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

Année : 2004/2005
N ° d'ordre : 103/Mag/2005
Série : 010/inf/2005

Mémoire de Magistère

En vue de l'obtention du diplôme de Magistère en Informatique
Option : Information and Computation

Présenté par

Nabil BELALA

Thème

Formalisation des systèmes temps-réel avec durées d'actions

Soutenu le 01/06/2005 devant le jury composé de :

Dr Souham MESHOUL	**Président**	**M. C**	**U. M. Constantine**
Dr Djamel-Eddine SAIDOUNI	**Rapporteur**	**M. C**	**U. M. Constantine**
Dr Mohamed BENMOHAMMED	**Examinateur**	**M. C**	**U. M. Constantine**
Dr Faïza BELALA	**Examinateur**	**M. C**	**U. M. Constantine**
M^{me} Nadia ZEGHIB	**Invité**	**C. C**	**U. M. Constantine**

Mémoire préparé au Laboratoire LIRE, Université Mentouri, 25000 Constantine, Algérie

Formalisation des systèmes temps-réel avec durées d'actions

Nabil Belala

Equipe Vision et Infographie
Laboratoire LIRE, Université de Mentouri, 25000 Constantine

A mon défunt père...

Remerciements

Je remercie en tout premier lieu les membres de mon jury : M^{me} Souham Meshoul, qui a accepté la lourde tâche d'être président du jury ; M. Djamel-Eddine Saïdouni, qui a accepté d'être rapporteur de mon mémoire ; et M. Mohamed Benmohammed, M^{me} Faïza Belala et M^{me} Nadia Zeghib, pour le temps qu'ils ont accordé à l'examen de ce travail en dépit de leurs nombreuses activités.

Je remercie tout particulièrement M. 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, sa patience et sa disponibilité ont beaucoup facilité et agrémenté mon travail.

Bien entendu, je n'oublie pas M. Mohamed Chaouki Batouche, qui m'a chaleureusement accueilli dans son équipe Vision et Infographie.

Merci aussi à ma mère, mon grand-père, mes sœurs et toute ma famille pour s'être 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

Les méthodes formelles sont de plus en plus utilisées pour répondre aux exigences auxquelles sont soumis les systèmes temps-réel. Ces méthodes reposent sur l'utilisation de modèles formels de spécification dotés de sémantiques rigoureuses et de techniques de vérification formelle. Parmi ces dernières, nous nous intéressons à l'approche de vérification logique exploitant le graphe de comportement afin de vérifier les propriétés qualitatives et/ou quantitatives requises du système. D'autre part, les sémantiques de vrai parallélisme, comme la sémantique de maximalité, conviennent à être employées lorsqu'on s'abstrait de l'hypothèse de l'atomicité temporelle et structurelle des actions ; des travaux antérieurs ont largement montré l'aptitude du modèle des systèmes de transitions étiquetées maximales (STEM) à prendre en considération et le comportement parallèle des systèmes et la prise en compte du temps. Parmi ces travaux, nous citons le langage D-LOTOS dont le but est la spécification des systèmes temps-réel.

Notre travail qui s'inscrit dans le cadre de la spécification et la vérification formelle des systèmes temps-réel consiste à définir un modèle sémantique temps-réel basé sur la sémantique de maximalité, exprimant les comportements parallèles et supportant à la fois contraintes temporelles, durées explicites des actions, non-atomicité structurelle et temporelle des actions et notion d'urgence, tout en proposant les règles de génération de manière opérationnelle de ce nouveau modèle (le modèle des DATA*'s) à partir de spécifications D-LOTOS. Enfin, nous montrons comment se servir de ce modèle dans une technique de vérification formelle, le model checking.

Mots-clés Systèmes temps-réel, Durées d'actions, Spécification formelle, Vérification formelle, Langage D-LOTOS, Modèle des DATA*'s.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

Les systèmes temps-réel, ou à temps contraint, sont de plus en plus utilisés dans la vie quotidienne. De plus, ils font souvent partie des applications critiques dans le domaine de l'aviation, le contrôle de processus industriels, le contrôle de centrales nucléaires, etc. Les systèmes temps-réel sont en général caractérisés par des interactions complexes avec l'environnement et par des contraintes temporelles strictes dont la violation peut entraîner des conséquences graves. Par conséquent, une propriété importante de ces systèmes est leur fiabilité.

Les méthodes formelles sont de plus en plus utilisées pour répondre aux exigences auxquelles sont soumis ce type de systèmes. Ces méthodes reposent sur l'utilisation de modèles *formels* de spécification dotés de sémantiques rigoureuses et de techniques de *vérification* formelle.

Précisément, la vérification formelle de systèmes temps-réel nécessite généralement la réunion de quatre éléments :

- Un langage de haut niveau de description du système, avec des opérateurs pour exprimer les contraintes temporelles. Nous pouvons citer les extensions temporelles et temporisées des réseaux de Petri et les extensions temporelles des algèbres de processus comme TCCS, RT-LOTOS, D-LOTOS, etc.

- Un modèle sémantique de bas niveau exprimant le comportement du système. Des règles de génération à partir du langage de spécification vers ce modèle sont définies en obéissant à une sémantique formelle qui associe une signification précise à la notion de comportement.

  Les modèles sémantiques sont divisés en deux catégories selon leur manière d'exprimer le parallélisme :

  - Les modèles d'entrelacement qui considèrent l'hypothèse de l'atomicité temporelle et structurelle des actions (les actions sont de durée nulle et indivisibles). Cette hypothèse peut s'avérer contraignante vis-à-vis de la réalité du système que l'on veut modéliser.

  - Les modèles non entrelacés ou de vrai parallélisme qui abandonnent les hypothèses d'atomicité et d'entrelacement et se définissent par rapport à une notion de causalité ou de simultanéité entre les exécutions d'actions. Bien que la description de ces modèles est plus abstraite, ils permettent d'éviter les difficultés liées aux modèles entrelacés, et de conserver la nature parallèle des programmes concurrents.

- Un langage de spécification pour décrire les aspects qualitatifs et quantitatifs des propriétés temps-réel du système. Ces formalismes de spécification se caractérisent par la volonté d'exprimer les propriétés des systèmes tout en restant simple et expressif. Les logiques temporelles sont bien adaptées aux comportements infinis et aux notions de sûreté et de vivacité, essentielles pour les systèmes étudiés. Elles allient une grande puissance d'expression à l'utilisation d'opérateurs dont le sens reste intuitif.

- Une relation de satisfaction qui définit formellement la comparaison entre le système à vérifier et sa spécification, munie d'un algorithme de vérification pour la mettre en œuvre. Une des approches de vérification formelle des systèmes concurrents qui se sert de la logique temporelle est l'approche automatique du *model checking*. Récemment, l'intérêt pour la vérification automatique se déplace vers les systèmes concurrents temps-réel. Les propriétés à vérifier sont exprimées en une logique temporelle comme CTL [CES86, Eme90], ou dans sa version temporisée TCTL [ACD93].

Problématique

L'application pratique du model checking est fortement limitée par le problème de l'explosion combinatoire de l'espace d'états des systèmes à vérifier, principalement provoqué par l'expression entrelacée d'actions concurrentes. Dès lors, un des problèmes majeurs de la sémantique des langages de spécification est de pouvoir donner au parallélisme un sens précis qui soit en accord avec les hypothèses réelles portant sur les interactions entre les composantes du système que l'on souhaite décrire. A titre d'exemple, la sémantique classique d'entrelacement ne distingue pas entre exécutions séquentielles et exécutions parallèles d'actions. Cette interprétation de la concurrence souffre également de plusieurs inconvénients majeurs dès que l'on veut en particulier s'abstraire de l'hypothèse d'atomicité temporelle et structurelle des actions. Pour résoudre ces problèmes, différentes sémantiques alternatives dites de vrai parallélisme ont été étudiées, comme par exemple, les sémantiques de causalité [Coe93] et de maximalité [Saï96]. Pour remédier à l'atomicité dans un contexte temps-réel, on a tenté de spécifier des actions avec *durées explicites*. Cependant, on trouve des modèles non temporisés, comme les systèmes de transitions étiquetées maximales [Saï96, SB03a], où on avait la possiblité d'exprimer l'exécution potentielle d'une action qui dure dans le temps. Des règles de génération de manière opérationnelle à partir de spécifications du langage Basic LOTOS ont été définies dans le contexte de la sémantique de maximalité. Le pas vers la considération des durées d'actions de manière explicite a été franchi naturellement, et a fait naître le langage de spécification des systèmes temps-réel D-LOTOS [SC03]. L'un des objectifs de ce travail

consiste en la considération explicite des durées des actions dans un modèle sémantique de bas niveau basé sur la sémantique de maximalité, tout en définissant des règles de passage de l'un de ces langages (comme D-LOTOS) vers ce nouveau modèle.

Pour modéliser les systèmes temps-réel, plusieurs modèles sémantiques d'états-transitions, dotés généralement d'un mécanisme décrivant les contraintes temporelles, ont été étudiés dans la littérature. L'un des modèles les plus étudiés pour les systèmes temps-réel est le modèle des *automates temporisés* [AD90, AD94], qui étend les automates classiques en ajoutant des variables mesurant le temps, appelées *horloges*. Les transitions sont gardées par les *contraintes d'horloge*, qui comparent la valeur d'une horloge à une certaine constante, et exécutent les *mises à jour* des horloges, qui initialisent une horloge à la valeur initiale 0. Malgré que ce modèle a été largement utilisé dans la vérification, de nombreuses propriétés impliquant des informations sur les actions s'exécutant en parallèle, restent non vérifiables, dû toujours à la considération de la sémantique d'entrelacement.

D'une part, deux approches sont utilisées pour échapper à l'hypothèse d'atomicité des actions. Dans la première approche, une action est modélisée par son début suivi de sa fin, ces événements sont toujours atomiques. La durée d'une action est introduite via un laps de temps séparant les événements début et fin relatifs à cette action. Quoique cette approche permet la modélisation de comportements concurrents, elle souffre de l'explosion combinatoire aggravée par l'éclatement de chaque action. Une deuxième approche consiste à faire correspondre à toute action de durée non nulle une transition non instantanée. Tel qu'il sera discuté ultérieurement, cette approche n'est pas suffisante à la modélisation de comportements concurrents.

D'autre part, des notions comme l'urgence des actions ne sont pas supportées par le modèle de base des automates temporisés. Notre approche repose sur l'association à chaque action urgente d'une contrainte d'urgence qui, une fois satisfaite, oblige l'action à être exécutée tout en empêchant le temps à progresser.

Notre travail consiste à définir un modèle sémantique temps-réel basé sur la sémantique de maximalité, exprimant les comportements parallèles et supportant à la fois contraintes temporelles, durées explicites des actions, non-atomicité structurelle et temporelle des actions et notion d'urgence, tout en proposant les règles de génération de manière opérationnelle de ce nouveau modèle (qui sera appelé modèle des DATA*'s) à partir de spécifications D-LOTOS. Enfin, nous montrons comment se servir de ce modèle dans une technique de vérification formelle, le model checking.

Contributions

Nos contributions peuvent se résumer en :

- L'étude de différents modèles temporels et temporisés de bas niveau, basés notamment sur le modèle des automates temporisés, pour la spécification formelle des systèmes temps-réel. Ces modèles vont être confrontés à nos besoins de pouvoir spécifier et vérifier

des comportements parallèles temps-réel et supportant la non-atomicité structurelle et temporelle des actions.

- La proposition d'un modèle dénotationnel de spécification des systèmes temps-réel permettant l'expression de comportements parallèles et supportant à la fois contraintes temporelles, durées explicites des actions, non-atomicité structurelle et temporelle des actions et notion d'urgence.
- L'utilisation de ce modèle pour la vérification de comportements qualitatifs de systèmes temps-réel.

Plan du document

Le mémoire est organisé de la manière suivante :

- Le Chapitre 2 introduit quelques modèles utilisés dans le contexte temps-réel, à savoir les extensions de réseaux de PETRI, les automates temporises, et les algèbres de processus. Nous détaillerons le langage RT-LOTOS, l'une des extensions temporisées de l'algèbre de processus LOTOS, ainsi que le modèle des automates temporisés.
- Le Chapitre 3 considère la notion de durées d'actions dans l'algèbre de processus Basic LOTOS, et introduit un modèle sémantique appelé DATA (pour *Durational Action Timed Automata*). La génération opérationnelle de DATA's relatifs à des expressions Basic LOTOS avec durées d'actions étant donnée.
- Le Chapitre 4 considère le langage temps-réel D-LOTOS. La particularité de ce langage étant la considération conjointe d'opérateurs exprimant les contraintes temporelles et l'attribution de durées aux actions. La sémantique de ce langage est exprimée à travers l'extension du modèle des DATA's pour la prise en compte des contraintes temporelles ; le modèle ainsi défini est nommé DATA*. De manière similaire, la génération de DATA*'s relatifs à des expressions de comportement D-LOTOS est définie.
- Le Chapitre 5 introduit la notion de model checking. La logique adoptée étant la logique temporelle CTL, et le modèle est celui des systèmes de transitions étiquetées maximales. L'accent est mis sur l'apport de la sémantique de maximalité pour la vérification de propriétés qu'on ne pouvait pas vérifier dans une approche d'entrelacement. Un algorithme de model checking est proposé. La vérification de propriétés qualitatives des applications temps-réel est possible par l'abstraction du temps (après génération du modèle des DATA*'s) et l'application de l'algorithme proposé.
- Le Chapitre 6 donne un cas d'étude, le brûleur à gaz.
- Le Chapitre 7 donne finalement quelques conclusions et perspectives des travaux présentés dans ce mémoire.

Chapitre 2

Formalismes de spécification des systèmes temps-réel

Afin de modéliser les systèmes temps-réel, plusieurs modèles ont été étudiés dans la littérature. On trouve les modèles de bas niveau, comme les modèles d'états-transitions, dotés d'un mécanisme décrivant les contraintes temporelles, et les modèles de haut niveau permettant de modéliser de manière concise les systèmes ayant de comportements assez complexes. Ce chapitre introduit quelques modèles parmi les plus connus.[1]

2.1 Modèles de haut niveau

2.1.1 Extensions temporisées des réseaux de Petri

Les réseaux de Petri, du fait de leur sémantique du parallélisme et de leur représentation graphique, apparaissent comme particulièrement attractifs pour la modélisation des comportements des systèmes. Le modèle initial des réseaux de Petri est atemporel. Plusieurs extensions ont été proposées pour prendre en compte le temps. On distingue principalement quatre grandes «familles» de réseaux :

1. Les *réseaux de* Petri *temporisés* (*Timed* Petri *Nets* [Ram74]), qui associent une *durée* aux transitions ou aux places.

2. Les *réseaux de* Petri *temporels* (*Time* Petri *Nets* [Mer74]), qui préfèrent la notion de *délai* (entre événements) à celle de durée (d'un état ou d'une action).

3. Les systèmes à gardes algébriques, comme les *réseaux RT* [Bou99], qui utilisent des contraintes temporelles de la forme $(3 \leq x) \vee (x \leq 6)$ (où x représente une valeur d'horloge) plutôt que de simples durées ou délais.

[1] Certaines parties de ce chapitre s'intéressant aux algèbres de processus temps-réel sont extraites de [SBA04, Loh02].

4. Les systèmes à synchronisation, comme les *réseaux de* PETRI *à flux temporels* [DS93], qui sont des modèles conçus pour modéliser un système comme la composition de sous-éléments en les «synchronisant».

En plus de la représentation graphique des réseaux de PETRI qui donne une description claire du système spécifié, ils permettent de représenter les aspects de conflit (choix), de séquencement et de parallélisme entre comportements. Or, deux points cruciaux peuvent limiter leur utilisation dans la pratique, et ainsi leurs extensions :

Pouvoir d'expression Dans [BB93], il a été montré que le modèle des réseaux de PETRI de base a exactement le pouvoir d'expression qu'un sous-ensemble du langage LOTOS.

Vérification Dans la vérification comportementale (Voir Section 5.1), plusieurs relations d'équivalence, telles que les bissimulations, nécessitent la traduction des réseaux de PETRI à analyser dans d'autres structures du genre système de transitions. Dans un contexte de vérification logique, le model checking de propriétés de branchement sur les réseaux de PETRI temporels nécessitent l'utilisation de constructions plus élaborées comme les *régions géométriques* [YR98].

2.1.2 Algèbres de processus

Les algèbres de processus sont des langages abstraits conçus pour la spécification, la conception, et l'analyse fonctionnelle de systèmes concurrents. De nombreux modèles ont été proposés. Les modèles émergents sont LOTOS (*Language of Temporal Ordering Specification* [BB87, ISO88]), CCS (*Calculus of Communication Systems* de R. MILNER [Mil89]), CSP (*Communicating Sequential Process* de C.A.R. HOARE [Hoa85]), ACP (*Algebra of Communicating Processes* de J.A. BERGSTRA [BK85]), auxquels peuvent être ajoutés des modèles qui introduisent le temps, comme TCCS (*Temporal* CCS [MT90]) ainsi que les langages du type SPA (*Stochastic Process Algebras* de U. HERZOG [HR98]).

Dans les algèbres de processus, les systèmes sont modélisés par un ensemble de processus, des entités appelées *agents*, qui exécutent des actions. Ces actions sont les briques de base du langage, et des opérateurs sont utilisés pour décrire des comportements séquentiels qui peuvent s'exécuter de manière concurrente et se synchroniser (communiquer) entre eux.

Les algèbres de processus apparaissent comme des modèles de description formelle et de spécification «orientés contraintes», qui semblent bien adaptés à l'intuition et à la compréhension humaine. Ajoutée à cela la faculté d'automatiser le processus de vérification, ces modèles connaissent un certain succès.

Présentation informelle de 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 2.1.

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 :

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

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 ($Act = \mathcal{G} \cup \{i, \delta\}$).

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)

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	$P_1 \|[g_1, ...g_n]\| P_2$	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

TAB. 2.1 – *Principaux opérateurs de Basic LOTOS*

- 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 $\mathbb{Q}^+$. 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 $\mathbb{R}^+$. 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 («*opérateur* $< \ldots >$ P»), soit par une extension de l'opérateur de préfixe (préfixe d'un processus par une action : «$g < \ldots >; P$»), 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 2.2 reprend quelques points de comparaison entre les différentes extensions temporelles à LOTOS.

	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

TAB. 2.2 – *Comparaison des extensions temporelles à LOTOS*

2.1.3 L'extension RT-LOTOS

Le langage RT-LOTOS [CdS93, CSLO00] est l'ancêtre de D-LOTOS [SC03], c'est une extension temporelle de LOTOS qui a été définie en considérant une sémantique d'entrelacement. Nous en évoquerons quelques aspects dans cette section, dans la perspective d'expliquer certains opérateurs temporels qui ont été repris dans D-LOTOS avant d'introduire celui-ci. Le lecteur intéressé par de plus amples détails pourra se référer à [Loh02].

L'introduction de RT-LOTOS a été faite dans le but de permettre la description de l'aspect quantitatif (et non pas seulement de l'aspect qualitatif de l'ordonnancement des événements) des instants auxquels les événements se produisent réellement. L'extension temporelle RT-LOTOS (*Real-Time* LOTOS) reprend les concepts et l'essentiel des opérateurs de LOTOS, et apporte de nouveaux opérateurs permettant d'exprimer des contraintes temporelles quantifiées.

L'occurrence des actions peut être contrainte de la manière suivante :

- en retardant l'occurrence d'un processus de manière déterministe ;
- en retardant l'occurrence d'un processus de manière non-déterministe ;
- en limitant le temps pendant lequel une action est offerte à son environnement.

RT-LOTOS propose essentiellement trois opérateurs pour décrire, de manière intuitive, l'expression du temps dans le comportement de processus LOTOS.

- L'opérateur de délai (noté `delay`(d) ou Δ^d) permet de retarder un processus d'une certaine quantité de temps d.

- L'opérateur de latence (noté `latency`(l) ou Ω^l) permet de retarder un processus d'une certaine quantité de temps choisie de manière non-déterministe au sein de l'intervalle de latence $[0, l]$. Notons qu'en fait, cet opérateur porte sur la ou les premières actions du processus auquel il est appliqué. Par ailleurs, il n'a d'effet que s'il porte sur une action interne, l'occurrence d'une action observable étant, en tout état de cause, soumise à la date de l'offre faite par l'environnement. En d'autres termes, l'opérateur de latence permet de relâcher la contrainte d'urgence d'une action interne. Il permet d'introduire de manière générale le non-déterminisme temporel.

- L'opérateur de restriction temporelle (noté $g\{t\}$) limite le temps pendant lequel une action observable g peut être offerte à son environnement. Le délai d'expiration commence à courir à partir du moment où l'action est offerte.

Présentation informelle des opérateurs de RT-LOTOS

Les opérateurs temporels introduits par RT-LOTOS sont présentés de manière intuitive au moyen des graphiques décrits dans la Figure 2.1[2], où les axes temporels sont supposés avoir pour origine l'instant où le processus associé devient sensibilisé. Ces schémas illustrent les moyens offerts par les opérateurs temporels pour décrire les dates d'offre et d'occurrence des actions internes (cachées à l'environnement du processus par l'opérateur `hide`) et observables (offertes à l'environnement et dépendant donc de lui).

Par convention, nous avons représenté sur ces schémas les différents intervalles temporels de la manière suivante :

- une boîte rectangulaire représente l'intervalle de délai, c'est-à-dire l'intervalle de temps pendant lequel il n'y a ni offre ni occurrence d'action ;

- une zone grisée représente l'intervalle de temps dans lequel a lieu l'occurrence de l'action ;

- un ressort représente l'intervalle de latence, c'est-à-dire l'intervalle de temps dans lequel est choisi une date pour l'offre de l'action ; cette zone est également grisée (gris foncé) car l'occurrence de l'action peut éventuellement y avoir lieu ;

- un trait vertical épais représente l'instant à partir duquel l'action devient urgente (cas des schémas à gauche) ;

- un triangle blanc marque un instant possible pour l'offre de l'action ;

- un triangle noir marque un instant possible pour l'occurrence de l'action ; si le triangle noir est seul sur la ligne temporelle, alors l'offre et l'occurrence ont lieu au même instant.

[2] Cette figure est tirée de [CSLO00].

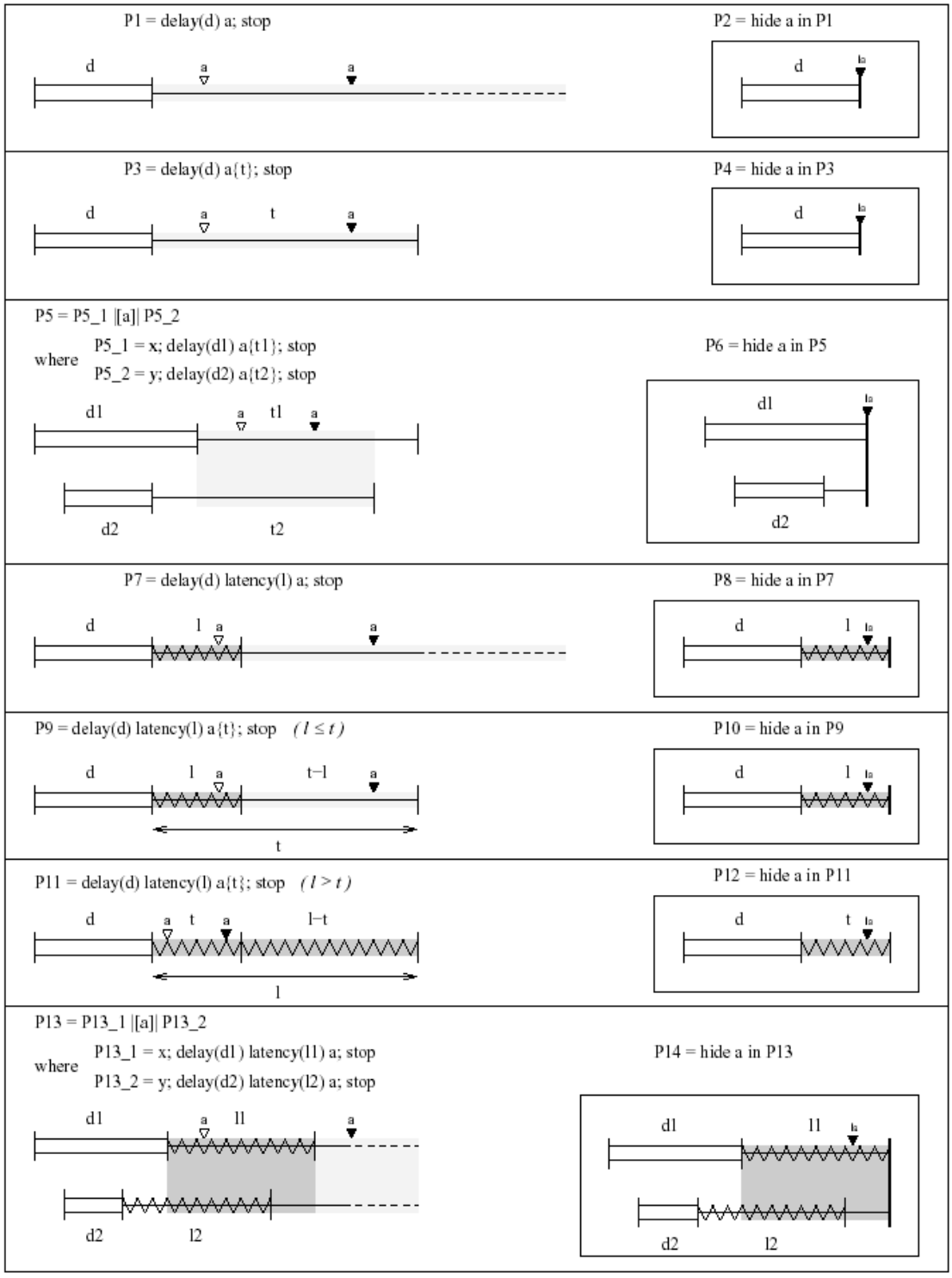

FIG. 2.1 – *Illustration des opérateurs RT-LOTOS*

Syntaxe formelle et sémantique opérationnelle de Basic RT-LOTOS

La syntaxe de Basic RT-LOTOS est donné par la Figure 2.2.

$$\begin{aligned} P ::= & \texttt{stop} \mid \texttt{exit} \mid X[L] \mid i;P \mid g;P \\ & \mid P \ [] \ P \mid P \ |[L]| \ P \mid \texttt{hide}\ L\ \texttt{in}\ P \\ & \mid P \gg P \mid P \ [> P \\ & \mid \Delta^u P \mid \Omega^u P \\ & \mid i\{u\};P \mid g\{u\};P \end{aligned}$$

FIG. 2.2 – *Syntaxe de Basic RT-LOTOS*

Les notations suivantes seront utilisées lors de l'expression des différentes règles d'inférence de la sémantique opérationnelle :

- $P \xrightarrow{a}$ signifie que $\exists P'$ tel que $P \xrightarrow{a} P'$.
- $P \xnrightarrow{a}$ signifie que le processus P ne peut réaliser l'action a.
- $P \xrightarrow{t} P'$ signifie que le processus P ne peut exécuter aucune action pendant une période de t unités de temps et se comporte ensuite comme le processus P'.

Afin de traiter les actions urgentes et non urgentes dans le modèle sémantique, deux actions sémantiques, notées g_w et g_s, sont associées à chaque action $g \in \mathcal{G}$ du modèle syntaxique :

- g_s, appelée action g forte (ou g *strong*),
- g_w, appelée action g faible (ou g *weak*).

Ces actions sémantiques présentent les caractéristiques d'urgence suivantes :

1. g_s et g_w ($g \in \mathcal{G}$) ne sont pas urgentes par définition dès lors que g est observable
2. i_s est urgente, de même que g_s ($g \in \mathcal{G}$) lorsqu'elle est cachée, ainsi que δ_s lorsqu'elle apparaît à droite de l'opérateur $\gg$
3. i_w n'est pas urgente, de même que g_w ($g \in \mathcal{G}$) lorsqu'elle est cachée, ainsi que δ_w lorsqu'elle apparaît à droite de l'opérateur $\gg$

L'*urgence* en RT-LOTOS est définie formellement par l'assertion suivante :

$$P \xrightarrow{i_s} \ \Rightarrow \forall t \neq 0,\ P \xnrightarrow{t}$$

La sémantique opérationnelle d'entrelacement de Basic RT-LOTOS est définie dans le Tableau 2.3.

(1.a)	$exit \xrightarrow{\delta_s} stop$	(1.b)	$stop \xrightarrow{t} stop$
		(1.c)	$exit \xrightarrow{t} exit$
(2.a)	$g\{u\};P \xrightarrow{g_s} P \ (g \in \mathcal{G},\ u > 0)$	(2.b)	$g\{u+t\};P \xrightarrow{t} g\{u\};P \ (g \in \mathcal{G})$
		(2.c)	$g\{0\};P \xrightarrow{t} stop \ (g \in \mathcal{G})$
(3.a)	$i\{v\};P \xrightarrow{i_w} P \ (v > 0)$	(3.c)	$i\{u+t\};P \xrightarrow{t} i\{u\};P$
(3.b)	$i\{0\};P \xrightarrow{i_s} P$		
(4.a)	$\frac{P \xrightarrow{g} P'}{P[]Q \xrightarrow{g} P'} \ (g \in \mathcal{G}^{i,\delta})$	(4.b)	$\frac{P \xrightarrow{t} P' \quad Q \xrightarrow{t} Q'}{P[]Q \xrightarrow{t} P'[]Q'}$
(5.a)	$\frac{P \xrightarrow{g_x} P' \quad Q \xrightarrow{g_y} Q'}{P\|[L]\|Q \xrightarrow{g_{x\wedge y}} P'\|[L]\|Q'} \ (g \in L \cup \{\delta\})$	(5.c)	$\frac{P \xrightarrow{t} P' \quad Q \xrightarrow{t} Q'}{P\|[L]\|Q \xrightarrow{t} P'\|[L]\|Q'}$
(5.b)	$\frac{P \xrightarrow{g} P'}{P\|[L]\|Q \xrightarrow{g} P'\|[L]\|Q} \ (g \in \mathcal{G}^{i,\delta} \setminus L)$		
(6.a)	$\frac{P \xrightarrow{g} P' \quad (g \in \mathcal{G}^{i,\delta} \setminus L)}{hide\ L\ in\ P \xrightarrow{g} hide\ L\ in\ P'}$	(6.c)	$\frac{P \xrightarrow{t} P' \quad P \xrightarrow{g_s} (\forall g \in L)}{hide\ L\ in\ P \xrightarrow{t} hide\ L\ in\ P'}$
(6.b)	$\frac{P \xrightarrow{g_x} P' \quad (g \in L)}{hide\ L\ in\ P \xrightarrow{i_x} hide\ L\ in\ P'}$		
(7.a)	$\frac{P \xrightarrow{g} P'}{P \gg Q \xrightarrow{g} P' \gg Q} \ (g \in \mathcal{G}^{i})$	(7.c)	$\frac{P \xrightarrow{t} P' \quad P \xrightarrow{\delta_s} \not\rightarrow}{P \gg Q \xrightarrow{t} P' \gg Q}$
(7.b)	$\frac{P \xrightarrow{\delta_x} P'}{P \gg Q \xrightarrow{i_x} Q}$		
(8.a)	$\frac{P \xrightarrow{g} P'}{P[>Q \xrightarrow{g} P'[>Q} \ (g \in \mathcal{G}^{i})$	(8.d)	$\frac{P \xrightarrow{t} P' \quad Q \xrightarrow{t} Q'}{P[>Q \xrightarrow{t} P'[>Q'}$
(8.b)	$\frac{Q \xrightarrow{g} Q'}{P[>Q \xrightarrow{g} Q'} \ (g \in \mathcal{G}^{i,\delta})$		
(8.c)	$\frac{P \xrightarrow{\delta} P'}{P[>Q \xrightarrow{\delta} P'}$		
(9.a)	$\frac{P_X[g_1/g'_1 \cdots g_n/g'_n] \xrightarrow{h} Q' \quad X[g'_1 \cdots g'_n] := P_X}{X[g_1 \cdots g_n] \xrightarrow{h} Q'} \ (h \in \{g_s, g_w \mid g \in \mathcal{G}^{i,\delta}\} \cup D)$		
(9.b)	$\frac{P \xrightarrow{g} P'}{P\phi \xrightarrow{\phi(g)} P'\phi} \ (\phi = [g_1/g'_1 \cdots g_n/g'_n] \text{ et } g \in G^{i,\delta})$	(9.c)	$\frac{P \xrightarrow{t} P'}{P\phi \xrightarrow{t} P'\phi} \ (\phi = [g_1/g'_1 \cdots g_n/g'_n])$
(10.a)	$\frac{P \xrightarrow{g} P'}{\Delta^0 P \xrightarrow{g} P'} \ (g \in \mathcal{G}^{i,\delta})$	(10.b)	$\Delta^{u+t} P \xrightarrow{t} \Delta^u P$
		(10.c)	$\frac{P \xrightarrow{t} P'}{\Delta^0 P \xrightarrow{t} P'}$
(11.a)	$\frac{P \xrightarrow{g} P'}{\Omega^u P \xrightarrow{g_w} P'} \ (g \in \mathcal{G}^{\delta})$	(11.d)	$\frac{P \xrightarrow{t} P'}{\Omega^{u+t} P \xrightarrow{t} \Omega^u P'}$
(11.b)	$\frac{P \xrightarrow{i} P'}{\Omega^u P \xrightarrow{i} P'}$	(11.e)	$\frac{P \xrightarrow{t} P'}{\Omega^0 P \xrightarrow{t} P'}$
(11.c)	$\frac{P \xrightarrow{g} P'}{\Omega^0 P \xrightarrow{g} P'} \ (g \in \mathcal{G}^{i,\delta})$		

TAB. 2.3 – *Sémantique opérationnelle d'entrelacement de Basic RT-LOTOS*

2.2 Modèles de bas niveau

Cette section rappelle l'intuition et la définition formelle du modèle des automates temporisés (TA's). Puis, nous exposons de manière succincte les différents travaux sur ce modèle.

2.2.1 Automates temporisés

Tout comportement d'un système réel peut être représenté par un système de transitions, c'est-à-dire un ensemble de nœuds et de transitions entre ces nœuds. Les transitions sont décorées par des étiquettes représentant les actions exécutées, tandis que les nœuds représentent les états du système. Les *systèmes de transitions étiquetées* (*STEs*) sont des modèles très intuitifs 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 STEs, 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* (*TA's*) a été introduit par Rajeev Alur et David Dill dans [AD90, AD94]. Sa définition fournit un simple moyen pour doter 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.

Définitions préliminaires

Définition 2.1 *Une séquence de temps est définie comme étant un ensemble de points aléatoires appartenant à l'ensemble* $\mathbb{R}^+$: $t = t_1 t_2 t_3 ...$ *avec la satisfiabilité des deux conditions suivantes :*

- *La monotonie :* $t_i < t_i + 1$ *pour* $i \geq 1$*, cette propriété nous assure que l'instant qui va venir dans le futur est tout à fait supérieur à celui de présent. C'est pour assurer que le temps évolue toujours.*
- *La progression : pour chaque instant* t *qui appartient à* $\mathbb{R}^+$*, il existe un instant* t' *appartenant au même ensemble et qui vérifie la relation* $t' > t$*.*

Définition 2.2 *Un mot temporisé sur un alphabet* Σ *est un couple* (a, t) *tel que* $a = a_1 a_2 ...$ *est un mot infini sur* Σ *et* t *est une séquence de temps. Un langage temporisé sur* Σ *est un ensemble de mots temporisés engendrés par* Σ*.*

Remarque 2.1 *Notons que l'occurrence des symboles a_i arrivent et terminent à l'instant t_i, entre autres, ils obéissent à l'hypothèse d'atomicité temporelle.*

Définition 2.3 *Soit H un ensemble d'horloges. L'ensemble des gardes sur X est l'ensemble $\Phi(H)$ engendré par la grammaire suivante :*

$$\delta ::= \texttt{true} \mid x \sim c \mid \delta_1 \wedge \delta_2 \mid \delta_1 \vee \delta_2 \mid \neg\delta_1$$

où δ_1 et δ_2 sont des gardes, $x \in H$, $c \in \mathbb{R}^+$ et $\sim \in \{=, <, >, \leq, \geq\}$. Une garde δ est dite vérifiée pour les valeurs des horloges $(a_i)_{1 \leq i \leq n}$ si lorsque x_i est remplacé par a_i dans δ la proposition devient vraie.[3]

Définition 2.4 *Une valuation des horloges de H est une fonction $v : H \to \mathbb{R}^+$. Pour $D \subseteq H$, $[D \to 0]\, v$ désigne la valuation qui à $x \in D$ associe 0 et à $x \notin D$ associe $v(x)$. Pour v valuation d'horloges, la relation de satisfaction suivante est définie, δ étant une garde :*

$$v \models \delta \iff \delta\left(x_i / v(x_i)\right) \text{ est vraie}$$

où $\delta\left(x_i / v(x_i)\right)$ désigne la valeur booléenne de δ dans lequel x_i est remplacé par sa valeur $v(x_i)$. L'ensemble de toutes les valuations d'horloges sur H est noté $\Xi(H)$. Si $d \in \mathbb{R}^+$, $v + d$ désigne la valuation qui associe $v(x) + d$ à chaque $x \in H$.

Formalisation du modèle des automates temporisés

Définition 2.5 *Un automate temporisé $\mathcal{A}$ est un quintuplet (Σ, S, S_0, H, E) tel que :*

- *Σ est un ensemble d'alphabet,*
- *S est un ensemble fini d'états,*
- *$S_0 \subseteq S$ est l'ensemble d'états initiaux,*
- *H est un ensemble fini d'horloges, et*
- *$E \subseteq S \times S \times \Sigma \times 2^H_{fn} \times \Phi(H)$ est l'ensemble des transitions. Un arc $\langle s, s', a, \lambda, \delta \rangle$ représente une transition de l'état s à l'état s' en lisant le symbole a. L'ensemble $\lambda \subseteq H$ contient les horloges à réinitialiser à zéro par cette transition, et δ est une contrainte temporelle sur H. $\langle s, s', a, \lambda, \delta \rangle$ peut être écrit $s \xrightarrow{a,\lambda,\delta} s'$.*

Remarque 2.2 *Quelques définitions ajoutent un ensemble d'états finals, et un ensemble d'états répétés comme dans le cas des automates de* Büchi [Büc62] *d'où la nomination :* automates temporisés de Büchi *(TBA's)* [AD94]. *D'autres définitions joignent à chaque état un invariant devant être satisfait pour que le système puisse demeurer dans l'état correspondant, comme par exemple dans une autre définition des automates temporisés dans* [Alu99],

[3] Nous utilisons indifféremment les expressions *contraintes d'horloges*, *gardes* et *contraintes temporelles*.

les graphes temporisés [Yov93][4]*, ou encore dans les* Timed Safety Automata [HNSY94]*. Dans* [AH92]*, les états des automates temporisés sont étiquetés par des* contraintes propositionnelles *sur un ensemble de propositions.*

Un *chemin* dans l'automate temporisé $\mathcal{A}$ est une suite finie ou infinie de transitions consécutives

$$s_0 \xrightarrow{a_1,\lambda_1,\delta_1} s_1 \xrightarrow{a_2,\lambda_2,\delta_2} \cdots \xrightarrow{a_n,\lambda_n,\delta_n} s_n \cdots$$

Si $u = (a_1, t_1) \cdots (a_n, t_n) \cdots$ est un mot temporisé de $(\Sigma \times \mathbb{R}^+)^\infty$, une *exécution* r sur le chemin précédent pour le mot u est :

$$\langle s_0, v_0 \rangle \xrightarrow[t_1]{a_1,\lambda_1,\delta_1} \langle s_1, v_1 \rangle \xrightarrow[t_2]{a_2,\lambda_2,\delta_2} \cdots \xrightarrow[t_n]{a_n,\lambda_n,\delta_n} \langle s_n, v_n \rangle \cdots$$

où $(v_i)_{i\geq 0}$ est une suite de valuations d'horloges telle que pour tout $x \in H$, $v_0(x) = 0$ et pour tout $i \geq 0$,

$$\begin{cases} v_{i+1} = [\lambda_{i+1} \rightarrow 0](v_i + t_{i+1} - t_i) \\ v_i + t_{i+1} - t_i \models \delta_i \end{cases}$$

Remarque 2.3 *Par convention, la date t_0 correspond à la date de début d'exécution du mot temporisé, on suppose donc que $t_0 = 0$.*

Une telle exécution est dite *acceptante* pour u si s_0 est un état initial et

- soit le chemin est fini et se termine dans un état final,
- soit le chemin est infini et passe infiniment souvent par au moins un état répété[5].

Un mot u est *accepté* par l'automate $\mathcal{A}$ s'il existe une exécution acceptante pour u dans $\mathcal{A}$. Le *langage* accepté par $\mathcal{A}$, noté $L(\mathcal{A})$, est l'ensemble de tous les mots (finis ou infinis) acceptés par $\mathcal{A}$.

La sémantique d'un TA est un système de transitions temporisé où chaque *état du système* ou *configuration* est la réunion d'un *état du TA* et les valeurs actuelles des horloges. Il existe deux types de transitions entre les états. Le TA peut rester dans un état un certain temps (*une transition de délai*), ou passer d'un état du TA à l'autre (*une transition d'action*).

Dans [Alu99], Un TA est de la forme $\langle \Sigma, S, S_0, H, E, I \rangle$ où I est une fonction d'étiquetage de chaque état s par un *invariant* $I(s)$ dans $\Phi(H)$. Un système peut rester dans un état s tant que les valeurs actuelles des horloges satisfont l'invariant $I(s)$.

Définition 2.6 *Une configuration d'un automate temporisé $\mathcal{A}$ est un couple $\langle s, v \rangle$ avec s état de $\mathcal{A}$ et v valuation des horloges de $\mathcal{A}$.*

[4] Dans [ACD93], l'appellation *graphes temporisés* (*Timed Graphs*) est attribuée à une structure de Kripke où chaque arc est muni d'une contrainte temporelle.

[5] Pour un chemin r, l'ensemble des états répétés $inf(r)$ est l'ensemble des états $s \in S$ tels que $s = s_i$ pour infiniment souvent $i \geq 0$.

Définition 2.7 *La sémantique d'un automate temporisé* $\mathcal{A} = \langle \Sigma, S, S_0, H, E, I \rangle$ *est un système de transitions temporisé dont les états sont des configurations* $\langle s, v \rangle$*, et les transitions sont définies par les règles :*

- $\langle s, v \rangle \xrightarrow{a} \langle s', v' \rangle$ *si* $s \xrightarrow{a,\lambda,\delta} s' \in E$*,* $v \models \delta$*,* $v' = [\lambda \rightarrow 0]\, v$ *et* $v' \models I(s')$*.*
- $\langle s, v \rangle \xrightarrow{d} \langle s, v + d \rangle$ *si pour tout* $0 \leq d' \leq d$*,* $v + d' \models I(s)$ *avec* $d, d' \in \mathbb{R}^+$*.*

Remarque 2.4 *Il est évident qu'il existe une infinité de configurations pour un système de transitions temporisé vu la densité du domaine sous-jacent (l'ensemble des réels positifs* $\mathbb{R}^+$*).*

Considérons l'exemple de la Figure 2.3. L'état initial est s_0. Il existe une seule horloge x. La notation $x := 0$ correspond à l'action de réinitialisation de l'horloge x au moment où la transition est franchie. La notation $(x < 2)?$ représente la contrainte temporelle associée à la transition. Un exemple d'exécution de cet automate est :

$$(s_0, 0) \xrightarrow[0.5]{a} (s_1, 0) \xrightarrow[0.3]{b} (s_0, 0.3) \xrightarrow[5.4]{a} (s_1, 0) \xrightarrow[1.7]{b} (s_0, 1.7)$$

A travers cette exécution, l'automate accepte le mot temporisé $(a, 0.5)\,(b, 0.3)\,(a, 5.4)\,(b, 1.7)$, cela en considérant que s_0 est l'état initial et final. Formellement, le langage reconnu par cet automate est : $\{((ab)^\omega, t) \,|\forall i.\ (t_{2i} < t_{2i-1} + 2)\}$.

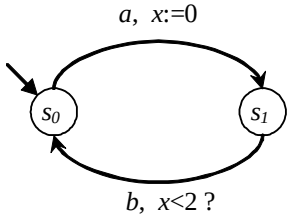

FIG. 2.3 – *Exemple d'un automate temporisé*

Problèmes de décidabilité

Une fois que les systèmes sont modélisés, le problème de tester le vide du langage accepté par le modèle est fondamental. En effet, le problème de l'accessibilité d'un état (c'est-à-dire tester s'il existe une exécution dans le modèle qui permet d'atteindre un état donné) est strictement équivalent à la non-vacuité du langage accepté par ce même modèle en prenant comme état final l'état dont on cherche à tester l'accessibilité. Par exemple, pour assurer une propriété d'exclusion mutuelle, il faut tester que deux processus ne peuvent pas aller simultanément dans leur section critique, ce qui revient à tester l'accessibilité de l'un des états du système où deux processus sont simultanément en section critique. Le problème de tester le vide d'un langage accepté par un automate est appelé : *le problème du vide.*

Théorème 2.1 [AD94] *Le problème du vide est décidable pour la classe des automates temporisés.*

Théorème 2.2 [AD94] *Tester le vide d'un langage accepté par un automate temporisé est un problème* PSPACE*-complet*[6].

Ce résultat est fondamental, car il permet d'utiliser le modèle des automates temporisés dans les approches de vérification basées sur le test du vide. Par contre, le Corollaire 2.1 montre qu'il n'est pas possible de tester si un modèle représenté par un automate temporisé vérifie une propriété donnée par un autre automate temporisé, ce qui est vital dans une autre approche de vérification.

Théorème 2.3 [AD94] *Le problème universel*[7] *est indécidable pour les automates temporisés (*Π_1^1*-difficile*[8]*).*

Le corollaire suivant est une conséquence de ce théorème :

Corollaire 2.1 [AD94] *Étant donnés deux automates temporisés* $\mathcal{A}$ *et* $\mathcal{B}$*, le problème de savoir si* $L(\mathcal{A}) \subseteq L(\mathcal{B})$ *est un problème indécidable (*Π_1^1*-difficile).*

Le résultat négatif du Corollaire 2.1 limite l'utilisation des automates temporisés pour des problèmes de vérification. Ce résultat va inciter à se restreindre à une sous-classe des automates temporisés pour tenter d'obtenir un ensemble dans lequel l'inclusion serait décidable. Cette sous-classe est la classe des automates temporisés *déterministes*. Malheureusement, cette dernière est beaucoup moins expressive que celle des automates temporisés classiques.

Définition 2.1 (Automate temporisé déterministe) *Soit* $\mathcal{A} = \langle \Sigma, S, S_0, H, E \rangle$ *un automate temporisé. Il sera dit déterministe s'il contient un seul état initial* s_0*, et si pour toutes les transitions* $\langle s, s', a, \lambda, \delta \rangle$ *et* $\langle s, s'', a, \lambda', \delta' \rangle$*, on a* $\delta \wedge \delta'$ *qui est toujours faux.*

Théorème 2.4 [AD94] *Étant donnés deux automates temporisés déterministes* $\mathcal{A}$ *et* $\mathcal{B}$*, le problème de savoir si* $L(\mathcal{A}) \subseteq L(\mathcal{B})$ *est un problème décidable (*PSPACE*-complet).*

En dépit de tout ces résultats, le modèle des automates temporisés est tout de même largement utilisé pour la vérification des systèmes temporisés, notamment dans les approches basées sur le test du vide.

Alur et Dill ne sont pas les seuls à s'être intéressés aux automates temporisés. Dans ce qui suit, nous citons quelques travaux qui ont été réalisés sur ce modèle.

[6] Les problèmes *complets* d'une classe C constituent les problèmes les plus «*difficiles*» dans C.

[7] C'est-à-dire, étant donné un automate temporisé $\mathcal{A}$ sur un alphabet Σ, est-ce que $\mathcal{A}$ accepte tous les mots temporisés sur Σ ?

[8] La classe Π_1^1 constitue les problèmes extrêmement indécidables.

2.2.2 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].

2.2.3 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*[9], dans lesquels le temps ne peut que diverger (c'est-à-dire ne comportant que des séquences d'exécution non-Zénon (Voir Section 4.2.3)). 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*

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

[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.

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.

2.3 Conclusion

Nous avons introduit dans ce chapitre quelques formalismes de spécification des systèmes à temps contraint. Ce qu'on constate essentiellement est que ces formalismes sont généralement inspirés d'autres plus simples en associant le concept *temps*. Pour les algèbres de processus, nous nous sommes focalisés sur un formalisme standard de spécification qui est l'algèbre de processus LOTOS ; ainsi, nous avons mentionné quelques extensions de ce dernier tout en étudiant l'une d'elles : RT-LOTOS.

Le langage de spécification formelle RT-LOTOS a été défini en considérant une sémantique d'entrelacement où l'atomicité temporelle et structurelle des actions sont des propriétés indispensables à son utilisation. Cependant, dans la pratique, une telle considération induit des restrictions d'expression au niveau des spécifications des systèmes, entre autre, dire qu'une action est de durée nulle est une hypothèse qui peut être largement contestée dans certains cas. Pour palier à de telles restrictions, nous préférons se servir du langage temps-réel D-LOTOS [SC03] qui a été défini en considérant les deux aspects à savoir les contraintes temporelles et la non atomicité des actions ; ceci est rendu possible grâce à l'utilisation de la sémantique de maximalité. Ce langage va être présenté dans le Chapitre 3.

Dans la deuxième partie de ce chapitre, nous avons passé en revue le modèle des automates temporisés au tant que modèle de bas niveau permettant la considération de l'aspect «temps» dans les automates classiques, puis nous avons exposé quelques extensions et sous-classes de ce modèle. On note que l'introduction de certaines extensions et sous-classes des automates temporisés a pour but de comparer ces modèles avec le notre, ainsi, des discussions sont avancées dans les sections 3.2 et 4.2.5 qui concernent principalement l'expression de la non-atomicité structurelle et temporelle des actions, et les contraintes de sensibilisation et d'urgence des actions.

Chapitre 3

Modèle sémantique pour les systèmes avec durées d'actions

Dans ce chapitre, nous discutons le modèle des automates temporisés et son aptitude à modéliser la non-atomicité des actions, tout en définissant et motivant le modèle sémantique des DATA's (pour *Durational Action Timed Automata*) qui est un modèle de bas niveau. Puis nous étalons la manière de construire opérationnellement un DATA à partir d'une spécification écrite dans un formalisme de haut niveau, à savoir le langage Basic LOTOS avec durées d'actions.

Le modèle des DATA's s'inspire de la sémantique de maximalité [CS95, Saï96], donc il nous paraît naturel d'introduire en premier lieu cette sémantique.

3.1 Sémantique de maximalité

3.1.1 Principe de la sémantique de maximalité

La sémantique d'un système concurrent peut être caractérisée par l'ensemble des états du système et des transitions par lesquelles le système passe d'un état à un autre. Dans l'approche basée sur la maximalité, les transitions sont des événements qui ne représentent que le début de l'exécution des actions. En conséquence, l'exécution concurrente de plusieurs actions devient possible, c'est-à-dire que l'on peut distinguer exécutions séquentielles et exécutions parallèles d'actions.

Etant donné que plusieurs actions qui ont le même nom peuvent s'exécuter en parallèle (autoconcurrence), nous associons, pour distinguer les exécutions de chacune des actions, un identificateur à chaque début d'exécution d'action, c'est-à-dire à la transition ou à l'événement associé. Dans un état, un événement 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 là. Associer des noms d'événements maximaux aux états nous conduit à la notion de configuration qui sera formalisée dans la Définition 3.2.

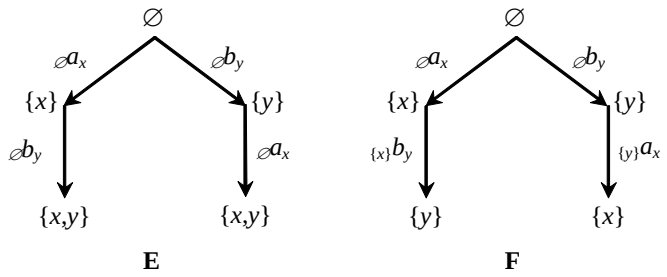

FIG. 3.1 – *Arbres de dérivation de E et F*

Pour illustrer la maximalité et cette notion de configuration, considérons les expressions de comportement $E = a; stop \;|||\; b; stop$ et $F = a; b; stop \;[]\; b; a; stop$. Dans l'état initial, aucune action n'a encore été exécutée, donc l'ensemble des événements maximaux est vide, d'où les configurations initiales suivantes associées à E et F : $_{\emptyset}[E]$ et $_{\emptyset}[F]$. En appliquant la sémantique de maximalité, les transitions suivantes sont possibles : $_{\emptyset}[E] \xrightarrow{\emptyset a_x}_m {}_{\{x\}}[stop] \;|||\; _{\emptyset}[b; stop] \xrightarrow{\emptyset b_y}_m {}_{\{x\}}[stop] \;|||\; _{\{y\}}[stop]$.

x (respectivement y) étant le nom de l'événement identifiant le début de l'action a (respectivement b). Etant donné que 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 maximaux dans cette configuration. Notons que x est également maximal dans l'état intermédiaire représenté par la configuration $_{\{x\}}[stop] \;|||\; _{\phi}[b; stop]$.

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 événement maximal dans la configuration $_{\{x\}}[b; stop]$. Il est clair que, au vu de la sémantique de l'opérateur de préfixage, le début de l'exécution de l'action b n'est possible que si l'action a a terminé son exécution. Par conséquent, x ne reste plus maximal lorsque l'action b commence son exécution ; l'unique événement maximal dans la configuration résultante est donc celui identifié par y qui correspond au début de l'exécution de l'action b. L'ensemble des noms des événements maximaux a donc été modifié par la suppression de x et l'ajout de y, ce qui justifie la dérivation suivante : $_{\{x\}}[b; stop] \xrightarrow{\{x\}b_y}_m {}_{\{y\}}[stop]$.

La configuration $_{\{y\}}[stop]$ est différente de la configuration $_{\{x\}}[stop] \;|||\; _{\{y\}}[stop]$, car la première ne possède qu'un seul événement maximal (identifié par y), alors que la deuxième en possède deux (identifiés par x et y). Les arbres de dérivation des expressions de comportement E et F obtenus par l'application de la sémantique de maximalité sont représentés dans la Figure 3.1. Ces structures sont appelées des STEMs (pour *systèmes de transitions étiquetées maximales*).

La définition formelle du modèle des STEMs sera donnée dans la Section 5.3.1.

3.1.2 Sémantique de maximalité de Basic LOTOS

Définition 3.1 *L'ensemble des noms des événements est un ensemble dénombrable noté $\mathcal{M}$. Cet ensemble est parcouru par $x, y, ...$ $M, N, ...$ 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'événement peut se faire de manière déterministe par l'utilisation de toute fonction get : $2^{\mathcal{M}} - \{\emptyset\} \rightarrow \mathcal{M}$ satisfaisant $get(M) \in M$ pour tout $M \in 2^{\mathcal{M}} - \{\emptyset\}$.*

Définition 3.2 *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 $\mathcal{F} \in \mathcal{C}$ $\quad op \in \{ [] , ||| , || , |[L]| , [> \}$*
- *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 3.3 *La fonction $\psi : C \rightarrow 2^{\mathcal{M}}_{fn}$, qui détermine l'ensemble des noms des événements dans une configuration, est définie récursivement par :*

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

Définition 3.4 *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 :*

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

Définition 3.5 *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 :

$$
\begin{array}{ll}
({}_M[E])\,\sigma = {}_{M\sigma}[E] & (\mathcal{E}\ []\ \mathcal{F})\,\sigma = \mathcal{E}\sigma\ []\ \mathcal{F}\sigma \\
(\mathcal{E}\ ||[L]||\ \mathcal{F})\,\sigma = \mathcal{E}\sigma\ ||[L]||\ \mathcal{F}\sigma & (hide\ L\ in\ \mathcal{E})\,\sigma = hide\ L\ in\ \mathcal{E}\sigma \\
(\mathcal{E} \gg F)\,\sigma = \mathcal{E}\sigma \gg F & (\mathcal{E}\ [>\ \mathcal{F})\,\sigma = \mathcal{E}\sigma\ [>\ \mathcal{F}\sigma \\
(\mathcal{E}\,[b_1/a_1, ..., b_n/a_n])\,\sigma = \mathcal{E}\sigma\,[b_1/a_1, ..., b_n/a_n] &
\end{array}
$$

Définition 3.6 *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. $\dfrac{\mathcal{E} \xrightarrow{M^{a}_{x}} \mathcal{E}'}{\mathcal{F}\ []\ \mathcal{E} \xrightarrow{M^{a}_{x}} \mathcal{E}' \qquad \mathcal{E}\ []\ \mathcal{F} \xrightarrow{M^{a}_{x}} \mathcal{E}'}$

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) $\dfrac{\mathcal{E} \overset{M^a x}{\longrightarrow} \mathcal{E}' \qquad a \notin L}{hide\ L\ in\ \mathcal{E} \overset{M^a x}{\longrightarrow} hide\ L\ in\ \mathcal{E}'}$

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

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

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

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

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

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

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

3.2 Explicitation des durées d'actions

Dans cette section qui s'appuie en grande partie sur les résultats de [BS05b], 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 *automates temporisés avec durées d'actions* (*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. Pour cela, nous rappelons tout d'abord l'intuition et la définition formelle des automates temporisés. 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 3.2.(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 3.2.(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

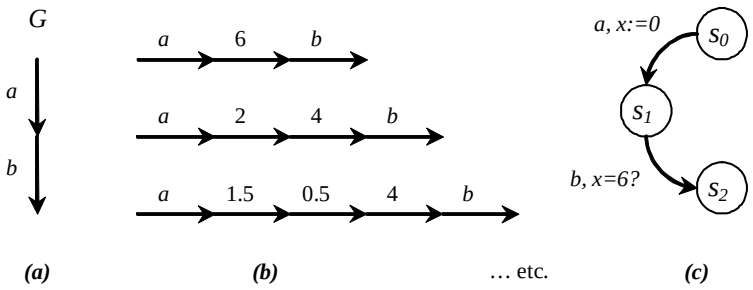

FIG. 3.2 – *Comportement infini de G*

temporelles dans les états. Le comportement infini de G peut être représenté par l'automate temporisé de la Figure 3.2.(c).

Dans la Figure 3.2.(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.

3.2.1 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 le TA de la Figure 3.3.(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 $début(a_i)$ et avoir le comportement de la Figure 3.3.(b).

Le comportement de la Figure 3.3.(b) paraît 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'exé-

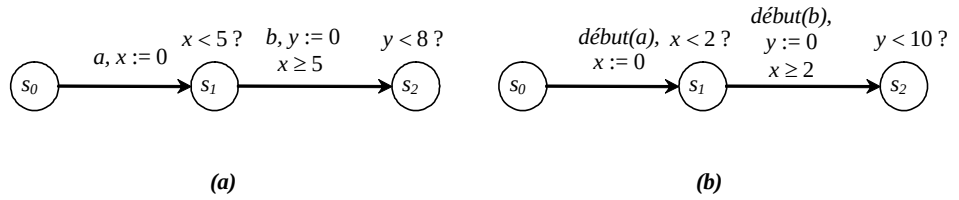

FIG. 3.3 – *Considération des débuts d'exécutions des actions dans les TA's*

cutant 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 3.4.(a).

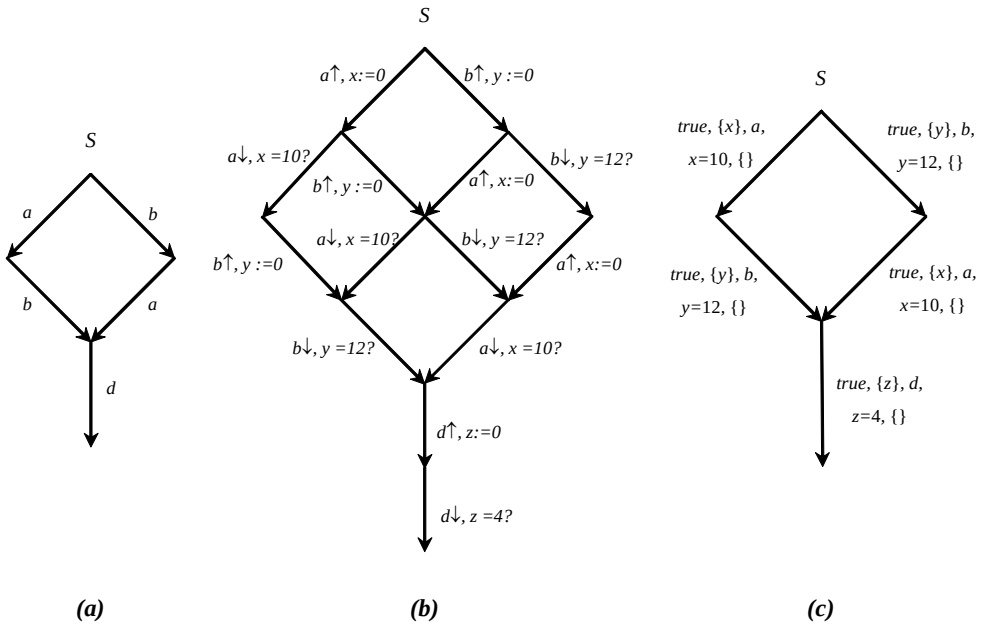

FIG. 3.4 – *Comportement de S*

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 événements : début et fin. Le symbole $\uparrow$ représente l'événement *début* tandis que $\downarrow$ représente l'événement *fin*. Le comportement de S pourra être exprimé en terme d'un TA dans la Figure 3.4.(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 tran-

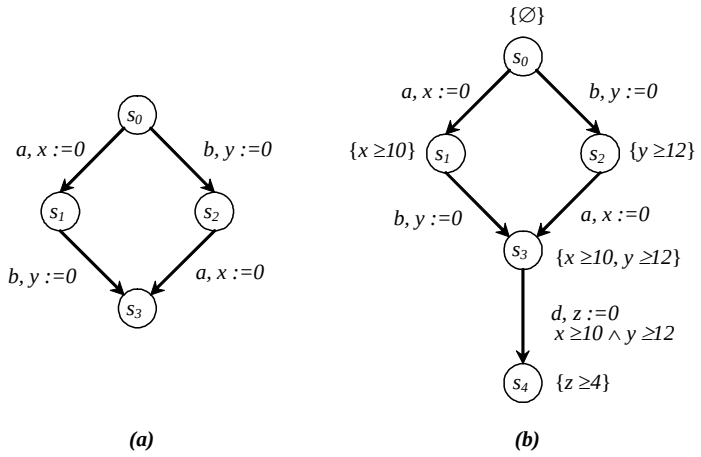

FIG. 3.5 – *Comportement de S, utilisant les conditions de terminaison*

sitions 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, *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 3.4.(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 possibles : $s_0 \xrightarrow{a,\ x:=0} s_1$ 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.

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 3.5.(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 3.3.(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 3.3.(a) sera exprimé par l'automate de la Figure 3.6. Une telle structure sera appelée *Durational Action Timed Automata* (*DATA*).

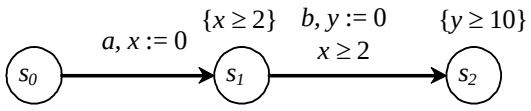

FIG. 3.6 – *Exemple d'un DATA*

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.

3.2.2 Formalisation des DATA's

Définition 3.7 *$\mathcal{H}$, 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 3.8 *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 3.9 *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,*
- *$L_S : S \to 2_{fn}^{\Phi_t(\mathcal{H})}$ 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.*
- *$T \subseteq S \times 2_{fn}^{\Phi_t(\mathcal{H})} \times Act \times \mathcal{H} \times S$ 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 3.10 *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 3.1 *Soit $\mathcal{A}$ le DATA de la Figure 3.6. Un exemple de transitions possibles de la forme $\langle s, v\rangle$ de $\mathcal{S}_{\mathcal{A}}$ est :*

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

3.2.3 Construction opérationnelle des DATA's

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 3.1.2 pour la génération des STEMs.

Désormais, l'ensemble $\mathcal{M}$ des événements 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 événements.

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 3.2.1 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, $\setminus$ et la fonction de substitution introduites dans la Section 3.1.2 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'événements 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 $\psi\,(_M\,[E]) = M$ est remplacée par $\psi\,(_F\,[E]) = \psi_{\mathcal{H}}\,(F)$, avec $\psi_{\mathcal{H}}$ donnée par la Définition 3.11.

Définition 3.11 *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 :*

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

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 $({}_F[E])\backslash K = {}_{F\backslash K}[E]$ qui remplace $({}_M[E])\backslash N = {}_{M-N}[E]$, avec $F\backslash K$ donnée dans la Définition 3.12.

Définition 3.12 *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 3.5.

Définition 3.13 *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) $\dfrac{\mathcal{E} \xrightarrow{F,a,x}_{\perp} \mathcal{E}' \qquad a \notin \{a_1, \ldots, a_n\}}{\mathcal{E}[b_1/a_1, \ldots, b_n/a_n] \xrightarrow{F,a,x}_{\perp} \mathcal{E}'[b_1/a_1, \ldots, b_n/a_n]}$

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

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 3.6.

Notons que les ensembles F au niveau des transitions peuvent être masqués, s'il n'y a pas risque d'ambiguïté, 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.

3.3 Discussion

L'étude du modèle des automates temporisés et des Timed Automata with non-Instantaneous Actions nous a montré que le problème de l'explosion combinatoire du graphe d'états et la difficulté d'exprimer certains comportements parallèles sont respectivement inhérents à la structure de chacun de ces modèles. Le modèle des DATA's nous a permis de surmonter chacun de ces problèmes. Dans ce qui suit, nous étendons notre étude à d'autres modèles ;

l'intérêt principal sera porté sur l'aptitude de chacun des modèles étudiés à prendre en considération la non-atomicité temporelle et structurelle des actions.

Dans les *Timed Safety Automata* (*TSA's*) de [HNSY94], au lieu d'utiliser les conditions d'acceptation de BÜCHI, tel est le cas dans le modèle original ([AD94]), les états de l'automate peuvent contenir des contraintes temporelles locales fermées en arrière sur les horloges appelées *invariants d'états*.

Un TSA peut rester dans un état aussi longtemps que les valeurs d'horloges satisfont l'invariant d'état. *Le problème de la Figure 3.3.(b) persiste toujours* vu que ce TA peut être considéré comme un TSA car les invariants utilisés sont tous fermés en arrière.

Dans le modèle des Event-Recording Automata [AFH94], à chaque occurrence d'une action a, L'horloge correspondante à a (notée x_a) est remise à zéro automatiquement. Le modèle global des Event-Clock Automata exige que les événements soient de durée nulle, donc, nous allons confronter de nouveau le problème de l'explosion combinatoire lors de l'éclatement des actions en début et fin d'exécution.

Une remarque importante est celle concernant le cas où on a plusieurs actions du même nom s'exécutant en parallèle. Celles-ci doivent partager par définition, dans le cas des Event-Recording Automata, une seule horloge, ce qui rend impossible l'expression de l'autoconcurrence. Pour distinguer de près ce problème, considérons l'exemple dans lequel deux actions concurrentes ont le même nom a et la même durée 5. Dans l'Event-Recording Automaton de la Figure 3.7.(a), l'horloge x_a correspond à l'action a. Cette association définie au départ entre x_a et a empêche l'expression de l'autoconcurrence dans le cas où une action commence son exécution alors que l'autre n'a pas encore terminé. Autrement dit, une seule horloge ne peut nous informer de l'évolution de chaque action du moment où on perd toute information sur l'exécution d'une action a juste après le lancement d'une autre du même nom a. Ce comportement peut être exprimé correctement par le DATA de la Figure 3.7.(b) où le nombre d'horloges suit le nombre d'entités concurrentes (quelque soit leurs noms) dans le système. Ainsi, on peut toujours garder l'information sur les évolutions de toutes les actions s'exécutant en parallèle.

Concernant le modèle des *Dynamic Timed Automata* (*DTA's* [CdO95b, Loh02]), il est à noter qu'il est très proche au modèle des automates temporisés. Nous pouvons constater que le modèle des DATA's, en plus de son aptitude à prendre en compte les actions non atomiques, permet l'utilisation des horloges dynamiques dont la construction se fait en une seule passe. Cependant, dans les DTA's, la construction des horloges se fait en deux passes [CdO95b], la première passe est destinée essentiellement à construire une horloge virtuelle pour chaque composante parallèle de la spécification RT-LOTOS, tandis que la deuxième est consacrée au mapping entre ces horloges virtuelles et les horloges actuelles du DTA à générer.

En résumé, nous avons proposé dans ce chapitre un modèle, très proche syntaxiquement du modèle des automates temporisés, pour la prise en compte de la non-atomicité structurelle et temporelle des actions. L'idée est basée sur le principe de la sémantique de maximalité dans laquelle seuls les débuts des actions sont modélisés. Les fins d'exécution des actions

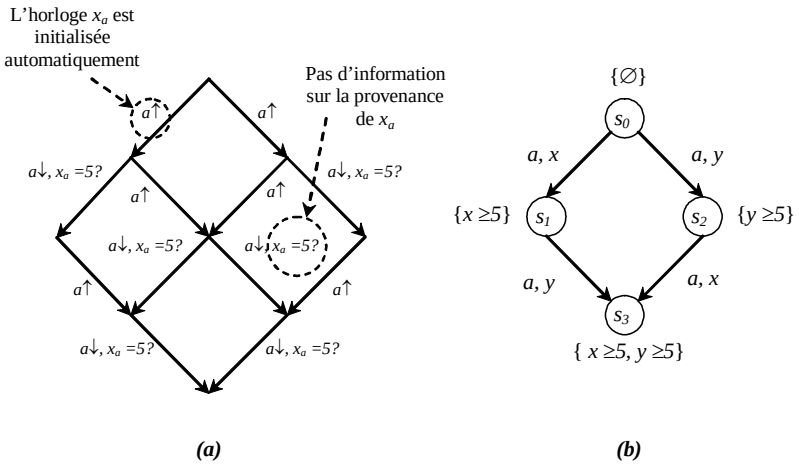

FIG. 3.7 – *Autoconcurrence dans les Event-Recording Automata*

étant capturés par les durées correspondantes.

Chapitre 4

Modèle sémantique pour les systèmes temps-réel avec durées d'actions

Après avoir introduit le modèle des DATA's pour la prise en compte des durées explicites des actions, nous étendons notre approche aux systèmes à temps contraint pour pouvoir prendre en considération les contraintes temporelles et l'urgence des actions. Cela est réalisé par une extension plus générale du modèle des DATA's, appelée DATA*.

Pour spécifier des systèmes de plus en plus complexe à temps contraint avec durées d'actions, nous avons besoin d'un langage de haut niveau intégrant à la fois contraintes temporelles et durées d'actions. Le lanage D-LOTOS répond bien à ce besoin.

Avant d'introduire les DATA*'s, nous présentons le langage D-LOTOS [SC03][1]. Nous expliquerons à la fin de ce chapitre comment une structure de DATA* est construite à partir de spécifications D-LOTOS.

4.1 Le langage D-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 $\tau : Act \rightarrow \mathcal{D}$ 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

[1] Nous reprenons la définition formelle de D-LOTOS de [SC03].

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 pouvons 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 [CdS93, 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 4.1 [2] *L'ensemble $\mathcal{C}_t$ des configurations temporelles est donné par :*

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

[2] 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* $F \in \mathcal{B}$ *alors* $\mathcal{E} \gg F \in \mathcal{C}_t$
- *si* $\mathcal{E}, \mathcal{F} \in \mathcal{C}_t$ *alors* $\mathcal{E}$ *op* $\mathcal{F} \in \mathcal{C}_t \qquad op \in \{ \; [] \; , \; ||| \; , \; || \; , \; |[L]| \; , \; [> \; \}$
- *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.

4.1.1 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 4.2.

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

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

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 3, 4(a)i, 4(a)ii, 4b, 5a, 5b, 6a, 7a, 7b, 7c, 8a et 9 de la Définition 3.6, dans lesquelles les configurations sont temporelles et la relation de transition est remplacée par $\rightarrow_\tau$, complétées par les règles suivantes :

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

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

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

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

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

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

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

Définition 4.2 *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].

4.1.2 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 4.1.1 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 :

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

4.2 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.[3]

4.2.1 Intuition

Considérons le système S décrit dans la Section 3.2.1. A partir de l'état s_3 de la Figure 3.5.(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

[3] Cette section s'appuie sur les résultats de [BS05a].

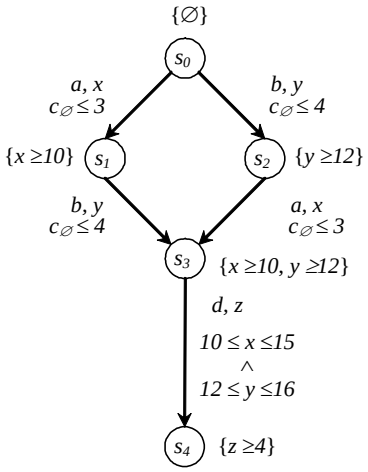

FIG. 4.1 – *DATA* du comportement de S*

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 offerte dans le sous-système S_1 (resp. S_2) dans les 5 (resp. 4) unités de temps, le comportement global de S est représenté par le DATA* de la Figure 4.1.

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 $y \in [12, 16]$.[4]

4.2.2 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

[4] $x \in [\min, \max]$, $\min \leq x \leq \max$, $x \geq \min \wedge x \leq \max$, 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.

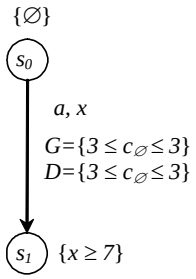

FIG. 4.2 – *Expression de l'urgence par un DATA**

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 4.1, 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 4.2. 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.

4.2.3 Formalisation

Définition 4.3 *Un DATA* $\mathcal{A}$ est un quintuplet $(S, L_S, s_0, \mathcal{H}, T_D)$ 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.*
- $T_D \subseteq S \times 2^{\Phi_t(\mathcal{H})}_{fn} \times 2^{\Phi_t(\mathcal{H})}_{fn} \times Act \times \mathcal{H} \times S$ *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 4.4 *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 (règles* RA1* *et* RA2**) et au tirage d'une transition de $\mathcal{A}$ (règle* RD**).*

$$\text{(RA1*)}\ \frac{d \in \mathbb{R}^+ \qquad \forall d' \leq d,\ v + d' \not\models D}{\langle s, v\rangle \xrightarrow{d} \langle s, v + d\rangle} \qquad \text{(RA2*)}\ \frac{\varepsilon \in \mathbb{R}^+ \qquad v + \varepsilon \vDash D \quad et \quad \varepsilon \leq \eta}{\langle s, v\rangle \xrightarrow{\varepsilon} \langle s, v + \varepsilon\rangle}$$

$$\text{(RD*)}\ \frac{(s, G, D, a, x, s') \in T_D \qquad v \models G}{\langle s, v\rangle \xrightarrow{a} \langle s', [\{x\} \mapsto 0]\, v\rangle}$$

Où η *est la plus petite quantité réelle de temps dans laquelle aucune action ne se produit.*

Nous aurions pu exprimer le passage de temps entre les configurations de $\mathcal{S}_\mathcal{A}$ par la seule règle RA1*. Or, cela nous posera un problème engendré entre autre par l'utilisation des prémisses négatives. Ce problème fréquent, dit de *Zénon*[5], est la divergence de temps dans un intervalle fini. Une infinité d'occurrences d'actions ayant lieu aux instants 1/2, 2/3, 3/4, ... est un exemple d'une séquence de Zénon. Ce problème peut aussi être la source d'inconsistence [LL98] ou d'incomplétude [AL94] pour les méthodes de preuve.

L'expression de la règle RA2* de la Définition 4.4 est rendue possible grâce à une hypothèse, garantissant qu'il y a un nombre fini d'actions dans un intervalle fini de temps. Cette hypothèse largement acceptée (comme dans [ACD93, AD94, HNSY94][6]) est basée sur l'existence d'un *voisinage temporel* d'une action. Ainsi, notre étude se focalise sur les systèmes ayant des comportements non-Zénon.

[5] Du nom de *Zénon d'Élée* (V^e siècle av. J.-C.), mathématicien et philosophe grec célèbre pour ses paradoxes philosophiques.

[6] Dans [HNSY94], cette hypothèse est appelée *finite variability condition.*

Propriété 4.1 *Soit t l'instant du début d'exécution d'une action a. Un* voisinage temporel *de cette action est le domaine temporel $]t-\eta, t+\eta[$ tel que η est la plus petite quantité de temps dans laquelle aucune action ne se produit dans un système.*

La notion de voisinage peut être aussi trouvée dans la littérature (le voisinage ϵ de [BGS04]).

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 4.1 *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.

4.2.4 Construction opérationnelle des DATA*'s

Cette section explique comment construire opérationnellement un DATA* à partir d'une spécification D-LOTOS. La démarche est comparable à celle de la Section 3.2.3 relatant les DATA's. Nous gardons les mêmes fonctions déjà évoquées dans la Section 3.2.3. 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 4.5 *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)* $$\frac{}{\emptyset[exit\{u\}] \xrightarrow{G=\{c_\emptyset \leq u\}, D=\{\texttt{false}\}, \delta, x}_* {}_{\{x\geq 0\}}[stop]} \; x=get(\mathcal{M})$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Notation 4.1 *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 = `true`*. Si la contrainte d'urgence D est omise au niveau d'une transition, alors $D =$* `{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 4.5, toutes les actions ne sont pas urgentes, ce qui explique que les contraintes D sont toujours mises à `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 4.5.

La transition suivante est alors possible :

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

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 :

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

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 4.2 *Notons que selon la Remarque 4.1, 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 4.3 *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.*

4.2.5 Discussion

Cette section dégage quelques remarques sur quelques modèles basés sur les automates temporisés et leur aptitude à spécifier des systèmes à temps contraint.

Il est important de souligner que les modèles cités dans cette section se sont avérés définis en supposant que les actions sont de durée nulle ; donc, toutes les réflexions faisant sujet de la discussion sur l'aptitude des modèles à spécifier des actions atomiques de la Section 3.3 s'appliquent conséquemment sur ces modèles. Inversement, tous les modèles vus dans la Section 3.3 n'ont pas la possibilité de spécifier l'urgence.

La notion d'urgence a été introduite dans les TADs (*Timed Automata with Deadlines*) de [BS98, BST97]. Ce modèle permet l'expression de l'urgence au niveau des transitions par des contraintes d'échéance fermées à gauche comme pour le modèle des DATA*'s, sauf que les TADs sont exprimés sur un alphabet d'actions instantanées.

Examinons maintenant le modèle des automates temporisés avec mises à jour. Ce modèle a le même pouvoir d'expression que le modèle des automates temporisés avec ε-transitions (ou *actions silencieuses*) qui est strictement plus expressif que les automates temporisés classiques (Voir [BDGP98]). Dans, [Bou02, BDFP04], il a été prouvé par exemple qu'il existe un automate temporisé avec mises à jour (celui de la Figure 4.4) qui reconnaît le même langage reconnu par l'automate avec ε-transitions de la Figure 4.3.

Certes, le modèle général des automates temporisés avec mises à jour est plus expressif que le modèle original des automates temporisés, mais à l'inverse, le problème du vide

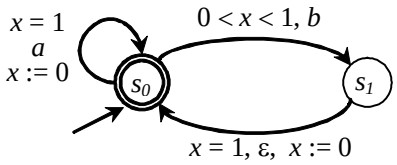

FIG. 4.3 – *Automate temporisé avec ε-transitions*

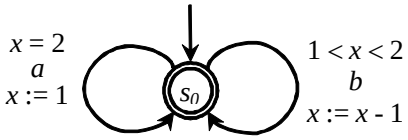

FIG. 4.4 – *Automate temporisé avec mises à jour*

dans ce modèle s'est avéré indécidable [BDFP00]. Cependant, en utilisant une extension de la construction des graphe de régions, on a montré quelques classes décidables où la décidabilité dépend de la nature des contraintes d'horloge employées, diagonales[7] ou pas. Dans [BDFP04], on a montré que n'importe quel automate de mise à jour appartenant à une certaine sous-classe décidable peut être efficacement transformé en un automates temporisé sans mises à jour mais avec des transitions silencieuses (*automates temporisé avec ε-transitions*). Néanmoins, cette transformation souffre d'une énorme explosion de combinatoire qui semble inévitable.

Vu que le tirage des transitions dans le modèle des DATA*'s implique une remise à zéro automatique des horloges correspondantes aux actions, le comportement de la Figure 4.4 n'a pas pu être exprimé à l'aide d'un DATA*.

Pour spécifier des protocoles s'appuyant sur les mises à jour comme ABR (*Available Bit Rate* [BF99]), une alternative consiste à modéliser la mise à jour par l'utilisation d'opérations plus générales sur les variables, comme cela est implanté dans l'outil HYTECH [HHWT97].

Nous pouvons conclure qu'une comparaison théorique du modèle des DATA*'s avec les modèles existants serait d'un intérêt capital pour pouvoir cerner le modèle le moins expressif mais le plus suffisant pour répondre aux besoins des applications temps-réel avec durées associées aux actions.

[7] Une contrainte d'horloges diagonale implique une comparaison entre plusieurs horloges. Si par exemple, x et y sont des horloges et c est une constante réelle, $x - y \leq c$ est une contrainte diagonale tandis que $x \leq c$ n'est pas une contrainte diagonale.

4.3 Conclusion

Le modèle de spécification des systèmes temps-réel D-LOTOS a été introduit dans ce chapitre. Il intègre deux concepts à savoir la spécification des contraintes temporelles et la prise en compte des durées d'action. D'un point de vue syntaxique, il reste très proche du formalisme ET-LOTOS, mais d'un point de vue sémantique, on s'est abstrait de l'hypothèse d'atomicité des actions imposée par la sémantique d'entrelacement du parallélisme. Cela a pu être réalisé par l'utilisation de la sémantique de maximalité, introduite dans la Section 3.1.

On note également que dans [SC03], des relations de bissimulation temporelle pour analyser le comportement de systèmes temps-réel d'une part ainsi qu'une relation de performance permettant de caractériser des systèmes concurrents ayant des performances identiques quelque soit l'architecture sous-jacente sur laquelle ces systèmes s'exécutent, ont été définies. L'introduction de durées dynamiques a permis en outre de formaliser la notion de latence telle qu'elle a été définie dans le formalisme RT-LOTOS.

Dans une autre partie de ce chapitre, nous avons proposé une extension du modèle des DATA's qui nous a permis de prendre en compte certaines notions rudimentaires des systèmes temps-réel, à savoir les contraintes temporelles et la notion d'urgence des actions, tout en supportant la non-atomicité temporelle et structurelle des actions. A travers les discussions présentées dans les sections 3.3 et 4.2.5, nous pouvons conclure que le modèle des DATA*'s est suffisant pour spécifier le comportement des systèmes à temps contraint avec durées associées aux actions. De plus, il permet de combler certaines carences qui peuvent altérer une spécification plus fidèle du comportement d'un système temps-réel.

Chapitre 5

Vérification formelle basée sur la sémantique de 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.

5.1 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. Tout cela est illustré dans la Figure 5.1.

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.

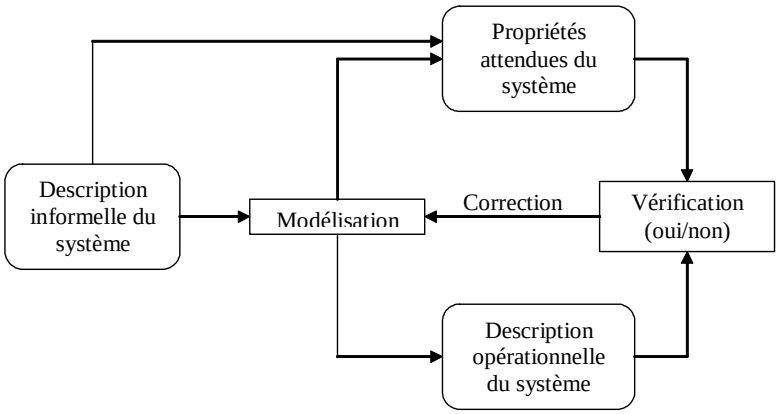

FIG. 5.1 – *Vérification formelle*

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 particu-

lièrement 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.

5.2 Model checking

5.2.1 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. L'architecture globale d'un model checker est illustrée par la Figure 5.2.

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

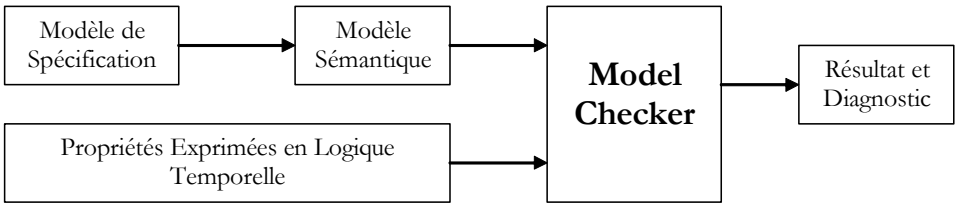

FIG. 5.2 – *Architecture d'un model checker*

simples de la logique temporelle permettent d'exprimer les propriétés usuelles des programmes parallèles.

5.2.2 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].

Par exemple dans le graphe de la Figure 5.3, pour que la formule **EG**p soit vraie à l'état s, il faut que p soit vraie au moins dans tous les nœuds d'un chemin sortant de s.

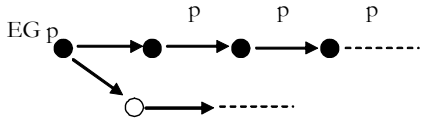

FIG. 5.3 – *La formule* **EG**p *est vraie à l'état initial*

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.

5.2.3 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) : $\mathbf{FG}(req_i) \Rightarrow \mathbf{GF}(done_i)$.

5.3 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 3.1.1 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.

5.3.1 Formalisation des STEMs

Une présentation détaillée de la sémantique de maximalité peut être trouvée dans [CS95, Saï96]. Dans cette section, nous nous contentons de rappeler la définition de la structure de STEMs et nous illustrons le concept par un exemple simple.

Définition 5.1 *$\mathcal{M}$ étant un ensemble dénombrable de nom d'événements, un système de transitions étiquetées maximales de support $\mathcal{M}$ est un quintuplet $(\Omega, A, \mu, \xi, \psi)$ avec :*

- *$\Omega = \langle S, T, \alpha, \beta \rangle$ est un système de transitions tels que :*
  - *S : l'ensemble des états dans lesquels peut se trouver le système, cet ensemble peut être fini ou infini.*
  - *T : l'ensemble des transitions indiquant le changement d'états que peut réaliser le système, cet ensemble peut aussi être fini ou infini.*
  - *α et β sont deux applications de T dans S tel que pour toute transition $t \in T$ on a : $\alpha(t)$ désigne l'origine de la transition et $\beta(t)$ son but.*
- *(Ω, A) est un système de transitions étiquetées par un alphabet A.*
- *$\psi : S \longrightarrow 2^{\mathcal{M}}_{fn}$ est une fonction qui associe à chaque état l'ensemble fini des noms des événements maximaux présents au niveaux de cet état.*
- *$\mu : T \longrightarrow 2^{\mathcal{M}}_{fn}$ est une fonction qui associe à chaque transition l'ensemble fini des noms des événements correspondant aux actions qui ont commencé leur exécution et dont la terminaison sensibilise cette transition.*
- *$\xi : T \longrightarrow \mathcal{M}$ est une fonction qui associe à chaque transition le nom de l'événement qui identifie son occurrence.*

Tel que pour toute transition $t \in T$, $\mu(t) \subseteq \psi(\alpha(t))$, $\xi(t) \notin \psi(\alpha(t)) - \mu(t)$ et $\psi(\beta(t)) = (\psi(\alpha(t)) - \mu(t)) \cup \{\xi(t)\}$.

Entre autre, un STEM permet d'exprimer le comportement des systèmes concurrents par la détermination de l'ensemble des actions qui sont potentiellement en cours d'exécution dans chaque état (voir Figure 5.4).

Soit l'expression de comportement Basic LOTOS `H = a;b;stop|||(c;stop[]a;stop)`. Intuitivement, nous pouvons remarquer que durant le comportement d'une telle spécification, avec l'hypothèse que les actions ne sont pas atomiques, dans certains états les actions a et c, a et a, b et c ainsi que b et a peuvent s'exécuter en parallèle. Il est clair que la sémantique d'entrelacement ne permet pas de voir de telles situations. Cependant l'application de la sémantique opérationnelle de maximalité de Basic LOTOS de la Section 3.1.2 permet la génération du STEM de la Figure 5.4.

On peut constater que les états 2, 3, 4, 5, 7, 8, 10, 11, 13 et 14 représentent de tels cas d'exécutions concurrentes d'actions.

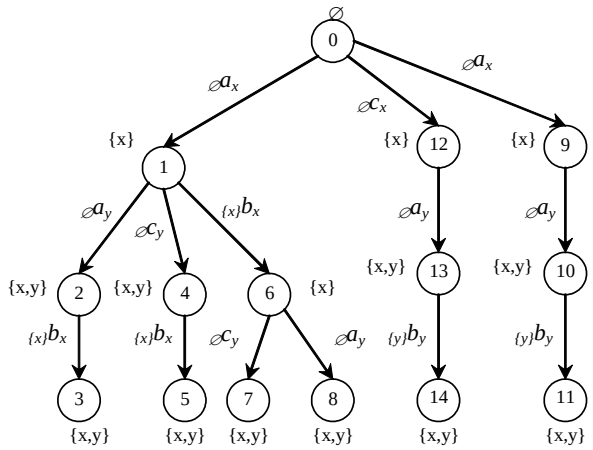

FIG. 5.4 – *STEM de l'expression H*

5.3.2 La 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 $\mathbf{AG}f$ si sur tous les chemins à partir de cet état, f est toujours satisfaite.
- Un état satisfait $\mathbf{EG}f$ s'il existe un chemin issu de cet état où f est toujours satisfaite.
- $\mathbf{A}(f\mathbf{U}f')$ 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'.
- $\mathbf{E}(f\mathbf{U}f')$ 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'.

5.3.3 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 STEMs, 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 STEM 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 5.3.5, des exemples plus élaborés seront présentés par l'étude du problème du dîner des philosophes.

Remarque 5.1 *On a vu dans la structure des STEMs 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.

5.3.4 Algorithme de model-checking

Après avoir vu le genre de propriétés qu'on pouvait exprimer en utilisant le modèle des STEMs 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 STEMs 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 STEMs 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 = A[qUr] :
        pour chaque s ∈ S faire
        si r ∈ L(s) alors
            ajouter A[qUr] à L(s) ;
        fsi
        fpour
        pour j = 1 à Card(S) faire
          pour chaque s ∈ S faire
            si q ∈ L(s) et si ∀ successeur s' de s /
              A[qUr] ∈ L(s') alors
                ajouter A[qUr] à L(s) ;
            fsi
          fpour
        fpour
    p = E[qUr] :
        pour chaque s ∈ S faire
          si r ∈ L(s) alors
            ajouter E[qUr] à L(s) ;
          fsi
        fpour
        pour j = 1 à Card(S) faire
          pour chaque s ∈ S faire
            si q ∈ L(s) et si ∃ successeur s' de s /
              E[qUr] ∈ L(s') alors
                ajouter E[qUr] 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 STEM, alors on remarque bien que cet algorithme peut s'adapter à notre étude pour la vérification des

propriétés sur les STEMs. Pour cela, il suffit de considérer les actions dans les états du STEM 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 $x_1, x_2, ..., x_n$ sont des noms d'événements, q est une action et *act* est une fonction qui reçoit un nom d'événement 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` développé dans notre laboratoire.

5.3.5 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 dernière 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 précédente.
- 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)`.

5.4 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 STEMs 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*) que nous avons développé.

Chapitre 6

Etude de cas

Le but de ce chapitre est de montrer comment spécifier un exemple de comportement d'un système à temps contraint dans le modèle des DATA*'s. Puis, nous discutons une approche de vérification de propriétés qualitatives sur un système temps réel.

Nous prenons comme exemple celui du brûleur à gaz, qui est un exemple classique largement employé comme étude de cas pour la spécification des systèmes temps-réel. Cet exemple a été introduit dans [CHR91] où sa spécification est écrite dans le formalisme de spécification *Duration Calculus* [CHR91]. Un schéma simplifié est donné dans la Figure 6.1.

Un brûleur à gaz est soit actif dans le cas de la présence de la flamme soit inactif. Il alterne indéfiniment entre ces deux états. Lorsque le brûleur est inactif, il n'y a pas une fuite de gaz, cependant, au moment du passage à l'état actif, le gaz doit s'écouler pendant une courte durée avant qu'il soit brûlé ; et quand une panne de flamme apparaît, la soupape de gaz doit être fermée. Toutefois, on peut avoir le cas où le gaz s'écoule et la flamme est éteinte, c'est-à-dire, il y a une *fuite* de gaz. Une conception d'un brûleur à gaz doit assurer que les périodes de fuite ne doivent pas être excessivement longues. Une contrainte temps-réel possible est la suivante : «dans n'importe quel intervalle de temps supérieur à une minute, la proportion des durées totales de fuite ne doit pas dépasser 5% de cet intervalle». Pour garantir cette contrainte, certaines spécifications précisent que chaque fuite de gaz est repérée et stoppée dans un délai d'une seconde ; et pour éviter des fuites de plus en plus fréquentes, après chaque fuite le brûleur rejette pendant trente secondes toutes les requêtes d'allumage,

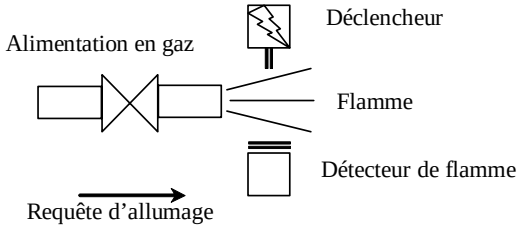

FIG. 6.1 – *Schéma primitif d'un brûleur à gaz*

et ainsi l'écoulement du gaz.

Le comportement simplifié du brûleur à gaz est illustré dans la Figure 6.2.

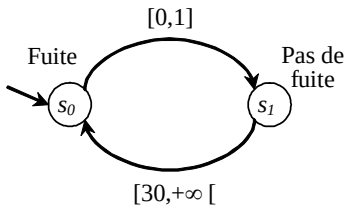

FIG. 6.2 – *Comportement simplifié d'un brûleur à gaz*

6.1 Spécification

Supposons que le brûleur à gaz a le comportement de la Figure 6.2, dans lequel chaque fuite de gaz est détectée et stoppée dans un délai d'une seconde, et après chaque fuite le brûleur attend au moins trente secondes avant que le gaz ait la possibilité de s'écouler à nouveau.

Afin de spécifier formellement ce système à l'aide du langage D-LOTOS, les actions observables suivantes seront considérées :

`HeatOn :` Indique une requête d'allumage du brûleur (*Heat Request*). Cette action a une durée nulle.

`Purge :` Cette action suspend le brûleur pour une durée de 30 secondes, et assure ainsi un intervalle suffisant entre les moments de fuite. Observons que cette action a une *durée* explicite non nulle. Dans d'autres modèles, l'entrelacement implique l'éclatement de cette action en deux.

`Ignite :` C'est l'action qui fait écouler le gaz et allumer le brûleur. Le gaz s'écoule pendant une courte durée d'une demi-seconde avant que l'étincelle ne soit déclenchée.

`FlOn :` Action de détection de présence de la flamme. La détection doit se réaliser dans une durée d'une demi-seconde après avoir déclenché l'étincelle.

`HeatOff :` Indique une requête d'arrêt du brûleur, se résumant en l'extinction de la flamme. Cette action a une durée nulle.

`FlOff :` Action de détection de l'absence de la flamme.

La spécification D-LOTOS du brûleur à gaz est la suivante :

```
system Burner[HeatOn[0],FlOn[0],HeatOff[0],FlOff[0]]:=
hide Purge[30], Ignite[0.5] in
```

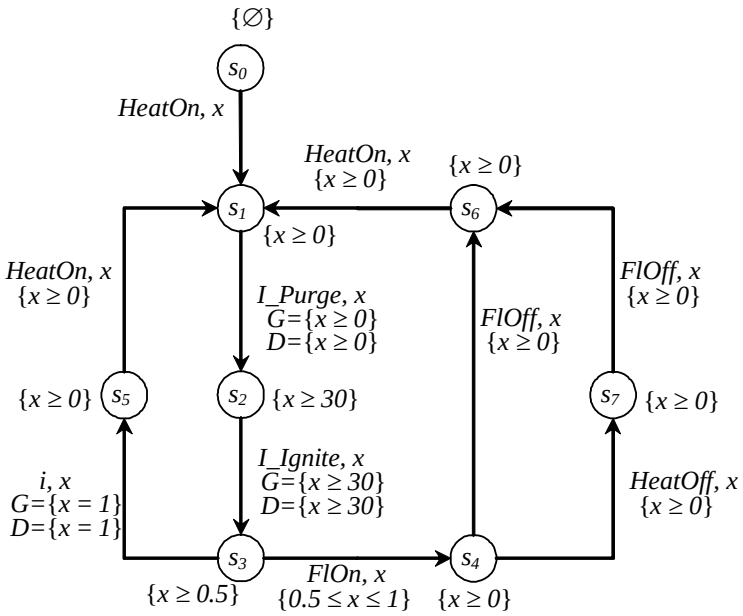

FIG. 6.3 – *DATA* du brûleur à gaz*

```
(
  HeatOn; Purge; Ignite;
  (
    (FlOn{0.5};
      (HeatOff; FlOff; Burner[HeatOn,FlOn,HeatOff,FlOff]
      []
      FlOff; Burner[HeatOn,FlOn,HeatOff,FlOff]))
    []
    (delay[0.5] i; Burner[HeatOn,FlOn,HeatOff,FlOff])
  )
)
endsys
```

Le DATA* correspondant au processus `Burner` est donné par la Figure 6.3.

Ce DATA* a été généré à partir de la spécification D-LOTOS du processus `Burner` en utilisant les règles de construction des DATA*'s de la Définition 4.5. Les actions cachées `Purge`, `Ignite` et `i` sont urgentes, ce qui explique la présence des contraintes d'échéance dans leurs transitions respectives. Sur une transition, l'absence de la contrainte G implique implicitement qu'elle est égale à `true` tandis que l'absence de D implique qu'elle est égale à `false` (dans le cas où l'action n'est pas urgente).

6.2 Vérification

Dans notre cas, nous adoptons une approche de vérification par model checking pour les raisons déjà mentionnées dans la Section 5.1. Conformément à l'approche de vérification décrite dans la Section 5.3, vérifier des propriétés qualitatives sur un système réactif revient à le modéliser par un STEM.

L'exemple du brûleur à gaz est un cas typique des systèmes réactifs à temps contraint (c'est-à-dire que le temps est un facteur déterminant du comportement global du système), dès lors notre spécification devra refléter cette propriété pour permettre une conception fidèle au cahier des charges.

Pour illustrer l'importance du temps dans le comportement de tels systèmes, considérons l'exemple suivant où nous allons faire abstraction du facteur temps et en observer les conséquences :

Soit la spécification Basic LOTOS suivante ayant comme comportement le STEM de la Figure 6.4.(a).

```
system example_1[a,b,c] :=
    a; exit |[a]| (a; b; exit [] a; c; exit)
endsys
```

Maintenant, considérons la spécification D-LOTOS suivante :

```
system example_2[a[10],b[3],c[2]] :=
    a{10}; exit |[a]| (delay(12) a; b; exit [] delay(5) a; c; exit)
endsys
```

Le DATA* correspondant à ce système est représenté dans la Figure 6.4.(b).

Il est clair que l'action b ne sera jamais déclenchée puisque la contrainte temporelle sur a (qui est : $12 \leq c_{\emptyset} \leq 10$) ne le permettra pas (Dans le DATA* de la Figure Figure 6.4.(b), la transition $s_0 \rightarrow s_1$ n'aura jamais lieu). Mais dans le cas de la Figure 6.4.(a), b peut survenir au même titre que c, ce qui ne reflète pas le comportement exact du système global exprimé dans la deuxième spécification.

Nous avons donc un comportement indésirable qui peut se produire et qui peut être fatal dans les systèmes critiques, comme nous pouvons avoir des comportements attendus qui n'aurons jamais lieu ; d'où la nécessité de faire appel à des compilateurs qui considèrent les aspects temporels de ce genre d'applications.

Dès que les propriétés que nous cherchons à vérifier dans notre étude sont qualitatives, et par souci de respect du cahier des charges d'un système à temps contraint, nous avons divisé la procédure de vérification en trois étapes :

1. Génération des DATA*'s pour les spécifications D-LOTOS avec l'outil `SMART` (qui est un compilateur pour D-LOTOS).

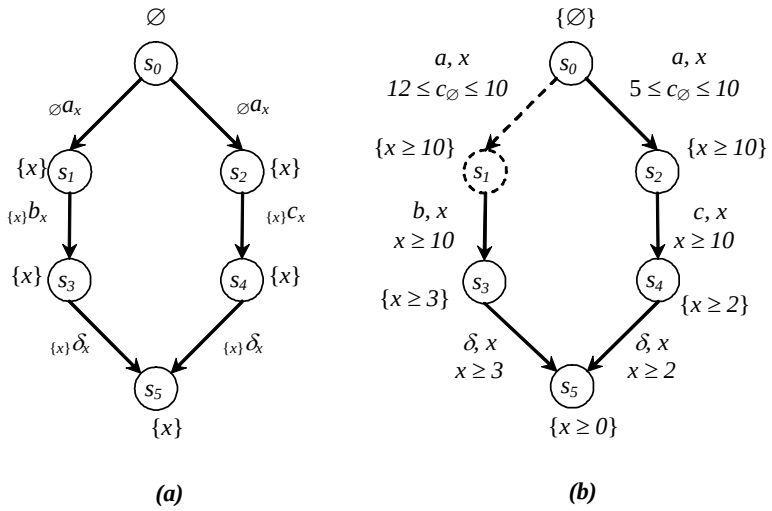

FIG. 6.4 – *Prise en compte des contraintes temporelles*

2. Faire abstraction des valeurs d'horloges (qui ne véhiculent pas une information pertinente pour la vérification des propriétés qualitatives), tout en respectant la structure du DATA*.

3. Vérification des propriétés sur le STEM résultant avec l'outil `LotoStem` qui implante un model checker dont les propriétés à vérifier sont exprimées en CTL, une logique temporelle du temps arborescent.

Pour éliminer des comportements non désirables et plus généraux que celui de l'exemple précédent, nous pouvons envisager la construction d'un *graphe de régions* en regroupant les états dans des classes d'équivalences selon les valeurs d'horloges et leurs évolutions futures (Voir [AD94]).

Dans ce qui suit, nous allons présenter quelques propriétés (qualitatives) typiques du brûleur à gaz, tout en donnant leurs spécifications dans la logique CTL.

Propriétés de sûreté

1. «A partir de l'état initial, la flamme ne peut être allumée sans la requête de l'utilisateur.»

   ```
   A G not((not HeatOn) U FlOn)
   ```

Propriétés de vivacité

1. «Le brûleur doit déclencher une étincelle après une requête d'allumage de l'utilisateur.»

   ```
   A G (HeatOn => A F Ignite)
   ```

2. «Une requête d'allumage a toujours la possibilité d'être satisfaite.»

   ```
   A G (HeatOn => E F FlOn)
   ```

3. «Une requête d'extinction doit toujours être satisfaite le plutôt possible.»

   ```
   A G (HeatOff => A X FlOff)
   ```

6.3 Conclusion

Dans ce chapitre, nous avons présenté une étude d'un exemple de système à temps contraint avec durées explicites sur les actions. L'étude concerne principalement la spécification de ces systèmes à l'aide du langage D-LOTOS puis leur traduction dans le modèle sémantique des DATA*'s. Une dernière étape concerne la spécification des propriétés (qualitatives) attendues de ces systèmes pour être vérifiées ensuite à l'aide d'outils développés dans notre laboratoire.

Cette étude ne nous semble pas suffisante quant à la variété d'exemples étudiés dans la littérature. Donc, nous envisageons d'étudier d'autres exemples de systèmes temps-réel, tels que les protocoles de communication comme AMRHy [DSB04], un protocole de multicast fiable dont la conception formelle peut être trouvée dans [SDAB04].

Chapitre 7

Conclusion et perspectives

Le travail présenté dans ce mémoire est une contribution à la spécification des systèmes temps-réel avec durées explicites des actions. Nous avons mis comme objectif la possibilité d'exprimer à la fois la non-atomicité temporelle et structurelle des actions, les contraintes temporelles, les actions urgentes et les durées d'actions dans un même modèle sémantique. En plus, vu qu'il est basé sur une sémantique de vrai parallélisme, ce nouveau modèle nous a permis de spécifier des comportements parallèles d'actions concurrentes de manière naturelle.

D'abord, nous avons passé en revue de manière non exhaustive quelques langages de spécification de haut et bas niveau dans le Chapitre 2, tout en inspectant les algèbres de processus temps-réel et les automates temporisés. D'une part, nous nous sommes focalisés sur le langage temps-réel RT-LOTOS, un langage de spécification très commun qui permet d'exprimer, en plus de la puissance du langage LOTOS, les contraintes temporelles, les délais et l'urgence des actions. D'autre part, le modèle sémantique des automates temporisés est examiné de près du moment qu'il est le plus proche (syntaxiquement) de notre modèle.

Pour introduire notre modèle, nous avons divisé notre travail en deux parties, l'une a pour but d'exprimer les durées explicites des actions dans un modèle sémantique de bas niveau, et l'autre pour prendre en compte les aspects temps-réel comme l'urgence et les contraintes temporelles. Cela a permis de dégager respectivement le modèle des DATA's (pour *Durational Action Timed Automata*) et son extension temps-réel, le modèle des DATA*'s.

Dans le Chapitre 3, le modèle des DATA's est défini afin de prendre en compte la non-atomicité structurelle et temporelle des actions. L'idée est basée sur le principe de la sémantique de maximalité dans laquelle seuls les débuts des actions sont modélisés. Les fins d'exécution des actions étant capturés par les durées correspondantes.

Ce modèle nous a permis entre autre de capturer les durées des actions, qui sont implicitement prises en compte par la sémantique de maximalité, de manière explicite. Ceci permettra à d'autres catégories de propriétés liées à des aspects quantitatifs de comportements, tels que la borne maximale d'exécution d'une trace donnée, d'être vérifiées. En plus de la définition du modèle des DATA's, nous avons proposé une approche assez générale pour l'attribution de sémantique opérationnelle aux algèbres de processus, incluant les durées explicites des

actions, dont le domaine d'interprétation est l'ensemble des DATA's. Ensuite, nous avons discuté la différence entre les DATA's et quelques sous-classes et extensions des automates temporisés quant à l'aptitude à expliciter les durées des actions.

L'extension temps-réel du modèle des DATA's a fait naître le modèle des DATA*'s qui a été présenté dans le Chapitre 4. Ce modèle nous a permis, en plus de la considération que les actions sont non atomiques (ayant des durées non nulles, et raffinables), de prendre en compte les spécificités principales des systèmes temps-réel, à savoir les contraintes temporelles et la notion d'urgence des actions. Le modèle des DATA*'s nous a permis de loin d'exprimer la sémantique d'un langage temps-réel avec durées d'actions, le langage D-LOTOS. Donc, notre modèle est en mesure d'être considéré comme le modèle de base dans un environnement de vérification formelle des systèmes temps-réel ayant D-LOTOS comme langage de spécification.

Afin de vérifier formellement les systèmes à temps contraint, nous avons opté pour une approche particulièrement attractive, celle basée sur les modèles (model checking). Dans cette approche, l'application à traiter est d'abord spécifiée dans un langage de spécification de haut niveau (comme LOTOS). Cette spécification est ensuite traduite vers un modèle sémantique (graphe d'états ou automate) sur lequel les propriétés attendues du système, exprimées dans une logique temporelle, sont vérifiées à l'aide de model checkers. L'avantage principal de cette approche est qu'elle est en général complètement automatisable, bien que restreinte à des systèmes ayant un nombre fini d'états. Pour remédier au problème d'expression naturelle du parallélisme d'actions, nous avons utilisé la sémantique de maximalité qui permet d'éviter les difficultés liées aux modèles entrelacés, et de conserver la nature parallèle des programmes concurrents.

Un model checker basé sur la logique temporelle CTL qui a été développé dans notre laboratoire, nous a permis grâce à la sémantique de maximalité d'exprimer de nouvelles propriétés portant sur l'exécution parallèle des actions, telles que la non compatibilité ou la compatibilité de deux actions et le degré d'autoconcurrence. En plus de ça, certaines propriétés classiques (de sûreté et de vivacité) ont pu être exprimées de manière plus naturelle et plus élégante. Dans notre travail, nous n'avons considéré que les propriétés qualitatives des applications temps-réel. Cela est rendu possible par l'abstraction du temps (après génération du modèle des DATA*'s) et l'application de l'algorithme proposé.

Le Chapitre 6 donne la spécification formelle d'un système réel à temps contraint qui se résume en un brûleur à gaz. Nous envisageons d'appliquer notre étude sur d'autres systèmes tels que les protocoles de communication, et/ou des exemples «académiques» comme le passage à niveau (Train-Gate-Controller).

Perspectives

Pour conclure, notre travail peut être considéré comme un premier pas pour le développement d'un environnement de vérification des systèmes temps-réel dont D-LOTOS est le langage de spécification.

Bien évidemment, dans la littérature il y a beaucoup d'autres formalismes de spécification

des systèmes à temps contraint qui ne sont pas considérés dans notre contexte, tels que la logique de réécriture [Mes92] qui a été utilisée dans le contexte temps-réel [Ölv00, ÖM04, DGLAM$^+$99], et le Duration Calculus [CHR91, Cha98]. Le choix de l'étude de modèles basés sur les automates temporisés est poussé par leur rapport d'expressivité/décidabilité, Or, comme nous l'avons déjà mentionné, une comparaison théorique du modèle des DATA*'s avec les modèles existants serait d'un intérêt capital pour pouvoir cerner le modèle le moins expressif mais le plus suffisant pour répondre aux besoins des applications temps-réel avec durées explicites associées aux actions.

En ce qui concerne la vérification, l'expression des propriétés quantitatives peut se faire dans une logique temporelle à la TCTL [ACD93]. Nous pensons que le passage vers la vérification des systèmes temps-réel peut se réaliser en prenant comme modèle sémantique les DATA*'s et en adaptant les algorithmes de model checking pour les systèmes temps-réel, utilisant notamment une logique comme la TCTL. Puisque nous exprimons les durées des actions de façon naturelle sans éclatement des actions, la vérification de propriétés quantitatives peut tirer profit de notre modèle pour vérifier par exemple la durée cumulée dans une trace d'exécution [ACH93].

En marge de ce travail, des algorithmes de vérification formelle basés sur les structures des BDDs sont en cours, dès lors, une représentation symbolique des DATA*'s serait envisageable. D'autres algorithmes traitent les aspects abstraction et réduction au niveau des STEMs. Le modèle des DATA*'s peut lui aussi être l'objet de techniques de réduction préservant la sémantique de maximalité. De telles mécanismes se baseront notamment sur les équivalences de bissimulation définies dans [SC03].

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.

[ACH93] R. Alur, C. Courcoubetis, and T. A. Henzinger. Computing accumulated delays in real-time systems. In *5th International Conference on Computer-Aided Verification (CAV'93)*, volume 818 of *LNCS*, pages 181–193. Springer-Verlag, 1993.

[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. Event-clock automata: A determinizable class of timed automata. In *Proc. 6th International Conference on Computer Aided Verification (CAV'94)*, volume 818 of *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.

[AL94] M. Abadi and L. Lamport. An old-fashioned recipe for real time. *ACM Transactions on Programming Languages and Systems*, 16(5):1543–1571, September 1994.

[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.

[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. *Vérification 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.

[BDGP98] B. Bérard, V. Diekert, P. Gastin, and A. Petit. Characterization of the expressive power of silent transitions in timed automata. *Fundamenta Informaticae*, 36(2-3):145–182, 1998.

[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.

[BGS04] H. Bowman, R. Gomez, and L. Su. How to stop time stopping (preliminary version). Technical Report 9-04, Computing Laboratory, University of Kent, Canterbury, Kent, CT2 7NF, UK, May 2004.

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

[Bou99] H. Boucheneb. *Conception et Analyse D'un Modèle Temporisé Unificateur À Base de Réseaux de Petri de Haut Niveau.* PhD thesis, USTHB, Alger, February 1999.

[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.

[BS05a] N. Belala and D. E. Saïdouni. Impact des durées d'actions sur les modèles temps-réel. Research report, Laboratoire LIRE, Université Mentouri, 25000 Constantine, Algérie, April 2005.

[BS05b] N. Belala and D. E. Saïdouni. Non atomicité dans les modèles temporisés. Research Report IPG05-001, Laboratoire LIRE, Université Mentouri, 25000 Constantine, Algérie, February 2005.

[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.

[Büc62] R. Büchi. On a decision method in restricted second-order arithmetic. In *Proc. Internat. Cong. On Logic, Methodology, and Philosophy of Science 1960*, pages 1–12. Stanford University Press, Stanford, 1962.

[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.

[CdS93] 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.

[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 Etiquetés Causaux: Une Nouvelle Approche pour la Description du Comportement Evénementiel 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.

[DGLAM+99] G. Denker, J. J. Garcia-Luna-Aceves, J. Meseguer, P. C. Ölveczky, Y. Raju, B. Smith, and C. L. Talcott. Specification and analysis of a reliable broadcasting protocol in Maude. In B. Hajek and R. S. Sreenivas, editors, *37th*

Annual Allerton Conference on Communication, Control, and Computation. University of Illinois, 1999.

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

[DS93] M. Diaz and P. Sénac. Time stream Petri nets, a model for multimedia streams synchronisation. In *Proceedings of Multimedia Modeling'93*. Singapore, 1993.

[DSB04] L. Derdouri, D. E. Saïdouni, and M. Benmohammed. Protocole hybride de transport actif multicast fiable. In *Proceedings of International Conference on Complex Systems (CISC'2004)*. University of Jijel, Algeria, September 6-8th 2004.

[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.

[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.

[HR98] J. Hillston and J. Ribaudo. Stochastic process algebra: A new approach to performance modeling. In *Modeling and Simulation of Advanced Computer Systems*. Gordon Breach, 1998.

[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.

[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.

[LL98] L. Léonard and G. Leduc. A formal definition of time in LOTOS - extended abstract. *Formal Aspects of Computing*, 10(3):248–266, 1998.

[Loh02] C. 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.

[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.

[Ölv00] P. C. Ölveczky. *Specification and Verification of Real-Time and Hybrid Systems in Rewriting Logic*. PhD thesis, Department of Computer Science, University of Bergen, Norway, Research done at the Computer Science Laboratory, SRI International, Menlo Park, USA, December 2000.

[ÖM04] P. C. Ölveczky and J. Meseguer. Specification and analysis of real-time systems using Real-Time Maude. In T. Margaria and M. Wermelinger, editors, *Fundamental Approaches to Software Engineering (FASE'2004)*, volume 2984 of *LNCS*, pages 354–358. Springer-Verlag, 2004.

[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 propriétés exprimées en CTL sur le modèle des systèmes de transitions étiquetées maximales. In *Conférence Internationale de la Productique (CIP'2003)*, Alger, October 2003. CDTA.

[SB03b] D. E. Saïdouni and N. Belala. Vérification de propriétés exprimées en CTL sur le modèle des systèmes de transitions étiquetées maximales (version étendue). 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):198–204, To Appear in July 2005.

[SBA04] D. E. Saïdouni, N. Belala, and A. Alidra. Formalismes de spécification des systèmes temps-réel. Technical report, Laboratoire LIRE, Université Mentouri, 25000 Constantine, Algérie, Juin 2004.

[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 *Ingénierie Des Protocoles (CFIP'2003)*. Hermes, France, 2003.

[SDAB04] D. E. Saïdouni, L. Derdouri, A. Alidra, and N. Belala. Conception formelle du protocole AMRHy. Technical report, Laboratoire LIRE, 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.

[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.

[YR98] T. Yoneda and H. Ryuba. CTL model checking of time Petri nets using geometric regions. *IEICE Trans. Inf. Syst.*, E99-D(3):1–11, March 1998.

Mémoire de Nabil Belala

Formalisation des systèmes temps-réel avec durées d'actions

Résumé : Les méthodes formelles sont de plus en plus utilisées pour répondre aux exigences auxquelles sont soumis les systèmes temps-réel. Ces méthodes reposent sur l'utilisation de modèles formels de spécification dotés de sémantiques rigoureuses et de techniques de vérification formelle. Parmi ces dernières, nous nous intéressons à l'approche de vérification logique exploitant le graphe de comportement afin de vérifier les propriétés qualitatives et/ou quantitatives requises du système. D'autre part, les sémantiques de vrai parallélisme, comme la sémantique de maximalité, conviennent à être employées lorsqu'on s'abstrait de l'hypothèse de l'atomicité temporelle et structurelle des actions ; des travaux antérieurs ont largement montré l'aptitude du modèle des systèmes de transitions étiquetées maximales (STEM) à prendre en considération et le comportement parallèle des systèmes et la prise en compte du temps. Parmi ces travaux, nous citons le langage D-LOTOS dont le but est la spécification des systèmes temps-réel.

Notre travail qui s'inscrit dans le cadre de la spécification et la vérification formelle des systèmes temps-réel consiste à définir un modèle sémantique temps-réel basé sur la sémantique de maximalité, exprimant les comportements parallèles et supportant à la fois contraintes temporelles, durées explicites des actions, non-atomicité structurelle et temporelle des actions et notion d'urgence, tout en proposant les règles de génération de manière opérationnelle de ce nouveau modèle (le modèle des DATA*'s) à partir de spécifications D-LOTOS. Enfin, nous montrons comment se servir de ce modèle dans une technique de vérification formelle, le model checking.

Mots-clés : Systèmes temps-réel, Durées d'actions, Spécification formelle, Vérification formelle, Langage D-LOTOS, Modèle des DATA*'s.

Formalization of real-time systems with action durations

Abstract: Formal methods are more and more used to fulfill real-time systems requirements. These methods rely on the use of formal specification models equipped with rigorous semantics and formal verification techniques. Among these last, we are interested in the logical approach of verification which operates on the behavior graph in order to check the required qualitative and/or quantitative properties. In addition, true concurrency semantics, like the maximality-based semantics, are appropriate to be used when one abstracts oneself from the temporal and structural actions atomicity hypothesis; former works largely showed the aptitude of the maximality-based labeled transitions system (MLTS) to take into account at once the true concurrent systems behavior and the consideration of time aspect. Among these works, we quote the D-LOTOS language of which the goal is the specification of real-time systems.

Our work which falls under the framework of formal specification and verification of real-time systems consists in defining a real-time semantic model based on the maximality-based semantics, expressing the true concurrent behaviors and supporting at the same time temporal constraints, explicit actions durations, structural and temporal not-atomicity of actions and actions urgency. We propose operational rules of generation of this new model (the DATA*'s model) starting from D-LOTOS specifications. Lastly, we show the use of this model in model checking technique.

Keywords: Real-time systems, Action durations, Formal specification, Formal verification, D-LOTOS language, DATA*'s model.