. L'extension aux configurations temporelles se déduit de manière directe.

Sémantique opérationnelle structurée de D-LOTOS

La fonction de durée τ étant fixée, la relation de transition temporelle de maximalité entre les configurations temporelles est notée $\rightarrow_\tau \subseteq \mathcal{C}_t \times Atm \cup \mathcal{D} \times \mathcal{C}_t$.

Processus *stop*

Considérons la configuration ${}_M[stop]$. A la différence du processus *stop* de LOTOS, cette configuration représente des évolutions potentielles en fonction des actions indexées par l'ensemble M. L'évolution cesse dès que toutes ces actions terminent, ce qui est caractérisé au moyen du prédicat $Wait : 2^{\mathcal{M}\times Act\times\mathcal{D}}_{fn} \longrightarrow \{true, false\}$ défini sur tout $M \in 2^{\mathcal{M}\times Act\times\mathcal{D}}_{fn}$ par : $Wait(M) = \exists\, x : a : d \in M \;\; tel\ que\ d > 0$. Intuitivement, $Wait(M) = true$ s'il existe au moins une action référencée dans M qui est en cours d'exécution. Ainsi, tant que $Wait(M) = true$, le passage du temps a un effet sur la configuration ${}_M\,[stop]$, d'où la règle sémantique ${}_M\,[stop] \xrightarrow{d}_\tau {}_{M^d}[stop]$ avec M^d donné par la Définition 3.4.

Processus *exit*

Considérons maintenant la configuration ${}_M[exit]$. La terminaison avec succès ne peut se produire qu'une fois les actions indexées par l'ensemble M ont terminé leur exécution, ce qui est conditionné par la valeur de $Wait(M)$ qui doit être égale à $false$ dans la règle 1. Les règles 2 et 3 expriment le fait que le temps attaché au processus *exit* ne peut commencer à s'écouler que si toutes les actions référencées par M sont terminées. La règle 4 impose que l'occurence de l'action δ ait lieu dans la période d, dans le cas contraire la terminaison avec succès ne se produira jamais.

1. $$\frac{\neg Wait(M)}{{}_M[exit\{d'\}] \xrightarrow{M^\delta x}_\tau {}_{\{x:\delta:0\}}[stop]} \; x{=}get(\mathcal{M})$$

2. $$\frac{Wait(M^d)\ or\ (\neg Wait(M^d)\ and\ \forall\varepsilon{>}0.\ Wait(M^{d-\varepsilon}))}{{}_M[exit\{d'\}] \xrightarrow{d}_\tau {}_{M^d}[exit\{d'\}]} \qquad d{>}0$$

3. $\dfrac{\neg Wait(M)}{{}_M[exit\{d'+d\}] \xrightarrow{d}_\tau {}_M[exit\{d'\}]}$

4. $\dfrac{\neg Wait(M)\ and\ d'>d}{{}_M[exit\{d\}] \xrightarrow{d'}_\tau {}_M[stop]}$

Opérateur de préfixage

Les mêmes contraintes sont imposées à l'occurrence d'une action observable préfixant un processus que celles imposées à l'occurrence de l'action δ à partir de la configuration ${}_M[exit\{d\}]$. Avec l'hypothèse que les actions ne sont pas urgentes, nous considérons l'opérateur @ introduit dans ET-LOTOS. L'expression SP dans les règles suivantes représente un prédicat sur l'exécution de l'action g. Les règles 1, 2 et 5 montrent que la prise en compte de l'écoulement du temps dans E commence uniquement lorsque l'action g est sensibilisée ou a commencé son exécution. La règle 3 exprime le fait qu'une fois que l'action g est sensibilisée et que le prédicat SP est vrai à cet instant, l'action g peut commencer son exécution. L'expression de comportement E ne peut évidemment évoluer que si l'action g se termine, ce qui est exprimé par la règle 4. Il est à noter que le préfixage peut être soit par une action observable ou interne avec la condition que l'action interne i et de durée nulle ($\tau(i) = 0$).

1. $\dfrac{Wait(M^d)\ or\ (\neg Wait(M^d)\ and\ \forall\varepsilon{:}0{<}\varepsilon{<}d.\ Wait(M^{d-\varepsilon}))\qquad d{>}0}{{}_M[g@t[SP];E] \xrightarrow{d}_\tau {}_{M^d}[g@t[SP];E]}$

2. $\dfrac{\neg Wait(M)\qquad d{>}0}{{}_M[g@t[SP];E] \xrightarrow{d}_\tau {}_M[g@t[[t{+}d/t]SP];[t{+}d/t]E]}$

3. $\dfrac{\neg Wait(M)\ and \vdash[0/t]SP}{{}_M[g@t[SP];E] \xrightarrow{M\,g\,x}_\tau {}_{\{x:g:\tau(g)\}}[E(t)]}\ x{=}get(\mathcal{M})$

4. $\dfrac{\neg Wait(x{:}g{:}d)\ and\ {}_{\{x:g:0\}}[[0/t]E] \xrightarrow{M\,a\,x} \mathcal{E}}{{}_{\{x:g:d\}}[E(t)] \xrightarrow{M\,a\,x}_\tau \mathcal{E}}$

5. $\dfrac{}{{}_{\{x:g:d'+d\}}[E(t)] \xrightarrow{d}_\tau {}_{\{x:g:d'\}}[[t{+}d/t]E(t)]}$

Les autres opérateurs

La sémantique des opérateurs de délai, de choix, de composition parallèle, d'intériorisation, de séquencement, d'interruption, de renommage de portes et d'instanciation de processus est donnée par les règles 3a, 4(a)i, 4(a)ii, 4b, 5a, 5b, 6a, 7a, 7b, 7c, 8a et 9 de la Définition 3.5, dans lesquelles les configurations sont temporelles et la relation de transition est remplacée par $\rightarrow_\tau$, complétées par les règles suivantes :

$$\frac{}{\Delta^{d'+d}\mathcal{E} \xrightarrow{d}_{\tau} \Delta^{d'}\mathcal{E}} \qquad \frac{\mathcal{E} \xrightarrow{d} \mathcal{E}' \qquad \forall d'{<}d.\ \mathcal{E}^{d'} \not\xrightarrow{a}_{\tau} \qquad \forall a{\in}L}{hide\ L\ in\ \mathcal{E} \xrightarrow{d}_{\tau} hide\ L\ in\ \mathcal{E}'}$$

$$\frac{\mathcal{E} \xrightarrow{d} \mathcal{E}'}{\Delta^{0}\mathcal{E} \xrightarrow{d}_{\tau} \mathcal{E}'} \qquad \frac{\mathcal{E} \xrightarrow{M^{\delta}x}_{\tau} \mathcal{E}'}{\mathcal{E}{>}{>}F \xrightarrow{M^{i}x}_{\tau} {}_{\{x:i:0\}}[F]}$$

$$\frac{\mathcal{E} \xrightarrow{M^{a}x}_{\tau} \mathcal{E}'}{\Delta^{0}\mathcal{E} \xrightarrow{M^{a}x}_{\tau} \mathcal{E}'} \qquad \frac{\mathcal{E} \xrightarrow{d}_{\tau} \mathcal{E}' \qquad \mathcal{E} \not\xrightarrow{\delta}_{\tau}}{\mathcal{E}{>}{>}F \xrightarrow{d}_{\tau} \mathcal{E}'{>}{>}F}$$

$$\frac{\mathcal{E} \xrightarrow{d}_{\tau} \mathcal{E}' \qquad \mathcal{F} \xrightarrow{d}_{\tau} \mathcal{F}'}{\mathcal{E}\ []\ \mathcal{F} \xrightarrow{d}_{\tau} \mathcal{E}'[]\mathcal{F}'} \qquad \frac{\mathcal{E} \xrightarrow{d}_{\tau} \mathcal{E}' \qquad \mathcal{F} \xrightarrow{d}_{\tau} \mathcal{F}'}{\mathcal{E}[{>}\mathcal{F} \xrightarrow{d}_{\tau} \mathcal{E}'[{>}\mathcal{F}'}$$

$$\frac{\mathcal{E} \xrightarrow{d}_{\tau} \mathcal{E}' \qquad \mathcal{F} \xrightarrow{d}_{\tau} \mathcal{F}'}{\mathcal{E}\ |[L]|\ \mathcal{F} \xrightarrow{d}_{\tau} \mathcal{E}'|[L]|\mathcal{F}'} \qquad \frac{\mathcal{E} \xrightarrow{d}_{\tau} \mathcal{E}'}{\mathcal{E}[b_1/a_1,...,b_n/a_n] \xrightarrow{d}_{\tau} \mathcal{E}'[b_1/a_1,...,b_n/a_n]}$$

$$\frac{P{:=}E \qquad {}_{M}[E] \xrightarrow{d}_{\tau} \mathcal{F}}{{}_{M}[P] \xrightarrow{d}_{\tau} \mathcal{F}} \qquad \frac{\mathcal{E} \xrightarrow{M^{a}x}_{\tau} \mathcal{E}' \qquad a{=}a_i\ (1{\le}i{\le}n)}{\mathcal{E}[b_1/a_1,...,b_n/a_n] \xrightarrow{M^{bi}x}_{\tau} \mathcal{E}'[x{:}bi{:}dbi/x{:}a{:}da][b_1/a_1,...,b_n/a_n]}$$

Ces règles sont similaires à celles introduites dans les algèbres de processus temps-réel.

Définition 3.4 *L'opération d'écoulement de temps* $(.)^d$ *dans une configuration est définie récursivement par :*

$$\emptyset^d = \emptyset \qquad (\mathcal{E}\ []\ \mathcal{F})^d = \mathcal{E}^d[]\mathcal{F}^d$$

$$(x:a:d')^d = x:a:d'-d \quad tel\,que\ d'-d=0\ si\ d>d' \qquad ({}_{M}[E])^d = {}_{M^d}[E]$$

$$(M \cup \{x:a:d'\})^d = M^d \cup \{(x:a:d')^d\} \qquad (\mathcal{E}\ |[L]|\ \mathcal{F})^d = \mathcal{E}^d\ |[L]|\ \mathcal{F}^d$$

$$(hide\ L\ in\ \mathcal{E})^d = hide\ L\ in\ \mathcal{E}^d \qquad (\mathcal{E} \gg F)^d = \mathcal{E}^d \gg F$$

$$(\mathcal{E}\,[b_1/a_1,...,b_n/a_n])^d = \mathcal{E}^d\,[b_1/a_1,...,b_n/a_n] \qquad (\mathcal{E}\ [{>}\mathcal{F})^d = \mathcal{E}^d[{>}\mathcal{F}^d$$

$${}_{\{x:g:d'\}}[E(t)]^d = {}_{\{x:g:d'\}^d}[[t+d/t]E(t)]$$

Quelques relations de bissimulation caractérisant le comportement des applications concurrentes ont été définies pour le langage D-LOTOS, le lecteur peut se référer à [SC03].

Spécification de la latence

L'opérateur de latence de RT-LOTOS est d'un intérêt considérable pour la spécification de systèmes temps réel [Cd94, CdO95a, CdOA95, Loh02, Sam03]. Il peut être introduit sans problème particulier dans le langage D-LOTOS. Mais dans [SC03], on a montré comment l'utilisation conjointe de la notion de durée d'action associée à l'opérateur @ permet de spécifier une latence similaire à celle exprimée par RT-LOTOS sans avoir recours à l'opérateur «artificiel» Ω.

Expliquons tout d'abord la finalité de l'opérateur de latence à travers l'exemple de deux expressions RT-LOTOS $E_1 = a\{u\};F$ et $E_2 = \Omega^l a\{u\};F$ en faisant l'hypothèse que $l < u$. Dans E_1, l'action a peut s'exécuter dans l'interval temporel $[0,u]$ sous

condition que l'environnement accepte de se synchroniser sur cette action dans cet intervalle. Au delà de cet intervalle, l'action a ne pourra plus être offerte, dans ce cas E_1 se transforme en le processus *stop*. Dans l'expression E_2, nous distinguons deux intervalles temporels, $I = [0, l]$ et $J = [l, u]$; durant l'intervalle I, l'action a peut s'exécuter sous condition que l'environnement et le processus E_2 acceptent de se synchroniser ensemble sur cette action ; ainsi, nous pouvons spécifier que le processus peut refuser de se synchoniser pour des raisons non explicitées ! Par contre, durant l'intervalle J , l'action a peut ne pas s'exécuter en raison d'un refus de l'environnement. Cependant, dans le cas où l'action a ne pourra pas s'exécuter, nous ne pouvons pas avoir connaissance de l'origine de ce refus d'exécution. Donc, d'un point de vue observable, ces deux systèmes sont identiques. La différence de comportement peut être perçue dès que l'on intériorise l'action a. Dans l'expression $E_1' = hide\ a\ in\ E_1$, l'action a est urgente et sera executée dès qu'elle est offerte. Par contre dans l'expression $E_2' = hide\ a\ in\ E_2$, l'action a peut ne pas s'exécuter durant l'intervalle I, et elle ne devient urgente qu'à la fin de cet intervalle. L'utilité de l'opérateur de latence réside donc dans la préservation de l'indéterminisme d'exécution des actions intériorisées.

Remarquons que la levée de l'hypothèse d'atomicité des actions implique que toute action peut être considérée comme un processus en exécution, cependant la durée d'éxécution du processus n'a pas nécessairement une valeur déterminée due au comportement non déterministe éventuel de ce processus. Ceci a amené à considérer des durées d'action variables de la forme $[m, M]$ indiquant que la durée d'exécution d'une action est comprise entre une durée minimale égale à m et une durée maximale égale à M. Donc, si une action a munie d'une durée $[m, M]$ débute son exécution à un instant ta, cette action peut terminer son exécution dans l'intervalle temporel $[ta + m, ta + M]$. Cette idée est facilement réalisable en considérant les points suivants :

- Deux fonctions temporelles $min, max : \mathcal{G} \rightarrow \mathcal{D}$ associant respectivement une durée minimale et une durée maximale à toute action observable vont être définies. L'action interne i et l'action δ sont par hypothèse de durée nulle. Evidemment, pour toute action $g \in \mathcal{G} : min(g) \leq \max(g)$.

- A chaque fois qu'une action observable g commence son exécution, on lui associe la durée $max(g)$ dans la configuration résultante (voir les règles sémantiques).

- Etant donné un ensemble $M \in 2_{fn}^{\mathcal{M} \times Act \times \mathcal{D}}$, désormais, le prédicat $wait$ est défini par $wait(M) = \exists x : g : d \in M$ tel que $\max(g) - d < \min(g)$.

Il est clair que la sémantique présentée dans la Section 3.1.2 est un cas particulier

de celle-ci dans laquelle $min(g) = max(g)$ pour toute action observable $g \in \mathcal{G}$.

Soit l'exemple de la spécification d'un médium de communication, introduit dans [Cd94], dont le délai de transmission est compris dans un intervalle $[m, M]$. Soit a l'action correspondant à l'émission d'un message sur le médium, et b l'action de réception de ce message après un délai non déterministe. L'action *error* caractérise la situation d'erreur dans laquelle l'environnement n'est pas prêt à recevoir le message offert par le médium de transmission. En utilisant l'opérateur de latence de RT-LOTOS, la spécification peut être exprimée ainsi [Cd94] :

$$Medium = a; (\Delta^{m} \Omega^{M-m} b\{M-m\};\ Medium\ []\ \Delta^{M+e} error;\ stop)$$

En considérent la notion de durée des actions, nous pouvons distinguer deux types d'actions : d'une part, les actions a et b, qui représentent respectivement l'émission et la réception d'un message, et que nous pouvons considérer de durée nulle, et l'opération de transmission proprement dite qui peut durer entre m et M, qui est représentée par l'action c de durée comprise entre m et M. Ceci a conduit à la spécification suivante avec D-LOTOS :

$$Medium = hide\ c\ in\ a; (c@t; (b; medium\ []\ \Delta^{M-t+e} error;\ stop))$$

3.2 Réseaux de Petri

Les réseaux de Petri ont été développés en tant que formalisme opérationnel pour la spécification des systèmes concurrents. Ils donnent une représentation dynamique d'un état du système par l'utilisation de jetons. Les réseaux de Petri ont été utilisés avec succès pour modéliser une variété de systèmes industriels. Ils peuvent spécifier différents composants du système modélisé dans de différentes étapes d'exécution ou dans de différents instants de temps, ceci rend ce formalisme particulièrement attractif pour modéliser les systèmes embarqués interagissant avec leur environnement extérieur.

Plus précisément, un réseau de Petri, ou un réseau de Petri place/transition, consiste en quatre composants de base : places, transitions, arcs et jetons. Une place est un état dans lequel le système spécifié (ou une de ses parties) peut se situer. Les arcs relient les transitions aux places et les places aux transitions. Si un arc se dirige d'une place à une transition, la place est une entrée (*input*) de cette transition et l'arc est un arc d'entrée vers cette transition. Si un arc se dirige d'une transition à une place, la place est une sortie (*output*) de cette transition et l'arc est un arc de sortie à partir de cette transition. Plusieurs arcs peuvent exister d'une place à une transition. Une place peut

être vide et peut contenir un ou plusieurs jetons. L'état d'un réseau de Petri, connu sous le nom de *marquage*, est défini par le nombre de jetons dans chaque place.

Définition 3.6 *Un* réseau de Petri *est un triplet* (S, T, W) *tel que* S *est l'ensemble de places,* T *est l'ensemble des transitions avec* $S \cap T = \emptyset$*, est* $W : ((S \times T) \cup (T \times S)) \to \mathbb{N} = \{0, 1, 2, ...\}$ *est la fonction de* poids.

Graphiquement, les transitions de T sont représentées par des rectangles ou des barres, les places de S par des cercles est la fonction de poids par des arcs associés à leurs poids. Nous supposons que tous les réseaux de Petri considérés sont finis, c'set-à-dire $|S \cup T| \in \mathbb{N}$.

Pour chaque $x \in S \cup T$, le *pré-ensemble* ${}^\bullet x$ est défini par ${}^\bullet x = \{y \in S \cup T | W(y, x) \neq 0\}$ et le *post-ensemble* $x^\bullet$ est défini par $x^\bullet = \{y \in S \cup T | W(x, y) \neq 0\}$.

Définition 3.7 *Le* marquage *d'un réseau de Petri* (S, T, W) *est défini par une fonction* $M : S \to \mathbb{N}$*. Un marquage est généralement représenté par la déposition de jetons dans les places.*

En tant que formalisme opérationnel, un réseau de Petri retrace un état particulier du système et évolue au prochain état selon les règles suivantes. Une transition t est *sensibilisée* par un marquage M ssi $M(s) \geq W(s, t)$ pour tout $s \in S$. Le tir[2] de la transition t produira le nouveau marquage M' défini par $M'(s) = M(s) - W(s, t) + W(t, s)$ pour tout $s \in S$. L'occurrence de t est notée par $M[t\rangle M'$. Deux transitions t_1 et t_2 (non nécessairement distinctes) sont sensibilisées de manière concurrente par un marquage M ssi $M(s) \geq W(s, t_1) + W(s, t_2)$ pour tout $s \in S$.

Définition 3.8 *Un* réseau de Petri marqué (S, T, W, M_0) *est un réseau de Petri* (S, T, W) *avec un marquage initial* M_0.

Un alphabet $\mathcal{A}$ est un ensemble fini ; nous supposons que $\tau \notin \mathcal{A}$ (τ désignera l'action *invisible*, dite aussi action silencieuse). L'étiquetage d'un réseau de Petri $PN = (S, T, W)$ est une fonction $\lambda : T \to \mathcal{A} \cup \{\tau\}$. Si $\lambda(t) \in \mathcal{A}$ alors t est dite *observable* ; dans le cas contraire, t est dite *silencieuse* ou *invisible*.

Définition 3.9 $\Sigma = (S, T, W, M_0, \lambda)$ *est un système étiqueté ssi* (S, T, W, M_0) *est un réseau de Petri marqué et* λ *est une fonction d'étiquetage de* (S, T, W).

[2]Les mots tir, franchissement, occurrence et exécution de transition seront utilisés indifféremment.

Une action $a \in \mathcal{A}$ d'un système $\Sigma = (S, T, W, M_0, \lambda)$ est dite *auto-concurrente* à un marquage M ssi M sensibilise de manière concurrente deux transitions observables t_1 et t_2 (non nécessairement distinctes) telles que $\lambda(t_1) = \lambda(t_2) = a$.

Une séquence $\sigma = M_0 t_1 M_1 t_2 \dots$ est une séquence d'occurrences ssi $M_{i-1}[ti\rangle M_i$ pour $1 \leq i$. Une séquence $t_1 t_2 \dots$ est une séquence de transitions débutant par M ssi il existe une séquence d'occurrences $M t_1 M_1 t_2 \dots$. Si une séquence finie $t_1 t_2 \dots t_n$ mène au marquage M' à partir du marquage M, on écrit $M[t_1 t_2 \dots t_n\rangle M'$.

L'ensemble des marquages atteignables d'un réseau marqué (S, T, W, M_0) est défini comme $[M_0\rangle = \{M | \exists t_1 t_2 \dots t_n : M_0[t_1 t_2 \dots t_n\rangle M\}$.

3.3 Systèmes de transitions étiquetées maximales

Cette section introduit le modèle des systèmes de transitions étiquetées maximales. Ce modèle est basé sur celui des systèmes de transitions étiquetées, donc il convient de commencer par formaliser ce dernier modèle.

Définition 3.10 *Un* système de transitions $T = (Q, \rightarrow)$ *consiste en un ensemble d'états Q (éventuellement non dénombrable) et une relation de transition $\rightarrow \subseteq Q \times Q$.*

Définition 3.11 *Un* système de transitions étiquetées $T = (Q, \Sigma, \rightarrow)$ *consiste en un ensemble d'états Q (éventuellement non dénombrable), un alphabet fini d'évènements Σ, et une relation de transition $\rightarrow \subseteq Q \times \Sigma \times Q$.*

Une *transition* $(q_1, a, q_2) \in \rightarrow$ est notée $q_1 \xrightarrow{a} q_2$. Un système de transitions est *fini* si Q est fini. Si l'alphabet d'évènements est réduit en un singleton, $\Sigma = \{a\}$, nous pouvons noter ce système de transitions $(Q, \rightarrow)$ et omettre l'évènement a.

Définition 3.12 *Etant donné un système de transitions étiquetées $T = (Q, \Sigma, \rightarrow)$, une* exécution infinie *(ou un* chemin infini*) de T est une séquence infinie d'états $(q_i)_{i \in \mathbb{N}}$ telle que pour tout $i \in \mathbb{N}_{>0}$ il existe $a_i \in \Sigma$ tel que $q_{i-1} \xrightarrow{a_i} q_i$. Nous dénotons cette exécution comme suit :*

$$\rho = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \dots \xrightarrow{a_i} q_i \dots$$

Nous définissons de la même manière la notion d'*exécution finie* (ou *chemin fini*) de *longueur* n de T, et la dénotons par $(q_i)_{0 \leq i \leq n}$.

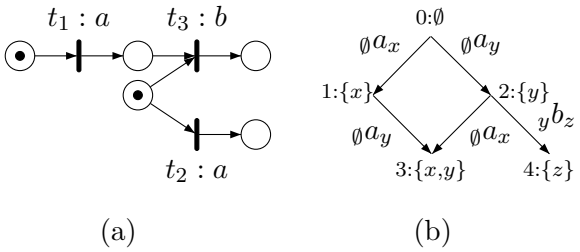

FIGURE 3.1 – Système de transitions étiquetées maximales

3.3.1 Principe des systèmes de transitions étiquetées maximales

Un système de transitions étiquetées maximales n'est autre qu'un graphe d'état bi-étiqueté tel que les transitions sont des évènements qui représentent le début d'exécution des actions. Une transition est donc étiquetée par le nom de l'action qui lui correspond. Un état est étiqueté par l'ensemble des actions qui sont potentiellement en cours d'exécution au niveau de cet état là, ces actions sont dites *maximales*. Etant donné que plusieurs actions de même nom peuvent s'exécuter en parallèle (auto-concurrence), nous identifions chaque début d'action par un identificateur distinct, appelé *nom d'évènement*. Les noms des évènements sont choisis à partir d'un ensemble dénombrable noté $\mathcal{M}$. Comme illustration, considérons le réseau de Petri de la Figure 3.1.(a). Dans cet exemple, seules les transitions t_1 et t_2, peuvent s'exécuter en parallèle, ce qui correspond à l'exécution parallèle de deux actions de nom a. À titre illustratif, nous admettons que les débuts d'exécution des actions a sont identifiés par les noms d'évènements x et y alors que le début d'exécution de l'action b est identifié par le nom d'évènement z. La Section **??** présentera une méthode de choix des noms d'évènements permettant leur réutilisation dans un même système de transitions étiquetées maximales ; ceci permettra de traiter les systèmes à comportements cycliques.

Le système de transitions étiquetées maximales qui représente la sémantique du réseau de Petri de la Figure 3.1.(a) est donné par la Figure 3.1.(b). Dans l'état initial, aucune action n'a commencé son exécution, l'état initial est donc étiqueté par l'ensemble vide. À partir de cet état, chacune des actions a peut commencer son exécution, d'où les transitions identifiées respectivement par les noms d'évènement x et y. L'état 1, étiqueté par l'ensemble $\{x\}$ signifie que l'action a est potentiellement en cours d'exécution au niveau de cet état. La transition identifiée par le nom d'évènement y correspond au

début de l'exécution de l'autre action a. L'état 3, étiqueté par l'ensemble $\{x, y\}$, montre que les deux actions a peuvent être en cours d'exécution *simultanément* ; alors que l'état 2, étiqueté par l'ensemble $\{y\}$, montre que dans cet état seule l'action a peut être en cours d'exécution. À partir de l'état 2, deux scénarios se présentent : soit la deuxième action a commence son exécution, dans ce cas on aboutit à l'état 3 ; soit c'est l'action b qui commence son exécution. Il est clair que l'action b ne peut commencer son exécution qu'après la fin de la première exécution de l'action a. Cette dépendance causale entre l'exécution de l'action a et l'exécution de l'action b est capturée par l'ensemble $\{y\}$ associé à la transition menant le système de l'état 2 à l'état 4. Dans l'état résultant, seule l'action b peut être en cours d'exécution, d'où l'étiquetage de cet état par l'ensemble $\{z\}$.

3.3.2 Formalisation

Pour formaliser le modèle des systèmes de transitions étiquetées maximales, nous utilisons la notation de [Arn92] pour définir le modèle sous-jacent des systèmes de transitions étiquetées.

Définition 3.13 *Soit $\mathcal{M}$ un ensemble dénombrable de noms d'évènements, un* système de transition étiquetées maximales *de support $\mathcal{M}$ est un quintuplet $(\Omega, \lambda, \mu, \xi, \psi)$ avec :*

- *$\Omega = \langle S, T, \alpha, \beta, s_0 \rangle$ est un système de transitions tel que :*
  - *S est l'ensemble des états dans lesquels le système peut être trouvé, cet ensemble peut être fini ou infini.*
  - *T est l'ensemble des transitions indiquant les changements d'états que le système peut effectuer, cet ensemble peut être fini ou infini.*
  - *α et β sont deux applications de T dans S telles que pour chaque transition t nous avons : $\alpha(t)$ est l'origine de la transition et $\beta(t)$ est sont but.*
  - *s_0 est l'état initial du système de transition Ω.*
- *(Ω, λ) est un système de transitions étiquetés par la fonction λ sur un alphabet Act appelé support de (Ω, λ). ($\lambda : T \to Act$).*
- *$\psi : S \to 2^{\mathcal{M}}$ est une fonction qui associe à chaque état l'ensemble fini d'évènements maximaux présents dans cet état.*
- *$\mu : T \to 2^{\mathcal{M}}$ est une fonction qui associe à chaque transition l'ensemble fini des noms d'évènements correspondant aux actions qui ont déjà commencé leurs exécutions et pour lesquelles leurs fins sensibilisent cette transition.*

- $\xi : T \to \mathcal{M}$ *est une fonction qui associe à chaque transition le nom d'évènement identifiant son occurrence.*

tel que $\psi(s_0) = \emptyset$ *et pour toute transition* t, $\mu(t) \subseteq \psi(\alpha(t))$, $\xi(t) \notin \psi(\alpha(t)) - \mu(t)$ *et* $\psi(\beta(t)) = (\psi(\alpha(t)) - \mu(t)) \cup \{\xi(t)\}$.

Notation 3.1 *Dans ce qui suit, nous utilisons les notations suivantes :*

- *Soit* $mlts = (\Omega, \lambda, \mu, \xi, \psi)$ *un système de transition étiquetées maximales tel que* $\Omega = \langle S, T, \alpha, \beta, s_0 \rangle$. $t \in T$ *est une transition pour chaque* $\alpha(t) = s$, $\beta(t) = s'$, $\lambda(t) = a$, $\mu(t) = E$ *et* $\xi(t) = x$. *La transition* t *sera notée* $s \xrightarrow{E a_x} s'$.

- *Soit* $f : E \to F$ *une function de domaine* $Dom(f) = E$ *et de co-domaine* $Cod(f) = F$, *et soit* D *(respectivement* C*) un sous-ensemble de* E *(respectivement de* F*). Les restrictions de* f *vis-à-vis son domaine et son co-domaine sont définies par :*

  - $f \lceil D = \{(x, y) \in f | x \in D\}$
  - $f \lfloor C = \{(x, y) \in f | y \in C\}$

- $\mathfrak{F} \subseteq 2^{\mathcal{M} \times \mathcal{M}}$ *est l'ensemble de toutes les fonctions bijectives entre les sous-ensembles de* $\mathcal{M}$.

- Id_A *est la fonction identité sur les éléments d'un ensemble* A.

Définition 3.14 *Soit* $mlts_1 = (\Omega_1, \lambda_1, \mu_1, \xi_1, \psi_1)$ *et* $mlts_2 = (\Omega_2, \lambda_2, \mu_2, \xi_2, \psi_2)$ *deux systèmes de transitions étiquetées maximales tels que* $\Omega_1 = \langle S_1, T_1, \alpha_1, \beta_1, s1_0 \rangle$ *et* $\Omega_2 = \langle S_2, T_2, \alpha_2, \beta_2, s2_0 \rangle$. $mlts_1$ *et* $mlts_2$ *sont dits* maximallement bisimilaires, *noté* $mlts_1 \approx_m mlts_2$, *s'il existe une relation* $\mathfrak{R} \subseteq S_1 \times S_2 \times \mathfrak{F}$ *avec*

1. $(s1_0, s2_0, \emptyset) \in \mathfrak{R}$. *Les états initiaux de* $mlts_1$ *et* $mlts_2$ *sont reliés par la relation. Puisque les ensembles des évènements maximaux dans les états initiaux sont vides, la fonction reliant ces deux ensembles est vide.*

2. *Si* $(s1, s2, f) \in \mathfrak{R}$ *alors*

   (a) $Dom(f) \subseteq \psi(s1)$ *et* $Cod(f) \subseteq \psi(s2)$.

   (b) *Si* $s1 \xrightarrow{E a_x} s1'$ *alors il existe* $s2 \xrightarrow{F a_y} s2'$ *tel que*

      i. $\forall (u, v) \in f$, *si* $u \notin E$ *alors* $v \notin F$

      ii. $(s1', s2', f') \in \mathfrak{R}$ *avec* $f' = (f \lceil (\psi(s1') - \{x\})) \lfloor (\psi(s2') - \{y\}) \cup \{(x, y)\}$

(c) Si $s2 \xrightarrow{F a_y} s2'$ *alors il existe* $s1 \xrightarrow{E a_x} s1'$ *tel que*

i. $\forall (u, v) \in f$*, si* $v \notin F$ *alors* $u \notin E$

ii. $(s1', s2', f') \in \mathfrak{R}$ *avec* $f' = (f \lceil (\psi(s1') - \{x\})) \lfloor (\psi(s2') - \{y\}) \cup \{(x, y)\}$

3.4 Automates temporisés

Les systèmes de transitions étiquetées sont un modèle très intuitif lorsqu'on fait abstraction du temps lors de la spécification des systèmes réels. Le besoin de spécifier des systèmes temps-réel a fait naître l'idée d'incorporer le temps dans les systèmes de transitions étiquetées, d'où la définition des *systèmes de transitions temporisés* où les transitions dénotent cette fois-ci soit une exécution d'une action soit un passage de temps.

Le formalisme des *automates temporisés* a été introduit pour la première fois par Rajeev Alur et David Dill dans [AD90, AD94]. Sa définition fournit un simple moyen pour munir les systèmes de transitions d'un ensemble de contraintes temporelles exprimées à l'aide de variables réelles appelées *horloges*.

Les automates temporisés peuvent alors être décrits comme des automates finis auxquels ont été rajoutées des horloges. Il existe deux types d'évolutions possibles pour un tel système. Une évolution du temps exprimée par la progression des valeurs des horloges au sein d'un état de l'automate et une évolution entre états de l'automate. Une transition entre deux états de l'automate exprime une exécution d'une action impliquant la remise à zéro d'un ensemble d'horloges, une telle transition est conditionnée par la satisfaction d'une contrainte de franchissement en fonction des valeurs actuelles des horloges.

3.4.1 Notations et définitions préliminaires

Nous commençons par introduire quelques notations afin de définir formellement les automates temporisés.

Nous notons par $X = \{x_1, \ldots, x_n\}$ un ensemble de n *horloges*. Une *valuation d'horloges* est une fonction $\nu : X \to \mathbb{T}$, où $\mathbb{T}$ est égal à l'ensemble $\mathbb{R}_{\geq 0}$ de réels non négatifs ou l'ensemble $\mathbb{N}$ d'entiers naturels, selon le fait que le temps soit *dense* ou *discret*. Soit une valuation d'horloges ν, pour $i = 1, \ldots, n$, nous dénotons par ν_i l'image de l'horloge x_i par la fonction ν, c'est-à-dire $\nu(x_i) = \nu_i$. Par la suite, les deux notations $\nu(x_i)$ et ν_i seront utilisées.

Etant donnée une valuation d'horloges ν, s'il n'y a pas de confusion, nous dénotons

aussi par ν l'élément de $\mathbb{T}^n$ donné par $(v_1, \ldots, v_n)$. Etant donnée une valuation ν et $\tau \in \mathbb{T}$, $\nu + \mathbb{T}$ est la valuation d'horloges définie par $(v_1 + \tau, \ldots, v_n + \tau)$.

Une *garde* (ou *contrainte temporelle*) est n'importe quelle conjonction finie d'expressions de la forme $x_i \sim c$ ou $x_i - x_j \sim c$ où x_i et x_j sont des horloges, $c \in \mathbb{N}$ est un constant entier, et $\sim \in \{<, \leq, =, >, \geq\}$.

Nous dénotons par $\mathcal{G}$ l'ensemble des gardes. Soit g une garde et ν une valuation d'horloges, la notation $\nu \models g$ signifie que $(v_1, \ldots, v_n)$ satisfait g. Une *remise à zéro* ou *réinitialisation* $Y \in 2^X$ indique les horloges remises à zéro.

Nous utilisons la notation AP pour l'ensemble des *propositions atomiques*.

Définition 3.15 *Un* automate temporisé $\mathcal{A} = (L, X, \Sigma, E, \mathcal{I}, \mathcal{L})$ *est constitué des composants suivants : (i)* L *est un ensemble fini d'*emplacements*, (ii)* X *est un ensemble fini d'*horloges*, (iii)* Σ *est un ensemble fini d'*actions*, (iv)* $E \subseteq L \times \Sigma \times \mathcal{G} \times 2^X \times L$ *est un ensemble fini d'*arcs*, (v)* $\mathcal{I} : L \to \mathcal{G}$ *fait correspondre un* invariant *à chaque emplacement, et (vi)* $\mathcal{L} : L \to 2^{\mathsf{AP}}$ *est la fonction d'*étiquetage*.*

Nous définissons maintenant les notions d'automates temporisés *bornés* et *non diagonaux*.

Définition 3.16 *Un automate temporisé* $\mathcal{A}$ *est* non diagonal *si les gardes utilisées dans les arcs et les invariants ne contiennent pas des expressions de la forme* $x_i - x_j \sim c$*, avec* x_i *et* x_j *des horloges,* $c \in \mathbb{N}$ *et* $\sim \in \{<, \leq, =, \geq, >\}$*.*

Définition 3.17 *Un automate temporisé* $\mathcal{A}$ *est* borné *si pour chaque emplacement* l*, toutes les horloges utilisées dans l'invariant* $\mathcal{I}(l)$ *sont bornées.*

La sémantique d'un automate temporisé $\mathcal{A}$ est donnée par un système de transitions étiquetées $T_{\mathcal{A}}$. Afin de définir ce système de transitions, nous avons besoin de définir les *états* de $\mathcal{A}$ et les *transitions* entre deux états de $\mathcal{A}$.

Définition 3.18 *Un* état *d'un automate temporisé* $\mathcal{A}$ *est une paire* $q = (l, \nu)$ *telle que* $l \in L$ *et* $\nu \models \mathcal{I}(l)$*. Nous dénotons l'ensemble de tous les états par* Q*.*

Remarque 3.1 *Lorsque* $\mathcal{A}$ *est un automate temporisé borné, il existe une constante* m *telle que chaque état* (l, ν) *de* $T_{\mathcal{A}}$ *satisfait* $\nu_i \leq m$ *pour tout* $i \in 1, \ldots, n$*.*

Nous distinguons deux types de transitions : *transitions temporelles* et *transitions de changement d'état*.

Définition 3.19 *Etant donnés $q = (l, \nu)$ et $q' = (l', \nu')$ deux états de $\mathcal{A}$, il y a une* transition temporelle *dans $\mathcal{A}$ entre q et q' s'il existe $\tau \in \mathbb{T}$ tel que $l = l'$, $\nu' = \nu + \tau$ et $\nu + \tau' \models \mathcal{I}(l)$ pour n'importe quel τ', $0 \leq \tau' \leq \tau$. Nous notons cette transition $q \xrightarrow{\tau} q'$.*

Définition 3.20 *Etant donnés $q = (l, \nu)$ et $q' = (l', \nu')$ deux états de $\mathcal{A}$, il y a une* transition de changement d'état *dans $\mathcal{A}$ entre q et q' s'il existe $e = (l, a, g, Y, l') \in E$ tel que $\nu \models g$ et ν' est donnée par :*

$$\nu'_i = \begin{cases} 0 & \text{if } x_i \in Y \\ \nu_i & \text{if } x_i \notin Y \end{cases}$$

Nous notons cette transition de changement d'état $q \xrightarrow{e} q'$. Pour mettre l'accent sur l'action a, nous pouvons aussi utiliser la notation $q \xrightarrow{a} q'$.

Nous définissons maintenant le système de transitions étiquetées $T_{\mathcal{A}}$.

Définition 3.21 *Etant donné un automate temporisé $\mathcal{A}$, le* système de transitions étiquetées associé à $\mathcal{A}$ *est donné par $T_{\mathcal{A}} = (Q, \Sigma \cup \mathbb{T}, \rightarrow)$ où la relation de transition est donnée par*

$$\rightarrow = \bigcup_{\tau \in \mathbb{T}} \xrightarrow{\tau} \cup \bigcup_{e \in E} \xrightarrow{e}$$

Notation 3.2 *Etant donnés q, q' deux états de $T_{\mathcal{A}}$, une transition (temporelle ou de changement d'état) entre q et q' est uniformément dénotée par $q \rightarrow q'$. En d'autres termes, quand on écrit $q \rightarrow q'$ on veut dire qu'il existe soit un temps τ tel que $q \xrightarrow{\tau} q'$ soit un arc e tel que $q \xrightarrow{e} q'$.*

Remarque 3.2 *Nous pouvons remarquer que la notation $(l, \nu) \rightarrow (l', \nu')$ est ambigüe dans des cas très particuliers tant qu'elle peut représenter une transition temporelle ou une transition de changement d'état. En effet, on peut avoir $(l, \nu) \xrightarrow{\tau} (l, \nu')$ avec $\tau = 0$ et $(l, \nu) \xrightarrow{e} (l, \nu')$ pour certain $e \in E$. Cependant, nous utilisons cette notation pour éviter d'utiliser de lourdes notations.*

Remarque 3.3 *Nous pouvions aussi définir* un système de transitions abstrait $T^{ta}_{\mathcal{A}} = (Q, \Sigma \cup \{t\}, \rightarrow)$, *où $t \notin \Sigma$ et $q \xrightarrow{t} q'$ si et seulement s'il existe $\tau \in \mathbb{T}$ tel que $q \xrightarrow{\tau} q'$.*

Nous définissons maintenant la notion d'exécution d'un automate temporisé.

Définition 3.22 *Soit $\mathcal{A}$ un automate temporisé. Une* exécution finie de $\mathcal{A}$ *est un chemin fini $\rho = (q_i)_{i \in \{0,\ldots,l\}}$ de $T_{\mathcal{A}}$ dénoté par*

$$\rho = q_0 \rightarrow q_1 \rightarrow \cdots \rightarrow q_l$$

Nous pouvons écrire aussi $\rho = q_0 \rightsquigarrow q_l$.

Définition 3.23 *Soit $\mathcal{A}$ un automate temporisé. Une* exécution infinie de $\mathcal{A}$ *est un chemin infini $\rho = (q_k)_{k\in\mathbb{N}}$ de $T_\mathcal{A}$ dénoté par*

$$\rho = q_0 \to q_1 \to \cdots \to q_i \ldots$$

Notation 3.3 *Une exécution finie ou infinie est uniformément dénotée par $\rho = (q_i)_{i\in I}$.*

Définition 3.24 *Une exécution $\rho = (q_k)_{k\in K}$ est appelée* initiale *si q_0 est de la forme $(l, 0)$ où toutes les valeurs d'horloges sont nulles.*

Définition 3.25 *Une exécution ρ est* canonique *si elle est de la forme $q_0 \xrightarrow{\tau_1} q_1 \xrightarrow{e_1} q_2 \xrightarrow{\tau_2} q_3 \xrightarrow{e_2} q_4 \ldots$ où les transitions temporelles et les transitions de changement d'état alternent.*

Remarque 3.4 *Nous pouvons associer à n'importe quelle exécution (éventuellement initiale) une exécution canonique (éventuellement initiale). En effet, deux transitions temporelles consécutives $q_k \xrightarrow{\tau} q_{k+1} \xrightarrow{\tau'} q_{k+2}$ peuvent être remplacées par la transition temporelle $q_k \xrightarrow{\tau+\tau'} q_{k+2}$ telle que la transition temporelle $q_k \xrightarrow{\tau} q_{k+1}$ avec $\tau = 0$ est permise dans la Définition 3.21.*

Définition 3.26 *Etant donnée une exécution canonique $q_0 \xrightarrow{\tau_0} q_1 \xrightarrow{e_0} q_2 \xrightarrow{\tau_1} q_3 \xrightarrow{e_1} q_4 \ldots$, une* position *dans ρ est n'importe quel état q_k, ou bien $q_{2k} + \tau$ avec $0 < \tau < \tau_k$. L'ensemble des positions dans ρ est totalement ordonné. Etant donnée une exécution ρ, q et q' deux positions dans ρ, nous dénotons le fait que q est « avant » q' dans ρ par $q < q'$.*

3.4.2 Graphes de régions

Dans cette section, nous définissons un *graphe de régions* [AD90, AD94] d'un automate temporisé $\mathcal{A}$. Le graphe de régions est une abstraction finie des automates temporisés d'un intérêt particulier.

Afin d'introduire le graphe de régions, nous commençons par rappeler une équivalence définie dans [AD90, AD94] sur les valuations d'horloges et son extension sur les états de $T_\mathcal{A}$. Nous dénotons par $\approx_t$ cette équivalence. Pour une horloge x_i, soit c_i la plus grande constante comparée avec x_i dans l'ensemble des gardes de E et des invariants de $\mathcal{I}$, et soit $\mathbf{c}$ la plus grande valeur des c_i. Pour $\tau \in \mathbb{R}_{\geq 0}$, $\lfloor\tau\rfloor$ dénote sa partie intégrale et $\mathsf{frac}(\tau)$ sa partie fractionnelle.

Définition 3.27 *Deux valuations d'horloges ν et ν' sont équivalentes, noté $\nu \approx_t \nu'$, si et seulement si ces conditions suivantes sont satisfaites :*

- $\lfloor \nu_i \rfloor = \lfloor \nu'_i \rfloor$ *ou bien* $\nu_i, \nu'_i > c_i$, *pour tout* $i \in \{1, \ldots, n\}$;
- $\mathsf{frac}(\nu_i) = 0$ *si et seulement si* $\mathsf{frac}(\nu'_i) = 0$, *pour tout* $i \in \{1, \ldots, n\}$ *avec* $\nu_i \leq c_i$;
- $\nu_i - \nu_j \sim c$ *si et seulement si* $\nu'_i - \nu'_j \sim c$, *pour tout* $i \neq j \in \{1, \ldots, n\}$ *où* $\sim \in \{<, =, >\}$ *et* $c \in \{0, \ldots, \mathbf{c}\}$.

Le relation d'équivalence $\approx_t$ *est étendue aux états de* $T_{\mathcal{A}}$ *comme suit :*

$$q = (l, \nu) \approx_t q' = (l', \nu') \text{ si et seulement si } l = l' \text{ et } \nu \approx_t \nu'$$

Notons qu'étant donné un automate temporisé $\mathcal{A}$, le nombre de classes d'équivalence induites par la relation $\approx_t$ est fini.

Remarque 3.5 *Lorsque nous considérons les automates temporisés non diagonaux, la troisième condition de la Définition 3.27 est remplacée par ce qui suit :*

- $\mathsf{frac}(\nu_i) \leq \mathsf{frac}(\nu_j)$ *si et seulement si* $\mathsf{frac}(\nu'_i) \leq \mathsf{frac}(\nu'_j)$, *pour tout* $i \neq j \in \{1, \ldots, n\}$ *avec* $\nu_i \leq c_i$, $\nu_j \leq c_j$.

Nous utilisons $[\nu]$ (respectivement $[q]$) pour noter la classe d'équivalence dans laquelle appartient ν (respectivement q). Une *région* est une classe d'équivalence $[q]$. L'ensemble de toutes les régions est noté R. Une région $[q]$ est *fermée* si $q + \tau \not\approx_t q$ pour n'importe quel $\tau > 0$, sinon elle est *ouverte*. Une région $[q]$ est *non bornée* si elle satisfait $q = (l, \nu)$ avec $\nu_i > c_i$ pour tout $i \in \{1, \ldots, n\}$.

Nous définissons dans ce qui suit le graphe de régions d'un automate temporisé $\mathcal{A}$. Ce graphe est le quotient du système de transitions abstraites $T^{ta}_{\mathcal{A}}$ sur la relation d'équivalence sur les valuations d'horloges $\approx_t$.

Définition 3.28 *Etant donné un automate temporisé* $\mathcal{A} = (L, X, \Sigma, E, \mathcal{I}, \mathcal{L})$. *Le* graphe de régions $R_{\mathcal{A}} = (R, F)$ *est un graphe étiqueté fini où*

- *l'ensemble des nœuds est égal à l'ensemble des régions* R ;
- *l'ensemble des arcs* F *est composé d'arcs de deux types :*
  - $r \xrightarrow{t} r'$ *est un* arc temporel *s'il existe deux états* $q \in r$, $q' \in r'$ *et une transition temporelle* $q \xrightarrow{\tau} q'$ *dans* $T_{\mathcal{A}}$, *pour un certain* $\tau \in \mathbb{R}_{\geq 0}$,
  - $r \xrightarrow{e} r'$ *est un* arc de changement d'état *s'il existe deux états* $q \in r$, $q' \in r'$ *et une transition de changement d'état* $q \xrightarrow{e} q'$ *dans* $T_{\mathcal{A}}$, *pour un certain* $e \in E$.

Lemme 3.1 ([AD94]) *Le nombre de régions d'un automate* $\mathcal{A} = (L, X, \Sigma, E, \mathcal{I}, \mathcal{L})$ *est borné par* $|X|!.2^{|X|}.\Pi_{x \in X}\,(2\mathbf{c} + 2)$.

Notation 3.4 *Etant données* r, r' *deux régions de* $R_{\mathcal{A}}$, *un arc (temporel ou de changement d'état) entre* r *et* r' *est uniformément dénoté par* $r \to r'$.

Etant donnée une exécution $\rho = (q_k)_{k \in K}$ de $T_{\mathcal{A}}$, nous dénotons par $[\rho]$ le chemin correspondant $([q_k])_{k \in K}$ dans $R_{\mathcal{A}}$. On dit qu'un chemin ρ_R dans $R_{\mathcal{A}}$ est *canonique* (respectivement *initial*) si $\rho_R = [\rho]$ pour une certaine exécution canonique (respectivement initiale) ρ de $T_{\mathcal{A}}$.

3.4.3 Travaux sur le modèle original

Plusieurs travaux ont été élaborés sur le modèle d'Alur et Dill. Ce modèle a été étudié sous multiples aspects tels que la déterminisation (les *Event-Clock Automata* [AFH94]), la minimisation [ACH+92] et le pouvoir d'expression des horloges [ACH94], et d'autre part ce modèle a été vivement utilisé dans la spécification et la vérification de systèmes temporisés ([ACD93], les *Timed Safety Automata* [HNSY94]). Plus tard, les contraintes diagonales d'horloge qui comparent les valeurs de deux horloges, et les mises à jour constantes (les *Updatable Timed Automata* [Bou02, BDFP04]) qui initialisent la valeur d'une horloge à une certaine constante, ont été introduites. Dans un but d'optimisation, des travaux ont proposé de réduire le nombre d'horloges. Une première approche consiste à essayer de repérer les horloges non utilisées et les horloges qui sont toujours égales, via la notion d'horloges actives [DY96]. Une autre approche consiste à définir un ensemble d'horloges non plus globales à tout le système mais locales à chacun des états de contrôle de l'automate, comme dans les DTA's (*Dynamic Timed Automata* [CdO95b, Loh02]). Des travaux ont proposé une extension radicale du modèle en introduisant des horloges ne progressant plus nécessairement toutes au même rythme, elle peuvent consister en horloges *continues* et horloges *discrètes*. Ces automates sont appelés automates *hybrides* [ACHH93, NSY93, ACH+95]. Afin de prendre en compte les actions avec durées non nulles, une approche consiste à modéliser chaque action par une transition non instantanée. Ainsi, le modèle des *Timed Automata with non-Instantaneous Actions* a été introduit [BFT01].

3.4.4 Quelques sous-classes et extensions des automates temporisés

Timed Safety Automata

Le modèle des *Timed Safety Automata* (*TSA's*) est introduit dans [HNSY94] notamment pour pouvoir spécifier et vérifier les systèmes temps-réel «implémentables», dits *divergence-safe real-time systems*[3], dans lesquels le temps ne peut que diverger (c'est-à-dire ne comportant que des séquences d'exécution non-Zénon (Voir Section **??**)). Les TSA's sont une version non standard du modèle original des automates temporisés (introduit dans [AD94]) pour les raisons suivantes :

- Ils accordent les contraintes temporelles aussi bien dans les états et les transitions du système. La différence entre les TSA's et les automates temporisés impliquant les contraintes sur les états de [Alu99] est que les invariants d'états des TSA's sont fermés en arrière (*past-closed*), c'est-à-dire

$$\forall s \in S, v \in \Xi(H), t \in \mathbb{R}^+. (v + t \models I(s)) \Rightarrow (v \models I(s))$$

- Ils sont interprétés à travers un temps faiblement monotone (*weakly monotonic time* [AH92]) permettant l'expression des exécutions simultanées d'événements *ponctuels* dans le temps.

- Tous les états d'un TSA sont Büchi-acceptés, afin de supporter les systèmes divergence-safe.

Bien que le modèle des TSA's est moins expressif par rapport au modèle original de [AD94], sa simplicité a entraîné son adoption dans plusieurs outils de vérification d'automates temporisés, tels que Uppaal [LPY97] et Kronos [Yov97].

Event-Clock Automata

La classe des *Event-Clock Automata* englobe les *Event-Recording Automata* et les *Event-Predicting Automata* [AFH94]. Ce modèle a été introduit dans le but de rendre décidable le problème universel des TA's et ainsi le problème d'inclusion des langages. Un automate Event-Recording est un TA contenant, pour chaque action (ou symbole d'entrée) a, une horloge qui enregistre l'instant de la dernière occurrence de a, donc on attribue au départ à chaque action l'horloge correspondante. La notion duale d'enregistrement est la prévision, un automate Event-Predicting est un TA qui contient des horloges qui prévoient l'instant de la prochaine occurrence d'un événement.

[3] Dans [Yov93], ces systèmes sont appelés *systèmes bien temporisés*.

Updatable Timed Automata

L'introduction des automates temporisés avec mises à jour ou *Updatable Timed Automata* [BDFP00, BDFP04, Bou02] est motivée entre autre par l'étude des automates permettant l'affectation de valeurs non nulles aux horloges, l'affectation de valeurs appartenant à un intervalle temporel de façon non déterministe, et l'attribution à une horloge la valeur d'une autre. Ainsi, cette extension a permis par exemple la spécification de protocoles s'appuyant sur les mises à jour comme ABR (*Available Bit Rate* [BF99]).

Dynamic Timed Automata

Dans les *Dynamic Timed Automata* (*DTA's*) [CdO95b, Loh02], l'approche consiste à définir un ensemble d'horloges non plus globales à tout le système mais locales à chacun des états de l'automate. L'automate est dit *dynamique* car le nombre d'horloges peut varier d'un état à un autre.

Timed Automata with non-Instantaneous Actions

Les *Timed Automata with non-Instantaneous Actions*, introduit dans [BFT01], ont été proposés pour permettre la spécification des systèmes temps-réel de manière plus naturelle en contournant l'hypothèse de l'atomicité temporelle des actions (où les actions sont de durée nulle). Chaque transition dans se modèle prend un certain temps pour être tirée. Il a été prouvée dans que les Timed Automata with non-Instantaneous Actions sont plus expressifs que les automates temporisés et moins expressifs que les automates temporisés avec des ε-transitions.

Timed Automata with Deadlines

Les *Timed Automata with Deadlines* (*TADs*) ont été initialement introduits par Bornot and Sifakis [BS98, BST97] afin d'exprimer l'urgence des actions. Cette notion existe déjà pour les algèbres de processus qui supposent en général que les actions cachées (intériorisée par l'opérateur `hide`) sont urgentes. Or, l'absence d'un opérateur approprié dans le modèle des automates temporisés ne permet pas à ce dernier à spécifier l'urgence des actions, d'où l'introduction des TADs.

L'urgence dans les TADs est supportée par l'ajout aux transitions des automates temporisés d'une autre contrainte appelée *échéance* (*deadline*) qui, au moment de sa satisfaction, la transition correspondante doit être tirée tout en empêchant le temps à progresser.

3.5 Conclusion

Nous avons donné dans ce chapitre des notions préliminaires sur les formalismes liés à notre travail. Ainsi, les modèles des algèbles de processus, en l'image de Basic LOTOS et D-LOTOS, des réseaux de Petri, des systèmes de transitions étiquetées maximales et des automates temporisés sont introduits.

Chapitre 4

Vérification logique basée sur la maximalité

Dans ce chapitre, nous montrons qu'une adaptation d'une approche de vérification, à savoir l'approche logique, par le passage vers l'utilisation d'une sémantique de vrai parallélisme, est directe.

4.1 Introduction

Par vérification formelle, nous entendons toute technique permettant de confronter un système (sa description opérationnelle) à ses spécifications (aux propriétés que l'on attend de lui). Ce type d'approches nécessite trois ingrédients :

1. Une description opérationnelle du système (son graphe de comportement), générée à partir d'un modèle de spécification.
2. Un langage de spécification permettant d'exprimer les propriétés du système que l'on souhaite vérifier.
3. Une procédure de décision qui permet de contrôler la conformité entre la description opérationnelle du système et sa spécification, c'est-à-dire une procédure qui permet de vérifier que le système satisfait effectivement les propriétés que l'on attend de lui.

La relation entre ces trois éléments est la suivante : Après avoir spécifié formellement un système dans un modèle de spécification, on génère les comportements possibles exprimés dans un modèle sémantique. Ensuite, à l'aide de la procédure de vérification,

les propriétés de bon fonctionnement du système peuvent être vérifiées sur le modèle généré.

Il existe plusieurs classes de méthodes de vérification formelle divisées selon le moyen utilisé pour exprimer les propriétés attendues du système. On distingue deux grandes classes : celle basée sur l'approche comportementale et celle basée sur l'approche logique.

L'approche comportementale Dans cette approche, la description opérationnelle du système (son graphe de comportement) ainsi que sa spécification, sont tous les deux exprimés par des comportements. La procédure de décision revient alors à décider de l'équivalence entre deux graphes. De nombreuses relations d'équivalence ont été proposées pour la comparaison et l'analyse des systèmes concurrents, allant de l'équivalence langage à l'équivalence observationnelle, en passant par les modèles de refus et les équivalences de test. Cette diversité s'explique d'une part, par la variété des propriétés spécifiques aux systèmes étudiés, et d'autre part, de la difficulté de définir formellement une sémantique des systèmes de processus.

L'approche logique Dans cette approche, les propriétés attendues du système sont exprimées par des assertions dans une logique, par exemple la *logique temporelle*. Dans ce contexte on distingue deux approches : une basée sur la preuve de théorèmes (*theorem proving* ou *proof checking*) et une autre basée sur le contrôle (évaluation) de modèle (*model checking*).

i. Méthodes basées sur la preuve

Les méthodes basées sur la preuve consistent à modéliser le programme dans un système formel, puis prouver la correction du programme avec des raisonnements syntaxiques. L'avantage de ces méthodes est qu'elles permettent de traiter des systèmes ayants un nombre infini d'états. En plus, l'intuition humaine peut guider le processus de vérification. En revanche, elles ne peuvent pas être complètement automatisées et nécessitent beaucoup d'effort et d'ingéniosité humaine.

ii. Méthodes basées sur l'évaluation

Les techniques basées sur l'évaluation, bien que restreintes à des systèmes ayant un nombre fini d'états, permettent une vérification (relativement) simple et efficace, qui s'avère particulièrement utile dans les premières phases du processus de conception, quand les erreurs sont susceptibles d'être plus fréquents.

Dans cette classe de méthodes, l'application à vérifier est d'abord décrite dans un langage de spécification de parallélisme de haut niveau ayant une sémantique opérationnelle bien définie. Ensuite, cette description est traduite vers un modèle sous-jacent, qui est souvent un système de transitions étiquetées (STE), c'est-à-dire un graphe (ou automate) contenant éventuellement avec un certain niveau d'abstraction, tous les comportements possibles du programme. Finalement, les propriétés de bon fonctionnement du système, exprimées dans une logique temporelle sont vérifiées sur le modèle à l'aide de model checkers. Dans ce cas, la procédure de décision consiste à vérifier si la description opérationnelle du système (le graphe de comportement) est un modèle des formules qui expriment les propriétés de correction du système. Cette procédure de décision revient alors à effectuer du «contrôle de modèle» (*model checking*) permettant d'associer à chaque formule l'ensemble des états du modèle qui la satisfont. Ces méthodes offrent l'avantage d'être complètement automatisables, mais souffrent du problème de l'explosion combinatoire d'espace d'états des comportements possibles du système. Pour réduire l'impacte de ce problème, plusieurs solutions ont été proposé : utiliser une sémantique de vrai parallélisme, des structures compactées au niveau de la présentation (*Symbolic Model Checking* [BCM+92, McM92] utilisant la structure des BDDs [Bry86]) ou des algorithmes de réduction à la volée au niveau génération.

Pour notre part, nous avons adopté une approche logique de vérification utilisant une logique temporelle vu que les logiques temporelles sont bien adaptées pour spécifier les propriétés, car elles permettent d'obtenir des spécifications abstraites et modulaires du système. L'abstraction signifie l'indépendance de la spécification par rapport à toute implémentation : les propriétés requises sont exprimées séparément, sans indiquer la manière dont elles sont implémentées. La modularité signifie qu'une spécification est facilement modifiable en rajoutant, enlevant ou modifiant une des propriétés ; de plus, le processus de vérification sur un modèle peut aussi être effectué de façon modulaire, en vérifiant chaque propriété séparément.

4.2 Model checking

4.2.1 Principe

La démarche du model checking consiste à vérifier si un spécification satisfait une propriété de bon fonctionnement, par la confrontation de l'automate (modèle) associé à cette spécification à une formule de logique temporelle, exprimant la propriété voulue vérifier. On doit noter que la vérification s'effectue sur un modèle du futur système, et

non sur le système lui-même.

L'avantage le plus important du model checking est qu'il est complètement automatisable, ce qui a engendré des outils appelés : model checkers.

Lorsqu'on souhaite vérifier les propriétés d'un système, il est nécessaire de pouvoir les exprimer dans un langage adéquat. La logique temporelle répond à ce besoin. Des formules simples de la logique temporelle permettent d'exprimer les propriétés usuelles des programmes parallèles.

4.2.2 Logique temporelle

Contrairement à la logique classique, où il existe une fonction d'interprétation unique à partir de laquelle il est possible de déduire la valeur de vérité de chaque formule ; dans la logique temporelle, la fonction d'interprétation est définie sur un graphe. Dans ce graphe, chaque nœud correspond à une fonction classique. A cause de l'introduction d'opérateurs temporels (**A** : quelque soit le chemin, **E** : pour un chemin donné, **G** : toujours, **F** : parfois, **X** : prochain et **U** : jusqu'à), l'interprétation d'une formule dans un nœud ne dépend pas seulement des connaissances de ce nœud, mais peut dépendre des connaissances des autres nœuds du graphe d'interprétation. Les opérateurs précédents sont ceux de la logique temporelle arborescente CTL [CES86, Eme90].

En prenant comme graphe d'interprétation (modèle) le système d'états-transitions associé à un programme, il est possible d'exprimer des propriétés sur les valeurs des formules logiques du programme à chaque état.

4.2.3 Expression de propriétés de bon fonctionnement

A cause de leur aspect critique, les programmes concurrents doivent être de très haut niveau, robustes, et doivent présenter des propriétés de bon fonctionnement. Certaines propriétés assurent que quelque chose de mauvais ne se produira jamais durant l'exécution du système. D'autres propriétés expriment le fait que quelque chose de bon se produira immanquablement durant l'exécution du système. La première classe de ces propriétés est appelée : *propriétés de sûreté* (propriétés d'invariance). L'autre classe est la classe des *propriétés de vivacité*. Généralement, les propriétés de bon fonctionnement des programmes concurrents sont inclues à l'une des deux grandes classes. Une propriété classique de sûreté est la propriété de l'exclusion mutuelle. Soit un ensemble de processus utilisant un objet commun. Cet objet est appelé : ressource critique. On dit que ces processus sont en exclusion mutuelle lorsque chacun d'eux a une section critique dans son programme. A un instant donné, un et un seul processus au plus exécute

sa section critique. C'est-à-dire que durant toute l'exécution du système (un ensemble de processus), on ne doit pas avoir le cas où plus d'un processus en train d'exécuter sa section critique. Alors le problème de l'exclusion mutuelle entre deux processus peut être exprimé en logique temporelle comme suit : $\mathbf{G}(\neg(CS_1 \wedge CS_2))$: les deux processus p_1 et p_2 ne peuvent jamais être en même temps en section critique.

Une autre propriété de sûreté est l'absence de blocage dans un ensemble de processus. Cette propriété exprime le fait qu'à un moment donné, au moins un processus n'est pas en attente. Cette propriété peut être exprimée par : $\mathbf{G}(prêt_1 ... prêt_n)$.

Les propriétés de vivacité signifient qu'après l'initialisation du système, alors dans cet état et ans tous les états suivants, si un requête est lancée, cette requête sera satisfaite plus tard. Ces propriétés sont exprimées en général sous la forme : $\mathbf{G}(req_i \Rightarrow \mathbf{F}\ done_i)$. La terminaison est un exemple d'une propriété de vivacité. Elle exprime que toute opération qui commence son exécution doit se terminer à un moment donné. On peut exprimer cela en logique temporelle comme suit : Pour un processus P_i : $\mathbf{G}(atC_i \Rightarrow \mathbf{F}\ end_i)$: si un processus P_i exécute un opération C_i, alors il doit la terminer à un instant donné.

Une autre classe de propriétés de vivacité est la classe des *propriétés d'équité*. Quand un processus demande l'entrée dans sa section critique, son désir sera satisfait éventuellement à un moment donné, on est ici devant une propriété d'équité forte (réponse à la persistance), on peut exprimer cela par : $\mathbf{GF}(req_i) \Rightarrow \mathbf{GF}(done_i)$. Maintenant, si un processus est prêt à être exécuté, alors il sera exécuté à un moment donné, c'est un exemple de propriété d'équité faible (réponse à l'insistance) : $\mathbf{FG}(req_i) \Rightarrow \mathbf{GF}(done_i)$.

4.3 Model checking basé sur la sémantique de maximalité

Dans cette section, basée principalement sur les résultats de [SB03a, SB04, SB05], nous nous intéressons à une approche de vérification formelle basée sur une sémantique de vrai parallélisme pour exprimer la concurrence. Dans notre contexte, nous adoptons la sémantique de maximalité. Nous avons vu dans la Section 2.3 que cette sémantique nous permet de distinguer entre exécutions séquentielles et exécutions parallèles d'actions. Ainsi, nous montrons comment adapter un algorithme de model checking pour pouvoir vérifier d'autres propriétés ne pouvant pas être vérifiées dans une approche de vérification basée sur l'entrelacement des actions concurrentes.

4.3.1 Logique CTL

CTL est une logique temporelle propositionnelle de branchement utilisée fréquemment dans les techniques de model checking [CE81, BAMP83, CES86, BBL+99]. CTL contient les opérateurs temporels usuels : **X** (le prochain instant), **F** (éventuellement), **G** (toujours) et **U** (jusqu'à) qui doivent être immédiatement précédés par l'un des quantificateurs de chemin qui sont **A** (pour tous les chemins) ou **E** (il existe un chemin). Par exemple, **AG**p est satisfaite dans un état si pour tous les chemins à partir de cet état, p est toujours vrai.

La logique temporelle CTL permet d'exprimer des formules sur les états, notées φ, et des formules sur les chemins, notées ω. Leur syntaxe est comme suit :

$$\varphi ::= p \mid True \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{A}\omega \mid \mathbf{E}\omega$$

$$\omega ::= \mathbf{F}\varphi \mid \mathbf{G}\varphi \mid \mathbf{X}\varphi \mid \varphi\mathbf{U}\varphi$$

où $p \in AP$ est une proposition atomique.

Nous pouvons distinguer huit opérateurs de base dans la logique CTL : **AX**, **EX**, **AG**, **EG**, **AF**, **EF**, **AU** et **EU** définis comme suit :

- **AX**f est satisfaite dans un état si la formule f est satisfaite dans tous ses successeurs.
- **EX**f est satisfaite dans un état si au moins un de ses successeurs satisfait la formule f.
- Un état satisfait **AF**f si sur chaque chemin issu de cet état, il y a au moins un état qui satisfait f.
- Un état satisfait **EF**f s'il existe un chemin issu de cet état contenant au moins un état qui satisfait f.
- Un état satisfait **AG**f si sur tous les chemins à partir de cet état, f est toujours satisfaite.
- Un état satisfait **EG**f s'il existe un chemin issu de cet état où f est toujours satisfaite.
- **A**(f**U**f') est satisfaite dans un état si sur tous les chemins à partir de cet état f est toujours vérifiée jusqu'à un état qui vérifie f'.
- **E**(f**U**f') est satisfaite dans un état s'il existe un chemin à partir de cet état où f est toujours vérifiée jusqu'à un état qui vérifie f'.

4.3.2 Sémantique de maximalité et vérification logique

Type de propriétés à vérifier

Dans la section précédente nous avons vu le genre de propriétés qu'on pouvait exprimer en utilisant la logique temporelle CTL ; le raisonnement portait sur des propositions logiques appartenant aux états du système, et les propriétés qu'on voulait vérifier sont généralement divisées en deux grandes classes qui sont : la classe des propriétés de vivacité et celle des propriétés de sûreté [Lam83]. Evidemment, ce sont des propriétés classiques bien connues et plusieurs model checkers ont été développés pour les vérifier.

Pour notre part, grâce à la sémantique de maximalité et l'utilisation du modèle des systèmes de transitions étiquetées maximales, les informations incluses dans les états du modèle représentent les actions qui sont potentiellement en cours d'exécution. De ce fait, on peut exprimer des propriétés appartenant au classes citées antérieurement telles que *l'exclusion mutuelle* de manière plus naturelle, ainsi que de nouvelles propriétés qui portent sur les actions et leur exécution parallèle. L'expression de ces propriétés ne nécessite pas l'utilisation d'une nouvelle logique ou l'introduction de nouveaux opérateurs, car on peut utiliser la logique temporelle arborescente CTL et considérer les actions dans les états comme étant des formules atomiques. Cependant, ce qui change c'est l'intuition derrière les formules. A titre d'exemple, la formule $\mathbf{EF}\,(a \wedge b)$ où a et b sont des noms d'actions, signifie qu'il existe au moins un chemin dans lequel l'exécution en parallèle de a et b peut avoir lieu, de manière similaire on peut expliquer intuitivement toutes les formules de la logique CTL dont le modèle sur lequel elles vont être vérifiées est un système de transitions étiquetées maximales comme suit :

- $a \wedge b$ dans un état S signifie que a et b peuvent être exécutées en parallèle dans l'état S.
- $\neg a$ dans un état S signifie que l'exécution de a dans l'état S ne peut pas avoir lieu.
- $\mathbf{EX}a$ dans un état S_0 signifie qu'il existe au moins un chemin $(S_0, S_1, ...)$ où a pourra s'exécuter dans l'état S_1.
- $\mathbf{AX}a$ dans un état S_0 signifie que quelque soit le chemin $(S_0, S_1, ...)$ issu de l'état S_0, a pourra s'exécuter à l'état S_1.
- $\mathbf{E}[a\mathbf{U}b]$ dans un état S_0 signifie qu'il existe un chemin $(S_0, S_1, ..., S_k, ...)$ où b pourra s'exécuter dans l'état S_k et a pourra s'exécuter dans chaque état de ce chemin qui précède l'état S_k.

- $\mathbf{A}[a\mathbf{U}b]$ dans un état S_0 signifie que quelque soit le chemin issu de l'état S_0, il existe un état appartenant à ce chemin où b pourra s'exécuter et a pourra s'exécuter dans chaque état de ce chemin qui précède cet état.

Ainsi, pour exprimer des propriétés de sûreté ou de vivacité, on n'a plus besoin d'utiliser les formules logiques pour indiquer l'état d'évolution d'un processus, mais on raisonne directement sur les actions. En procédant ainsi, les propriétés seront plus faciles à exprimer et leur signification paraît plus naturelle.

Si nous prenons comme exemple d'exclusion mutuelle le problème des lecteurs-rédacteurs, en supposant qu'il existe une seule variable où un rédacteur écrit une information et un lecteur la lit, on sait que les accès à cette variable sont mutuellement exclusifs, donc au lieu d'exprimer l'exclusion mutuelle comme : $\mathbf{AG}\neg(atC_1 \wedge atC_2)$, on peut l'exprimer tout simplement par $\mathbf{AG}\neg(red\_ecrire \wedge lect\_lire)$, où :

- red_ecrire est l'action d'écriture du rédacteur.
- $lect\_lire$ est l'action de lecture du lecteur.
- atC_1 est une formule logique. Elle est vraie si le rédacteur est dans sa section critique.
- atC_2 est une formule logique. Elle est vraie si le lecteur est dans sa section critique.

Dans la Section 4.3.4, des exemples plus élaborés seront présentés par l'étude du problème du dîner des philosophes.

Remarque 4.1 *On a vu dans la structure des systèmes de transitions étiquetées maximales qu'à chaque action est associé un nom d'événement qui permet de distinguer entre plusieurs actions de même nom qui sont en exécution parallèle. Par conséquent, on peut avoir plusieurs actions du même nom dans un même état mais les noms des événements associés seront différents. En partant de ce point, on peut envisager des formules qui nous permettront de raisonner sur le nombre d'occurrences d'une action qu'on peut avoir en parallèle, c'est-à-dire vérifier le degré d'autoconcurrence d'une action.*

On peut utiliser la notation $a : 5$ pour dire qu'on peut avoir 5 occurrences de l'action a en parallèles, $a : 5$ sera donc considérée comme étant une proposition atomique ; ce qui évitera l'introduction d'un nouvel opérateur à la logique. On aura alors deux formes de propositions atomiques, soit de la forme a soit de la forme $a : n$ où n est une valeur entière.

En s'appuyant sur ces aspects intuitifs et en utilisant cette dernière notation, on peut exprimer de nouvelles propriétés telles que :

L'incompatibilité de deux actions On peut exprimer que a et b sont incompatibles par la formule

$$\mathbf{AG}\neg(a \wedge b)$$

c'est-à-dire qu'elles ne pourront jamais s'exécuter en parallèle. De manière similaire, vérifier que des actions peuvent s'exécuter en parallèle s'exprime comme suit :

$$\mathbf{EF}\,(a \wedge b \wedge ... \wedge z)$$

où $a, b, ...$et z sont des noms d'actions.

Le degré d'autoconcurrence d'une action Par exemple $\mathbf{EF}\,(a : n)$ est vraie s'il existe un état dans lequel on a n actions s'exécutant en parallèle et dont le nom est a.

Il est clair que ces propriétés n'auraient pas pu être exprimées sur un modèle d'entrelacement.

4.3.3 Algorithme de model checking

Après avoir vu le genre de propriétés qu'on pouvait exprimer en utilisant le modèle des systèmes de transitions étiquetées maximales pour la représentation des comportements possibles et la logique temporelle CTL comme langage de spécification des propriétés, dans ce qui suit nous illustrons la méthode d'évaluation des formules de la logique CTL sur le modèle des systèmes de transitions étiquetées maximales à travers l'adaptation de l'algorithme de model checking presenté dans [CES86]. Le choix de cet algorithme est fait à titre d'illustration, il est clair que différents algorithmes de model checking peuvent être adaptés au modèle des systèmes de transitions étiquetées maximales de manière similaire.

Fonctionnement de l'algorithme

Supposons qu'on a une structure (modèle) fini $M = (S, R, L)$, et une formule CTL p_0. Le but est de déterminer les états s de M où on a $M, s \models p$.

Cet algorithme est conçu pour être exécuté en étapes : la première étape traite toutes les sous-formules de p_0 de longueur 1, la seconde étape traite toutes les sous-formules de p_0 de longueur 2, et ainsi de suite... A la fin de la $i^{\text{ème}}$ étape, chaque état

sera étiqueté par l'ensemble de toutes les sous-formules de longueur i vraies dans cet état. Pour élaborer l'étiquetage à l'étape i, on a besoin des informations collectées dans les étapes précédentes. Par exemple, l'état s doit être étiqueté par la sous-formule $(q \wedge r)$ exactement si l'état s est étiqueté par q et r.

Pour la sous-formule $\mathbf{A}[q\mathbf{U}r]$, on aura besoin des informations sur les états successeurs de s ainsi que sur l'état s lui-même, puisque $\mathbf{A}[q\mathbf{U}r] = r \vee (q \wedge \mathbf{AXA}[q\mathbf{U}r])$. Initialement, $\mathbf{A}[q\mathbf{U}r]$ est ajoutée à tous les états déjà étiquetés par r. Ensuite, $\mathbf{A}[q\mathbf{U}r]$ va être propagée et ajoutée à tout état étiqueté par q dont tous ses successeurs sont étiquetés par $\mathbf{A}[q\mathbf{U}r]$.

De la même manière on raisonne pour $\mathbf{E}[q\mathbf{U}r]$.

On doit noter que les autres opérateurs modaux sont implicites et définis autant qu'abréviations :

$$\begin{aligned}
q \vee r &\equiv \neg(\neg q \wedge \neg r) \\
q \Rightarrow r &\equiv \neg q \vee r \\
q \Leftrightarrow r &\equiv (q \Rightarrow r) \wedge (r \Rightarrow q) \\
\mathbf{AX}q &\equiv \neg\mathbf{EX}\neg q \\
\mathbf{EF}p &\equiv \mathbf{E}(true\mathbf{U}p) \\
\mathbf{AG}p &\equiv \neg\mathbf{EF}\neg p \\
\mathbf{AF}p &\equiv \mathbf{A}(true\mathbf{U}p) \\
\mathbf{EG}p &\equiv \neg\mathbf{AF}\neg p
\end{aligned}$$

Algorithme

Entrée Une structure temporelle $M = (S, R, L)$ comme modèle sémantique ; et une formule p_0 écrite en CTL.

Sortie Ensemble d'états de M qui satisfont la formule p_0.

Début
pour $i = 1$ **à** $longueur(p_0)$ **faire**
pour chaque sous-formule p de p_0 de longueur i **faire**
cas forme de p **faire**
$p = P$: une proposition atomique :
/* ne rien faire */
$p = q \wedge r$:
pour chaque $s \in S$ **faire**
si $q \in L(s)$ **et** $r \in L(s)$ **alors**

ajouter $(q \wedge r)$ à $L(s)$;
fsi
fpour
$p = \neg q$:
pour chaque $s \in S$ **faire**
si $q \notin L(s)$ **alors**
ajouter $\neg q$ à $L(s)$;
fsi
fpour
$p = \mathbf{EX}q$:
pour chaque $s \in S$ **faire**
si $\exists$ successeur s' de s / $q \in L(s')$ **alors**
ajouter $\mathbf{EX}q$ à $L(s)$;
fsi
fpour
$p = \mathbf{A}[q\mathbf{U}r]$:
pour chaque $s \in S$ **faire**
si $r \in L(s)$ **alors**
ajouter $\mathbf{A}[q\mathbf{U}r]$ à $L(s)$;
fsi
fpour
pour $j = 1$ **à** $Card(S)$ **faire**
pour chaque $s \in S$ **faire**
si $q \in L(s)$ **et si** $\forall$ successeur s' de s /
$\mathbf{A}[q\mathbf{U}r] \in L(s')$ **alors**
ajouter $\mathbf{A}[q\mathbf{U}r]$ à $L(s)$;
fsi
fpour
fpour
$p = \mathbf{E}[q\mathbf{U}r]$:
pour chaque $s \in S$ **faire**
si $r \in L(s)$ **alors**
ajouter $\mathbf{E}[q\mathbf{U}r]$ à $L(s)$;
fsi
fpour
pour $j = 1$ **à** $Card(S)$ **faire**

pour chaque $s \in S$ **faire**
si $q \in L(s)$ **et si** $\exists$ successeur s' de s /
$\mathbf{E}[q\mathbf{U}r] \in L(s')$ **alors**
ajouter $\mathbf{E}[q\mathbf{U}r]$ à $L(s)$;
fsi
fpour
fpour
fcas
fpour
fpour
Fin

Cette version de l'algorithme possède une complexité temporelle linéaire en fonction de la longueur de la formule à vérifier et quadratique en fonction de la taille de la structure M [Eme90].

Puisqu'on a choisi la logique CTL comme langage d'expression des propriétés, et puisque les informations sur lesquelles on veut raisonner sont incluses dans les états du système de transitions étiquetées maximales, alors on remarque bien que cet algorithme peut s'adapter à notre étude pour la vérification des propriétés sur les systèmes de transitions étiquetées maximales. Pour cela, il suffit de considérer les actions dans les états du système de transitions étiquetées maximales à la place des propositions atomiques et rajouter à l'algorithme le cas où p est de la forme $q : n$ comme suit :

cas $P = q : n$:
pour chaque $s \in S$ **faire**
si $\exists\, x_1, x_2, ..., x_n$ / $x_1, x_2, ..., x_n \in L(s)$
et $act\,(x_1, s) = q$, $act\,(x_2, s) = q$, ..., $act\,(x_n, s) = q$
et $card\,(\{x_1, x_2, ..., x_n\}) = n$ **alors**
ajouter $q : n$ à $L(s)$;
fsi
fin pour

tel que $x_1, x_2, \ldots, x_n$ sont des noms d'événements, q est une action et act est une fonction qui reçoit un nom d'évé nement et un état et elle retourne le nom d'action associé à l'événement donné comme paramètre dans cet état.

Cet algorithme adapté est implanté dans l'outil `LotoStem` qui fait partie de l'environnement `FOCOVE` [SBBA08] développé dans notre laboratoire.

4.3.4 Exemple

Cette section présente nos résultats de l'étude du problème du dîner des philosophes [Dij71]. Les spécifications formelles détaillées de ce problème en Basic LOTOS peuvent être trouvées dans [SB03b].

Exclusion mutuelle

L'exclusion mutuelle concerne l'utilisation de chaque fourchette (ressource non partageable). Pour deux philosophes, la propriété s'exprime par :

```
A G (not (Philo1Prendf1 and Philo2Prendf1) and
     not (Philo1Prendf2 and Philo2Prendf2))
```

Nous avons vérifié le cas de deux, trois et quatre philosophes en utlisant l'outil `LotoStem`, et nous avons eu comme résultat la satisfiabilité des trois spécifications.

Abscence de l'interblocage

Le cas de l'interblocage se produit lorsque chaque processus du système possédant une ressource (fourchette dans cet exemple) demande une ressource déjà allouée à un autre processus. Les requêtes des processus forment ainsi une attente circulaire. L'absence de l'interblocage stipule que le système pourra toujours progresser dans le futur, et ne sera jamais bloqué. Cette propriété s'exprime par : `A G((E X true) or delta)`.

Cette expression CTL signifie que chaque état du système aura au moins un état successeur. Dans le cas contraire, il faut que la derni ère action qui est potentiellement en cours d'exécution dans cet état, selon la sémantique de maximalité, soit l'action *delta* (δ) qui signifie la terminaison d'un processus avec succès. Les résultats obtenus sont les suivants :

- Dans un premier temps nous avons écrit les spécifications répondant à l'hypothèse que les fourchettes sont prises dans le même ordre (élimination d'une des conditions nécéssaires de l'interblocage). Ce résultat est confirmé par l'outil `LotoStem` par la vérification de la satisfiabilité de la formule pr écédente.

- Dans un deuxième temps, nous avons spécifié la demande des ressources dans n'importe quel ordre. Dans ce cas, la formule n'est pas vérifiée, et des états d'interblocage ont été détectés [SB03b].

Absence de famine

On peut exprimer qu'à chaque fois qu'un philosophe veut manger alors il viendra un moment où il pourra le faire. Pour un philosophe i, l'absence de famine peut être exprimée en CTL par : `A G(Philo_i_VeutManger => A F philo_i_mange)`.

4.4 Conclusion

Dans ce chapitre, et après avoir évoqué succinctement le principe d'une approche de vérification formelle, à savoir la vérification logique, nous avons discuté l'intérêt du modèle des systèmes de transitions étiquetées maximales pour la vérification des propriétés du parallélisme. Cet apport a été principalement induit par la présence des informations de maximalité liées aux états et aux transitions. En particulier, nous avons montré comment ces informations facilitent l'écriture de propositions atomiques exprimant les propriétés à vérifier. Une attention particulière a été portée sur la lecture naturelle et intuitive de ces propriétés. A titre d'exemple, nous avons souligné la vérification du degré de parallélisme en général et de l'autoconcurrence en particulier dans une application concurrente donnée. Dans le but de concrétiser cette étude, nous avons montré comment un algorithme classique de model-checking [CES86] peut être adapté de manière directe au modèle des STEMs. Concernant cette étude, nous avons adopté une approche d'évaluation globale.

Pour clarifier les idées, nous avons présenté quelques résultats de la vérification du problème du dîner des philosophes. La validation des résultats a été réalisée par l'utilisation de l'outil `LotoStem` de l'environnement de vérification formelle `FOCOVE` (pour *FOrmal COncurrency Verification Environment*)[1].

[1] `http://www.focove.new.fr`

Deuxième partie

Contributions

Chapitre 5

Durational Action Timed Automata(*)

Ce chapitre, qui s'appuie en grande partie sur les résultats de [BS05] et de [SB06], propose deux l'étude de deux considérations liées à la spécification des systèmes temps-réel. La première étant la spécification de systèmes ayant des actions temporellement et structurellemnt non atomiques. La deuxième est la spécification de systèmes temps-réel proprement dits, c'est-à-dire des systèmes dont leur comportement est contraint par le temps par la prise en considération de contraintes temporelles et de contraintes d'urgence. Pour cela, nous proposons respectivement la définition des structures de DATA (*Durational Action Timed Automata*) et leur extension DATA*.

5.1 Modèle des DATA's

Dans cette section, nous proposons une méthode permettant la prise en compte de la non-atomicité temporelle et structurelle des actions dans les automates temporisés sans passer par l'éclatement des actions, grâce au modèle des DATA (pour Durational Action Timed Automata) qui sera défini. Cependant, nous montrerons que les DATA's sont structurellement une sous-classe des automates temporisés [AD90, AD94, Alu99], néanmoins, la différence à souligner est celle qui concerne la sémantique associée au modèle.

L'intérêt d'un tel modèle est la représentation des comportements infinis induits par les transitions temporelles tel qu'il est montré par l'exemple suivant. Considérons l'expression de comportement $G = a; b; stop$. En supposant que les actions sont ici de durées nulles, le comportement de G est donné par la Figure 5.1.(a). Cependant, s'il

existe exactement 6 unités de temps entre les exécutions de a et de b, et en prenant en compte les transitions temporelles, le délai entre a et b peut être exprimé par n transitions (n peut tendre vers l'infini) comme cela est montré par la Figure 5.1.(b). Un moyen pour modéliser ce comportement d'une manière concise consiste à se servir du modèle des automates temporisés, en regroupant ainsi les transitions temporelles dans les états. Le comportement infini de G peut être représenté par l'automate temporisé de la Figure 5.1.(c).

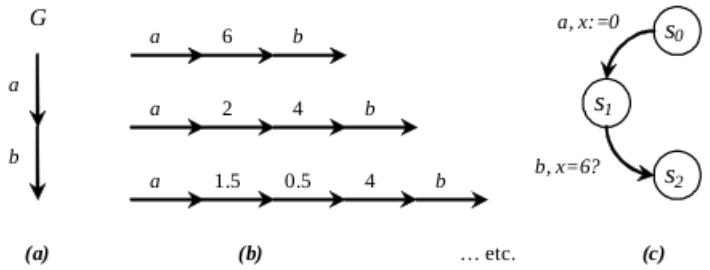

FIGURE 5.1 – Comportement infini de G

Dans la Figure 5.1.(c), le délai s'écoulant entre les occurrences des actions a et b est pris en compte par l'initialisation de l'horloge x lors de l'occurrence de l'action a et l'association d'une contrainte portant sur la transition relative à l'occurrence de l'action b.

5.1.1 Intuition

Le modèle des automates temporisés est construit en se conformant à l'hypothèse d'atomicité structurelle et temporelle des actions (actions indivisibles et de durées nulles).

Prenons l'automate temporisé de la Figure 5.2.(a) contenant une seule horloge x. L'état initial de cet automate est s_0. Après l'exécution instantanée de l'action a, l'horloge x va être remise à zéro, et l'automate passe à l'état s_1. Le système doit séjourner dans cet état 5 unités de temps, car il ne peut exécuter b que si l'horloge x atteint 5. Si les actions n'étaient pas de durée nulle en supposant par exemple que les durées respectives de a et b sont 2 et 10, la question qui se pose est comment parvenir à capturer l'exécution éventuelle de l'action a dans l'état s_1. Une alternative consiste à mentionner une action proprement dite à l'aide de deux événements : son début et sa terminaison. Certes, l'utilisation des horloges dans ce modèle permet le chevauchement des délais entre événements, toutefois, il ne permet pas intelligiblement le chevauchement de plusieurs actions proprement dites. Cela nous motive à considérer chaque

transition étiquetée par une action comme le début de l'exécution de cette dernière, et de garder dans certains états futurs une information sur l'exécution possible de cette action. Ainsi, nous pouvons par exemple remplacer chaque étiquette a_i par $début(a_i)$ (ou $a_i \uparrow$) et avoir le comportement de la Figure 5.2.(b).

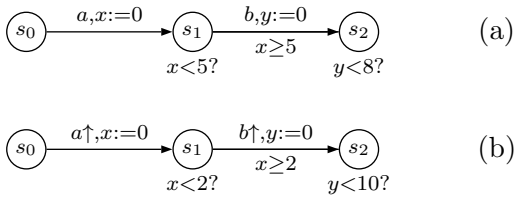

FIGURE 5.2 – Considération des débuts d'exécution des actions dans les automates temporisés

Le comportement de la Figure 5.2.(b) parait correct, mais considérons la situation suivante. Si le système passe à l'état s_1, il n'a que 2 unités de temps pour le quitter comme mentionne l'invariant $x < 2$. Donc, le système est obligé à lancer l'exécution de l'action b à l'instant $x = 2$, ce qui semble anormal vu que la contrainte liée à la transition $début(b)$ offre à b l'intervalle $[2, +\infty[$ pour entamer son exécution.

Considérons l'exemple d'un système S se résumant en deux sous-systèmes S_1 et S_2 s'exécutant en parallèle et se synchronisant sur une action d. Le sous-système S_1 exécute l'action a suivie de d, tandis que S_2 exécute b puis d. Si on suppose que toutes les actions ont une durée nulle, S aura le comportement de la Figure 5.3.(a).

Supposons maintenant que les actions a, b et d ont les durées respectives de 10, 12 et 4. Une approche possible d'exprimer les durées est de considérer chaque action, dans les automates temporisés classiques, comme deux évènements : début et fin. Le symbole $\uparrow$ représente l'évènement *début* tandis que $\downarrow$ représente l'évènement *fin*. Le comportement de S pourra être exprimé en terme d'un TA dans la Figure 5.3.(b). L'idée qui consiste à représenter une action par son début et sa fin d'exécution peut très bien modéliser la notion de durée. Toutefois, ce mécanisme affecte l'automate résultant en éclatant sa taille ainsi que son ensemble d'alphabet.

Une autre approche consiste à modéliser les actions de durées non nulles par des transitions non instantanées, c'est le cas des *Timed Automata with non-Instantaneous Actions* [BFT01, Tes04]. Dans ce modèle, chaque transition est étiquetée, en plus du nom d'action, par deux ensembles de remise à zéro, une remise à zéro de début de franchissement (*initiation reset*) et une remise à zéro de fin de franchissement (*completion reset*), ainsi que deux contraintes correspondantes au début et à la fin de franchissement,

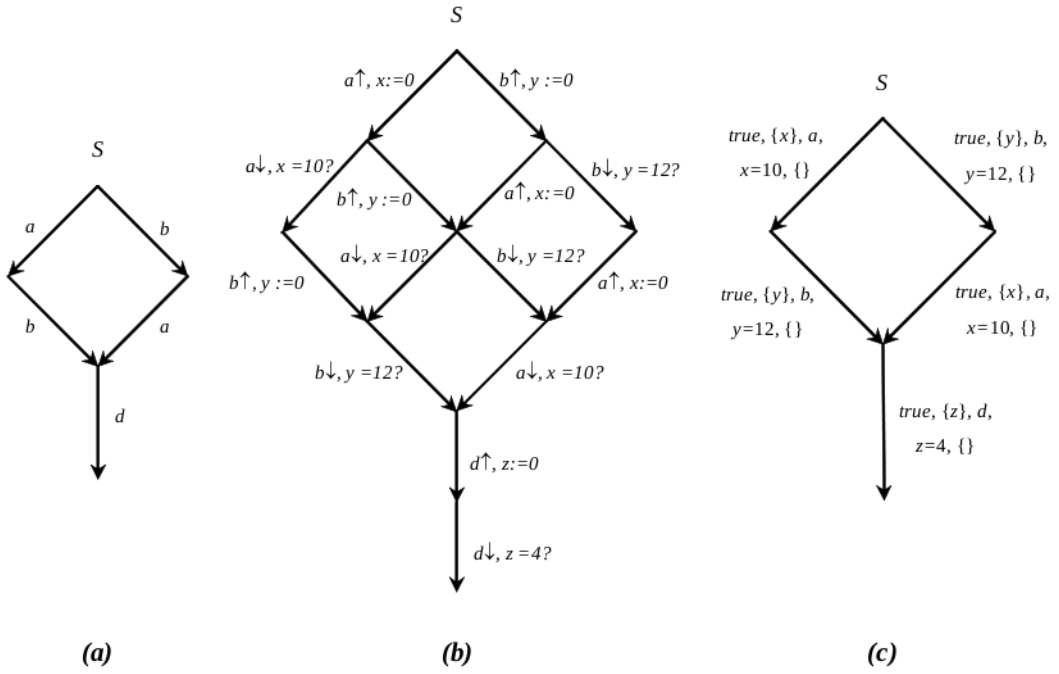

FIGURE 5.3 – Comportement de S

initiation constraint et *completion constraint*. Selon la sémantique du modèle [BFT01], les transitions sont indivisibles. Ceci impose l'atomicité structurelle des actions (les actions sont indivisibles), ce qui contraint l'exécution des actions concurrentes. Une modélisation du système S par le modèle des Timed Automata with non-Instantaneous Actions est donnée par la Figure 5.3.(c), où il est clair que l'exécution parallèle des actions a et b ne peut être capturée dans ce modèle.

L'idée de modéliser les durées associées aux actions peut être inspirée de la sémantique de maximalité dans laquelle une transition représente le début d'exécution d'une action. Dans l'état résultant on dit que l'action est éventuellement en cours d'exécution, aucune conclusion ne peut être tirée en ce qui concerne la fin de son exécution, cependant cette information peut être déduite dans un état ultérieur dans lequel une action qui lui est causalement dépendante est exécutée. L'association de durées explicites aux actions va nous permettre d'exprimer et le début et la fin d'exécution des actions. Considérons l'exemple du système S précédent. A partir de l'état initial s_0, les deux actions a et b peuvent commencer leur exécution indépendamment l'une de l'autre. Puisque nous pouvons avoir le cas où ces deux actions s'exécutent en parallèle, nous allons attribuer à chacune d'elles une horloge, x et y respectivement, pour distinguer leurs occurrences. Donc, à partir de l'état s_0, les deux transitions suivantes sont pos-

sibles : $s_0 \xrightarrow{a,\ x:=0} s_1$ et $s_0 \xrightarrow{b,\ y:=0} s_2$. Une transition étiquetée par a désigne le début d'exécution de l'action a, l'horloge qui lui est associée comptabilise l'évolution dans le temps de cette action.

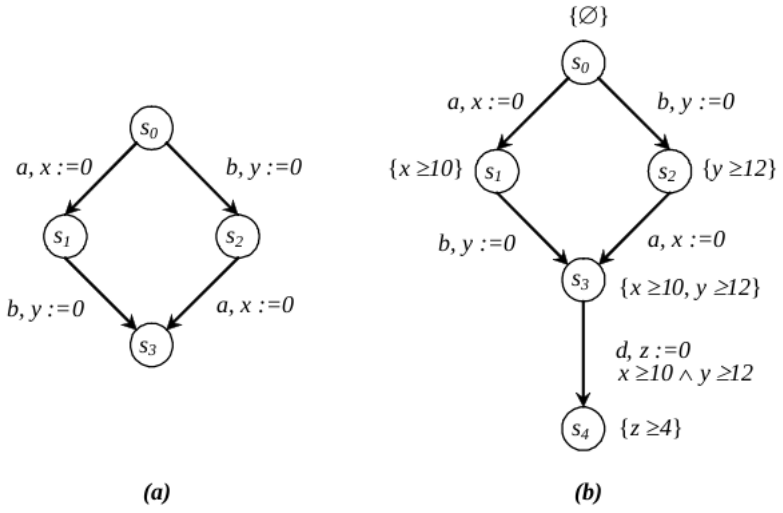

FIGURE 5.4 – Comportement de S utilisant les conditions de terminaison

En suivant le même raisonnement, les deux transitions suivantes sont possibles : $s_1 \xrightarrow{b,\ y:=0} s_3$ et $s_2 \xrightarrow{a,\ x:=0} s_3$. Le comportement du système S jusqu'ici est illustré par la Figure 5.4.(a). A partir de l'état s_3, l'action d ne peut évidemment commencer son exécution que si les deux actions a et b ont terminé leur exécution. Donc, la transition d ne peut être tirée que si une condition portant sur les exécutions de a et de b est satisfaite. Cette condition, appelée *condition sur les durées (DC, pour Duration Condition)*, est construite en fonction des durées de a et de b. D'abord, nous montrons la construction des conditions sur les durées pour s_0, s_1 et s_2. Après le tirage de la transition $s_0 \xrightarrow{a,\ x:=0} s_1$, nous avons besoin d'une information sur l'exécution éventuelle de l'action a dans l'état s_1. On est sûr que l'action a termine son exécution lorsque l'horloge correspondante x atteint la valeur 10, donc, on ajoute à l'état s_1 la condition sur la durée de a, $\{x \geq 10\}$, qui veut dire que si la valeur de x est supérieure ou égale à 10 alors on est sûr que l'action a a fini de s'exécuter. La même chose pour l'état s_2 qui sera étiqueté par $\{y \geq 12\}$. A l'état s_0, aucune action n'est en exécution, ce qui implique que l'ensemble des conditions sur les durées soit vide. A l'état s_3, les actions a et b peuvent éventuellement s'exécuter en parallèle, et chacune d'elles ne peut terminer que si son horloge atteint une valeur égale à sa durée. D'où l'ensemble des conditions

sur les durées $\{x \geq 10, y \geq 12\}$. La condition d'exécution de l'action d devient alors $x \geq 10 \wedge y \geq 12$. *A l'état s_3 la condition sur les durées des actions a et b implique la possibilité de leurs évolutions parallèles.*

Une différence intrinsèque entre les invariants associés aux états utilisés dans les automates temporisés et les conditions sur les durées est à noter. En effet, comme c'est illustré dans la Figure 5.2.(b), la résidence dans un état est impérativement conditionnée par la satisfiabilité de l'invariant à cet état là, le système doit changer d'état dès que l'invariant devient non vérifié. Cependant, les conditions sur les durées visent plutôt à décrire l'état d'évolution des actions au sein d'un état, le système n'étant pas forcé à quitter un état sous condition que les actions en cours terminent leur exécution.

En considérant la sémantique des conditions sur les durées, avec l'hypothèse que les actions a et b sont de durées respectives 2 et 10, le comportement de la Figure 5.2.(a) sera exprimé par l'automate de la Figure 5.5. Une telle structure sera appelée *Durational Action Timed Automata* (*DATA*).

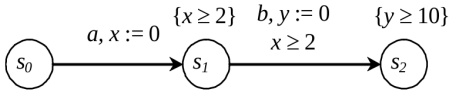

FIGURE 5.5 – Exemple d'un DATA

Après lancement de l'exécution de l'action a, la condition sur les durées présente dans l'état s_1 stipule le fait que l'action correspondante à l'horloge x (ici a) termine son exécution lorsque x atteint la valeur 2. Le système peut demeurer dans l'état s_1 *indéfiniment*, car la condition sur les durées $\{x \geq 2\}$ exprime uniquement l'instant de terminaison de l'action a (quand x est égale à 2). L'action b peut alors commencer son exécution qui dure 10 unités de temps.

5.1.2 Formalisation

Définition 5.1 *$\mathcal{H}$, parcouru par x, y... étant un ensemble d'horloges de valeurs dans $\mathbb{R}^+$. L'ensemble $\Phi_t(\mathcal{H})$ des contraintes temporelles γ sur $\mathcal{H}$ est défini par la syntaxe $\gamma ::= x \sim t$, où x est une horloge dans $\mathcal{H}$, $\sim \in \{=, <, >, \leq, \geq\}$ et $t \in \mathbb{R}^+$. F_x sera utilisée pour désigner une contrainte de la forme $x \sim t$.*

Une *valuation* (ou *interprétation*) v des horloges de $\mathcal{H}$ est une fonction qui associe à chaque $x \in \mathcal{H}$ une valeur dans $\mathbb{R}^+$. On dit qu'une valuation v des horloges de $\mathcal{H}$ satisfait une contrainte temporelle γ sur $\mathcal{H}$ ssi γ est vraie en utilisant les valeurs données par v.

Pour $\mathcal{I} \subseteq \mathcal{H}$, $[\mathcal{I} \mapsto 0]\, v$ dénote la valuation de $\mathcal{H}$ qui affecte la valeur 0 à chaque $x \in \mathcal{I}$, et maintient v pour les autres horloges de $\mathcal{H}$. L'ensemble de toutes les valuations des horloges de $\mathcal{H}$ est noté $\Xi(\mathcal{H})$.

Définition 5.2 *La relation de satisfaction $\models$ pour les contraintes temporelles est définie sur l'ensemble des valuations des horloges de $\mathcal{H}$, par $v \models x \sim t \iff v(x) \sim t$ tel que $v \in \Xi(\mathcal{H})$.*

Définition 5.3 *Un Durational Action Timed Automaton (DATA) $\mathcal{A}$ est un quintuplet $(S, L_S, s_0, \mathcal{H}, T)$ tel que :*

- *S est un ensemble fini d'états,*
- *$L_S : S \to 2^{\Phi_t(\mathcal{H})}_{fn}$ est une fonction qui fait correspondre à chaque état s l'ensemble F des conditions de terminaison des actions potentiellement en exécution dans s.*
- *$s_0 \in S$ est l'état initial,*
- *$\mathcal{H}$ est un ensemble fini d'horloges.*
- *$T \subseteq S \times 2^{\Phi_t(\mathcal{H})}_{fn} \times Act \times \mathcal{H} \times S$ est l'ensemble des transitions. Une transition (s, γ, a, x, s') représente le passage de l'état s à l'état s', en lançant l'exécution de l'action a et en réinitialisant l'horloge x. γ est la contrainte correspondante, qui doit être satisfaite pour tirer cette transition. (s, γ, a, x, s') peut être écrit $s \xrightarrow{\gamma, a, x} s'$.*

Définition 5.4 *La sémantique d'un DATA $\mathcal{A} = (S, L_S, s_0, \mathcal{H}, T)$ est définie en lui associant un système infini de transitions $\mathcal{S}_\mathcal{A}$ sur l'alphabet $Act \cup \mathbb{R}^+$. Un état de $\mathcal{S}_\mathcal{A}$ (ou configuration) est un couple $\langle s, v \rangle$ tel que s est un état de $\mathcal{A}$ et v est une valuation sur $\mathcal{H}$. Une configuration $\langle s_0, v_0 \rangle$ est initiale si s_0 est l'état initial de $\mathcal{A}$ et $\forall x \in \mathcal{H}, v(x) = 0$. Deux types de transitions entre les configurations de $\mathcal{S}_\mathcal{A}$ sont possibles, et qui correspondent respectivement au passage de temps (règle* RA*) et au tirage d'une transition de $\mathcal{A}$ (règle* RD*).*

$$\text{(RA)}\ \frac{d \in \mathbb{R}^+}{\langle s, v \rangle \xrightarrow{d} \langle s, v + d \rangle} \qquad \text{(RD)}\ \frac{(s, \gamma, a, x, s') \in T \qquad v \models \gamma}{\langle s, v \rangle \xrightarrow{a} \langle s', [\{x\} \mapsto 0]\, v \rangle}$$

Exemple 5.1 *Soit $\mathcal{A}$ le DATA de la Figure 5.5. Un exemple de transitions possibles de la forme $\langle s, v \rangle$ de $\mathcal{S}_\mathcal{A}$ est :*

$$\langle s_0, x = 0 \rangle \xrightarrow{2.33} \langle s_0, x = 2.33 \rangle \xrightarrow{a} \langle s_1, x = 0 \rangle \xrightarrow{4} \langle s_1, x = 4 \rangle \xrightarrow{b} \langle s_2, x = 0 \rangle \xrightarrow{6.25} \cdots$$

5.2 Modèle des DATA*'s

Le modèle des DATA's a été introduit dans le but d'exprimer la non-atomicité temporelle et structurelle des actions. En général, les systèmes à temps contraint ne peuvent être complètement spécifiés si l'on considère pas des notions comme l'urgence, les délais, les contraintes, etc. Pour prendre en compte ces nouveaux concepts, nous avons besoin de passer vers les DATA*'s que nous introduisons dans cette section.

5.2.1 Intuition

Considérons le système S décrit dans la Section 5.1.1. A partir de l'état s_3 de la Figure 5.4.(b), l'action d ne peut commencer son exécution que si a et b ont terminé leurs exécutions, d'où la contrainte $x \geq 10 \wedge y \geq 12$ correspondante à la transition d. Cette contrainte est une contrainte sur les durées des actions a et b. Une fois la contrainte $x \geq 10 \wedge y \geq 12$ satisfaite, l'action d peut s'exécuter à n'importe quel moment dans l'intervalle ouvert de sensibilisation $x \in [10, +\infty[\,, y \in [12, +\infty[$ que nous appelons *domaine de sensibilisation*.

Le type de contraintes que nous voulons exprimer est celui impliquant des restrictions sur le domaine de sensibilisation. Dans un contexte de temps contraint, ces restrictions peuvent ne pas être dues uniquement aux durées des actions antérieures, formant ainsi des domaines ouverts comme pour $x \geq 10 \wedge y \geq 12$, mais peuvent borner le domaine de sensibilisation indépendamment des durées des autres actions en retardant par exemple une action d'une certaine quantité de temps ou en limitant le temps pendant lequel une action est offerte à son environnement (restriction temporelle).

Ainsi, supposons que l'action a ne peut commencer son exécution que dans les trois premières unités de temps, c'est-à-dire dans le domaine $[0, 3]$. Une action peut éventuellement commencer son exécution si la valeur d'une certaine horloge appartient à son domaine de sensibilisation. L'action a ne peut commencer son exécution que si une horloge particulière n'a pas encore atteint la valeur 3. Cette horloge est initialisé au moment de la sensibilisation du système S (c'est-à-dire à l'instant 0). Etant donné que l'action a n'attend la fin d'aucune autre action, cette horloge est désignée $c_\emptyset$. Conséquemment, la transition a dans le DATA* résultant sera étiquetée par la contrainte $c_\emptyset \leq 3$ (c'est-à-dire $c_\emptyset \in [0, 3]$). Cette contrainte, appelée *garde*, devra être satisfaite pour que l'action a puisse s'exécuter. Si en plus, l'action a est retardée d'un laps de temps égal à 1 (comme fait l'opérateur Δ^d de D-LOTOS), la garde sur la transition a sera $1 \leq c_\emptyset \leq 4$ (c'est-à-dire $c_\emptyset \in [0 + 1, 3 + 1]$).

Le même raisonnement s'applique sur les autres actions. En admettant que les

actions a et b du système S sont offertes à l'environnement dans les quantités de temps respectives 3 et 4 depuis leurs sensibilisation, et que l'action d est retardée dans le sous-système S_1 de 5 et dans S_2 de 4 unités de temps, le comportement global de S est représenté par le DATA* de la Figure 5.6.

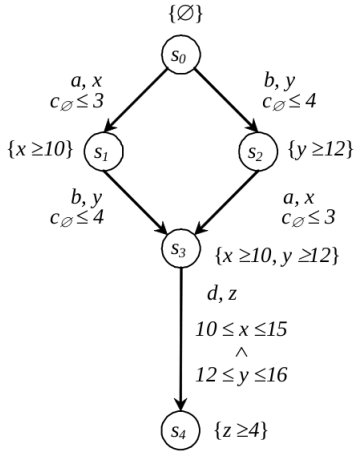

FIGURE 5.6 – DATA* du comportement de S

A partir de l'état s_3, les deux sous-systèmes S_1 et S_2 peuvent se synchroniser sur l'action d à condition que les actions a et b ont terminé leurs exécutions. Le début de l'action d est conditionné d'une part par la contrainte sur les durées de a et b : $x \geq 10 \wedge y \geq 12$, et d'autre part par la restriction temporelle du domaine de sensibilisation de l'action d de 5 et 4 unités de temps respectivement suivant la provenance de d (de S_1 ou S_2). L'action d provenant de S_1 attend la terminaison de a (qui a comme horloge x), c'est-à-dire, elle attend que x atteint la valeur 10. Une fois cette valeur atteinte, le délai d'expiration de l'offre de l'action d de S_1 commence, et termine après 5 unités de temps, c'est-à-dire après que x atteint la valeur 15. Donc, le domaine de sensibilisation de cette action est $x \in [10, 15]$. La même chose pour l'autre action d ayant comme domaine de sensibilisation $y \in [12, 16]$.[1]

[1] $x \in [\min, \max]$, $\min \leq x \leq \max$, $x \geq \min \wedge x \leq \max$, ou encore $\{x \geq \min, x \leq \max\}$ signifient la même chose. Cette observation a pour but d'assurer que les fonctions sur les domaines, qui seront définies par la suite, peuvent être appliquées à toutes les formes précédentes de domaines, même si elles seront explicitement définies en utilisant une seule forme.

Expression de l'urgence

Nous avons constaté que le nouveau modèle des DATA*'s est capable d'exprimer les contraintes temporelles dues aux restrictions sur un domaine de sensibilisation d'une action. Observons à présent l'expression d'actions urgentes dans le modèle des DATA*'s. Une action urgente doit s'exécuter dès qu'elle est sensibilisée, tout en stoppant la progression du temps.

Notons qu'il faut distinguer entre actions urgentes et actions dont le domaine de sensibilisation est formé d'un seul instant dans le temps. Considérons l'exemple d'une action a ayant une durée de 7 et ayant comme domaine de sensibilisation $x \in [3,3]$. Cette action ne peut s'exécuter que si l'horloge x atteint la valeur 3 ; au-delà de ce domaine (par exemple dans le cas d'un refus de l'environnement), l'action a ne peut plus s'exécuter. Si par contre a est une action urgente ayant toujours comme domaine de sensibilisation $x \in [3,3]$, une fois x est égale à 3, *le temps s'arrête de progresser* jusqu'à ce que l'action a commence à s'exécuter. Donc, le *domaine d'urgence* de a est $x \in [3,3]$.

En considérant l'hypothèse de la monotonie du temps, au *moment de la satisfaction* de la condition correspondante au domaine d'urgence d'une action, cette dernière doit être tirée immédiatement.

Dans cet exemple, le domaine $x \in [3,3]$ désigne à la fois le domaine de sensibilisation et le domaine d'urgence sauf que sa sémantique diffère dans les deux contextes. Cette observation nous ramène à introduire le domaine d'urgence au niveau des transitions des DATA*'s. Ainsi, toutes les transitions seront étiquetées en plus de la contrainte de sensibilisation G (pour *guard*, ou garde) par la contrainte d'urgence D (pour *deadline*, ou échéance). Dans le cas du comportement du système S représenté par la Figure 5.6, toutes les transitions seront étiquetées par $D = \{\texttt{false}\}$, à cause de l'absence d'actions urgentes. La contrainte d'urgence D peut être occultée sans aucune ambiguïté dans le cas où $D = \{\texttt{false}\}$.

Le comportement de l'exemple précédent dans lequel l'action a est urgente est illustré par la Figure 5.7. L'horloge $c_\emptyset$ est utilisée à la place de x dans le cas où a est la première action que le système peut exécuter, ce qui est effectivement le cas dans notre exemple. En fait, l'horloge x sera associée à l'action a, la contrainte sur la durée de a placée sur l'état s_1 est écrite ainsi en fonction de x.

5.2.2 Formalisation

Définition 5.1 *Un DATA* $\mathcal{A}$ est un quintuplet $(S, L_S, s_0, \mathcal{H}, T_D)$ tel que :*

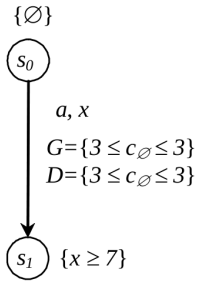

FIGURE 5.7 – Expression de l'urgence par un DATA*

- S *est un ensemble fini d'états,*
- $L_S : S \to 2^{\Phi_t(\mathcal{H})}_{fn}$ *est une fonction qui fait correspondre à chaque état* s *l'ensemble* F *des conditions de terminaison des actions potentiellement en exécution dans* s*.*
- $s_0 \in S$ *est l'état initial,*
- $\mathcal{H}$ *est un ensemble fini d'horloges.*
- $T_D \subseteq S \times 2^{\Phi_t(\mathcal{H})}_{fn} \times 2^{\Phi_t(\mathcal{H})}_{fn} \times Act \times \mathcal{H} \times S$ *est l'ensemble des transitions. Une transition* (s, G, D, a, x, s') *représente le passage de l'état* s *à l'état* s'*, en lançant l'exécution de l'action* a *et en réinitialisant l'horloge* x*.* G *est la contrainte correspondante, qui doit être satisfaite pour tirer cette transition.* D *est l'échéance correspondante qui exige, au moment de sa satisfaction, que l'action* a *doit être tirée.* (s, G, D, a, x, s') *peut être écrit* $s \xrightarrow{G,D,a,x} s'$

Définition 5.2 *La sémantique d'un DATA* $\mathcal{A} = (S, L_S, s_0, \mathcal{H}, T_D)$ *est définie en lui associant un système infini de transitions* $\mathcal{S}_\mathcal{A}$ *sur l'alphabet* $Act \cup \mathbb{R}^+$*. Un état de* $\mathcal{S}_\mathcal{A}$ *(ou configuration) est un couple* $\langle s, v\rangle$ *tel que* s *est un état de* $\mathcal{A}$ *et* v *est une valuation sur* $\mathcal{H}$*. Une configuration* $\langle s_0, v_0\rangle$ *est initiale si* s_0 *est l'état initial de* $\mathcal{A}$ *et* $\forall x \in \mathcal{H}, v(x) = 0$*. Deux types de transitions entre les configurations de* $\mathcal{S}_\mathcal{A}$ *sont possibles, et qui correspondent respectivement au passage de temps (règle* RA*) et au tirage d'une transition de* $\mathcal{A}$ *(règle* RD*).*

$$\text{(RA*)}\ \frac{d \in \mathbb{R}^+ \qquad \forall d' \leq d,\ v + d' \not\models D}{\langle s, v\rangle \xrightarrow{d} \langle s, v + d\rangle} \qquad \text{(RD*)}\ \frac{(s, G, D, a, x, s') \in T_D \qquad v \models G}{\langle s, v\rangle \xrightarrow{a} \langle s', [\{x\} \mapsto 0]\, v\rangle}$$

Selon la sémantique de maximalité, l'*étiquette* a dans la règle RD* implique le *début d'exécution* de l'action a et non l'exécution totale de a.

Dans la règle RA*, $D = \bigvee_{i \in I} D_i$ où $\{(s, G_i, D_i, a_i, x_i, s_i)\}_{i \in I}$ est l'ensemble de toutes les transitions issues de l'état s. En effet, chaque fois qu'une contrainte d'urgence D_i est satisfaite, le temps ne peut progresser indépendamment de la valeur des autres D_i. La fonction L_S, qui donne les conditions sur les durées au niveau d'un état est utile pour la construction des contraintes temporelles et des contraintes d'urgence. Par construction, si D_i ou G_i sont satisfaites alors les actions causales ont terminé leurs exécutions. L'intérêt de L_S sera montré dans la Section 6.2 dans le processus de translation entre les expressions D-LOTOS et les structures de DATA*.

Notons bien que si on veut garantir qu'au moins une transition pourrait être tirée à partir d'un état dans le cas où le temps ne peut plus progresser au sein de cet état, on exige que la formule $D \Rightarrow G$ soit vérifiée.

Observons un autre problème dans lequel une action a doit être tirée le plutôt possible à un instant supérieur *strictement* à 1 (c'est-à-dire à l'intervalle $]1, +\infty[$). Cette instant ne peut évidemment être connu, ce qui nous ramène à exiger que les domaines d'urgence ne doivent pas être ouverts à gauche.

Remarque 5.1 *Les domaines d'urgence sont toujours fermés à gauche, c'est-à-dire qu'ils sont de la forme* [.] *ou* [.[.

Le problème précédent peut être aussi percé dans les algèbres de processus, c'est le cas par exemple de l'expression D-LOTOS

```
hide a in (a@t[t>1];stop)
```

spécifiant le comportement précédent.

5.3 Conclusion

Dans ce chapitre, nous avons introduit et défini formellemement deux modèles basés sur la maximalité appelés respectivement DATA et DATA* (pour Durational Action Timed Automata(*)). Le premier modèle est destiné principalement à la tâche de spécification de systèmes, temps-réel ou non, où on s'intéresse beaucoup plus aux durées d'actions. Le deuxième étant une extension des DATA mais cette fois-ci en donnant la possibilité à un concepteur de pouvoir spécifier aussi bien les durées explicites d'actions que les contraintes temporelles et les contraintes d'urgence agissant sur le comportement du système temps-réel à analyser. Dans le Chapitre 6, nous donnerons une manière de générer des structures de DATA et DATA* à partir de formalismes de haut niveau (langages Basic LOTOS et D-LOTOS).

Chapitre 6

Sémantiques autour de la maximalité

Dans ce chapitre, nous proposons des méthodes opérationnelles pour la génération de structures, temporelles ou non, basées sur la maximalité à partir de modèles de spécification de haut niveau, à l'image des langages Basic LOTOS avec durées d'actions, D-LOTOS et les réseaux de Petri place/transition. Ce chapitre se base en grande partie sur des résultats publiés dans [BS05, SB06, SBB08, SBB09b, SBB09a, SBB09c].

6.1 Pour Basic LOTOS

Dans cette section, nous donnons la manière de générer opérationnellement un DATA à partir d'une spécification Basic LOTOS dans laquelle les durées d'actions sont rendues explicites (on appelle cette extension *Basic LOTOS avec durées d'actions*). La démarche est très proche à celle de la Section 3.1.1 pour la génération des systèmes de transitions étiquetées maximales.

Désormais, l'ensemble $\mathcal{M}$ des évènements maximaux sera utilisé pour désigner l'ensemble de toutes les horloges. Ceci est motivé par le choix dynamique des horloges correspondant à l'occurrence des évènements.

Afin de voir de manière informelle comment générer à partir d'une expression Basic LOTOS avec durées d'actions le DATA correspondant, reprenons l'exemple du comportement du système S de la Section 5.1.1 qui se traduit en la composition parallèle de deux sous-systèmes S_1 et S_2 avec synchronisation sur la porte d. Le sous-système S_1 exécute l'action a suivie de d, tandis que S_2 exécute b puis d. En supposant que les durées respectives des actions a, b et d sont 10, 12 et 4, le comportement du système S peut être

exprimé par l'expression Basic LOTOS avec durées d'actions `E[a[10],b[12],d[4]] = a;d;stop |[d]| b;d;stop`.

Comme nous l'avons déjà vu, une structure de DATA comporte un ensemble fini d'états contenant les conditions sur les durées. Dans un contexte de génération d'un DATA à partir d'une expression de comportement d'une algèbre de processus, une *configuration d'un DATA* est identifiée par un état du DATA ainsi qu'une sous-expression de comportement. C'est-à-dire que les configurations correspondantes aux états des DATA's vont être réécrites en fonction des conditions sur les durées sous la forme ${}_F\,[E]$ où $F \in 2^{\Phi_t(\mathcal{H})}_{fn}$.

A l'état initial du système S, aucune action n'est en exécution, ce qui explique que l'ensemble des conditions sur les durées est vide. Nous faisons correspondre à cet état l'expression E pour former la configuration initiale ${}_\emptyset\,[E]$ du DATA, où aucune action n'a encore été tirée. A partir de cette configuration, l'action a peut être tirée. Une horloge x, choisie par la fonction *get* et ayant la valeur initiale 0, est associée à cette occurrence de a. L'action a n'attend la fin d'aucune autre action pour pouvoir s'exécuter, d'où l'ensemble vide associé à la transition

$$\underbrace{{}_\emptyset\,[E]}_{config_0} \xrightarrow{\emptyset,a,x} \underbrace{{}_{\{x\geq 10\}}\,[d;stop] \;\; |[d]| \;\; {}_\emptyset\,[b;d;stop]}_{config_1}$$

A partir de la configuration $config_1$ correspondante à l'état s_1, la seule transition possible est celle correspondante au tirage de l'action b, d'où la dérivation

$$\underbrace{{}_{\{x\geq 10\}}\,[d;stop] \;\; |[d]| \;\; {}_\emptyset\,[b;d;stop]}_{config_1} \xrightarrow{\emptyset,b,y} \underbrace{{}_{\{x\geq 10\}}\,[d;stop] \;\; |[d]| \;\; {}_{\{y\geq 12\}}\,[d;stop]}_{config_3}$$

Le même raisonnement s'applique sur l'autre branche où l'action b commence son exécution avant l'action a de la manière suivante :

$$\underbrace{{}_\emptyset\,[E]}_{config_0} \xrightarrow{\emptyset,b,y} \underbrace{{}_\emptyset\,[a;d;stop] \;\; |[d]| \;\; {}_{\{y\geq 12\}}\,[d;stop]}_{config_2} \xrightarrow{\emptyset,a,x} \; config_3$$

A partir de la configuration $config_3$, l'action d ne peut commencer son exécution que si les deux actions a et b auraient terminé leur exécution, autrement dit, que si les conditions sur les durées faisant partie de l'ensemble $\{x \geq 10, y \geq 12\}$ sont toutes satisfaites, d'où l'ensemble $\{x \geq 10, y \geq 12\}$ correspondant à la condition $x \geq 10 \wedge y \geq 12$. La transition suivante devient alors possible :

$$\underbrace{{}_{\{x\geq 10\}}\,[d;stop] \;\; |[d]| \;\; {}_{\{y\geq 12\}}\,[d;stop]}_{config_3} \xrightarrow{\{x\geq 10, y\geq 12\},d,z} \underbrace{{}_{\{z\geq 4\}}\,[stop] \;\; |[d]| \;\; {}_{\{z\geq 4\}}\,[stop]}_{config_4}$$

Le système ne peut tirer aucune action à partir de la configuration $config_4$, néanmoins, nous avons l'information sur le moment de la fin d'exécution de l'action d.

Notons bien que nous pouvons faire correspondre à l'action d l'horloge x suite à la libération des deux horloges x et y au moment du tir de d, nous pouvons réutiliser l'une de ces deux dernières (soit x) pour la faire correspondre à d. Ce mécanisme de réutilisation des horloges est offert par la fonction *get*.

Pour formaliser la génération opérationnelle des DATA's à partir de spécifications Basic LOTOS avec durées d'actions, les fonctions ψ, $\backslash$ et la fonction de substitution introduites dans la Section 3.1.1 se généralisent directement aux configurations relatives aux états des DATA's comme suit.

Les configurations correspondantes aux états des DATA's font partie de l'ensemble $\mathcal{C}_\perp$. La fonction ψ d'extraction de l'ensemble d'évènements maximaux d'une configuration est redéfinie sur l'ensemble $\mathcal{C}_\perp$ de la même manière afin de déterminer l'ensemble des horloges utilisées dans une configuration de DATA, sauf que l'équation $\psi\left({}_M\left[E\right]\right) = M$ est remplacée par $\psi\left({}_F\left[E\right]\right) = \psi_{\mathcal{H}}\left(F\right)$, avec $\psi_{\mathcal{H}}$ donnée par la Définition 6.1.

Définition 6.1 *La fonction $\psi_{\mathcal{H}} : 2^{\Phi_t(\mathcal{H})}_{fn} \rightarrow 2^{\mathcal{H}}_{fn}$, qui détermine l'ensemble des horloges utilisées dans un ensemble de conditions de terminaison, est définie récursivement par :*

$$\begin{aligned}
&\psi_{\mathcal{H}}\left(\emptyset\right) = \emptyset \\
&\psi_{\mathcal{H}}\left(\{x \sim t\}\right) = \{x\} \\
&\psi_{\mathcal{H}}\left(F_1 \cup F_2\right) = \psi_{\mathcal{H}}\left(F_1\right) \cup \psi_{\mathcal{H}}\left(F_2\right)
\end{aligned}$$

tel que $F_1, F_2 \in 2^{\Phi_t(\mathcal{H})}_{fn}$, $x \in \mathcal{H}$, $\sim \in \{=, <, >, \leq, \geq\}$ et $t \in \mathbb{R}^+$.

Afin de déterminer l'ensemble des conditions sur les durées d'une configuration $\mathcal{C}_\perp$, nous utilisons la fonction $\psi_{DC} : \mathcal{C}_\perp \rightarrow 2^{\Phi_t(\mathcal{H})}_{fn}$.

Si $\mathcal{E}$ est une configuration d'un DATA ; $\mathcal{E}\backslash K$, qui dénote la configuration obtenue par la suppression de l'ensemble des conditions sur les durées K écrites en fonction des horloges utilisées par l'ensemble F de la configuration $\mathcal{E}$ reste la même, excepté l'équation $\left({}_F\left[E\right]\right)\backslash K = {}_{F\backslash K}\left[E\right]$ qui remplace $\left({}_M\left[E\right]\right)\backslash N = {}_{M-N}\left[E\right]$, avec $F\backslash K$ donnée dans la Définition 6.2.

Définition 6.2 *Soit F un ensemble de conditions de terminaison ; $F\backslash K$ dénote l'ensemble obtenue par la suppression, à partir de l'ensemble F, de toutes les conditions sur les durées écrites en fonction des horloges de K. $F\backslash K$ est définie récursivement*

sur F comme suit :

$$\begin{aligned}
&\emptyset\backslash K = \emptyset\\
&(F_1 \cup F_2)\backslash K = F_1\backslash K \cup F_2\backslash K\\
&\{x \sim t\}\backslash K = \begin{cases} \emptyset & si\ x \in K\\ \{x \sim t\} & sinon \end{cases}
\end{aligned}$$

tel que $F_1, F_2 \in 2^{\Phi_t(\mathcal{H})}_{fn}$, $K \subseteq \mathcal{H}$, $x \in \mathcal{H}$, $\sim \in \{=, <, >, \leq, \geq\}$ *et* $t \in \mathbb{R}^+$.

La substitution simultanée reste inchangée à part l'équation $(_M [E])\,\sigma = {}_{M\sigma}[E]$ qui sera remplacée par $(_F [E])\,\sigma = {}_{F\sigma}[E]$. Si $x, y \in \mathcal{H}$ et $F \in 2^{\Phi_t(\mathcal{H})}_{fn}$ alors $F\sigma = \bigcup_{F_i \in F} \sigma F_i$ avec $\{x \sim t\}\,\sigma = \{\sigma(x) \sim t\}$, et σ est une fonction de substitution donnée par la Définition 3.4.

Définition 6.3 *La relation de transition des DATA's* $\longrightarrow_\perp \subseteq \mathcal{C}_\perp \times 2^{\Phi_t(\mathcal{H})}_{fn} \times Act \times \mathcal{H} \times \mathcal{C}_\perp$ *est définie comme étant la plus petite relation satisfaisant les règles suivantes :*

1. $\dfrac{}{{}_F[exit] \xrightarrow{F,\delta,x}_\perp {}_{\{x \geq 0\}}[stop]}$ $x{=}get(\mathcal{M})$

2. $\dfrac{}{{}_F[a;E] \xrightarrow{F,a,x}_\perp {}_{\{x \geq \tau(a)\}}[E]}$ $x{=}get(\mathcal{M})$

3. $\dfrac{\mathcal{E} \xrightarrow{F,a,x}_\perp \mathcal{E}'}{\mathcal{F} \,[]\, \mathcal{E} \xrightarrow{F,a,x}_\perp \mathcal{E}' \qquad \mathcal{E} \,[]\, \mathcal{F} \xrightarrow{F,a,x}_\perp \mathcal{E}'}$

4. *(a)* *i.* $\dfrac{\mathcal{E} \xrightarrow{F,a,x}_\perp \mathcal{E}' \qquad a \notin L \cup \{\delta\}}{\mathcal{E} \;|[L]|\; \mathcal{F} \xrightarrow{F,a,y}_\perp \mathcal{E}'[y/x] \;|[L]|\; \mathcal{F}\backslash F}$ $y{=}get(\mathcal{M} - ((\psi(\mathcal{E}) \cup \psi(\mathcal{F})) - \psi_{\mathcal{H}}(F)))$

   ii. $\dfrac{\mathcal{E} \xrightarrow{F,a,x}_\perp \mathcal{E}' \qquad a \notin L \cup \{\delta\}}{\mathcal{F} \;|[L]|\; \mathcal{E} \xrightarrow{F,a,y}_\perp \mathcal{F}\backslash F \;|[L]|\; \mathcal{E}'[y/x]}$ $y{=}get(\mathcal{M} - ((\psi(\mathcal{E}) \cup \psi(\mathcal{F})) - \psi_{\mathcal{H}}(F)))$

   (b) $\dfrac{\mathcal{E} \xrightarrow{F,a,x}_\perp \mathcal{E}' \qquad \mathcal{F} \xrightarrow{G,a,y}_\perp \mathcal{F}' \qquad a \in L \cup \{\delta\}}{\mathcal{E} \;|[L]|\; \mathcal{F} \xrightarrow{F \cup G,a,z}_\perp \mathcal{E}'[z/x]\backslash G \;|[L]|\; \mathcal{F}'[z/y]\backslash F}$ $z{=}get(\mathcal{M} - ((\psi(\mathcal{E}) \cup \psi(\mathcal{F})) - (\psi_{\mathcal{H}}(F) \cup \psi_{\mathcal{H}}(G))))$

5. *(a)* $\dfrac{\mathcal{E} \xrightarrow{F,a,x}_\perp \mathcal{E}' \qquad a \notin L}{hide\ L\ in\ \mathcal{E} \xrightarrow{F,a,x}_\perp hide\ L\ in\ \mathcal{E}'}$

   (b) $\dfrac{\mathcal{E} \xrightarrow{F,a,x}_\perp \mathcal{E}' \qquad a \in L}{hide\ L\ in\ \mathcal{E} \xrightarrow{F,i,x}_\perp hide\ L\ in\ \mathcal{E}'}$

6. *(a)* $\dfrac{\mathcal{E} \xrightarrow{F,a,x}_\perp \mathcal{E}' \qquad a \neq \delta}{\mathcal{E} \gg F \xrightarrow{F,a,x}_\perp \mathcal{E}' \gg F}$

   (b) $\dfrac{\mathcal{E} \xrightarrow{F,\delta,x}_\perp \mathcal{E}'}{\mathcal{E} \gg E \xrightarrow{F,i,x}_\perp {}_{\{x \geq 0\}}[E]}$

7. *(a)* $\dfrac{\mathcal{E} \xrightarrow{F,a,x}_\perp \mathcal{E}' \qquad a \neq \delta}{\mathcal{E} \;[>\; \mathcal{F} \xrightarrow{F,a,y}_\perp \mathcal{E}'[y/x] \;[>\; \mathcal{F}\backslash F}$ $y{=}get(\mathcal{M} - ((\psi(\mathcal{E}) \cup \psi(\mathcal{F})) - \psi_{\mathcal{H}}(F)))$

(b) $\dfrac{\mathcal{E} \xrightarrow{F,\delta,x}_{\perp} \mathcal{E}' \qquad \forall z{\in}\psi(\mathcal{F})}{\mathcal{E}\ [>\ \mathcal{F} \xrightarrow{F,\delta,y}_{\perp} \mathcal{E}'[y/x]\ [>\ _{\psi_{DC}(\mathcal{F})-F_z}[stop]} \; y{=}get(\mathcal{M}{-}((\psi(\mathcal{E}){\cup}\psi(\mathcal{F})){-}\psi_{\mathcal{H}}(F)))$

(c) $\dfrac{\mathcal{F} \xrightarrow{F,a,x}_{\perp} \mathcal{F}' \qquad \forall z{\in}\psi(\mathcal{E})}{\mathcal{E}\ [>\ \mathcal{F} \xrightarrow{F,a,y}_{\perp}\ _{\psi_{DC}(\mathcal{E})-F_z}[stop]\ [>\ \mathcal{F}'[y/x]} \; y{=}get(\mathcal{M}{-}((\psi(\mathcal{E}){\cup}\psi(\mathcal{F})){-}\psi_{\mathcal{H}}(F)))$

8. (a) $\dfrac{\mathcal{E} \xrightarrow{F,a,x}_{\perp} \mathcal{E}' \qquad a{\notin}\{a_1,...,a_n\}}{\mathcal{E}[b_1/a_1,...,b_n/a_n] \xrightarrow{F,a,x}_{\perp} \mathcal{E}'[b_1/a_1,...,b_n/a_n]}$

(b) $\dfrac{\mathcal{E} \xrightarrow{F,a,x}_{\perp} \mathcal{E}' \qquad a{=}a_i\ (1{\leq}i{\leq}n)}{\mathcal{E}[b_1/a_1,...,b_n/a_n] \xrightarrow{F,b_i,x}_{\perp} \mathcal{E}'[b_1/a_1,...,b_n/a_n]}$

9. $\dfrac{P{:=}E \qquad _F[E] \xrightarrow{F,a,x}_{\perp} \mathcal{F}}{_F[P] \xrightarrow{F,a,x}_{\perp} \mathcal{F}}$

La règle 1 implique que la terminaison avec succès ne peut commencer qu'après que toutes les actions qui correspondent aux horloges présentes dans l'ensemble F de conditions sur les durées aient terminé leur exécution. Cela peut être vu par la satisfaction de la condition $\bigwedge_{i\in\mathcal{H}} F_i$. La condition sur les durées de la configuration résultante est $\{x \geq 0\}$ parce que la durée de l'action δ est supposée nulle.

La règle 2 caractérise la sémantique des opérateurs de préfixage des actions ; comme le début de l'exécution de l'action a dépend de la terminaison des actions associées aux horloges présentes dans l'ensemble F, seul x, l'horloge associée au début de l'action a, apparaît dans la configuration $_{\{x\geq\tau(a)\}}[E]$. La condition sur les durées $\{x \geq \tau(a)\}$ implique qu'on ne peut tirer aucune action de l'expression E que si l'horloge x atteint la valeur $\tau(a)$.

Les autres règles sont une adaptation directe des règles de la Définition 3.5.

Notons que les ensembles F au niveau des transitions peuvent être masqués, s'il n'y a pas risque d'ambigüité, dans les cas où :

- l'ensemble F au niveau des transitions est vide $(F = \emptyset)$;
- une transition est gardée par une contrainte temporelle de la forme $\bigwedge_{i\in\mathcal{H}} F_i$, tel que $\bigcup_{i\in\mathcal{H}} F_i = F$, et F est présent au niveau de cette transition.

6.2 Pour D-LOTOS

Nous expliquons dans cette section comment construire opérationnellement un DATA* à partir d'une spécification D-LOTOS. La démarche est comparable à celle de la Section 6.1 relatant les DATA's. Nous gardons les mêmes fonctions déjà évoquées dans la Section

6.1. La spécification des délais et des contraintes temporelles requiert la définition d'une fonction $shift\,(G,t)$ notant le décalage d'un domaine par un temps t, c'est-à-dire

$$shift\,(G,t) \stackrel{déf}{=} \{(\min + t \sim i \sim \max + t) \mid (\min \sim i \sim \max) \in G\} \qquad \sim \in \{<, \leq\}$$

Dans le contexte des DATA*'s, les configurations correspondantes aux états font partie de l'ensemble $\mathcal{C}_*$.

Définition 6.1 *La relation de transition des DATA*'s* $\longrightarrow_* \subseteq \mathcal{C}_* \times 2^{\Phi_t(\mathcal{H})}_{fn} \times 2^{\Phi_t(\mathcal{H})}_{fn} \times Act \times \mathcal{H} \times \mathcal{C}_*$ *est définie comme étant la plus petite relation satisfaisant les règles suivantes :*

1. (a) $\dfrac{}{\emptyset[exit\{u\}] \xrightarrow{G=\{c_\emptyset \leq u\}, D=\{\texttt{false}\}, \delta, x}_* {}_{\{x\geq 0\}}[stop]} \quad x=get(\mathcal{M})$

   (b) $\dfrac{}{{}_F[exit\{u\}] \xrightarrow{G=\{\cup_{i\in\mathcal{H}}\{shift(0\leq i\leq u,t)\}\setminus F_i=\{i\geq t\}\}, D=\{\texttt{false}\}, \delta, x}_* {}_{\{x\geq 0\}}[stop]} \quad x=get(\mathcal{M})$

2. (a) $\dfrac{}{\emptyset[a\{u\};E] \xrightarrow{G=\{c_\emptyset \leq u\}, D=\{\texttt{false}\}, a, x}_* {}_{\{x\geq \tau(a)\}}[E]} \quad x=get(\mathcal{M})$

   (b) $\dfrac{}{{}_F[a\{u\};E] \xrightarrow{G=\{\cup_{i\in\mathcal{H}}\{shift(0\leq i\leq u,t)\}\setminus F_i=\{i\geq t\}\}, D=\{\texttt{false}\}, a, x}_* {}_{\{x\geq \tau(a)\}}[E]} \quad x=get(\mathcal{M})$

3. (a) $\dfrac{}{\emptyset[i\{u\};E] \xrightarrow{G=\{c_\emptyset \leq u\}, D:=G, i, x}_* {}_{\{x\geq 0\}}[E]} \quad x=get(\mathcal{M})$

   (b) $\dfrac{}{{}_F[i\{u\};E] \xrightarrow{G=\{\cup_{i\in\mathcal{H}}\{shift(0\leq i\leq u,t)\}\setminus F_i=\{i\geq t\}\}, D:=G, i, x}_* {}_{\{x\geq 0\}}[E]} \quad x=get(\mathcal{M})$

4. (a) $\dfrac{SP=\{\min \sim t \sim \max\} \qquad \sim \in \{<,\leq\}}{\emptyset[a@t[SP];E] \xrightarrow{G=\{[c_\emptyset/t]SP\}, D=\{\texttt{false}\}, a, x}_* {}_{\{x\geq \tau(a)\}}[[val(c_\emptyset)+\tau(a)/t]E]} \quad x=get(\mathcal{M})$

   (b) $\dfrac{SP=\{\min \sim t \sim \max\} \qquad \sim \in \{<,\leq\}}{{}_F[a@t[SP];E] \xrightarrow{G=\{\cup_{i\in\mathcal{H}}\{[i/t]shift(SP,d)\}\setminus F_i=\{i\geq d\}\}, D=\{\texttt{false}\}, a, x}_* {}_{\{x\geq \tau(a)\}}[[val(j)+\tau(a)-\theta/t]E]}$

   avec : $x=get(\mathcal{M}) \wedge F_j=\{j\geq\theta\} \wedge F_j \in F$

5. $\dfrac{\mathcal{E} \xrightarrow{G,D,a,x}_* \mathcal{E}'}{\mathcal{F} \,[]\, \mathcal{E} \xrightarrow{G,D,a,x}_* \mathcal{E}' \qquad \mathcal{E} \,[]\, \mathcal{F} \xrightarrow{G,D,a,x}_* \mathcal{E}'}$

6. (a) i. $\dfrac{\mathcal{E} \xrightarrow{G,D,a,x}_* \mathcal{E}' \qquad a \notin L \cup \{\delta\}}{\mathcal{E} \;|[L]|\; \mathcal{F} \xrightarrow{G,D,a,y}_* \mathcal{E}'[y/x] \;|[L]|\; \mathcal{F}\setminus G} \quad y=get(\mathcal{M}-((\psi(\mathcal{E})\cup\psi(\mathcal{F}))-\psi_{\mathcal{H}}(G)))$

   ii. $\dfrac{\mathcal{E} \xrightarrow{G,D,a,x}_* \mathcal{E}' \qquad a \notin L \cup \{\delta\}}{\mathcal{F} \;|[L]|\; \mathcal{E} \xrightarrow{G,D,a,y}_* \mathcal{F}\setminus G \;|[L]|\; \mathcal{E}'[y/x]} \quad y=get(\mathcal{M}-((\psi(\mathcal{E})\cup\psi(\mathcal{F}))-\psi_{\mathcal{H}}(G)))$

   (b) $\dfrac{\mathcal{E} \xrightarrow{G,D,a,x}_* \mathcal{E}' \qquad \mathcal{F} \xrightarrow{G',D',a,y}_* \mathcal{F}' \qquad a \in L \cup \{\delta\}}{\mathcal{E} \;|[L]|\; \mathcal{F} \xrightarrow{G\cup G', D\cup D', a, z}_* \mathcal{E}'[z/x]\setminus G' \;|[L]|\; \mathcal{F}'[z/y]\setminus G} \quad z=get(\mathcal{M}-((\psi(\mathcal{E})\cup\psi(\mathcal{F}))-(\psi_{\mathcal{H}}(G)\cup\psi_{\mathcal{H}}(G'))))$

7. (a) $\dfrac{\mathcal{E} \xrightarrow{G,D,a,x}_* \mathcal{E}' \qquad a \notin L}{hide\ L\ in\ \mathcal{E} \xrightarrow{G,D,a,x}_* hide\ L\ in\ \mathcal{E}'}$

(b) $\dfrac{\mathcal{E} \xrightarrow{G,D,a,x}_* \mathcal{E}' \qquad a \in L}{\textit{hide } L \textit{ in } \mathcal{E} \xrightarrow{G,D:=G,i,x}_* \textit{hide } L \textit{ in } \mathcal{E}'}$

8. $\dfrac{\mathcal{E} \xrightarrow{G,D,a,x}_* \mathcal{E}' \qquad d>0}{\Delta^d \mathcal{E} \xrightarrow{shift(G,d),shift(D,d),a,x}_* \mathcal{E}'}$

9. (a) $\dfrac{\mathcal{E} \xrightarrow{G,D,a,x}_* \mathcal{E}' \qquad a \neq \delta}{\mathcal{E} \gg F \xrightarrow{G,D,a,x}_* \mathcal{E}' \gg F}$

(b) $\dfrac{\mathcal{E} \xrightarrow{G,D,\delta,x}_* \mathcal{E}'}{\mathcal{E} \gg E \xrightarrow{G,D,i,x}_* \{x \geq 0\}[E]}$

10. (a) $\dfrac{\mathcal{E} \xrightarrow{G,D,a,x}_* \mathcal{E}' \qquad a \neq \delta}{\mathcal{E} \,[\!> \mathcal{F} \xrightarrow{G,D,a,y}_* \mathcal{E}'[y/x] \,[\!> \mathcal{F} \backslash G} \; y = get(\mathcal{M} - ((\psi(\mathcal{E}) \cup \psi(\mathcal{F})) - \psi_{\mathcal{H}}(G)))$

(b) $\dfrac{\mathcal{E} \xrightarrow{G,D,\delta,x}_* \mathcal{E}' \qquad \forall z \in \psi(\mathcal{F})}{\mathcal{E} \,[\!> \mathcal{F} \xrightarrow{G,D,\delta,y}_* \mathcal{E}'[y/x] \,[\!> \; \psi_{DC(\mathcal{F})-G_z}[stop]} \; y = get(\mathcal{M} - ((\psi(\mathcal{E}) \cup \psi(\mathcal{F})) - \psi_{\mathcal{H}}(G)))$

(c) $\dfrac{\mathcal{F} \xrightarrow{G,D,a,x}_* \mathcal{F}' \qquad \forall z \in \psi(\mathcal{E})}{\mathcal{E} \,[\!> \mathcal{F} \xrightarrow{G,D,a,y}_* \; \psi_{DC(\mathcal{E})-G_z}[stop] \,[\!> \mathcal{F}'[y/x]} \; y = get(\mathcal{M} - ((\psi(\mathcal{E}) \cup \psi(\mathcal{F})) - \psi_{\mathcal{H}}(G)))$

11. (a) $\dfrac{\mathcal{E} \xrightarrow{G,D,a,x}_* \mathcal{E}' \qquad a \notin \{a_1,\ldots,a_n\}}{\mathcal{E}[b_1/a_1,\ldots,b_n/a_n] \xrightarrow{G,D,a,x}_* \mathcal{E}'[b_1/a_1,\ldots,b_n/a_n]}$

(b) $\dfrac{\mathcal{E} \xrightarrow{G,D,a,x}_* \mathcal{E}' \qquad a = a_i \;\; (1 \leq i \leq n)}{\mathcal{E}[b_1/a_1,\ldots,b_n/a_n] \xrightarrow{G,D,b_i,x}_* \mathcal{E}'[b_1/a_1,\ldots,b_n/a_n]}$

12. $\dfrac{P := E \qquad {}_F[E] \xrightarrow{G,D,a,x}_* \mathcal{F}}{{}_F[P] \xrightarrow{G,D,a,x}_* \mathcal{F}}$

Avec, $c_\emptyset$ une horloge particulière créée et initialisée au moment de la sensibilisation du système.

Notation 6.1 *Nous avons adopté la notation suivante :*

Pour les actions observables ($\in \mathcal{G}$), nous avons

- $a@t\,[t \leq d]\,;E = a\,\{d\}\,;E$ *(le prédicat de sélection SP de la forme $t \leq d$ est remplacé par une restriction temporelle), avec t une variable non libre dans E.*
- $a@t\,[d \leq t \leq d']\,;E = \Delta^d a\,\{d'\}\,;E$, *avec $d \leq d'$ et t une variable non libre dans E.*

Si i est une action interne, alors

- $i@t\,\{0\}\,;E = i@t;E$ *(Si $\{d\}$ est omis, alors $d = 0$).*
- $i@t\,\{d\}\,;E = i\,\{d\}\,;E$ *si t une variable non libre dans E.*

Par convention, si l'intervalle de sensibilisation $\{d\}$ *est omis dans* $a\,\{d\}\,;E$*, alors* $d = \infty$*. De même, si* SP *est omis, alors* $SP = \texttt{true}$*. Si la contrainte d'urgence* D *est omise au niveau d'une transition, alors* $D = \{\texttt{false}\}$*.*

Dans les règles 1a et 2a, une action offerte dans un laps de temps égale à u et ne dépendant de la fin d'aucune autre peut s'exécuter à condition que la valeur de l'horloge $c_\emptyset$ ne dépasse pas la valeur u.

Dans le cas où le déclenchement d'une action a dépend de la fin d'au moins une autre, la contrainte de sensibilisation G est construite d'une part à partir des valeurs des durées des horloges correspondantes aux actions dont a y dépend, et d'autre part par l'intervalle de l'offre u de a, ce qui est exprimé par les règles 1b et 2b.

Dans les quatre premières règles de la Définition 6.1, toutes les actions ne sont pas urgentes, ce qui explique que les contraintes D sont toujours mises à $\texttt{false}$.

Les règles 3a et 3b expriment le tirage d'une action interne. Cette action devient urgente à la fin du laps de temps de sensibilisation d. Dans ces règles, la contrainte d'urgence D est construite à partir de la contrainte de sensibilisation G. Comme nous l'avons déjà mentionné, la sémantique des deux contraintes diffère : au moment de la satisfaction de D, l'action doit être tirée. Le même raisonnement s'applique aux actions cachées où l'urgence est exprimée à travers la règle 7b.

Nous allons maintenant expliquer l'utilisation de l'opérateur @ à travers l'exemple de l'expression D-LOTOS $E = a@t_1\,[t_1 \leq 3]\,;b@t_2\,[t_2 \leq t_1]\,;\,stop$. Supposons que les durées respectives de a et b sont 20 et 7. Dès que l'expression $a@t_1\,[t_1 \leq 3]\,;F$ est équivalente à $a\,\{3\}\,;\,[d/t_1]\,F$ (d est l'intervalle entre la sensibilisation et la fin d'exécution de a), et le début de l'action a ne dépend de la fin d'aucune autre action, nous allons juste substituer la variable t_1 au niveau du prédicat $t_1 \leq 3$ par l'horloge $c_\emptyset$ pour avoir la contrainte de sensibilisation $c_\emptyset \leq 3$. Dans l'expression restante $b@t_2\,[t_2 \leq t_1]\,;\,stop$, toutes les occurrences libres de la variable t_1 reçoivent le laps de temps entre la sensibilisation ($c_\emptyset = 0$) et la fin d'exécution de a (valeur actuelle de $c_\emptyset$, qui sera notée $val\,(c_\emptyset)$, plus la durée de a, $\tau\,(a)$). Donc, cet intervalle est égal à $val\,(c_\emptyset) + \tau\,(a) - 0 = val\,(c_\emptyset) + \tau\,(a)$. Cela est exprimé par la règle 4a de la Définition 6.1.

La transition suivante est alors possible :

$$\underbrace{\emptyset\,[E]}_{config_0} \xrightarrow{c_\emptyset \leq 3, a, x} \underbrace{_{\{x \geq 20\}}\,[b@t_2\,[t_2 \leq val\,(c_\emptyset) + 20]\,;\;stop]}_{config_1}$$

Dans la configuration $config_1$, le domaine de sensibilisation de b est $t_2 \in [0, val\,(c_\emptyset) + 20]$, et comme l'action b dépend de la terminaison de a qui dure 20 unités de temps et qui a

comme horloge x, le domaine de sensibilisation de b devient $x \in [0+20, val(c_\emptyset)+20+20]$. De manière plus générale, si nous avons plusieurs horloges dans la contrainte sur les durées F de la configuration source, le décalage de l'intervalle implique toutes ces horloges, ce qui est formulé par la règle 4b. Dans notre exemple, la contrainte sur le tir de l'action b n'implique que l'horloge x avec un décalage de 20 unités de temps.

La transition suivante devient possible :

$$\underbrace{\{x \geq 20\}\,[b@t_2\,[t_2 \leq val\,(c_\emptyset)+20]\,;\ stop]}_{config_1} \xrightarrow{20 \leq x \leq val(c_\emptyset)+40, b, x} \underbrace{\{x \geq 7\}\,[stop]}_{config_2}$$

Maintenant, si à la place de *stop* nous avons l'expression $c@t_3\,[t_3 \leq t_1 + t_2]\,;\ stop$ (c'est-à-dire l'expression E devient $a@t_1\,[t_1 \leq 3]\,; b@t_2\,[t_2 \leq t_1]\,; c@t_3\,[t_3 \leq t_1 + t_2]\,; stop$), avec c de durée 18, la transition représentant le début d'exécution de l'action c sera gardée par le domaine $x \in \left[0, \underbrace{val\,(c_\emptyset)+20}_{t_1} + \underbrace{val\,(x)+7-20}_{t_2}\right]$, décalé de 7 unités, c'est-à-dire $7 \leq x \leq \underbrace{val\,(c_\emptyset)+20}_{t_1} + \underbrace{val\,(x)+7-20}_{t_2} + 7$. Autrement dit, t_1 contient toujours la valeur $val\,(c_\emptyset)+20$, tandis que la variable t_2 contient $val\,(x)+7-20$ car l'action b est sensibilisée lorsque $x = 20$, et la fin d'exécution de b est mémorisé dans $\underbrace{val\,(x)}_{\text{début de } b} + \underbrace{7}_{\tau(b)}$. De manière plus générale, si la transition b est gardée par une contrainte construite de plusieurs horloges telle que $x \in [\min_x, \max_x] \wedge y \in [\min_y, \max_y] \wedge ...$, la valeur de t_2 dans l'expression restante $c@t_3\,[t_3 \leq t_1 + t_2]\,;\ stop$ peut être exprimée à l'aide d'*une seule* horloge parmi $\{x, y, ...\}$, soit j, puisque les laps de temps sont égaux quelque soit l'échelle de temps (c'est-à-dire, $val\,(x)+7-20 = val\,(y)+7-d_1 = val\,(z)+7-d_2 = ...$ et $d_1, d_2, ... \in \mathbb{R}^+$). Tout cela est exprimé par la règle 4b, d'où la transition suivante :

$$\underbrace{\{x \geq 7\}\,[c@t_3\,[t_3 \leq val\,(c_\emptyset)+val\,(x)+7]\,;\ stop]}_{config_2} \xrightarrow{7 \leq x \leq val(c_\emptyset)+val(x)+14, c, x} \underbrace{\{x \geq 18\}\,[stop]}_{config_3}$$

Concernant l'utilisation de l'opérateur @ avec les actions internes, les règles 3a et 3b s'appliqueront de la même manière sur l'expression $i@t\,\{d\}\,; E$ avec t non libre dans E.

La règle 8 concerne le retardement d'une action a d'une certaine quantité de temps d. On note que si l'opérateur de délai Δ est appliqué à une configuration $\mathcal{E}$, seule la *première* action tirée de la configuration $\mathcal{E}$ sera retardée.

Remarque 6.1 *Notons que selon la Remarque 5.1, nous ne considérons que les domaines d'urgence fermés à gauche, donc la forme des prédicats de sélection* $SP =$

$\{\min \sim t \sim \max\} \mid \sim \in \{<, \leq\}$ *employée dans les règles 4a et 4b se réduit en* $SP = \{\min \leq t \sim \max\} \mid \sim \in \{<, \leq\}$.

Remarque 6.2 *Sans perte de généralité, nous avons recouru au langage D-LOTOS dans lequel les durées des actions sont fixées une fois pour toute au moment de la sensibilisation du système. Le cas où les actions ont une durée choisie dans un intervalle* $[m, M]$ *ne sont pas considérés.*

6.3 Pour les réseaux de Petri

Nous essayons maintenant de donner une sémantique de maximalité en terme de systèmes de transitions étiquetées maximales au modèle des réseaux de Petri place/transition. Il est à noter qu'une sémantique de maximalité et une relation de bisimulation de maximalité ont été définies par R. Devillers [Dev92a, Dev92b] pour les réseaux de Petri place/transition en termes de réseaux d'occurrences. Ces derniers sont des structures infinies si le comportement du système correspondant est infini. Par conséquent, cette approche est intéressante du point de vue théorique. L'avantage de l'utilisation de modèle des systèmes de transitions étiquetées maximales est que ce dernier est un modèle implémentable.

Dans une autre partie de cette section, nous donnerons deux méthodes pour réduire un système de transitions étiquetées maximales généré à partir d'un réseau de Petri, à savoir l'agrégation de transitions et la réduction modulo la α-équivalence.

Nous commençons par introduire les notations et les fonctions utiles à la définition du graphe de marquage associé à un système étiqueté (voir Définition 3.9) dans une approche de maximalité, ce qui permettra de générer, pour tout système étiqueté, un système de transitions étiquetées maximales.

Soit le réseau de Petri marqué de la Figure 6.1.(a). Après le tir de la transition t_1, il est évident que les tirs des transitions t_2 et t_3 sont conditionnés par la fin de l'action liée à la transition t_1. Pour capturer cette dépendance causale entre les tirs des transitions, nous considérons que les jetons produits par le tir de la transition t_1 sont liés à cette transition, entre autres le jeton de la place s_2 et le jeton de la place s_3. On peut remarquer que, dans l'état initial, le jeton dans s_1 n'est lié à aucune transition, ce jeton est dit *libre* dans cet état là. Dans le cas où la transition t_2 serait tirée, on pourrait déduire que l'action associée au tir de t_1 s'est terminée. De ce fait, le jeton dans s_3 deviendra libre. Le marquage résultant du tir de la transition t_2 est donné par la Figure 6.1.(c).

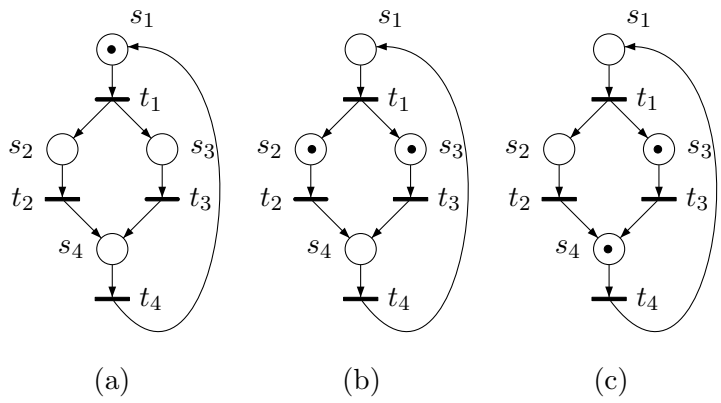

FIGURE 6.1 – Exemple d'un réseau de Petri marqué

Pour distinguer entre jetons libres et jetons liés dans une place, on peut imaginer qu'une place est composée de deux parties disjointes. La partie de gauche contiendra les jetons libres tandis que celle de droite contiendra les jetons liés. Dans une place, le nombre des jetons libres sera désigné par $\mathcal{FT}$ (pour *free tokens*) alors que l'ensemble des jetons liés sera désigné par $\mathcal{BT}$ (pour *bound tokens*). On obtiendra ainsi la succession des marquages de la Figure 6.2.

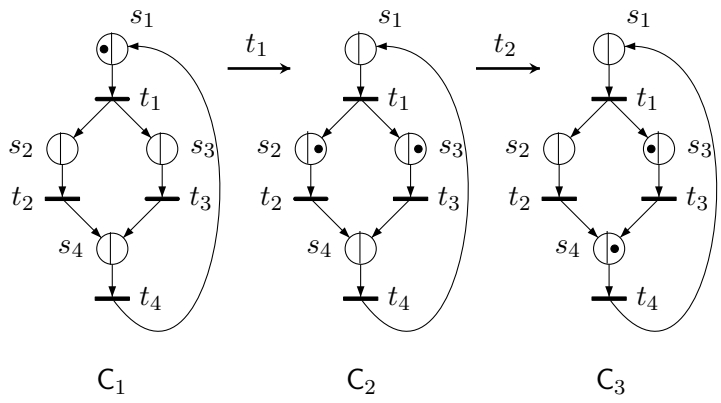

FIGURE 6.2 – Jetons libres et jetons liés dans un marquage

A partir de la configuration C_2 on aurait pu tirer la transition t_3. La configuration résultante serait C_4. (Voir Figure 6.3). Le jeton dans s_4 est maintenant lié à la transition t_3 et le jeton dans s_2 est désormais libre puisque la fin de l'action associée à la transition t_1 implique la libération du jeton de s_2.

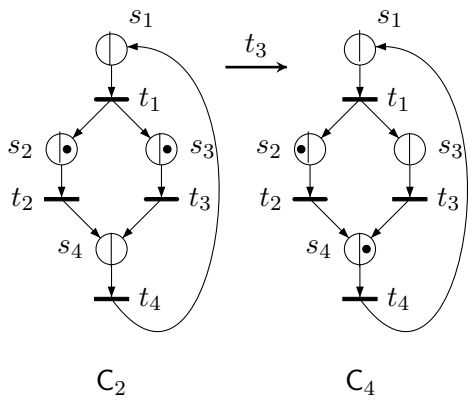

FIGURE 6.3 – Franchissement de la transition t_3

Une question qui se pose maintenant est comment lier un jeton à une transition ? Pour y répondre, on considère le réseau de Petri marqué de la Figure 6.4.(a).

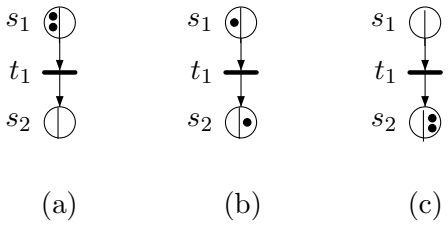

FIGURE 6.4 – Liaison des jetons

Par un tir de la transition t_1, on obtiendra le réseau marqué de la Figure 6.4.(b). A partir de ce marquage, on remarque que la transition t_1 est franchissable. Le franchissement de cette transition conduira à la configuration de la Figure 6.4.(c).

Les deux jetons de la place s_2 sont liés. En effet, l'un est lié au premier tir de la transition t_1 alors que l'autre est lié au second tir de la même transition (deux actions associées à t_1 pouvant être en exécution parallèle). Pour lever cette ambigüité, chaque tir de transition sera identifié par un nom d'évènement. De ce fait, le lien d'un jeton sera caractérisé aussi bien par la transition qui la produite que par le nom d'évènement identifiant le franchissement de cette transition. Les successions des franchissements de l'exemple précédent sont illustrées par la Figure 6.5.

Dans la configuration C_2, l'ensemble des jetons liés dans s_2 est $\mathcal{BT} = \{t_{1_x}\}$ alors

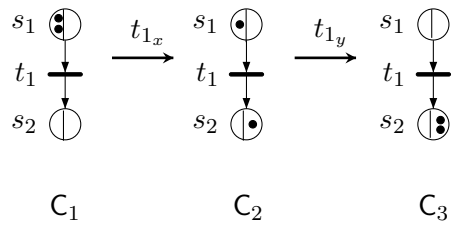

FIGURE 6.5 – Successions de franchissements de t_1

que l'ensemble des jetons liés dans s_2 de la configuration C_3 est $\mathcal{BT} = \{t_{1_x}, t_{1_y}\}$. Le nom de l'évènement x désigne le premier franchissement de la transition t_1 alors que y désigne le deuxième franchissement de cette même transition.

Un autre problème se pose quant aux jetons liés à un même tir de transition. Pour le voir, considérons le réseau de Petri de la Figure 6.6.(a).

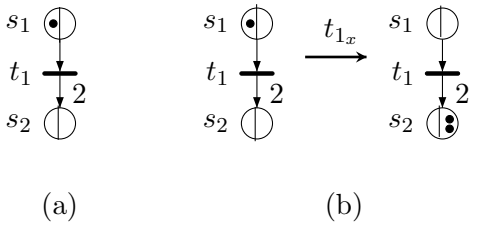

FIGURE 6.6 – Réseau de Petri avec arc sortant de poids supérieur à un

Par le franchissement de la transition t_1 on obtient la dérivation de la Figure 6.6.(b).

La partie droite $\mathcal{BT}$ de la place s_2 contient deux jetons liés au tir t_{1_x}, c'est-à-dire $\mathcal{BT} = \{t_{1_x}, t_{1_x}\}$. Etant donné que $\mathcal{BT}$ est un ensemble, nous considérons qu'un jeton lié est un triplet (n, t, x) de $\mathbb{N}\times T \times \mathcal{M}$, (noté aussi nt_x), où n est le nombre d'instances, t est la transition productrice de ce jeton et x est le nom d'évènement associé au franchissement de la transition t. On note $\mathcal{BT} = \{n_1 t_{1x_1}, n_2 t_{2x_2}, \ldots\}$ l'ensemble (éventuellement vide) des jetons liés. Dans l'exemple précédent $\mathcal{BT} = \{2t_{1_x}\}$.

Par conséquent, le *marquage* d'une place s est un couple $(\mathcal{FT}, \mathcal{BT})$ où $\mathcal{FT}$ est le nombre de *jetons libres* dans s.

Définition 6.4 *Etant donné un réseau de Petri $PN = (S, T, W)$, le marquage de PN est désormais une fonction $M : S \to \mathbb{N}\times 2^{\mathbb{N}\times T\times\mathcal{M}}$. Entre autre, le marquage $M(s)$*

d'une place $s \in S$ est un couple $(\mathcal{FT}, \mathcal{BT})$ tel que $\mathcal{FT} \in \mathbb{N}$ et $\mathcal{BT} \in 2^{\mathbb{N}\times T\times \mathcal{M}}$ désignent respectivement le nombre de jetons libres et l'ensemble (éventuellement vide) de jetons liés dans la place s.

Dans ce qui suit, un réseau de Petri muni d'un marquage sera appelé *configuration*. $|M(s)|$ désigne le nombre total de jetons dans la place s. Si $M(s) = (\mathcal{FT}, \mathcal{BT})$ tel que $\mathcal{BT} = \{n_1 t1_{x_1}, \ldots, n_m tm_{x_m}\}$ alors $|M(s)| = \mathcal{FT} + |\mathcal{BT}|$ avec $|\mathcal{BT}| = \sum_{i=1}^{m} n_i$ est le cardinal de l'ensemble des jetons liés dans la place s. Parfois on utilisera l'écriture $\mathcal{FT}(s)$ et $\mathcal{BT}(s)$ pour désigner les composantes du marquage $M(s)$ de la place s.

Après avoir montré les jetons produits par le franchissement d'une transition, un point reste à éclairer et qui concerne l'identification des jetons consommés par le franchissement de celle-ci. Pour cela, considérons le réseau de Petri de la Figure 6.7.

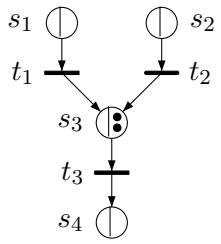

FIGURE 6.7 – Identification des jetons consommés

Nous admettons qu'un jeton de s_3 est lié au franchissement de t_1 (t_{1_x}) et que l'autre jeton est lié au franchissement de t_2 (t_{2_y}).

Parmi les jetons liés dans s_3, on veut connaitre le jeton consommé dans le premier franchissement de t_3 et celui consommé dans le second franchissement de cette même transition. Cette information est indispensable pour connaitre, dans chaque configuration, les actions (associées aux transitions) qui ont terminé leur exécution. Pour ce faire, nous associons au niveau d'un franchissement les noms d'évènements identifiant les jetons liés consommés par ce franchissement. Ceci nous donne la succession des franchissements de la Figure 6.8.

Dans l'exemple de la Figure 6.2, nous avons noté qu'après le franchissement de la transition t_2, le jeton qui était lié à la transition t_1 dans la place s_3 devient libre dans le marquage résultant. Ceci est dû au fait que le franchissement de la transition t_3 est conditionné par la fin de l'action relative à la transition t_1. Dans le paragraphe suivant, nous donnons quelques définitions préliminaires qui nous permettront la proposition de

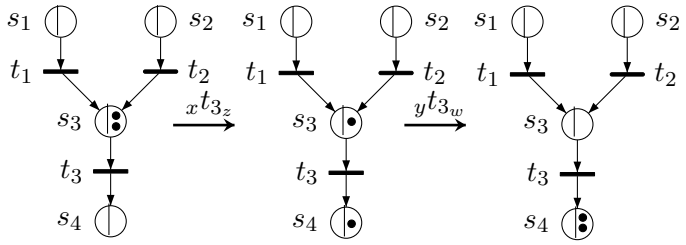

FIGURE 6.8 – Noms d'évènements identifiant les jetons consommés

la méthode de génération d'un graphe de marquage dans le contexte de la sémantique de maximalité.

6.3.1 Définitions préliminaires

Définition 6.5 *Etant donné un réseau de Petri (S, T, W) muni d'un marquage M :*

- *L'ensemble des noms des évènements maximaux dans M est l'ensemble de tous les noms d'évènements identifiant les jetons liés dans le marquage M. Formellement, la fonction ψ sera utilisée pour le calcul de cet ensemble, elle peut être définie de la manière suivante : $\psi(M) = \bigcup_{s \in S} \bigcup_{i=1}^{ms} xs_i$ tel que $M(s) = (\mathcal{FT}, \mathcal{BT})$ avec $\mathcal{BT} = \{(ns_1, ts_1, xs_1), \ldots, (ns_{ms}, ts_{ms}, xs_{ms})\}$.*

- *$\mathcal{N} \subset \mathcal{M}$ étant une ensemble fini non vide de noms d'évènements.* $\mathsf{makefree}(\mathcal{N}, M)$ *est définie récurssivement par :*

  - $\mathsf{makefree}(\{x_1, x_2, \ldots, x_n\}, M) = \mathsf{makefree}(\{x_2, \ldots, x_n\}, \mathsf{makefree}(\{x_1\}, M))$
  - $\mathsf{makefree}(\{x\}, M) = M'$ *tel que pour tout $s \in S$, si $M(s) = (\mathcal{FT}, \mathcal{BT})$ alors :*
    - *S'il existe $(n, t, x) \in \mathcal{BT}$ alors $M'(s) = (\mathcal{FT} + n, \mathcal{BT} - \{(n, t, x)\})$ (Conversion des n jetons liés identifiés par le nom d'évènement x en jetons libres).*
    - *Sinon, $M'(s) = M(s)$.*

- *t étant une transition de T ; t est dite sensibilisée par le marquage M ssi $|M(s)| \geq W(s,t)$ pour tout $s \in S$. L'ensemble de toutes les transitions sensibilisées par le marquage M sera désigné par* $\mathsf{enabled}(M)$.

- *Le marquage M est dit minimal pour le franchissement de la transition t ssi $|M(s)| = W(s,t)$ pour tout $s \in S$.*

- *Soient* M_1 *et* M_2 *deux marquages du réseau de Petri* (S,T,W). $M_1 \Subset M_2$ *ssi* $\forall s \in S$, *si* $M_1(s) = (\mathcal{FT}_1, \mathcal{BT}_1)$ *et* $M_2(s) = (\mathcal{FT}_2, \mathcal{BT}_2)$ *alors* $\mathcal{FT}_1 \leq \mathcal{FT}_2$ *et* $\mathcal{BT}_1 \Subset \mathcal{BT}_2$ *tel que la relation* $\Subset$ *et étendue aux ensembles des jetons liés de la manière suivante :*

  - $\mathcal{BT}_1 \Subset \mathcal{BT}_2$ *ssi* $\forall (n_1,t,x) \in \mathcal{BT}_1$, $\exists (n_2,t,x) \in \mathcal{BT}_2$ *tel que* $n_1 \leq n_2$.

- *Soient* M_1 *et* M_2 *deux marquages du réseau de Petri* (S,T,W) *tels que* $M_1 \Subset M_2$. *La différence* $M_2 - M_1$ *est un marquage* M_3 *(*$M_2 - M_1 = M_3$*) tel que pour tout* $s \in S$, *si* $M_1(s) = (\mathcal{FT}_1, \mathcal{BT}_1)$ *et* $M_2(s) = (\mathcal{FT}_2, \mathcal{BT}_2)$ *alors* $M_3(s) = (\mathcal{FT}_3, \mathcal{BT}_3)$ *avec :*

  - $\mathcal{FT}_3 = \mathcal{FT}_2 - \mathcal{FT}_1$,
  - $\forall (n_1,t,x) \in \mathcal{BT}_1$, $(n_2,t,x) \in \mathcal{BT}_2$ *si* $n_1 \neq n_2$ *alors* $(n_2 - n_1, t, x) \in \mathcal{BT}_3$.

- $\mathsf{Min}(M,t) = \{M' | M' \Subset M\}$ *et* M' *est minimal pour le franchissement de la transition* t.

- *Soit* $\mathcal{M}$ *un ensemble. La fonction get* : $2^{\mathcal{M}} - \{\emptyset\} \rightarrow \mathcal{M}$ *est une fonction qui satisfait* $get(E) \in E$ *pour* $\forall E \in 2^{\mathcal{M}} - \{\emptyset\}$.

- *Etant donnés un marquage* M, *une transition* t *et un nom d'évènement* $x \notin \psi(M)$ *;* $\mathsf{occur}(t,x,M) = M'$ *tel que* $\forall s \in S$, *si* $M(s) = (\mathcal{FT}, \mathcal{BT})$ *alors* $M'(s) = (\mathcal{FT}, \mathcal{BT}')$ *avec* $\mathcal{BT}' = \mathcal{BT} \cup \{W(t,s), t, x)\}$ *si* $W(t,s) \neq 0$ *et* $\mathcal{BT}' = \mathcal{BT}$ *sinon. Ainsi,* M' *est la marquage résultant de l'ajout au marquage* M *des jetons liés à la transition* t.

6.3.2 Construction du graphe de marquage

Soit la donnée d'un système étiqueté $\Sigma = (S,T,W,M_0,\lambda)$. Le graphe de marquage GM étiqueté par λ associé à Σ est celui dont les états sont définis par tous les marquages accessibles à partir du marquage initial M_0. La règle de dérivation des marquages est définie récurssivement par la Définition 6.6.

Définition 6.6 *M étant un marquage accessible du réseau de Petri marqué* (S,T,W,M_0), $t \in \mathsf{enabled}(M)$, *alors pour tout* $M" \in \mathsf{Min}(M,t)$, $E = \psi(M")$ *et* $M''' = \mathsf{makefree}(E, M - M")$ *; la dérivation* $M \xrightarrow{E\,t_x} M'$ *(notée aussi* $(M, {}_E\,t_x, M')$*) est possible telle que :*

- *E est l'ensemble des noms d'évènements maximaux associés aux actions dont les fins d'exécution sont impératives au début de l'action associée au franchissement de la transition t.*
- *$x = get(\mathcal{M} - \psi(M'''))$. On choisit un nom d'évènement de l'ensemble $\mathcal{M}$ qui n'est pas associé à une action pouvant rester en cours d'exécution après le franchissement de la transition t. Notons que les noms d'évènements appartenant à E peuvent être réutilisés. Cette méthode de choix des noms d'évènements permet la considération des systèmes à comportements cycliques.*
- *$M' = \mathsf{occur}(t, x, M''')$ est le marquage résultant du rajout des jetons liés à la transition t au marquage M'''.*

6.3.3 Propriétés

Proposition 6.1 *Etant donné un système étiqueté $\Sigma = (S, T, W, M_0, \lambda)$ dont le graphe de marquage GM est construit selon la Définition 6.6, alors la structure $\Sigma_{stem} = (GM, \lambda, \mu, \xi, \psi)$ est un système de transitions étiquetées maximales avec :*

- *$GM = \langle Sg, Tg, \alpha, \beta, M_0 \rangle$ est le graphe de marquages associé à Σ tel que*
  - *Sg est l'ensemble des états défini par l'ensemble des marquages accessibles à partir du marquage initial M_0.*
  - *$Tg = \{(M,_E t_x, M')\}$ tel que $M, M' \in Sg$ et $(M,_E t_x, M')$ est une dérivation valide.*
  - *Pour $(M,_E t_x, M') \in Tg$, $\alpha((M,_E t_x, M')) = M$ et $\beta((M,_E t_x, M')) = M'$.*
- *$\psi : Sg \to 2^{\mathcal{M}}$ est la fonction définie précédemment dans la Définition 6.5.*
- *Pour $d = (M,_E t_x, M') \in Tg$ on pose $\lambda(d) = \lambda(t)$, $\mu(d) = E$ et $\xi(d) = x$.*

Preuve. Remarquons tout d'abord que le marquage initial M_0 ne contient que des jetons libres, de ce fait $\psi(M_0) = \emptyset$. D'autre part, pour $d = (M,_E t_x, M') \in Tg$, D'après la Définition 6.6, $\mu(d) = E \subseteq \psi(\alpha(d)) = \psi(M)$, $\xi(t) \notin \psi(M) - \mu(d)$ et $\psi(\beta(t)) = \psi(M') = (\psi(M) - \mu(d)) \cup \{\xi(d)\}$. D'où le résultat. □

Proposition 6.2 *Etant donné un réseau de Petri marqué (S, T, W, M_0), alors l'ensemble des séquences de transitions généré dans une approche d'entrelacement est le même que celui généré dans l'approche de maximalité.*

Preuve. Dérive directement du fait que dans l'approche de maximalité, la condition de franchissement d'une transition ne prend en compte que le nombre de jetons dans les places et non la nature de ces jetons. Ce qui revient aux mêmes conditions de franchissement des transitions dans l'approche d'entrelacement. □

Définition 6.7 *Soient deux systèmes étiquetés* $\Sigma_1 = (S_1, T_1, W_1, M1_0, \lambda_1)$ *et* $\Sigma_2 = (S_2, T_2, W_2, M2_0, \lambda_2)$. Σ_1 *et* Σ_2 *sont dits* maximalement bisimilaires *ssi leurs systèmes de transitions étiquetées maximales respectifs sont maximalement bisimilaires.*

Exemple 6.1 *Considérons l'exemple des deux systèmes étiquetées* Sys_1 *et* Sys_2 *de la Figure 6.9.(a) (cet exemple est tiré de* [Dev92a]*). Par l'application de l'approche de la Section 6.3.2, les systèmes de transitions étiquetées maximales correspondant sont donnés par la Figure 6.9.(b). Le lecteur peut facilement vérifier que ces deux systèmes sont maximalement bisimilaires.*

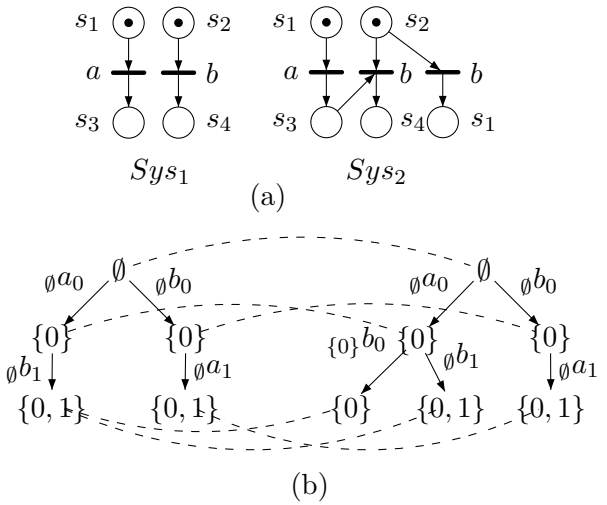

FIGURE 6.9 – Systèmes maximalement bisimilaires

6.3.4 Agrégation de transitions

Nous introduisons maintenant une manière de réduire le système de transitions étiquetées maximales généré à partir d'un réseau de Petri en définissant une relation de bisimulation maximale sur les transitions. Le but étant d'éliminer les dérivations équivalentes en accord avec cette dernière relation.

Sémantique opérationnelle de maximalité pour les réseaux de Petri avec agrégation de transitions

Le graphe de marquage est généré de la même manière que l'approche de la Section 6.3.2, donc nous gardons les mêmes définitions de base. Or, pour atteindre notre objectif, nous changeons la sémantique de la fonction Min. Dans ce cas, un marquage minimal pour le tir d'une transition t est considéré comme un élément de l'ensemble $\mathsf{Min}(M,t)$ seulement si pour chaque place du marquage, les jetons liés sont pris uniquement dans le cas où la partie libre ne satisfait pas la pré-condition de cette transition. Ainsi, nous pouvons s'assurer qu'une transition t s'exécutera de manière séquentielle après une transition t' si elle ne peut pas s'exécuter indépendamment de cette même transition t'.

Formellement, $\mathsf{Min}(M,t)$ est l'ensemble des marquages $M' \Subset M$ tels que pour n'importe quel état s avec $M(s) = (\mathcal{FT}, \mathcal{BT})$, $M'(s)$ est défini comme suit :

$$M'(s) = \begin{cases} (w(s,t), \emptyset) & si\ \mathcal{FT} \geqslant w(s,t) \\ (\mathcal{FT}, \mathcal{BT}') & sinon \end{cases}$$

avec $\mathcal{BT}' \Subset \mathcal{BT}$ et $|\mathcal{BT}'| = w(s,t) - \mathcal{FT}$.

Relation de bisimulation de maximalité sur les transitions

Définition 6.8 *Soit $Marq$ un ensemble de marquages, T un ensemble de transitions et $\rightarrow$ une relation de dérivation entre les marquages comme définie dans la Définition 6.6.*

- *Une relation $\Re \subseteq 2^{Marq} \times 2^{Marq} \times \mathfrak{F}$ est dite une* relation de bisimulation de maximalité sur un ensemble de transitions *T ssi $\forall (M_i, M_i^{'}, Id_{A_i}) \in \Re$, $A_i \subseteq \psi(M_i)$ et $A_i \subseteq \psi(M_i^{'})$,*

  1. *Si $M_i \overset{E_i ti_x}{\longrightarrow} M_j$ alors $\exists M_i' \overset{E_i' ti_y}{\longrightarrow} M_j'$ /$\forall u \in A_i$ si $u \notin E_i$ alors $u \notin E_i'$ et pour $z = get(\mathcal{M} - ((\psi(M_i) - E_i) \cup (\psi(M_i') - E_i')))$: $(M_j[z/x], M_j'[z/y], Id_{A_{i+1}}) \in \Re$/ $A_{i+1} = (A_i - E_i) \cup \{z\}$.*
  2. *Si $M_i' \overset{E_i' ti_y}{\longrightarrow} M_j'$ alors $\exists M_i \overset{E_i ti_x}{\longrightarrow} M_j$ /$\forall u \in A_i$ si $u \notin E_i'$ alors $u \notin E_i$ et pour $z = get(\mathcal{M} - ((\psi(M_i) - E_i) \cup (\psi(M_i') - E_i')))$: $(M_j[z/x], M_j'[z/y], Id_{A_{i+1}}) \in \Re$/ $A_{i+1} = (A_i - E_i') \cup \{z\}$.*

- *Deux systèmes étiquetés $\Sigma_1 = (S_1, T, W_1, M_0^1, \lambda_1)$ et $\Sigma_2 = (S_2, T, W_2, M_0^2, \lambda_2)$ sont dits* maximalement bisimilaires sur T, *noté $\Sigma_1 \approx_m^T \Sigma_2$, ssi il existe une relation de bisimulation de maximalité $\Re$ sur T telle que $(M_0^1, M_0^2, \emptyset) \in \Re$.*

- $M_1 \approx_m^T /f\ M_2$ *dénote le fait que* $(M_1, M_2, f) \in \Re$.

Propriétés

Corollaire 6.1 $\approx_m^T \subseteq\ \approx_m$

Proposition 6.3 *Soit* M *un marquage avec* $A \subseteq \psi(M)$ *alors*

1. $M \approx_m^T /Id_{\psi(M)}\ M$
2. $M \approx_m^T /Id_A\ M$
3. *Si* $B = \psi(M) - A$ *alors* $\mathsf{makefree}(B, M)\ \approx_m^T /Id_A\ \ M$

Proposition 6.4 *Soit* Σ *un système avec le marquage initial* M_0, *et soit* σ *une séquence de transitions telle que* $M_0[\sigma\rangle M$ *et* $M_0[\sigma\rangle M'$ *avec* $D \subseteq \psi(M)$, $C \subseteq \psi(M')$ *et* $f : D \longrightarrow C$ *une bijection, alors* $M \approx_m^T /f\ M' \Longrightarrow \mathsf{makefree}(\psi(M) - D, M) \approx_m^T /f\ \mathsf{makefree}(\psi(M') - C, M')$

Proposition 6.5 *Soit* $f = Id_A$ *telle que* $A \subseteq \psi(M)$. *Si* $M \xrightarrow{E_1 t_x} M_1$ *et* $M \xrightarrow{E_2 t_y} M_2$ *tel que* $A \cap E_1 \subseteq A \cap E_2$ *alors pour* $z = get(\mathcal{M} - ((\psi(M) - E_1) \cup (\psi(M) - E_2)))$ *et* $B = (A - E_2) \cup \{z\}$ *nous avons* $M_1[z/x]\ \approx_m^T /Id_B\ \ M_2[z/y]$.

Théorème 6.1 *Soit* $\Sigma = (S, T, W)$ *un système étiqueté et soit* $mlts_1$ *(respectivement* $mlts_2$*) un système de transitions étiquetées maximales généré à partir de* Σ *utilisant la sémantique de maximalité (respectivement la sémantique de maximalité avec agrégation des transitions), alors* $mlts_1$ *et* $mlts_2$ *sont maximalement bisimilaires pour l'ensemble de transitions* T.

6.3.5 Réduction à la volée modulo la α-équivalence

Une réduction consiste en l'élimination des informations redondantes tout en préservant les propriétés d'un système. Dans cette section, nous utiliserons la relation de la α-équivalence comme un critère de comportements redondants, et la relation de bisimulation de maximalité comme une propriété à préserver. Comme illustration, le système de transitions étiquetées maximales de la Figure 6.11.(a) représente le comportement du réseau de Petri donné par la Figure 6.10. Il a été généré par l'application directe de la sémantique opérationnelle de maximaité pour les réseaux de Petri définie dans la Section 6.3.2.

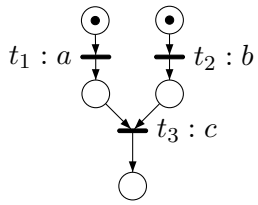

FIGURE 6.10 – Un réseau de Petri marqué

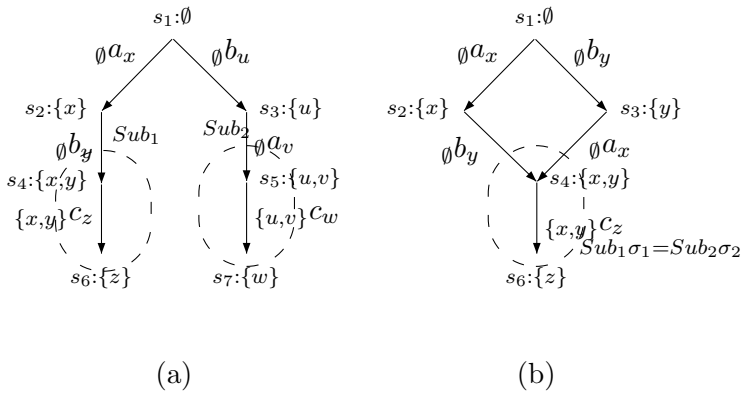

FIGURE 6.11 – Systèmes de transitions étiquetées maximales

Les deux sous-systèmes de transitions étiquetées maximales sub_1 et sub_2 de la Figure 6.11.(a) sont α-équivalents. En effet, il existe deux fonctions de substitution $\sigma_1 = \{x/x, y/y, z/z\}$ et $\sigma_2 = \{x/v, y/u, z/w\}$ telles que $sub_1\sigma_1 = sub_2\sigma_2$. Pour retirer cette redondance, nous devons au début (1) appliquer la fonction de substitution $\sigma_1 \cup \sigma_2$ sur le système de transitions étiquetées maximales de la Figure 6.11.(a), (2) grouper les états s_4 et s_5, et (3) retirer $sub_1\sigma_1$ ou bien $sub_2\sigma_2$. Comme résultat, nous obtenons le système de transitions étiquetées maximales de la Figure 6.11.(b).

Ce que nous avons vu dans l'exemple de la Figure 6.11.(b) est appliqué sur un système de transitions étiquetées maximales déjà généré. Cependant, notre but est de générer à-la-volée un système de transitions étiquetées maximales α-réduit. L'alternative consiste à vérifier, pour chaque configuration générée, si elle est α-équivalente avec une des configurations générées auparavant. Dans ce cas, une fonction de substitution est calculée et appliquée sur le système de transitions étiquetées maximales et une configuration est retirée. Ceci est donné par l'Algorithme 6.1 qui implémente la relation de la α-équivalence sur les configurations associées aux réseaux de Petri donnée par la

Définition 6.9.

Définition 6.9 *Soit $\Sigma = (S, T, W, M_0, \lambda)$ un système étiqueté. La relation de la α-équivalence est définie récursivement sur les configurations comme suit :*

- $M(s) =_\alpha M'(s)$ *ssi*
  - $\mathcal{FT}(s) = \mathcal{FT}'(s)$, *et*
  - $\forall (n, t, x) \in \mathcal{BT}(s), \exists (n, t, x') \in \mathcal{BT}'(s)$.
- $M =_\alpha M'$ *ssi* $\forall s \in S$, $M(s) =_\alpha M'(s)$.

L'Algorithme 6.1 permet, en commençant de la configuration initiale, de générer à-la-volée un système de transitions étiquetées maximales α-réduit. Cette construction est basée sur les configurations où pour chaque nouvelle configuration, nous extrayons de nouvelles configurations et de nouvelles transitions par l'utilisation de la sémantique opérationnelle de maximalité pour les réseaux de Petri introduite dans la Section 6.3.2.

Une configuration est considérée comme « nouvelle » si elle n'est pas α-équivalente avec l'une des configurations générées auparavant. Pour simplifier la construction de la fonction de substitution, nous avons associé, à chaque évènement maximal, son action appropriée.

6.4 Conclusion

Nous avons proposé dans ce chapitre une sémantique de maximalité pour trois modèles de spécification à savoir Basic LOTOS avec durées d'actions, D-LOTOS et les réseaux de Petri place/transition. Les modèles sémantiques sous-jacents ainsi générés sont les DATA (Durational Action Timed Automata), les DATA* et les systèmes de transitions étiquetées maximales. Dans la dernière partie, nous avons défini deux méthodes de réduction des structures de systèmes de transitions étiquetées maximales générées à partir de réseaux de Petri. Lorsque ces deux méthodes seront fusionnées, nous pouvons avoir un bon taux de réduction (voir les exemples données dans [SBB08, SBB09c]).

Algorithme 6.1 Génération à-la-volée de systèmes de transitions étiquetées maximales à partir de réseaux de Petri

Require: Un réseau de Petri places/transitions

Ensure: un système de transitions étiquetées maximales α-réduit

construire la configuration initiale $conf_0$
$list\_confs \leftarrow conf_0$
while il existe une configuration non traitée **do**
  $Sub \leftarrow \emptyset$
  sélectionner un élément $conf$ de $list\_confs$
  for all transition t **do**
    if $enabled(conf, t)$ **then**
      calculer l'ensemble Min
      for all élément $item$ de Min **do**
        $new\_edge \leftarrow (item, get\_transition\_name(t), get(\mathcal{M}))$
        $new\_conf \leftarrow firing(consume(makefree(item, conf), t))$
        if $\exists conf' = new\_conf$ tel que $conf'$ est un membre de $list\_confs$ **then**
          ajouter $new\_edge(conf, conf')$ à $list\_edges$
        else
          if $\exists conf" =_\alpha new\_conf$ tel que $conf"$ est un membre de $list\_confs$ **then**
            ajouter $new\_edge(conf, conf")$ à $list\_edges$
            calculer la liste de substitution Sub
          else
            ajouter new_conf à $list\_confs$
            ajouter $new\_edge(conf, new\_conf)$ à $list\_edges$
          end if
        end if
      end for
    end if
  end for
  substituer $list\_confs$, $list\_edges$ par l'utilisation de Sub
end while

Chapitre 7

Conclusion et prospective

Le travail présenté dans cette thèse est une contribution à la spécification des systèmes temps-réel. Le but prinicipal était l'étude de l'aptitude de quelques modèles de temps à exprimer, de manière naturelle, le comportement des systèmes temps-réel, à savoir les contraintes temporelles, les contraintes d'urgence, la non-atomicité temporelle et structurelle des actions ainsi que les durées explicites d'actions. En plus, nous avons étudié le côté *sémantique* de ces modèles en leur dotant d'une sémantique du vrai parallélisme appelée *sémantique de maximalité*.

D'abord, nous avons commencé par présenter l'activité de spécification et de vérification formelle des systèmes temps-réel, tout en s'intéressant à la sémantique de maximalité qui a été introduite.

Ensuite, nous avons passé en revue les modèles de base sur lesquels nous avons travaillé. Ces modèles sont les systèmes de transitions étiquetées maximales, les extensions temporelles et temporisés des algèbres de processus à l'image du langage D-LOTOS, le modèle des réseaux de Petri place/transition, et le modèle des automates temporisés avec quelques-unes de ses extensions et sous-classes. Nous notons que ces modèles cités, à part D-LOTOS, sont choisis à cause de leur utilisation extensive dans la littérature pour la spécification et la vérification formelle des systèmes à temps contraint. Le choix de l'étude du langage D-LOTOS est poussé par son aptitude à spécifier et les contraintes temporelles et les durées d'actions. D'autre côté, le modèle des réseaux de Petri est pris en considération à cause de son pouvoir d'expression de comportements concurrents et éventuellement parallèles. Enfin, le modèle des automates temporisés est un autre modèle parmi les plus utilisés dans la littérature. Nous nous inspirons essentiellement des notions d'horloges et de gardes pour la proposition de nos modèles de DATA et DATA*.

En ce qui concerne la vérification formelle, nous avons introduit dans le Chapitre 4 la technique du model checking dans laquelle nous avons montré que le passage vers l'utilisation d'une sémantique du vrai parallélisme était trivial. L'un des buts d'adopter la sémantique de maximalité pour le model checking étant la possibilité d'avoir un taux de réduction supérieur par rapport à l'utilisation de la sémantique d'entrelacement si on utilise des techniques de réduction comme dans [BS06].

Dans la partie contribution, nous avons introduit intuitivement et défini formellement deux modèles sémantiques de bas niveau : les DATA (Duational Action Timed Automata) et son extension temps-réel le modèle des DATA*. Le premier modèle est destiné à spécifier des systèmes avec durées explicites d'actions tandis que le deuxième est une extension du premier pour les systèmes temps-réel, c'est-à-dire les systèmes définis avec des exigences temporelles.

Dans une autre partie de cette thèse, nous avons donné une sémantique de maximalité en terme de DATA, DATA* et systèmes de transitions étiquetées maximales respectivement aux modèles de spécification Basic LOTOS, D-LOTOS et les réseaux de Petri place/transition. Le but de ces passagaes est l'exploitation des avantages de la sémantique de maximalité.

Pour conclure, notre travail peut être considéré comme un premier pas pour l'adoption d'une démarche de spécification et de vérification formelle des systèmes temps-réel basée sur l'expression de spécifications à la D-LOTOS ayant une sémantique de maximalité en terme de structures de bas niveau.

Nous envisageons aussi de donner une sémantique de maximalité à des extensions temporisées et/ou temporelles des réseaux de Petri. Nous pensons que ceci pourra être effectué en suivant le même principe que pour les réseaux de Petri place/transition.

Bibliographie

[ACD93] R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, May 1993.

[ACH+92] R. Alur, C. Courcoubetis, N. Halbwachs, D. L. Dill, and H. Wong-Toi. Minimization of timed transition systems. In *Proceedings of CONCUR'92*, volume 630 of *LNCS*, pages 340–354. Springer-Verlag, 1992.

[ACH94] R. Alur, C. Courcoubetis, and T. A. Henzinger. The observational power of clocks. In *Proceedings of CONCUR'94*, volume 836 of *LNCS*, pages 162–177. Springer-Verlag, 1994.

[ACH+95] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

[ACHH93] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho. Hybrid automata: An algorithmic approach to specification and verification of hybrid systems. In *Proc. Hybrid Systems*, volume 736 of *LNCS*, pages 209–229. Springer-Verlag, 1993.

[AD90] R. Alur and D. Dill. Automata for modeling real-time systems. In *Proc. 17th International Colloquium on Automata, Languages and Programming (ICALP'90)*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer-Verlag, 1990.

[AD94] R. Alur and D.L. Dill. A theory of timed automata. *TCS*, 126:183–235, 1994.

[AFH94] R. Alur, L. Fix, and T. A Henzinger. A determinizable class of timed automata. In *CAV*, LNCS, pages 1–13. Springer-Verlag, 1994.

[AH92] R. Alur and T. A. Henzinger. Logics and models of real time: A survey. In J. W. de Bakker, K. Huizing, W.-P. de Roever, and G. Rozenberg, editors, *Real Time: Theory in Practice*, volume 600 of *LNCS*, pages 74–106. Springer-Verlag, 1992.

[Alu99] R. Alur. Timed automata. In *Proc. 11th International Conference on Computer Aided Verification (CAV'99)*, volume 1633 of *LNCS*, pages 8–22. Springer-Verlag, 1999.

[Arn92] A. Arnold. *Systèmes de transitions finis et sémantique des processus communicants*. Masson, Paris, 1992.

[BAMP83] M. Ben-Ari, Z. Manna, and A. Pnueli. The temporal logic of branching time. *Acta Informatica*, 20:207–226, 1983.

[BB87] T. Bolognesi and E. Brinksma. Introduction to the ISO specification language LOTOS. *Computer Networks and ISDN Systems*, 14:25–59, 1987.

[BB93] F. Barbeau and G. V. Bochmann. A subset of LOTOS with the computational power of Place/Transition Nets. In Marco Ajmone Marsan, editor, *Application and Theory of Petri Nets*, volume 691 of *LNCS*, pages 49–68. Springer-Verlag, 1993.

[BBL+99] B. Bernard, M. Bidoit, F. Laroussine, A. Petit, and Ph. Schnoebelen. *Vi¿œification de logiciels: Techniques et outils du Model-Checking* . Paris, vuibert edition, 1999.

[BCM+92] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.

[BDFP00] P. Bouyer, C. Dufourd, E. Fleury, and A. Petit. Are timed automata updatable? In *Proc of CAV'2000*, volume 1855 of *LNCS*, pages 464–479. Springer-Verlag, 2000.

[BDFP04] P. Bouyer, C. Dufourd, E. Fleury, and A. Petit. Updatable timed automata. *Theoretical Computer Science*, 321(2-3):291–345, 2004.

[BF99] B. Bérard and L. Fribourg. Automatic verification of a parametric real-time program: The ABR conformance protocol. In *Proceedings of CAV'99*, volume 1633 of *LNCS*. Springer-Verlag, 1999.

[BFT01] R. Barbuti, N. De Francesco, and L. Tesei. Timed automata with non-instantaneous actions. *Fundamenta Informaticae*, 46:1–15, 2001.

[BK85] J. A. Bergstra and J. W. Klop. Algebra of communicating processes with abstraction. *TCS*, 37:77–121, 1985.

[Bou02] P. Bouyer. *Modèles et Algorithmes pour la Vérification des Systèmes Temporisés*. PhD thesis, Ecole Normale Supérieure de Cachan, France, April 2002.

[Bry86] R. E. Bryant. Graph-based algorithm for boolean function manipulation. *IEEE Transactions on Computer Sciences*, 37:77–121, 1986.

[BS98] S. Bornot and J. Sifakis. On the composition of hybrid systems. In *Proceedings of HSCC'98*, volume 1386 of *LNCS*, pages 69–83. Springer-Verlag, 1998.

[BS05] N. Belala and D. E. Saïdouni. Non-atomicity in timed models. In *Proceedings of International Arab Conference on Information Technology (ACIT'2005)*. Al-Isra Private University, Jordan, December 6-8th 2005.

[BS06] A. Benamira and D. E. Saïdouni. Consideration of the covering steps in the maximality-based labeled transition systems. In *Proceedings of International Arab Conference on Information Technology (ACIT'2006)*. Al-Yarmouk University, Irbid, Jordan, December 19-21st 2006.

[BST97] S. Bornot, J. Sifakis, and S. Tripakis. Modeling urgency in timed systems. In *Proc. International Symposium Compositionality (COMPOS'97)*, volume 1536 of *LNCS*. Springer-Verlag, 1997.

[Cd94] J. P. Courtiat and R. C. de Oliveira. About time nondeterminism and exception handling in a temporal extension of LOTOS. In Son T. Vuong and Samuel T. Chanson, editors, *PSTV'94*, pages 37–52. Chapman & Hall, 1994.

[CdO95a] J.P. Courtiat and R.C. de Oliveira. On RT-LOTOS and its application to the formal design of multimedia protocols. *Annals of Telecommunications*, 50:11–12, 1995.

[CdO95b] J.P. Courtiat and R.C. de Oliveira. A reachability analysis of RT-LOTOS specifications. In *FORTE*, London, 1995. Chapman and Hall.

[CdOA95] J.P. Courtiat, R.C. de Oliveira, and L. Andriantsiferana. Specification and validation of multimedia protocols using RT-LOTOS. In *Fifth IEEE International Conference on Future Trends of Distributed Computing Systems*, Cheju Island, Korea, 1995.

[CdS93a] J. P. Courtiat, M. S. de Camargo, and D. E. Saïdouni. RT-LOTOS: LOTOS temporisé pour la spécification de systèmes temps réel. In R. Dssouli, G.V. Bochmann, and L. Levesque, editors, *Ingénierie des Protocoles (CFIP'93)*, pages 427–441. Hermes, 1993.

[CdS93b] J. P. Courtiat, M. S. de Camargo, and D. E. Saïdouni. RT-LOTOS: LOTOS temporisé pour la spécification de systm̀es temps rél. Research Report 93040, LAAS-CNRS, 7 av. du Colonel Roche, 31077 Toulouse Cedex France, February 1993.

[CE81] E. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *IBM Workshop on logics of programs*, volume 131 of *LNCS*, pages 52–71. Springer-Verlag, 1981.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, April 1986.

[Cha98] Z. Chaochen. Duration calculus, a logical approach to real-time systems. In *AMAST*, volume 1548 of *LNCS*, pages 1–7. Springer-Verlag, 1998.

[CHR91] Z. Chaochen, C. A. R. Hoare, and A. P. Ravn. A calculus of durations. *Information Processing Letters*, 40(5):269–276, 1991.

[Coe93] R. J. Coelho da Costa. *Systèmes de transitions étiquetés causaux: une nouvelle approche pour la description du comportement événementiel de systèmes concurrents.* PhD thesis, LAAS-CNRS, 7 av. du Colonel Roche, 31077 Toulouse Cdex France, 1993.

[CS95] J. P. Courtiat and D. E. Saïdouni. Relating maximality-based semantics to action refinement in process algebras. In D. Hogrefe and S. Leue, editors, *IFIP TC6/WG6.1, 7th Int. Conf. on Formal Description Techniques (FORTE'94)*, pages 293–308. Chapman & Hall, 1995.

[CSLO00] J.P. Courtiat, C . A. S. Santos, C. Lohr, and B. Outtaj. Experience with RT-LOTOS, a temporal extension of the LOTOS formal description technique. *Computer Communications*, 23:1104–1123, 2000.

[DD89] P. Darondeau and P. Degano. Causal trees. In *ICALP'89*, volume 372 of *LNCS*, pages 234–248. Springer-Verlag, 1989.

[Dev92a] R. Devillers. Maximality preservation and the ST-idea for action refinement. In G. Rozenberg, editor, *Advances in Petri Nets*, volume 609 of *LNCS*, pages 108–151. Springer-Verlag, 1992.

[Dev92b] R. Devillers. Maximality preserving bisimulation. *TCS*, 102:165–183, 1992.

[Dij71] E. W. Dijkstra. Hierarchical ordering of sequential processes. *Acta Informatica*, 1(2):115–138, 1971.

[dV89] R. de Simone and D. Vergamini. Aboard AUTO. Technical report RT111, INRIA, 1989.

[DY96] C. Daws and S. Yovine. Reducing the number of clock variables of timed automata. In *Proc. 17th IEEE Real-Time Systems Symposium (RTSS'96)*, pages 73–81. IEEE Computer Society Press, 1996.

[EL86] E. A. Emerson and C-L. Lei. Efficient model checking in fragments of the propositional mu-calculus. In *1st LICS*, pages 267–278, 1986.

[EM85] H. Ehrig and B. Mahr. *Fondamentals of Algebraic Specification 1.* Springer-Verlag, 1985.

[Eme90] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science : Formal Models and Semantics, Ch. 16*, volume B, pages 995–1072. Elsevier, 1990.

[GRS95] R. Gorrieri, M. Roccetti, and E. Stancampiano. A theory of processes with durational actions. *Theoretical Computer Science*, 140:73–94, 1995.

[HHWT97] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: A model checker for hybrid systems. *Journal of Software Tools for Technology Transfer*, 1(1-2):110–122, October 1997.

[HNSY94] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model-checking for real-time systems. *Information and Computation*, 111:193–244, 1994.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[ISO88] ISO/IEC. *LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour, International Standard 8807* . International Organisation of Standardization - Information Processing Systems - Open Systems Interconnection, Genève, September 1988.

[Kha97] W. Khansa. *Ráseaux de Petri P-temporels : Contribution à l'Etude des Systèmes à Evènements Discrets*. PhD thesis, Université de Savoie, France, March 1997.

[Lam83] L. Lamport. What good is temporal logic ? . *Information processing letters*, 83:657–668, 1983.

[LL97] L. Léonard and G. Leduc. An introduction to ET-LOTOS for the description of time-sensitive systems. *Computer Networks and ISDN Systems*, 29:271–292, 1997.

[Loh02] Ch. Lohr. *Contribution À la Conception de Systèmes Temps Réel S'appuyant sur la Technique de Description Formelle RT-LOTOS*. PhD thesis, LAAS-CNRS, 7 avenue du colonel Roche, 31077 Toulouse Cedex France, 2002.

[LPY97] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *Journal of Software Tools for Technology Transfer (STTT)*, 1(1-2):134–152, 1997.

[McM92] K. L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*. PhD thesis, Carnegie Mellon University, 1992.

[Mer74] P. Merlin. *A Study of the Recoverability of Computer System*. PhD thesis, Dept. Computer Sciences, University of California, Irvine, 1974.

[Mes92] J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96:73–155, 1992.

[Mil80] R. Milner. *Communication and Concurrency*, volume 92 of *LNCS*. Springer-Verlag, 1980.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[MT90] F. Moller and C. Tofts. A temporal calculus of communicating systems. In J. C. Baeten and J. W. Klop, editors, *CONCUR*, volume 458 of *LNCS*, pages 401–415. Springer-Verlag, 1990.

[NSY93] X. Nicollin, J. Sifakis, and S. Yovine. From ATP to timed graphs and hybrid systems. *Acta Informatica*, 30(2):181–202, 1993.

[Pet62] C. Petri. *Kommunikation mit Automaten*. PhD thesis, Institut für Instrumentelle Mathematik, Bonn, Germany, 1962.

[Plo81] G. Plotkin. A structural approach to operational semantics. Report Daimi FN 19, Aarhus University, Denmark, 1981.

[Ram74] C. Ramchandani. *Analysis of Asynchronous Concurrent Systems by Timed Petri Nets*. PhD thesis, MIT, Cambridge, February 1974.

[Saï96] D. E. Saïdouni. *Sémantique de maximalité: Application au raffinement d'actions en LOTOS*. PhD thesis, LAAS-CNRS, 7 av. du Colonel Roche, 31077 Toulouse Cedex France, 1996.

[Sam03] P. N. M. Sampaio. *Conception Formelle de Documents Multimédia Interactifs: Une Approache S'appuyant sur RT-LOTOS*. PhD thesis, LAAS-CNRS, 7 avenue du colonel Roche, 31077 Toulouse cedex, France, 2003.

[SB03a] D. E. Saïdouni and N. Belala. Vérification de propriétes exprimées en ctl sur le modèle des systèmes de transitions étiquetées maximales. In *CIP'2003, Conférence Internationale de Productique*, Alger, Algérie, octobre 2003. CDTA.

[SB03b] D. E. Saïdouni and N. Belala. Vérification de propriétes exprimées en ctl sur le modèle des systèmes de transitions étiquetées maximales. Research report, Laboratoire LIRE, Université Mentouri, 25000 Constantine, Algérie, 2003.

[SB04] D. E. Saïdouni and N. Belala. Straightforward adaptation of interleaving-based solutions for true concurrency-based logic verification approaches. In *Proceedings of International Conference on Complex Systems (CISC'2004)*. University of Jijel, Algeria, September 6-8th 2004.

[SB05] D. E. Saïdouni and N. Belala. Using maximality-based labeled transition system model for concurrency logic verification. *The International Arab Journal of Information Technology (IAJIT)*, 2(3):199–205, July 2005. ISSN: 1683-3198.

[SB06] D. E. Saïdouni and N. Belala. Actions duration in timed models. In *Proceedings of International Arab Conference on Information Technology (ACIT'2006)*. Al-Yarmouk University, Irbid, Jordan, December 19-21th 2006.

[SBB08] D. E. Saïdouni, N. Belala, and M. Bouneb. Aggregation of transitions in marking graph generation based on maximality semantics for Petri nets. In *Proceedings of the Second International Workshop on Verification and Evaluation of Computer and Communication Systems (VECoS'2008), University of Leeds, UK*. eWiC Series, The British Computer Society (BCS), July, 2-3rd 2008. ISSN: 1477-9358.

[SBB09a] D. E. Saïdouni, N. Belala, and M. Bouneb. Maximality-based structural operational semantics for Petri nets. In *Proceedings of INTELLIGENT SYSTEMS AND AUTOMATION: 2nd Mediterranean Conference on Intelligent Systems and Automation (CISA'09), Zarzis, Tunisia*, volume 1107, pages 269–274. American Institute of Physics, March 23th-25th 2009. ISBN: 978-0-7354-0642-1.

[SBB09b] D. E. Saïdouni, N. Belala, and M. Bouneb. Using maximality-based labelled transition system as a model for Petri nets. *The International Arab Journal of Information Technology (IAJIT)*, 5(6):440–446, November 2009. Print ISSN: 1683-3198.

[SBB09c] D. E. Saïdouni, M. Bouneb, and N. Belala. On-the-fly generation algorithm of alpha-reduced maximality-based labeled transition systems for Petri nets. In *Proceedings of International Arab Conference on Information Technology (ACIT'2009), University of Science and Technology (UST), Sanaa, Yemen*, December 15-17th 2009.

[SBBA08] D. E. Saïdouni, A. Benamira, N. Belala, and F. Arfi. FOCOVE: Formal concurrency verification environment for complex systems. In *Proceedings of INTELLIGENT SYSTEMS AND AUTOMATION: 1st Mediterranean Conference on Intelligent Systems and Automation (CISA'08), Annaba, Algeria*, volume 1019, pages 375–380. American Institute of Physics Conference Proceedings, June 29-30th and July 1st 2008. ISBN: 978-0-7354-0540-0.

[SC03] D. E. Saïdouni and J. P. Courtiat. Prise en compte des durées d'action dans les algèbres de processus par l'utilisation de la sémantique de maximalité. In *CFIP'2003*. Hermes, France, 2003.

[SDAB04] D. E. Saïdouni, L. Derdouri, A. Alidra, and N. Belala. Conception formelle du protocole AMRHy. Technical report, Département d'Informatique, Université Mentouri, 25000 Constantine, Algérie, Juin 2004.

[Tes04] L. Tesei. *Specification and Verification Using Timed Automata*. PhD thesis, Università degli Studi di Pisa, Italy, May 2004.

[vG89] R. J. van Glabbeek and U. Goltz. Partial order semantics for refinement of actions - neither necessary nor sufficient but appropriate when used with care. In *Mathematical Foundations of Computer Science*, volume 38 of *Bulletin of the EATCS*, pages 154–163, 1989.

[Vog91] W. Vogler. Bisimulation and action refinement. In *STACS'91*, volume 480 of *LNCS*, pages 309–321. Springer-Verlag, 1991.

[vV87] R. J. van Glabbeek and F. W. Vaandrager. Petri net models for algebraic theories of concurrency. In *PARLE'87, Vol. II (Parallel Languages)*, volume 259 of *LNCS*, pages 224–242. Springer-Verlag, 1987.

[Win84] G. Winskel. Synchronization trees. *TCS*, 34:33–82, 1984.

[Win87] G. Winskel. Event structures. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *APN'86*, volume 255 of *LNCS*, pages 325–392. Springer-Verlag, 1987.

[Yov93] S. Yovine. *Méthodes et Outils pour la Vérification Symbolique des Systèmes Temporisés*. PhD thesis, Institut National Polytechnique de Grenoble, France, May 1993.

[Yov97] S. Yovine. KRONOS: A verification tool for real-time systems. *Journal of Software Tools for Technology Transfer*, 1(1-2):123–133, October 1997.