**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 : 2010/2011**
**N ° d'ordre :352/mag/2010**
**Série :006/inf/2010**

# Mémoire de Magister

**En vue de l'obtention du diplôme de Magister en Informatique**
**Option : Génie logiciel**

**Présenté par**

**Boumaza Amel**

**Dirigé par : Dr. SAIDOUNI Djamel-Eddine**

**Thème**

# Algorithmes de model checker opérant sur les DATA* (Durational Action Timed Automata)

**Membres du jury :**

| | | |
|---|---|---|
| **Dr. Kholladi M.Khireddine** | **Université Mentouri de Constantine** | **Président** |
| **Dr. Chaoui Allaoua** | **Université Mentouri de Constantine** | **Examinateur** |
| **Dr. Chikhi Salim** | **Université Mentouri de Constantine** | **Examinateur** |
| **Dr. Saidouni Djamel-Eddine** | **Université Mentouri de Constantine** | **Rapporteur** |

**Soutenu le : 07/11/2010**

*A ma famille...*

# Remerciements

Mes remerciements vont en premier lieu à ALLAH Tout Puissant qui a illuminé mon chemin de la lueur du savoir et de la science et pour la volonté, la santé et la patience qu'il m'a prodigué durant toutes ces années d'études.

Je tiens aussi à remercier le Docteur Djamel-Eddine Saïdouni pour avoir accepté de diriger ce travail. J'ai trouvé auprès de lui soutien et compréhension en toutes circonstances. Durant l'élaboration de ce travail, J'ai bénéficié sans cesse de sa grande expérience et de ses précieux conseils. Qu'il veuille trouver ici le modeste témoignage de ma gratitude et de mon profond respect.

Je tiens également à remercier vivement les membres du jury pour l'honneur qu'ils me font en acceptant de juger ce travail. Qu'ils trouvent ici l'expression de mon fidèle témoignage et l'assurance de mon profond respect.

J'aimerais également exprimer ma reconnaissance et gratitude à Mr Belala Nabil, pour son aide et orientation et précieuses remarques.

Enfin, mes remerciements vont particulièrement à Melle Guelleti souad et qu'elle trouve ici l'expression de ma profonde gratitude pour son soutient inconditionnel, tant sur le plan moral que sur le plan scientifique.

Que toutes les personnes que j'ai involontairement oubliées, trouvent ici, en cette heureuse et solennelle circonstance, l'expression de ma profonde gratitude et de mon indéfectible dévouement.

# Résumé

L'augmentation de la complexité des logiciels est incessante, ce phénomène est d'autant plus accentué lorsque des aspects temporels sont introduits, d'où la nécessité de méthodes de vérifications rigoureuses.

L'objectif de notre travail est de proposer un algorithme de vérification quantitative par modèle checking de propriétés exprimées en TCTL sur des systèmes temps-réel spécifiés dans un langage de haut niveau : D-LOTOS et dont le comportement est exprimé sous forme de systèmes étiquetés temporisés spécifiques à savoir le modèle des automates temporisés avec durées d'actions «DATA*», ces derniers supportent l'expression du comportement parallèle, la non-atomicité temporelle et structurelle et l'urgence.

Notre approche consiste à interpréter les comportements décrits par des DATA* à des Automates temporisés. L'environnement UPPAAL nous permettra par la suite la vérification de propriétés temporelles quantitatives, en particulier la vivacité bornée.

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

# Introduction

Avec l'accident du lanceur Ariane V, l'actualité récente nous remet une fois de plus en mémoire, et de manière spectaculaire, l'importance qu'a prise le logiciel dans le contrôle de systèmes variés, et par conséquent le prix que peuvent coûter des erreurs de spécifications, de conception ou d'implémentation de celui-ci. Or, la production de logiciel demeure une activité complexe, difficile à mettre en œuvre. Ce que nous rappelle cet accident, c'est la nécessité de plus en plus impérieuse de voir se développer une véritable ingénierie du logiciel, et aussi la difficulté d'y arriver.

Un nouveau facteur contribuant de plus en plus à cette complexité de la production du logiciel est la prise en compte du temps. Aujourd'hui la plupart des systèmes informatiques à construire comprennent l'aspect temporel. Cet aspect est rendu nécessaire par les tâches actuellement dévolues aux systèmes informatiques : Tâches de contrôles et tâches de communication. Cette caractéristique aujourd'hui inévitable rend la production de logiciel encore plus complexe, et donc, encore plus sujette aux erreurs.

En effet, là plus qu'ailleurs, se fait jour la nécessité de disposer d'outils aidant à la mise au point de systèmes corrects.

En particulier, l'activité de vérification devrait prendre de plus en plus d'importance. La vérification intervient en amont de la conception de systèmes. Etant donnée une spécification plus ou moins détaillée du système à réaliser, exprimée dans un langage possédant une sémantique formelle, il va s'agir de recenser les propriétés comportementales de la spécification, afin de détecter des erreurs précoces de conception. C'est dans ce cadre très général que va se situer notre travail. En effet, le terme «vérification» regroupe un ensemble de méthodes assez disparates suivant les modèles auxquels elles s'appliquent et les techniques mises en œuvre. Tout d'abord, nous nous intéressons à un langage temps réel : D-Lotos. L'intérêt de ce langage est la considération des contraintes temporelles et les durées associées aux actions. La sémantique de ce langage est exprimée à travers le modèle des DATA* (Durational Action Timed Automata).

D'autre part nous nous intéressons aux méthodes de vérification par modèle. Le premier objectif est alors de déterminer l'espace d'état du système étudié, c'est-à-dire l'ensemble des états auxquels le système peut accéder à partir de son état initial. Puis, la vérification s'applique à un graphe représentant le comportement du système, appelé le graphe d'accessibilité car ses sommets correspondent aux états accessibles, et ses arêtes aux évolutions

possibles du système passant d'un état à l'autre. D'autres approches opèrent sur des preuves des théorèmes sur le comportement du système. L'inconvénient de ce type de méthodes c'est qu'elles doivent être guidées par l'utilisateur pour obtenir des résultats. Il s'agit donc de méthodes semi-automatiques, en contraste avec les approches par modèle, qui nécessitent une intervention minimale de la part de l'utilisateur.

La vérification par modèle peut se diviser en deux grandes branches :

- Ou bien nous vérifions que le graphe d'accessibilité satisfait une certaine propriété, formalisée grâce à une logique adaptée. Nous parlons alors de model-checking. Les nombreuses logiques pouvant être utilisées sont regroupées sous le terme générique de logiques temporelles, car elles permettent d'exprimer des propriétés liées à la succession des états et/ou des actions dans le graphe d'accessibilité, ce qui correspond à l'évolution dans le temps du système.

- Ou bien nous vérifions que le graphe d'accessibilité satisfait une certaine relation, d'équivalence ou de pré-ordre, avec un graphe d'accessibilité plus petit dont nous savons qu'il vérifie une propriété donnée. Une relation très souvent employée est la bisimulation. Ce que nous prouvons alors, c'est que le système étudié présente un comportement acceptable.

Nous pouvons résumer l'activité de model-checking en trois phases : Une phase de modélisation permet de produire un modèle du comportement d'un système sous forme de graphe d'accessibilité. Parallèlement à cela, une phase de formalisation traduit une propriété à vérifier sur le système en une formule équivalente exprimée dans une logique temporelle. L'algorithme de model-checking nous permet de savoir si le graphe d'accessibilité satisfait bien un modèle de la formule donnée, c'est-à-dire de savoir si le système étudié vérifie bien la propriété qui nous intéresse.

Il est intéressant de distinguer entre deux grandes familles de logiques temporelles. Remarquons tout d'abord que, pour chaque état d'un graphe d'accessibilité, il existe a priori différents futurs possibles depuis cet état, constitués par les différentes séquences de transition tirables depuis cet état. Une première famille de logique temporelle permet de quantifier les propriétés vis-à-vis des futurs possibles : de distinguer entre le fait qu'il existe un futur vérifiant une propriété donnée, ou que tous les futurs possibles la vérifient. Nous parlons, alors de logique du temps arborescent. Si nous ne pouvons pas exprimer cette distinction, nous parlons de logique du temps linéaire.

Une autre distinction concerne le type d'algorithme de model-cheking mis en œuvre. Un premier type d'algorithme réalisera une exploration intégrale du système de transitions étudié, ce qui permettra ensuite de donner une réponse en ce qui concerne la satisfaction de la formule étudiée pour tous les états du graphe d'accessibilité. Nous parlons alors de model-checking global. Nous pouvons aussi se contenter d'une exploitation partielle, à partir d'un état donné, pour obtenir une réponse uniquement pour cet état : il s'agit alors de model-checking local.

Le principe du model-checking a été introduit par E. M. CLARKE, E. A. EMERSON et A. P. SISTLA en 1986 [CES86]. Depuis Il a été proposé pour de nombreuses logiques temporelles, et des algorithmes de model-checking sont largement disponibles.

De tels algorithmes commencent à être utilisés dans un contexte industriel, mêmes s'il s'agit toujours dune pratique marginale. L'utilisation d'outils de vérification semble en effet gagner du terrain et permettre de résoudre des problèmes de vraie grandeur, comme le montre une consultation effectuées lors de la conférence spécialisée Computer Aided Verification'96. Par exemple, le Concurrency Workbench a pu être utilisé pour formaliser et vérifier la correction de la phase de connexion du protocole UNI (version 3.0), présent dans les réseaux ATM [eS96]. De même, le Concurrency Factory a été appliqué à la vérification d'un protocole présent dans l'application de transfert de fichiers GNU UUCP [RC96].

D'autres exemples de programmes ont été formellement vérifiés par différents outils parmi lesquels nous trouvons un système d'échange ATM [DC96], un service de communication multipoint [eJMT96], un circuit de multiplication quatre étages [NB96] et l'arbitre du bus d'une architecture Bull [JCF96]. Précisons que cette liste n'est évidemment pas exhaustive.

**Problématique**

Les outils théoriques nécessaires à l'utilisation d'algorithmes de vérification par modèle sont maintenant bien établis dans le cas où le système étudié présente un graphe d'accessibilité fini. Ceci se comprend aisément. Dans le cas fini, les résultats de décidabilité sont en effet immédiats : pour décider de la satisfaction par une formule de logique temporelle ou de l'existence d'une relation similaire avec un autre graphe d'accessibilité, une exploration exhaustive suffira. Dans le cas fini, celle-ci peut être exécutée. Le problème sera alors de la réaliser de la manière la plus efficace possible, étant donné le problème de l'explosion combinatoire du nombre d'états accessibles.

Le problème qui va nous intéresser est le suivant : Quelles sont les possibilités de vérification lorsque nous souhaitons étudier un graphe d'accessibilité infini ?

Nous nous intéressons dans notre travail au modèle des DATA's, ce dernier 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 capturées par les durées correspondantes. Ce modèle permet 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.

L'extension temps-réel du modèle des DATA's a fait naître le modèle des DATA*'s. Ce modèle permet, 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 ainsi permet l'expression de la sémantique d'un langage temps-réel avec durées

d'actions, le langage D-LOTOS, ce qui fait des DATA* un modèle de base dans un environnement de vérification formelle des systèmes temps-réel ayant D-LOTOS comme langage de spécification.

L'approche que nous voulons optée pour la vérification formelle des systèmes à temps contraint, est basée sur les modèles (model checking). Dans notre cas, nous somme amenés d'abord à spécifier l'application à traiter dans le langage de spécification de haut niveau D-LOTOS. Cette spécification est ensuite traduite vers le modèle sémantique des DATA* sur lequel les propriétés attendues du système, exprimées dans la logique temporelle TCTL ou une de ses variantes, seront vérifiées à l'aide de model checkers. L'avantage principal de cette approche est qu'elle est en général complètement automatisable, sauf que, elle reste restreinte à des systèmes ayant un nombre fini d'états. Dû au fait que les valuations d'horloges produisent un nombre infini de configurations, le domaine du temps étant dense, pour un DATA ou un DATA*. Il est donc nécessaire d'utiliser des techniques particulières pour vérifier des propriétés classiques généralement décrites avec des logiques temporelles.

L'objectif de notre travail est de définir une structure fini à partir de l'espace d'états infini (tout en s'inspirant des travaux déjà existants pour les automates temporisés) généré par un DATA ou un DATA*. Une fois l'espace d'états est devenu fini, nous pouvons utiliser le modèle checker pour la vérification de propriétés qualitatives et/ou quantitatives.

### Contributions

Nos contributions peuvent se résumer en :

- L'étude du modèle checking TCTL proposé dans le cadre des automates temporisés. Ces modèles vont être confrontés à nos besoins de pouvoir vérifier par model checking des systèmes temps-réel souffrant de problème d'infinité d'espace d'états.
- L'interprétation des DATA* en automates temporisés.
- La proposition d'une approche de vérification de propriétés qualitatives et quantitatives sur des systèmes temps-réel via l'outil UPPAAL.

### Plan du document

Le Chapitre 1 considère la notion de durées d'actions dans l'algèbre de processus Basic LOTOS et le langage temps-réel D-LOTOS. La particularité de ce dernier étant la considération conjointe d'opérateurs exprimant les contraintes temporelles et l'attribution de durées aux actions. La sémantique de ces langages est exprimée à travers la sémantique de maximalité.

Le Chapitre 2 considère le modèle sémantique appelé DATA (pour Durational Action Timed Automata) ainsi que son extension pour la prise en compte des contraintes temporelles nommée DATA*. De manière similaire, la génération de DATA (respectivement DATA*'s) relatifs à des expressions de comportement Basic LOTOS avec durées d'actions (respectivement D-LOTOS) est exposée.

Le Chapitre 3 introduit les modèle des automates temporisés ainsi que les traveaux réalisés sur la vérification de propriétés quantitatives. Egalement une brève présentation de quelques outils de vérification des automates temporisées sont présentés notamment l'outil UPPAAL.

Le Chapitre 4 considère notre approche d'utilisation de l'outil UPPAAL pour la vérification de propriétés quantitatives sur les DATA*'s. Une étude de cas sur le Brûleur à Gaz est présentée.

# Table des matières

**Remerciements** ii

**Résumé** iii

**Introduction** iv

**1 Spécifcation formelle des systèmes temps réel avec durée d'action** 5
- 1.1 Sémantique de maximalité . . . 5
  - 1.1.1 Principe de la sémantique de maximalité . . . 6
  - 1.1.2 Sémantique de maximalité de Basic LOTOS . . . 7
- 1.2 Le langage D-LOTOS . . . 10
  - 1.2.1 Sémantique opérationnelle structurée de D-LOTOS . . . 11
  - 1.2.2 Spécification de la latence . . . 13
- 1.3 Conclusion . . . 15

**2 Modèles sémantiques pour les systèmes temps réel avec durée d'action** 16
- 2.1 Explicitation des durées d'actions . . . 16
  - 2.1.1 Intuition . . . 16
  - 2.1.2 Formalisation des DATA's . . . 18
  - 2.1.3 Construction opérationnelle des DATA's . . . 19
- 2.2 Modèle des DATA*'s . . . 23
  - 2.2.1 Intuition . . . 23
  - 2.2.2 Expression de l'urgence . . . 25
  - 2.2.3 Formalisation . . . 26
  - 2.2.4 Construction opérationnelle des DATA*'s . . . 27
- 2.3 Conclusion . . . 31

**3 Vérification formelle des systèmes temps réel** 32
- 3.1 Automates Temporisés . . . 32
  - 3.1.1 Définitions préliminaires . . . 32
  - 3.1.2 Présentation des automates temporisés . . . 33
- 3.2 Problèmes de décidabilité . . . 36

3.2.1 Description du problème . . . . . 36
3.2.2 Résultats de décidabilité . . . . . 36
3.2.3 Graphe des régions . . . . . 37
3.2.4 Automate des régions . . . . . 39
3.2.5 Bisimulation . . . . . 40
3.3 Vérification des systèmes temps réel . . . . . 40
3.3.1 Logiques temporelles quantitatives . . . . . 42
3.3.2 Algorithme d'analyse en avant . . . . . 45
3.4 Quelques environnements existants . . . . . 53
3.4.1 KRONOS . . . . . 53
3.4.2 HYTECH . . . . . 53
3.4.3 UPPAAL . . . . . 54
3.4.4 CMC . . . . . 54
3.4.5 Notre choix : UPPAAL . . . . . 54
3.5 Conclusion . . . . . 65

**4 Vérification des DATA*'s à la UPPAAL 67**
4.1 Interprétation des DATA*'s par des automates temprisés . . . . . 67
4.1.1 L'interprétation du parallélisme à l'aide d'un automate temporisé . . 69
4.1.2 L'interprétation de l'urgence à l'aide d'un automate temporisé : . . . . 70
4.1.3 L'interprétation de l'autoconcurrence à l'aide d'un automate temporisé 70
4.1.4 Formalisation du passage d'un DATA* vers un automates temporisé . 71
4.2 Exploitation d'UPPAAL pour la vérification des DATA*'s . . . . . 72
4.3 Etude de cas . . . . . 76
4.3.1 Spécification . . . . . 77
4.3.2 Vérification . . . . . 79
4.4 Conclusion . . . . . 80

**5 Conclusion et perspectives 82**
5.1 Perspectives . . . . . 83

# Table des figures


1.1 *Arbres de dérivation de E et F* . . . . . 7

2.1 *Comportement de S, utilisant les conditions de terminaison* . . . . . 17
2.2 *Exemple d'un DATA* . . . . . 19
2.3 *DATA* du comportement de S* . . . . . 24
2.4 *Expression de l'urgence par un DATA** . . . . . 25

3.1 *Exemple d'un automate temporisé* . . . . . 35
3.2 Graphe de région pour des contraintes $x \sim c(c_{max} = 3)$ et $y \sim c(c_{max} = 2)$ . . 38
3.3 L'ensemble de successeurs d'une région . . . . . 39
3.4 Automate temporisé $\mathcal{A}$ . . . . . 40
3.5 Automate des régions associé à $\mathcal{A}$ (Figure 3.4) . . . . . 41
3.6 *Vérification formelle* . . . . . 42
3.7 *Architecture d'un model checker* . . . . . 42
3.8 Zone représentant la formule $\varphi$ . . . . . 46
3.9 $(i)Z$ *et* $Z', (ii)Z \wedge Z', (iii)r(Z), r = \{x\}, (iv)Z \uparrow$ . . . . . 47
3.10 Automate temporisé dont le calcul en avant ne se termine pas . . . . . 48
3.11 Exemple d'une 2-approximation d'une zone . . . . . 49
3.12 Vu d'ensemble d'UPPAAL . . . . . 55
3.13 Editeur d'UPPAAL . . . . . 56
3.14 Simulateur d'Uppaal . . . . . 56
3.15 Vérificateur d'UPPAAL . . . . . 57
3.16 Architecture d'UPPAAL . . . . . 58
3.17 Fonctionnalités d'UPPAAL . . . . . 58
3.18 Les étiquettes dans UPPAAL . . . . . 59
3.19 Transmission de communication et Emplacement comité . . . . . 60
3.20 Exemple en UPPAAL : $x$ et $y$ sont des horloges, $n$ est un entier et $a$ est le canal. 61
3.21 Les formules de chemins sont supportées par UPPAAL. Les états en jaune indiquent ceux satisfaisants une certaine formule $\phi$ . . . . . 62
3.22 Le Train|-Barrière augmenté pour permettre la vérification de vivacité bornée. 65

4.1 DATA* représentant l'exécution de deux actions successives . . . . . 68

4.2 l'automate temporisé correspondant au DATA* de la Figure 4.1 . . . . . 69
4.3 L'addition de l'action $\delta$ . . . . . 69
4.4 L'automate temporisé correspondant au DATA* de la Figure 2.3 . . . . . 70
4.5 Automate temporisé exprimant l'urgence sur les actions . . . . . 71
4.6 Exécution d'actions séquentielles . . . . . 73
4.7 Augmentation du TA de la Figure 4.6 par *time* et $b$ . . . . . 74
4.8 Comportement parallèle . . . . . 74
4.9 Augmentation du TA de la Figure 4.8 . . . . . 75
4.10 Ecoulement du temps . . . . . 76
4.11 *Schéma primitif d'un brûleur à gaz* . . . . . 76
4.12 *Comportement simplifié d'un brûleur à gaz* . . . . . 77
4.13 *DATA* du brûleur à gaz* . . . . . 78
4.14 L'automate temporisé généré à partir du DATA* de la figure 4.13 . . . . . 79
4.15 L'augmentation de l'automate temporisé par l'hrloge *time* et le booléen $b$ . . 80
4.16 L'augmentation de l'automate temporisé par l'hrloge *time* et le booléen $c$ . . 81
4.17 Résultat de la vérification par UPPAAL . . . . . 81

# Chapitre 1

# Spécifcation formelle des systèmes temps réel avec durée d'action

La modélisation des systèmes temps réel nécessite des langages de spécification et des modèles sémantiques adéquats, dotés de mécanismes permettant la prise en compte des besoins tomporels qualitatifs et quantitatifs.

Dans ce but, il s'avère que l'intégration du temps pour la spécification des aspects temporels, par utilisation des opérateurs d'expression de contraintes temporelles ou par attribution de durées aux actions, soit importante, ce qui a donné naissance à de nombreuses extensions temporelles dont nous pouvant citer quelques unes [Bol91][BB91][NS91][CdS93b][Cd94][GRS95] [LL97].En revanche, cela reste insuffisant sans l'utilisation d'une sémantique de vrai parallélisme où l'hypothèse d'atomicité des actions soit levée.

Dans ce chapitre, nous allons nous intéresser à une approche intégrant à la fois les contraintes temporelles et les durées des actions dans le langage D-LOTOS pour lequel il a été défini une sémantique de vrai parallélisme, appelé sémantique temporelle de maximalité [SC03b]

## 1.1 Sémantique de maximalité

A l'inverse d'une sémantique de vrai parallélisme, comme la sémantique de maximalité, la sémantique d'entrelacement reste incompatible avec l'attribution de durées aux actions. En effet les deux expressions de comportement Basic LOTOS définies par $E = a; stop \;|||\; b; stop$ et $F = a; b; stop \;[]\; b; a; stop$ peuvent mettre en évidence cette incompatibilité.

Il est clair que ces deux expressions restent identiques tant que les durées des actions sont supposées nulles. Tandis que l'attribution de durées non nulles aux actions influence le temps d'exécution global qui sera égal à $\max\{durée(a), durée(b)\}$ pour l'expression $E$ et $durée(a) + durée(b)$ pour l'expression, $F$ d'où vient l'importance du choix du modèle sémantique lors de la levée de l'hypothèse d'atomicité temporelle.

### 1.1.1 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, la distinction entre exécutions séquentielles et exécutions parallèles d'actions devient possible

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 1.2.

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 1.1. Ces structures sont appelées des STEMs (pour *systèmes de transitions étiquetées*

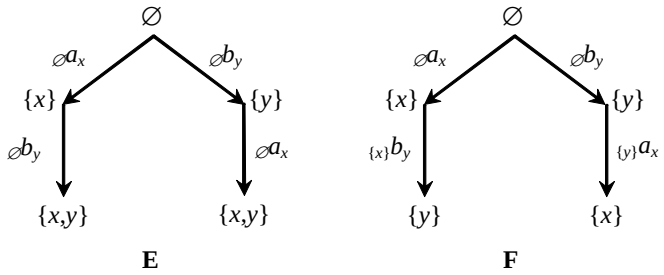


FIG. 1.1 – *Arbres de dérivation de E et F*

*maximales*).

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

### 1.1.2 Sémantique de maximalité 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 $Act$ ($Act = \mathcal{G} \cup \{i, \delta\}$).

**Définition 1.1** *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 1.2** *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* $\mathcal{F} \in \mathcal{C}$ $\qquad op \in \{ \; [] \; , \; ||| \; , \; || \; , \; |[L]| \; , \; [> \; \}$
- *si* $\mathcal{E} \in \mathcal{C}$ *et* $\{a_1, ..., a_n\}$, $\{b_1, ..., b_n\} \in 2^{\mathcal{G}}_{fn}$ *alors* $\mathcal{E}\,[b_1/a_1, ..., b_n/a_n] \in \mathcal{C}$

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

**Proposition 1.1** [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 1.3** *La fonction* $\psi : C \to 2^{\mathcal{M}}_{fn}$*, qui détermine l'ensemble des noms des événements dans une configuration, est définie récursivement par :*

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

**Définition 1.4** *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 :*

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

**Définition 1.5** *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* $\sigma$ *à* $x$ *sera écrite* $\sigma x$ *;*
- *La substitution identité* $\iota$ *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 $\sigma$ une fonction de substitution, la substitution simultanée de toutes les occurrences de $x$ dans $\mathcal{E}$ par $\sigma x$, est définie récursivement sur la configuration $\mathcal{E}$ comme suit :*

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

**Définition 1.6** *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. $\dfrac{}{{}_M[exit] \overset{M^{\delta}x}{\longrightarrow} {}_{\{x\}}[stop]}\ x{=}get(\mathcal{M})$

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

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

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

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

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

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

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

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

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

## 1.2 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 $\tau(i) = \tau(\delta) = 0$. $\tau_0$ est la fonction constante définie par $\tau_0(a) = 0$ pour tout $a \in Act$.

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

En prenant en compte la durée de l'action $a$, nous pouvons accepter la transition ${}_{\emptyset}[a; b; stop] \xrightarrow{\emptyset a_x} {}_{\{x:a:\tau(a)\}}[b; stop]$. La configuration résultante montre que l'action $b$ ne peut débuter son exécution que si une durée égale à $\tau(a)$ s'est écoulée, cette durée ne représentant rien d'autre que le temps nécessaire à l'exécution de l'action $a$. Nous 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 [CdS93a][CSLO00], la sémantique de ces opérateurs étant bien entendu exprimée dans notre contexte de maximalité. Du fait que les actions ne sont pas atomiques, les contraintes temporelles concernent dans ce contexte le début d'exécution des actions et non pas l'exécution complète des actions. Le langage ainsi défini est appelé *D-LOTOS*, pour LOTOS avec durées d'action.

La syntaxe de D-LOTOS est définie comme suit :

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

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

$t$ est une variable temporelle mémorisant le temps écoulé depuis la sensibilisation de l'action $g$ (resp. $i$) et qui sera substituée par zéro lorsque cette action termine son exécution.

**Définition 1.7** [1] *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* $\mathcal{E} \in \mathcal{C}_t$
- *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* $\{a_1, ..., a_n\}$, $\{b_1, ..., b_n\} \in 2_{fn}^{\mathcal{G}}$ *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 $Subs = \mathcal{M} \times Act \times \mathcal{D} \longrightarrow 2_{fn}^{\mathcal{M}\times Act\times\mathcal{D}}$. Par exemple $[y : a : 3/x : a : 4]x : a : 4 = y : a : 3$. L'extension aux configurations temporelles se déduit de manière directe.

### 1.2.1 Sémantique opérationnelle structurée de D-LOTOS

La fonction de durée $\tau$ 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 $Wait : 2_{fn}^{\mathcal{M}\times Act\times\mathcal{D}} \longrightarrow \{true, false\}$ défini sur tout $M \in 2_{fn}^{\mathcal{M}\times Act\times\mathcal{D}}$ 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 1.8.

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

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

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

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

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

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

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

**Les autres opérateurs**

La sémantique des opérateurs de délai, de choix, de composition parallèle, d'intériorisation, de séquencement, d'interruption, de renommage de portes et d'instanciation de processus est donnée par les règles 3a, 4(a)i, 4(a)ii, 4b, 5a, 5b, 6a, 7a, 7b, 7c, 8a et 9 de la Définition 1.6, dans lesquelles les configurations sont temporelles et la relation de transition est remplacée par $\rightarrow_\tau$, complétées par les règles suivantes :

$$\frac{}{\Delta^{d'+d}\mathcal{E} \xrightarrow{d}_\tau \Delta^{d'}\mathcal{E}} \qquad \frac{\mathcal{E} \xrightarrow{d} \mathcal{E}'}{\Delta^0\mathcal{E} \xrightarrow{d}_\tau \mathcal{E}'} \qquad \frac{\mathcal{E} \xrightarrow{M^a x}_\tau \mathcal{E}'}{\Delta^0\mathcal{E} \xrightarrow{M^a x}_\tau \mathcal{E}'}$$

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

$$\frac{P{:=}E \qquad {}_M[E] \xrightarrow{d}_\tau \mathcal{F}}{{}_M[P] \xrightarrow{d}_\tau \mathcal{F}}$$

$$\frac{\mathcal{E} \xrightarrow{d} \mathcal{E}' \qquad \forall d'{<}d.\ \mathcal{E}^{d'} \not\xrightarrow{a}_\tau \qquad \forall a{\in}L}{hide\ L\ in\ \mathcal{E} \xrightarrow{d}_\tau hide\ L\ in\ \mathcal{E}'} \qquad \frac{\mathcal{E} \xrightarrow{M^\delta x}_\tau \mathcal{E}'}{\mathcal{E}{>>}F \xrightarrow{M^i x}_\tau {}_{\{x:i:0\}}[F]}$$

$$\frac{\mathcal{E} \xrightarrow{d}_\tau \mathcal{E}' \qquad \mathcal{E} \not\xrightarrow{\delta}_\tau}{\mathcal{E}{>>}F \xrightarrow{d}_\tau \mathcal{E}'{>>}F} \qquad \frac{\mathcal{E} \xrightarrow{d}_\tau \mathcal{E}' \qquad \mathcal{F} \xrightarrow{d}_\tau \mathcal{F}'}{\mathcal{E}[{>}\mathcal{F} \xrightarrow{d}_\tau \mathcal{E}'[{>}\mathcal{F}'}$$

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

**Définition 1.8** *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 [SC03a].

### 1.2.2 Spécification de la latence

L'opérateur de latence de RT-LOTOS est d'un intérêt considérable pour la spécification de systèmes temps réel [Cd94, CdO95, CdOA95, Loh02, Sam03]. Il peut être introduit sans problème particulier dans le langage D-LOTOS. Mais dans [SC03a], 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» $\Omega$.

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 $\delta$ 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 **??** 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 :

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

## 1.3 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, nous nous sommes abstraits 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 1.1.

Nous notons également que dans [SC03a], 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.

# Chapitre 2

# Modèles sémantiques pour les systèmes temps réel avec durée d'action

Dans ce chapitre qui s'appuie en grande partie sur les résultats de [BS05], nous rappelons 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) ainsi que son extension DATA* qui prends en compte d'autres notions indispensables à la spécification des systèmes à temps contraint à savoir l'urgence, les délais, les contraintes, etc.

Pour spécifier des systèmes de plus en plus complexes 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 langage D-LOTOS répond bien à ce besoin.

## 2.1 Explicitation des durées d'actions

### 2.1.1 Intuition

Considérons l'exemple d'un système $S$ se résumant en deux sous-systèmes $S_1$ et $S_2$ s'exécutant en parallèle et se synchronisant sur une action $d$. Le sous-système $S_1$ exécute l'action $a$ suivie de $d$, tandis que $S_2$ exécute $b$ puis $d$ .

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. . 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.

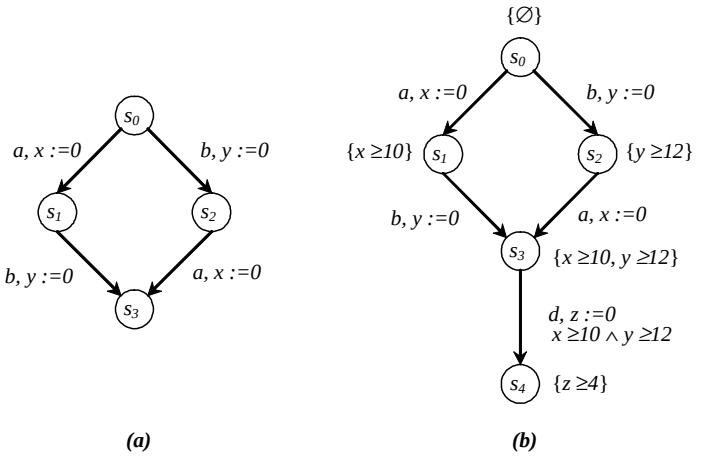


FIG. 2.1 – *Comportement de $S$, utilisant les conditions de terminaison*

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 2.1.(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.*

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.

### 2.1.2 Formalisation des DATA's

**Définition 2.1** *$\mathcal{H}$, parcouru par $x$, $y$... étant un ensemble d'horloges de valeurs dans $\mathbb{R}^+$. L'ensemble $\Phi_t(\mathcal{H})$ des contraintes temporelles $\gamma$ 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 $\gamma$ sur $\mathcal{H}$ ssi $\gamma$ 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 2.2** *La relation de satisfaction $\models$ pour les contraintes temporelles est définie sur l'ensemble des valuations des horloges de $\mathcal{H}$, par $v \models x \sim t \Longleftrightarrow v(x) \sim t$ tel que $v \in \Xi(\mathcal{H})$.*

**Définition 2.3** *Un Durational Action Timed Automaton (DATA) $\mathcal{A}$ est un quintuplet $(S, L_S, s_0, \mathcal{H}, T)$ tel que :*

- *$S$ est un ensemble fini d'états,*
- *$L_S : S \rightarrow 2_{fn}^{\Phi_t(\mathcal{H})}$ 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 $(s, \gamma, a, x, s')$ représente le passage de l'état $s$ à l'état $s'$, en lançant l'exécution de l'action $a$ et en réinitialisant l'horloge $x$. $\gamma$ est la contrainte correspondante, qui doit être satisfaite pour tirer cette transition. $(s, \gamma, a, x, s')$ peut être écrit $s \xrightarrow{\gamma, a, x} s'$.*

**Définition 2.4** *La sémantique d'un DATA* $\mathcal{A} = (S, L_S, s_0, \mathcal{H}, T)$ *est définie en lui associant un système infini de transitions* $\mathcal{S}_\mathcal{A}$ *sur l'alphabet* $Act \cup \mathbb{R}^+$. *Un état de* $\mathcal{S}_\mathcal{A}$ *(ou configuration) est un couple* $\langle s, v\rangle$ *tel que* $s$ *est un état de* $\mathcal{A}$ *et* $v$ *est une valuation sur* $\mathcal{H}$. *Une configuration* $\langle s_0, v_0\rangle$ *est initiale si* $s_0$ *est l'état initial de* $\mathcal{A}$ *et* $\forall x \in \mathcal{H}, v(x) = 0$. *Deux types de transitions entre les configurations de* $\mathcal{S}_\mathcal{A}$ *sont possibles, et qui correspondent respectivement au passage de temps (règle* RA*) et au tirage d'une transition de* $\mathcal{A}$ *(règle* RD*).*

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

**Exemple 2.1** *Soit* $\mathcal{A}$ *le DATA de la Figure 2.2. Un exemple de transitions possibles de la forme* $\langle s, v\rangle$ *de* $\mathcal{S}_\mathcal{A}$ *est :*

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

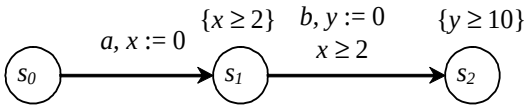


FIG. 2.2 – *Exemple d'un DATA*

### 2.1.3 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 1.1.2 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 2.1.1 qui se traduit en la composition parallèle de deux sous-systèmes $S_1$ et $S_2$ avec synchronisation sur la porte $d$. Le sous-système $S_1$ exécute l'action $a$ suivie de $d$, tandis que $S_2$ exécute $b$ puis $d$. En supposant que les durées respectives des actions $a$, $b$ et $d$ sont 10, 12 et 4, le comportement du système $S$ peut être exprimé par l'expression Basic LOTOS avec durées d'actions `E[a[10],b[12],d[4]] = a;d;stop |[d]| b;d;stop`.

Comme nous l'avons déjà vu, une structure de DATA comporte un ensemble fini d'états contenant les conditions sur les durées. Dans un contexte de génération d'un DATA à partir d'une expression de comportement d'une algèbre de processus, une *configuration d'un DATA*

est identifiée par un état du DATA ainsi qu'une sous-expression de comportement. C'est-à-dire que les configurations correspondantes aux états des DATA's vont être réécrites en fonction des conditions sur les durées sous la forme ${}_F[E]$ où $F \in 2^{\Phi_t(\mathcal{H})}_{fn}$.

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

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

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

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

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

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

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

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

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

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

Pour formaliser la génération opérationnelle des DATA's à partir de spécifications Basic LOTOS avec durées d'actions, les fonctions $\psi$, $\setminus$ et la fonction de substitution introduites dans la Section 1.1.2 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 $\psi$ 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 $\psi\left({}_M\left[E\right]\right) = M$ est remplacée par $\psi\left({}_F\left[E\right]\right) = \psi_{\mathcal{H}}\left(F\right)$, avec $\psi_{\mathcal{H}}$ donnée par la Définition 2.5.

**Définition 2.5** *La fonction $\psi_{\mathcal{H}} : 2^{\Phi_t(\mathcal{H})}_{fn} \rightarrow 2^{\mathcal{H}}_{fn}$, qui détermine l'ensemble des horloges utilisées dans un ensemble de conditions de terminaison, est définie récursivement par :*

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

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

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

Si $\mathcal{E}$ est une configuration d'un DATA ; $\mathcal{E}\backslash K$, qui dénote la configuration obtenue par la suppression de l'ensemble des conditions sur les durées $K$ écrites en fonction des horloges utilisées par l'ensemble $F$ de la configuration $\mathcal{E}$ reste la même, excepté l'équation $\left({}_F\left[E\right]\right)\backslash K = {}_{F\backslash K}\left[E\right]$ qui remplace $\left({}_M\left[E\right]\right)\backslash N = {}_{M-N}\left[E\right]$, avec $F\backslash K$ donnée dans la Définition 2.6.

**Définition 2.6** *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 :*

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

*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 $\left({}_M\left[E\right]\right)\sigma = {}_{M\sigma}\left[E\right]$ qui sera remplacée par $\left({}_F\left[E\right]\right)\sigma = {}_{F\sigma}\left[E\right]$. 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 $\{x \sim t\}\,\sigma = \{\sigma\left(x\right) \sim t\}$, et $\sigma$ est une fonction de substitution donnée par la Définition 1.5.

**Définition 2.7** *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 $\delta$ 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 1.6.

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.

## 2.2 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.

### 2.2.1 Intuition

Considérons le système $S$ décrit dans la Section 2.1.1. A partir de l'état $s_3$ de la Figure 2.1.(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*

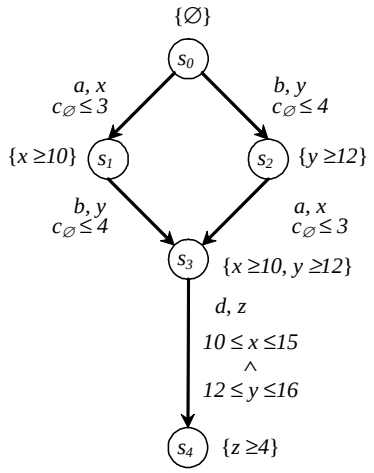


FIG. 2.3 – *DATA* du comportement de S*

résultant sera étiquetée par la contrainte $c_\emptyset \leq 3$ (c'est-à-dire $c_\emptyset \in [0,3]$). Cette contrainte, appelée *garde*, devra être satisfaite pour que l'action $a$ puisse s'exécuter. Si en plus, l'action $a$ est retardée d'un laps de temps égal à 1 (comme fait l'opérateur $\Delta^d$ de D-LOTOS), la garde sur la transition $a$ sera $1 \leq c_\emptyset \leq 4$ (c'est-à-dire $c_\emptyset \in [0+1, 3+1]$).

Le même raisonnement s'applique sur les autres actions. En admettant que les actions $a$ et $b$ du système $S$ sont offertes à l'environnement dans les quantités de temps respectives 3 et 4 depuis leurs sensibilisation, et que l'action $d$ est retardée dans le sous-système $S_1$ de 5 et dans $S_2$ de 4 unités de temps, le comportement global de $S$ est représenté par le DATA* de la Figure 2.3.

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

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

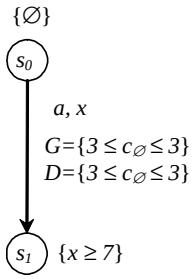


FIG. 2.4 – *Expression de l'urgence par un DATA**

### 2.2.2 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 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 2.3, 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 2.4. 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 $x$.

### 2.2.3 Formalisation

**Définition 2.8** *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 2.9** *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 $\eta$ 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*[2], 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 2.9 est rendue possible grâce à une hypothèse, garantissant qu'il y a un nombre fini d'actions dans un intervalle fini de temps. Cette

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

hypothèse largement acceptée (comme dans [ACD93, AD94, TAHY94][3]) 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.

**Propriété 2.1** *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 $\eta$ 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 $\epsilon$ 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 2.1** *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.

### 2.2.4 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 2.1.3 relatant les DATA's. Nous gardons les mêmes fonctions déjà évoquées dans la Section 2.1.3. 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 2.10** *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 :*

[3] Dans [TAHY94], cette hypothèse est appelée *finite variability condition.*

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

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

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

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

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

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

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

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

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 2.1** *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`*.*

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 2.10, 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 l'occurence de $a$), 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 $c_\emptyset \leq 3$. cela est exprimé par la règle 4a de la Définition 2.10.

La transition suivante est alors possible :

$$\underbrace{\emptyset\,[E]}_{config_0} \xrightarrow{G=\{c_\emptyset \leq 3\},D=\{\texttt{false}\},a,x} \underbrace{\{x \geq 20\}\,[b@t_2\,[t_2 \leq t_1]\,;\; stop]}_{config_1}$$

Dans la configuration $config_1$, $t_1$ renferme ainsi le laps de temps entre la sensibilisation (l'offre à l'environnement) et l'occurence (le début d'exécution) de l'action $a$. Le domaine de sensibilisation de $b$ est $t_2 \in [0, t_1]$, 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 $x \in [0 + 20, t_1 + 20]$. De manière plus générale, si nous avons plusieurs horloges dans la contrainte sur les durées $F$ de la configuration source, le décalage de l'intervalle implique toutes ces horloges, ce qui est formulé par la règle 4b. Dans notre exemple, la contrainte sur le tir de l'action $b$ n'implique que l'horloge $x$ avec un décalage de 20 unités de temps.

La transition suivante devient possible :

$$\underbrace{\{x \geq 20\}\,[b@t_2\,[t_2 \leq t_1]\,;\; stop]} \xrightarrow{G=\{20 \leq x \leq t_1+20\},D=\{\texttt{false}\},b,x} \underbrace{\{x \geq 7\}\,[stop]}_{config_2}$$

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 $\Delta$ est appliqué à une configuration $\mathcal{E}$, seule la *première* action tirée de la configuration $\mathcal{E}$ sera retardée.

**Remarque 2.2** *Notons que selon la Remarque 2.1, 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 peut se réduire en* $SP = \{\min \leq t \sim \max\} \mid \sim \in \{<, \leq\}$.

**Remarque 2.3** *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.*

## 2.3 Conclusion

Dans ce chapitre, nous avons rappelé le modèle des DATA's. 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 capturées par les durées correspondantes.

Ce modèle, qui reste très proche syntaxiquement du modèle des automates temporisés nous a apporté la possibilité de mieux exprimer des comportements parallèles ainsi que l'autoconcurrence, sous l'hypothèse de la levée d'atomicité structurelle et temporelle des actions, tout en évitant le problème d'explosion combinatoire. Et grâce à l'extension DATA*'s, d'autres concepts essentiels à la spécification des systèmes à temps contraint comme l'urgence ont pu être pris en compte.

Cependant, les horloges rendent l'ensemble des configurations infini sur lequel il n'est pas possible de faire d'évaluation algorithmique. Un problème qui a été rencontré également dans les automates temporisés.

# Chapitre 3

# Vérification formelle des systèmes temps réel

Dans ce chapitre, nous introduisons le modèle le plus couramment utilisé pour l'étude et l'analyse des systèmes temporisés, le modèle des automates temporisés. Nous définissons alors le modèle classique des automates temporisés tel qu'il a été introduit par Alur et Dill dans [AD94]. Par ailleurs, nous présentons quelques méthodes de vérification basées sur les modèles.

## 3.1 Automates Temporisés

### 3.1.1 Définitions préliminaires

**Définition 3.1** *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 l'instant 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 3.2** *Un mot temporisé sur un alphabet* $\Sigma$ *est un couple* $(a, t)$ *tel que* $a = a_1 a_2 ...$ *est un mot infini sur* $\Sigma$ *et* $t$ *est une séquence de temps. Un langage temporisé sur* $\Sigma$ *est un ensemble de mots temporisés engendrés par* $\Sigma$*.*

**Remarque 3.1** *Notons que les occurrences des symboles* $a_i$ *arrivent et terminent à l'instant* $t_i$*, entre autres, elles obéissent à l'hypothèse d'atomicité temporelle.*

**Définition 3.3** *Soit $H$ un ensemble d'horloges. L'ensemble des gardes sur $H$ est l'ensemble $\Phi(H)$ engendré par la grammaire suivante :*

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

*où $\delta_1$ et $\delta_2$ sont des gardes, $x \in H$, $c \in \mathbb{R}^+$ et $\sim \in \{=, <, >, \leq, \geq\}$. Une garde $\delta$ 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 $\delta$ la proposition devient vraie.*[1]

**Définition 3.4** *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, $\delta$ étant une garde :*

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

*où $\delta\left(v\left(x_i\right)/x_i\right)$ désigne la valeur booléenne de $\delta$ 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 $x \in H$.*

### 3.1.2 Présentation des automates temporisés

**Définition 3.5** *Un automate temporisé $\mathcal{A}$ est un quintuplet $(\Sigma, S, S_0, H, E)$ tel que :*

- *$\Sigma$ 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 $\delta$ est une contrainte temporelle sur $H$. $\langle s, s', a, \lambda, \delta \rangle$ peut être écrit $s \xrightarrow{a,\lambda,\delta} s'$.*

**Remarque 3.2** *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]*, les* graphes temporisés [Yov93][2]*, ou encore dans les* Timed Safety Automata [TAHY94]*. Dans* [AH92]*, les états des automates temporisés sont étiquetés par des* contraintes propositionnelles *sur un ensemble de propositions.*

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

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

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 3.3** *Par convention, la date $t_0$ correspond à la date de début d'exécution du mot temporisé, nous supposons 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[3].

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 3.6** *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 $\mathcal{A}$.*

**Définition 3.7** *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* $v' \models I(s')$.

[3]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 $i \geq 0$.

- $\langle s, v\rangle \xrightarrow{d} \langle s, v+d\rangle$ *si pour tout* $0 \leq d' \leq d$, $v + d' \models I(s)$ *avec* $d, d' \in \mathbb{R}^+$.

**Remarque 3.4** *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* $\mathbb{R}^+$*).*

Les automates temporisés sont souvent composés par un réseau sur un ensemble commun d'horloges et actions. Considérons un réseau de $n$ automates temporisés $\mathcal{A}_i = \langle \Sigma_i, S_i, s_{0i}, H, E_i, I_i\rangle$, $1 \leq i \leq n$. Un vercteur d'états $\overline{s} = (l_1, \cdots, l_n)$. Nous composons les invariants pour en obtenir un invariant commun sur les vecteurs d'états $I(\overline{s}) = \wedge_i I_i(s_i)$. Nous notons $\overline{s}[s'_i/s_i]$ le vecteur où l'$i$ème élément $s_i$ de $\overline{s}$ est remplacé par $s'_i$

**Définition 3.1** *soit* $\mathcal{A}_i = \langle \Sigma_i, S_i, S_{0i}, H, E_i, I_i\rangle, 1 \leq i \leq n$ *un réseau de* $n$ *automates temporisés. Soit* $\overline{s_0} = (s_{01}, \cdots, s_{0n})$ *le vecteur d'états initial. La sémantique est définie par un système de transition étiqueté* $S_A = (L, l_0, \rightarrow)$*, où* $L = (L_1 \times \cdots \times L_n) \times \mathbb{R}^H$ *est l'ensemble d'états* $l_0 = (\overline{s_0}, v_0)$ *est l'état initial et* $\rightarrow \subseteq L \times L$ *est la relation de transition définie par :*

- $(\overline{s}, v) \rightarrow (\overline{s}, v+d)$ *si* $\forall 0 \leq d' \leq d \Rightarrow v + d' \in I(\overline{s})$
- $(\overline{s}, v) \rightarrow (\overline{s}[s'_i/s_i], v')$ *s'il existe une transition* $s_i \xrightarrow{a,\lambda,\delta} s'_i$ *t.q.* $v \in \delta, v' = [\lambda \rightarrow 0]\, v$ *et* $v' \in I(\overline{s})$.
- $(\overline{s}, v) \rightarrow (\overline{s}[s'_i/s_i, s'_j/s_j], v')$ *s'il existe une transition* $s_i \xrightarrow{a!,\lambda_i,\delta_i} s'_i$ *et t.q.*$s_j \xrightarrow{a?,\lambda_j,\delta_j} s'_j$ $v \in (\delta_i \wedge \delta_j), v' = [\lambda_i \cup \lambda_j \rightarrow 0]\, v$ *et* $v' \in I(\overline{s})$.

Considérons l'exemple de la Figure 3.1. 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 : $\{((ab)^\omega, t)\,|\forall i.\ (t_{2i} < t_{2i-1} + 2)\}$.

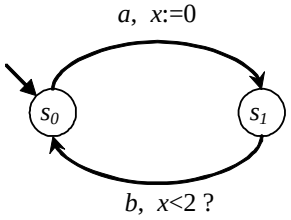


FIG. 3.1 – *Exemple d'un automate temporisé*

## 3.2 Problèmes de décidabilité

### 3.2.1 Description du problème

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 nous cherchons 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.*

### 3.2.2 Résultats de décidabilité

Dans la suite, pour une classe[4] de modèles, nous dirons que cette classe est décidable si le problème du vide est décidable pour cette classe. Le résultat fondamental démontré par Alur et Dill dans les travaux fondateurs [AD94] justifie l'intérêt important porté à ce modèle dans le domaine de la vérification de systèmes temporisés :

**Théorème 3.1** [AD94] *Le problème du vide est décidable pour la classe des automates temporisés.*

**Théorème 3.2** [AD94] *Tester le vide d'un langage accepté par un automate temporisé est un problème* PSPACE*-complet*[5]*.*

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.

**Théorème 3.3** [AD94] *Le problème universel*[6] *est indécidable pour les automates temporisés (*$\Pi_1^1$*-difficile*[7]*).*

Le corollaire suivant est une conséquence de ce théorème :

**Corollaire 3.1** [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 (*$\Pi_1^1$*-difficile).*

[4] Par classe de modèles, nous entendons une extension ou une restriction d'automates temporisés.

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

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

[7] La classe $\Pi_1^1$ constitue les problèmes extrêmement indécidables.

Le Corollaire 3.1 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.

Le résultat négatif du Corollaire 3.1 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 3.2 (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 $\langle s, s'', a, \lambda', \delta' \rangle$, nous avons $\delta \wedge \delta'$ qui est toujours faux.*

**Théorème 3.4** [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 tous ces résultats, le modèle des automates temporisés est largement utilisé pour la vérification des systèmes temporisés, notamment dans les approches basées sur le test du vide.

### 3.2.3 Graphe des régions

L'appartenance des horloges à des domaines infinis implique que l'automate produit à partir d'un automate temporisé donné possède une infinité d'états. Toutefois, les transitions purement temporelles, puisqu'elles se passent à l'intérieur d'un état normal, ne jouent pas, la plupart du temps, un rôle très effectif : seules celles qui déclenchent (ou peuvent déclencher) une transition d'état sont vraiment importantes. Nous allons donc pouvoir définir une équivalence entre états de l'automate produit qui sera d'indice fini, et l'automate quotient obtenu sera alors un automate fini sur lequel les techniques habituelles de vérification pourront s'appliquer.

Soit $\mathcal{A} = (\Sigma, S, s_0, H, E)$ un automate temporisé classique avec des constantes entières et des contraintes non diagonales. Pour toute horloge $x \in H$, nous notons $c_{\max}$ la plus grande constante $c$ apparaissant dans une contrainte d'horloges $x \sim c$ de $\mathcal{A}$.

Nous allons définir sur l'espace (infini) des valuations d'horloges sur $H$ une relation d'équivalence $\approx$ d'indice fini comme suit :

Deux valuations $v$ et $v\prime$ sont équivalentes, noté $v \approx v'$, si et seulement si elles satisfont les conditions suivantes :

- pour toute horloge $x \in H$, ou bien $v(x)$ et $v'(x)$ sont strictement supérieures à $c_{\max}$, ou bien $\lfloor v(x) \rfloor \approx \lfloor v'(x) \rfloor$[8],

[8] $\lfloor . \rfloor$représente la partie entière d'un réel alors que$\langle . \rangle$ représente sa partie fractionnaire .

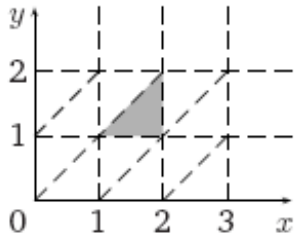


FIG. 3.2 – Graphe de région pour des contraintes $x \sim c(c_{max} = 3)$ et $y \sim c(c_{max} = 2)$

- pour toute horloge $x, x' \in H$,si à la fois $v(x)$ et $v(x')$ sont inférieures à $c_{\max}$, alors
  - $\langle v(x) \rangle \leq \langle v(x') \rangle$ ssi $v'(x) \rangle \leq \langle v'(x') \rangle$ et
  - $\langle v(x) \rangle = 0$ ssi $\langle v(x') \rangle = 0$.

Intuitivement, la première conditions expriment que deux valuations équivalentes satisfont exactement les mêmes contraintes de l'automate. Les deux dernières conditions expriment qu'à partir de deux configurations équivalentes, l'écoulement du temps permettra aux horloges d'atteindre de nouvelles valeurs entières dans le même ordre.

La relation $\approx$ est une relation d'équivalence. Elle vérifie la propriété suivante :

$$v \approx v' \Rightarrow \begin{cases} \text{pour toute contrainte } \delta \text{ de } A \ v \models \delta \Leftrightarrow v' \models \delta \\ \forall d \in \mathbb{R} \ \exists \ d' \in \mathbb{R} \text{ tel que } v + d \approx v' + d' \end{cases}$$

La première propriété ci-dessus indique que l'équivalence $\approx$ est compatible avec les contraintes de l'automate $\mathcal{A}$ alors que la deuxième propriété indique que l'équivalence $\approx$ est compatible avec l'écoulement du temps. Il est facile de voir que l'équivalence des régions est d'indece fini. Une classe de $\Xi(H) \diagup \approx$ est appelée une région. Nous notons $[v]$ la région contenant $v$.

**Exemple 3.1** *La Figure 3.2 représente l'ensemble des régions pour deux horloges dans le cas où seules des contraintes de la forme* $x \sim c$ *sont considérées*[9] *avec* $H = \{x,y\}, \sim \in \{=, \neq, <, \leq, >, \geq\}$ *et* $c_{\max} = 3$ *pour* $x$ *et* $2$ *pour* $y$.

Il y a alors 28 régions. Certaines sont réduites à une unique configuration (comme la région décrite par $x = y = 0$) d'autres sont des parties ouvertes du plan (comme le région décrite par la contrainte $0 < y < x < 1$), les autres sont des segments (comme la région décrite par la contrainte $0 < y = x < 1$).

[9] Uniquement les contraintes dites non diagonales,de la forme $x \sim c$, font l'objet de notre travail (une contrainte diagonale est de la forme $x - y \sim c$)

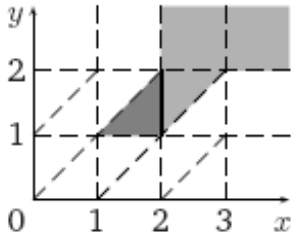


FIG. 3.3 – L'ensemble de successeurs d'une région

La région grisée est définie par la contrainte

$$1 < x < 2 \wedge 1 < y < 2 \wedge x > y$$

Le système démarre dans la région initiale $x = y = 0$. Lorsque le temps passe, nous accèdons à la région $0 < y = x < 1$, puis à la région $x = y = 1$ jusqu'à la région $x > 2 \wedge y > 3$. Si, au lieu de laisser passer le temps, nous effectuons une transition discrète, la remise à zéro de certaines horloges nous conduit aux régions situées sur les axes. Par exemple, la remise à zéro de $y$ dans la région $0 < y = x < 1$ nous conduit à la région $y = 0 \wedge x < 1$ d'où nous irons à la région $0 < y < x < 1$, etc.

**Lemme 3.1** *Le nombre de régions est borné par* $|S| \times |H|! \times 2^{|H|} \prod(2c_x + 2)$ *où* $c_x$ *est la constante maximale pour l'horloge* $x$*.*

### 3.2.4 Automate des régions

Nous avons vu que l'équivalence des régions est compatible avec l'écoulement du temps. Il est donc possible de définir une fonction successeur sur l'ensemble des régions. Si $R$ est une région, nous notons $Succ(R)$ l'ensemble des successeurs de $R$ pour l'écoulement du temps, c'est-à-dire l'ensemble de régions $Succ(R)$ vérifiant la propriété suivante :

$$R' \in Succ(R) \Leftrightarrow \exists v \in R. \exists t \in \mathbb{R} \text{ tels que } v + t \in R'$$

**Example 1** *Reprenons l'exemple* ***3.1.****Le premier successeur de la région en gris foncé est la région dessinée par une ligne épaisse noire. L'ensemble des autres successeurs de cette même région est dessiné en gris clair.*

Nous sommes maintenant en mesure de définir un automate fini $\mathcal{B} = (\Sigma, S', s'_0, H, E')$ de la manière suivante :

- $S' = S \times \Xi(H) \diagup \approx, s'_0 = (s_0, v_0)$,

- $E\prime = \left\{(s,R) \xrightarrow{a} (s',R') | \exists s \xrightarrow{a,\lambda,\delta} s' \in E \text{ et } \exists R" \in succ(R) \text{ tel que } R \subseteq \delta \text{ et } R' = R"[\lambda \leftarrow 0]\right\}$

Cet automate est appelé l'automate des régions associé à $\mathcal{A}$.

**Example 2** *([AD94]) Nous traitons un exemple de construction de l'automate des régions. Considérons l'automate $\mathcal{A}$ de la Figure **3.4** .*

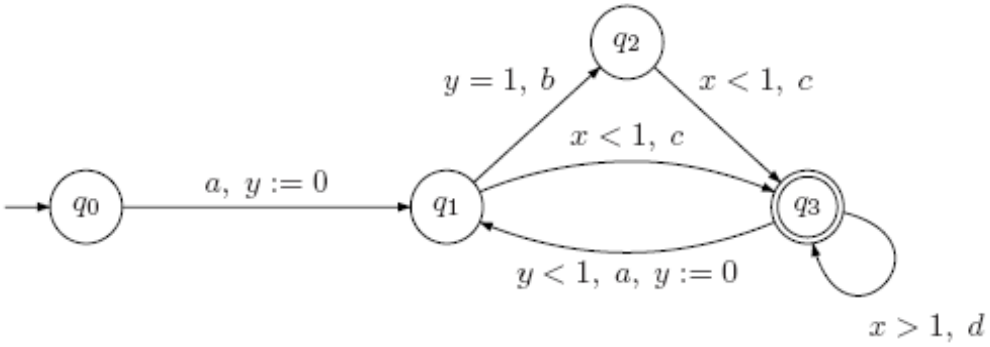


FIG. 3.4 – Automate temporisé $\mathcal{A}$

L'automate des régions que nous obtenons à partir de $\mathcal{A}$ en prenant la même valeur 1 pour les deux constantes maximales $c_{max}(x)$ et $c_{max}(y)$ est dessiné sur la Figure 3.5

L'automate des régions vérifie la propriété suivante :

**Propriété 3.1** *Le langage accepté par $\mathcal{B}$ correspond au Untime [10] du langage accepté par $\mathcal{A}$.*

### 3.2.5 Bisimulation

Une propriété de la relation d'équivalence $\approx$ est une bisimulation de temps abstrait :

- transitions d'action : si $w \approx v$ et $(s,w) \xrightarrow{a} (s',w')$ pour un certain $w'$, alors $\exists v' \approx w'$ t.q. $(s,v) \xrightarrow{a} (s',v')$.
- transitions de délai : si $w \approx v$ alors pour tout réel $d$, il exist $d'$ t.q. $w + d \approx v + d'$.

## 3.3 Vérification des systèmes temps réel

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 :

[10] Le *Untime* du mot temporisé $(\sigma,t)$ sur l'alphabet $\Sigma$ est le mot $\sigma$, obtenu par effacement de dates.

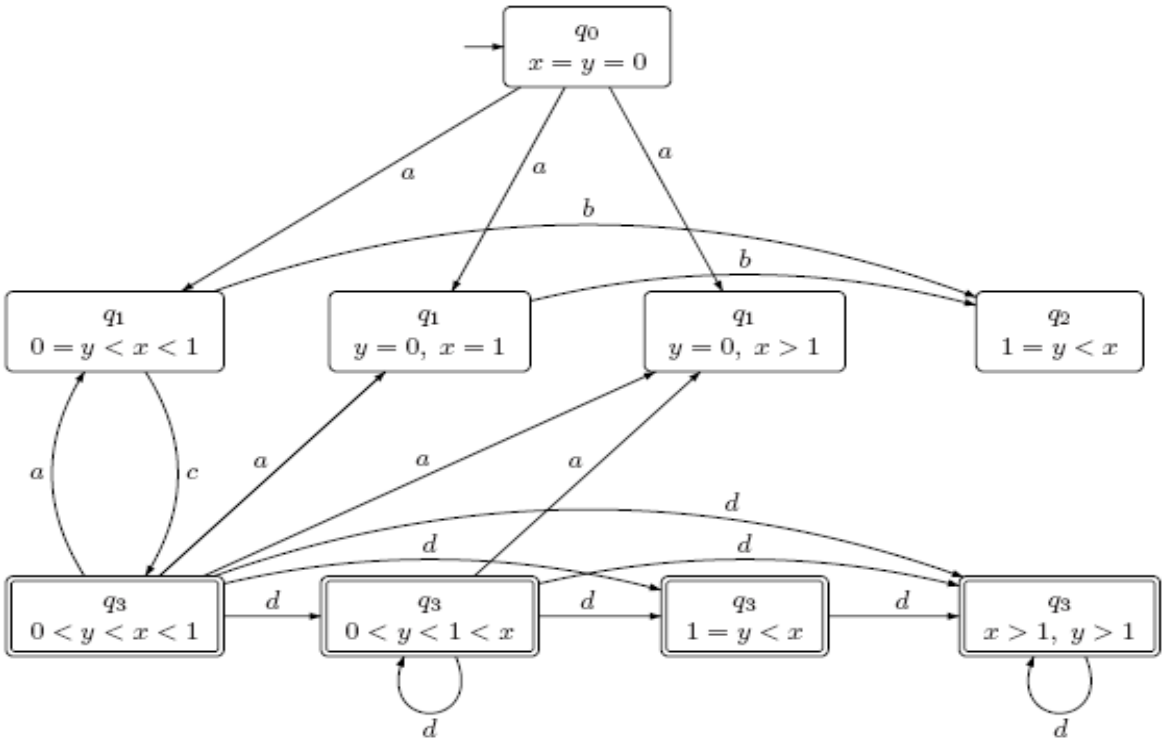


FIG. 3.5 – Automate des régions associé à $\mathcal{A}$ (Figure 3.4)

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. Cela est illustré par la Figure 3.6.

Une des techniques coramment utilisée est le modèle checking. Il consiste à vérifier si une 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.

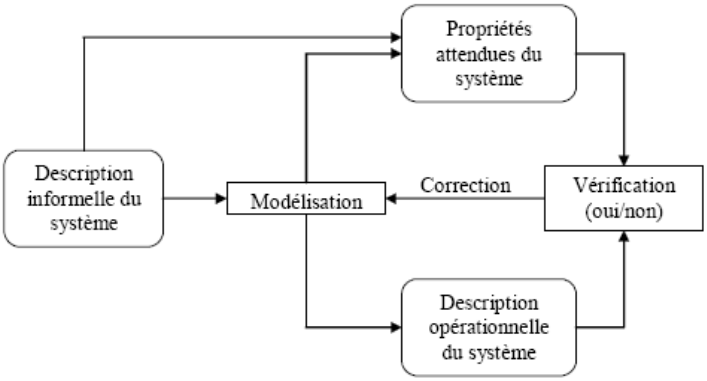


FIG. 3.6 – *Vérification formelle*

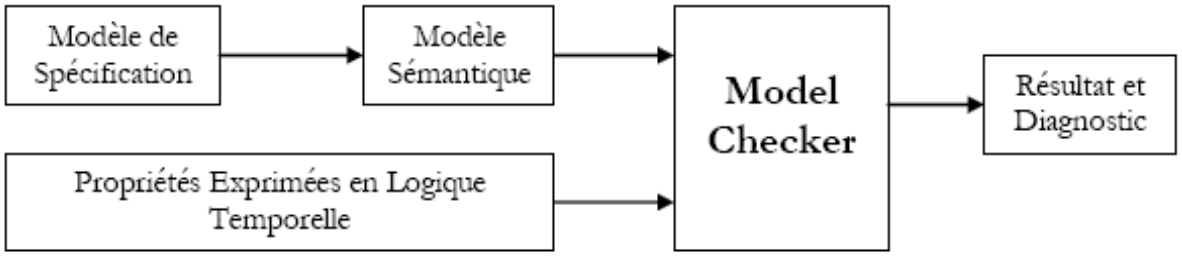


FIG. 3.7 – *Architecture d'un model checker*

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 3.7.

### 3.3.1 Logiques temporelles quantitatives

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 $\varphi$, et des formules sur les chemins, notées $\omega$. 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.

Pour intégrer des contraintes quantitatives dans les logiques temporelles, deux approches sont envisageables :

- rajouter des contraintes aux modalités classiques,
- rajouter des horloges appelées horloges de formule ainsi que des opérateurs pour les manipuler.

**Modalités avec contraintes quantitatives**

L'idée est de compléter l'opérateur temporel *Until* (noté **U**) avec une contrainte de la forme $\sim c$ avec $\sim \in \{=, <, \leq, \geq, >\}$ et $c \in \mathbb{N}$. La formule $\varphi U_{<c} \psi$ est vraie pour une exécution $\rho$ si et seulement si il existe un état $s$, situé à moins de $c$ unités de temps de l'état initial, et vérifiant $\psi$ et tel que tous les états précédents le long de $\rho$ vérifient $\varphi$. Il est aussi possible d'associer un intervalle $[a, b]$ à un opérateur **U**. Cette approche pour étendre les logiques temporelles est assez classique. Formellement, nous définissons la logique TCTL (Timed CTL) de la manière suivante :

**Définition 3.8** ***(Syntaxe de TCTL)*** *Nous ajoutons à la grammaire de CTL l'opérateur* $\mathbf{U}_{\sim c}$ :

$$\varphi, \psi ::= A\varphi \mathbf{U}_{\sim c} \psi | E\varphi \mathbf{U}_{\sim c} \psi$$

*avec* $\sim \in \{=, <, \leq, \geq, >\}$ *,* $c \in \mathbb{N}$.

Nous définissons aussi les abréviations courantes suivantes :$\varphi \vee \psi, \varphi \Rightarrow \psi, \ldots$ Nous utilisons la modalité **U** pour $\mathbf{U}_{\geq 0}$. Enfin, les contraintes $\sim c$ peuvent aussi s'utiliser dans les abréviations de CTL, nous avons donc :

- $EF_{\sim c}$ pour exprimer l'accessibilité de $\varphi$ dans un délai vérifiant $\sim c$.
- $AF_{\sim c}$ pour énoncer l'inévitabilité de $\varphi$ dans un délai vérifiant $\sim c$.
- $EG_{\sim c}$ le dual de $AF_{\sim c}$.
- $AG_{\sim c}$ le dual de $EF_{\sim c}$.

**Exemple 3.2** *Prenant l'énoncé suivant :*

*"L'alarme se déclenche au plus* 3 *secondes aprés l'apparition d'un problème"* (3.1)

La propriété énoncée ci-dessus s'écrit alors : $AG(\text{problème} \Rightarrow AF_{\leq 3}\text{alarm})$.

La sémantique des formules de TCTL est définie sur un état d'un système de transition temporisé :

**Définition 3.9** ***(Sémantique de TCTL)*** *Les clauses suivantes définissent la valeur de vérité des formules de TCTL sur un état $s$ d'un système de transition temporisé, noté* [11].

$s \models E\varphi U_{\sim c}\psi$ssi $\exists \rho \in Exec(s)$ avec$\rho = \sigma.\acute{\rho}$ et $s \xmapsto{\sigma} \acute{s}$ t.q.$Time(\sigma) \sim c, \acute{s} \models \psi$ et $\forall \acute{s} <_{\rho} \acute{s}, \acute{s} \models \varphi$

$s \models A\varphi U_{\sim c}\psi$ ssi $\forall \rho \in Exec(s) \exists \sigma$avec$\rho = \sigma.\acute{\rho}$ et $s \xmapsto{\sigma} \acute{s}, Time(\sigma) \sim c, \acute{s} \models \psi$ et $\forall \acute{s} <_{\rho} \acute{s}, \acute{s} \models \varphi$

Nous notons que l'opérateur $Next$ (noté $\mathbf{X}$) de CTL n'est pas présent dans cette définition de TCTL : en effet la notion de successeur immédiat n'est pas définie lorsque le modèle est de temps dense. Nous pouvons néanmoins ajouter cet opérateur lorsqu'il s'agit de temps discret.[12]

**Horloges de formule.**

Une autre méthode pour intégrer des aspects quantitatifs dans les logiques temporelles consiste à ajouter des horloges de formules (nous notons $H'$ l'ensemble de ces horloges) qui augmentent de manière synchrone avec le temps, un opérateur de remise à zéro ($\underline{in}$) et des contraintes simples $x \sim c$ ou $x - y \sim c$ avec$x, y \in H'$. La remise à zéro suivie, plus tard, d'une contrainte $x \sim c$ permet ainsi de mesurer le délai séparant deux états du système. Formellement, TCTL$_h$ est définie par :

**Définition 3.10** ***(Syntaxe de TCTL$_h$)*** *Les formules de $TCTL_h$ sont décrites par la grammaire suivante :*

$$\varphi, \psi ::= p | \neg\varphi | \varphi \wedge \psi | E\varphi U\psi | x \ \underline{in} \ \varphi | x \sim c | x - y \sim c$$

*avec* $\sim \in \{=, <, \leq, \geq, >\}$ *,* $c \in \mathbb{N}$*,* $x, y \in H$ *et* $p \in Prop$*.*

La sémantique des formules de $TCTL_h$ est définie sur un état d'un STT (Système de Transition Temporisé) et une valuation $v$ ($v : H \rightarrow \mathbb{R}$) pour les horloges de $H$. Etant donnée une valuation $v$ et $d \in \mathbb{R}$, nous notons $v + d$ la valuation qui associe la valeur $v(x) + d$ à chaque hologe $x \in H$ et $v[y \leftarrow 0]$ désigne la valuation qui associe à $y$ la valeur 0 et laisse les autres horloges inchangées par rapport à $v$.

**Définition 3.11** ***(Sémantique de TCTL$_h$)*** *Les clauses suivantes défiinissent la valeur de vérité des formules de $TCTL_h$ sur un état $s$ d'un système de transition temporisé et une valuation $v : H \rightarrow \mathbb{R}$, noté $s, v \models \varphi$.*

$s, v \models x \sim c$ ssi $v(x) \sim c$
$s, v \models x - y \sim c$ ssi $v(x) - v(y) \sim c$

[11] $Exec(s)$ sont les exécutions issues de $s, Time(\sigma)$ est la durée de temps associée à l'exécution $\sigma$.

[12] Nous nous contentons par la définition de l'opérateur $U_{\sim c}$, pour le reste des opérateurs, la sémantique est paraille à celle CTL.

$s, v \models x \ \underline{in}\ \varphi$ ssi $s, v\,[x \leftarrow 0] \models \varphi$

$s, v \models E\varphi U\psi$ ssi $\exists \rho \in Exec(s)$ avec $\rho = \sigma \cdot \rho$́et $s \overset{\sigma}{\longmapsto} s'$ t.q $.s', v + Time(\sigma) \models \psi$ et$\forall s' <_\rho s'$, t.q.$\rho = \sigma' \cdot \rho'$ et $s \overset{\sigma'}{\longmapsto} s'$ nous avons, $s', v + Time(\sigma') \models \varphi$

$s, v \models A\varphi U\psi$ ssi $\forall \rho \in Exec(s) \exists \sigma$ avec $\rho = \sigma \cdot \rho t.q.\ s \overset{\sigma}{\longmapsto} s', s', v + Time(\sigma \models \psi$ et $\forall s' <_\rho s'$,t.q. $s \overset{\sigma'}{\longmapsto} s'\, s', v + Time(\sigma') \models \varphi$

La propriété 3.1 s'écrit comme suit :

$AG$(problème $\Rightarrow \exists(x \ \underline{in}\ (AF(x \leq 3 \wedge$alarme$))))$.

L'opérateur $\underline{in}$ remet l'horloge $x$ à zéro lorsque nous rencontrons un état vérifiant *problème*, il suffit donc de vérifier que $x \leq 3$ lorsque nous rencontrons un état vérifiant *alarme* pour s'assurer que le délai séparant ces deux positions est inférieur à 3. Il est clair que, l'utilisation des horloges de formules permet d'exprimer tous les opérateurs de TCTL. Nous avons l'équivalence suivante lorsque $\varphi$ et $\psi$ sont des formules de TCTL :

$$E\varphi U_{\sim c}\psi \equiv x \ \underline{in}\ E\varphi U(\psi \wedge x \sim c)$$

La logique $TCTL_h$ permet d'exprimer des propriétés trés fines et était réputée être plus expressive que TCTL dans le temps dense, ce résultat a été récemment prouvé [CM05] : L'argument repose sur le fait que la formule suivante n'a pas d'équivalent en TCTL :

$$x \ \underline{in}\ EF(P_1 etx < 1 \wedge EG(x < 1 \Rightarrow \neg P_2))$$

Cette formule énonce le fait qu'il est possible d'atteindre un état vérifiant $P_1$ en moins de 1 unité de temps, à partir duquel il y a une exécution où il n'y a pas d'état vérifiant $P_2$ avant que $x$ ne valle 1. Nous pouvons aussi utiliser des *freeze variables* au lieu des horloges : une telle variable peut être instanciée avec la date associée à l'état courant, puis nous pouvons comparer la différence entre ces variables avec une valeur entière. Ces deux approches (freeze variables ou horloges de formule) sont équivalentes. L'ajout d'horloges peut aussi se faire dans des logiques de temps linéaire. Il existe notamment la logique TPTL [AH94] qui contient aussi des opérateurs du passé. Nous renvoyons à [AH92][ACD93][AH93][TAHY94] pour une présentation détaillée de plusieurs logiques temporelles quantitatives.

### 3.3.2 Algorithme d'analyse en avant

Il existe principalement deux algorithmes d'accessibilité à la volée : la recherche en avant, et la recherche en arrière. La recherche en avant (resp. en arrière) consiste à calculer tous les successeurs (resp. prédécesseurs) des configurations initiales (resp. finales). Rappelons que $\langle s, v \rangle$ est une configuration initiale si $s$ est un état initial et $v(x) = 0$ pour tout $x \in H$, et finale si $s$ est un état final.

L'algorithme d'analyse en arrière a de bonnes propriétés (terminaison, correction) mais souffre d'un défaut important : en pratique, les systèmes modélisés par des automates temporisés manipulent également des entiers, sur lesquels ils peuvent faire les opérations arithmétiques classiques $(+, -, \times, \div)$. Ces opérations sont aisément réalisables dans un calcul en

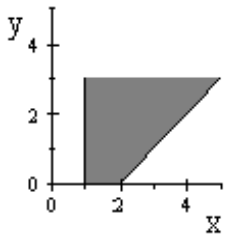


FIG. 3.8 – Zone représentant la formule $\varphi$

avant, mais au contraire elles sont difficiles à inverser : sachant qu'un entier $k = i \times j$ décrit un intervalle $[a, b]$, comment décrire les intervalles décrits par $i$ et $j$ ? Cette remarque justifie le fait que nous nous intéresserons désormais exclusivement à l'analyse à la volée en avant.

Voyons l'automate de région de la Figure 3.5. La région initiale contient un état unique $(s_0, x = y = 0)$ et il a trois successeurs avac l'état $s_1$ correspondant aux trois régions suivantes : $[y = 0 < x < 1], [y = 0, x = 1]$ et $[y = 0, x > 1]$. Une stratégie consiste à fusionner les trois régions pour obtenir l'union $[y = 0]$. De telles unions s'appellent zones.

En effet, la construction de l'automate des régions que nous avons vue dans la sous section 3.2.4 n'est pas utilisée en pratique car la notion de région est trop fine et rend les algorithmes inefficaces. Les outils utilisent les zones comme représentation symbolique et s'appuient sur des algorithmes à la volée.

Une zone est un ensemble de valuations définie par une contrainte d'horloges : la zone associée à la contrainte $\varphi$ est $\{v \in \Xi(H) \,|v \models \varphi\}$.

**Example 3** *La zone associée à la contrainte $\varphi = x > 1 \wedge y \leq 3 \wedge x - y < 2$ est illustrée par la Figure 3.8.*

Nous pouvons à présent décrire plus précisément le fonctionnement de l'algorithme de calcul en avant (Algorithme 3.1). Il consiste simplement, en partant de l'état initial (nous supposons qu'il est unique) $s_0$ de l'automate, et de la zone initiale $Z_0$ réduite à la seule valuation qui associe 0 à chaque horloge, à calculer les successeurs en un pas, selon l'opérateur $Post$, puis à itérer cette opération jusqu'à obtenir la stabilisation ou la découverte d'un état final.

**Algorithme 3.1** *Analyse en avant à la volée.*
***Entrée*** *: $\mathcal{A}$ un automate temporisé*
***Sortie*** *: $L(\mathcal{A}) = \varnothing$?*
$Visités := \varnothing$;
$En\_Attente := \{(s_0, Z_0)\}$;
***Répéter***
*Piocher $(s, Z)$ dans $En\_Attente$*;

***Si*** *s est un état final* ***Alors***
*Retourner "Un état final est accessible." ;*
***Sinon***
***Si*** *il n'existe pas* $(s, Z') \in Visités$ *tel que* $Z \subseteq Z'$ ***Alors***
$Visités := Visités \cup \{(s, Z)\}$ ;
$Successeurs := \{(s', Post(Z, e))|e$ *transition de* $s$ *à* $s'\}$;
$En\_Attente := En\_Attente \cup Successeurs$;
***Fin Si***
***Fin Si***
***Jusqu'à*** $En\_Attente = \varnothing$;
*Retourner "Aucun état final n'est accessible." .*

**L'opérateur** *Post*. Deux types d'évolutions sont possibles dans un automate temporisé : laisser écouler du temps en restant dans l'état courant et franchir une transition. Considérons un couple $(s; Z)$ ($s$ est un état et $Z$ une zone), et une transition $e$ de cet état $s$ vers un autre état $s\prime$.$Post(Z, e)$ retourne l'ensemble des valuations d'horloges accessibles à partir de celles appartenant à $Z$ en laissant écouler du temps dans $s$, puis en franchissant $e$. Plus précisément :

$$Post(Z, e) = \{v'|\exists(v, t) \in (Z \times \mathbb{R})|\, \langle s, v + t\rangle \xrightarrow{e} \langle s', v'\rangle\}$$

Ce calcul peut se décomposer en plusieurs étapes : notons $e = (s \xrightarrow{g,a,C} s'))$, alors nous avons $Post(Z, e) = [C \leftarrow 0](g \cap \overrightarrow{Z})$ où $\overrightarrow{Z}$ désigne le *futur de* $Z$, i.e.$Z = \{v + t|v \in Z$ et $t \in \mathbb{R}\}$.

Les opérations à réaliser sur les zones sont donc les suivantes : calcul du futur d'une zone, de l'instersection de deux zones, et remise à zéro au sein d'une zone de certaines horloges. Ces opérations, qui sont stables pour les zones (l'image d'une zone est une zone), sont illustrées sur la Figure 3.9

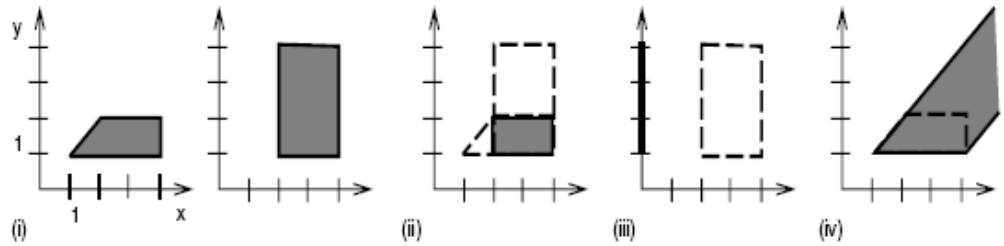


FIG. 3.9 – $(i)Z$ *et* $Z', (ii)Z \wedge Z', (iii)r(Z), r = \{x\}, (iv)Z \uparrow$

### Abstraction

Bien que l'utilisation des zones permet une représentation symbolique d'ensembles de valuations, le nombre d'objets manipulés par l'algorithme risque d'être infini. L'algorithme risque

donc de ne pas terminer. Cela est en fait possible, comme le montre le contre-exemple [13] représenté par la Figure 3.10.

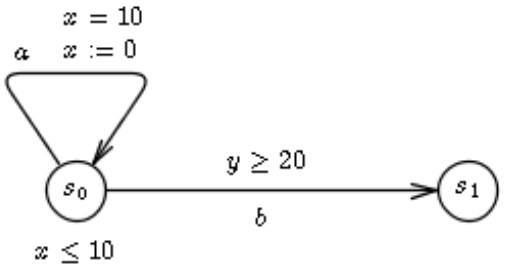


FIG. 3.10 – Automate temporisé dont le calcul en avant ne se termine pas

En effet, le temps s'écoule d'une façon non bornée pour l'horloge $y$ dans l'état $s_0$, générant un nombre infini d'états symboliques $(s_0, Z_i)$ :

$$
\begin{aligned}
Z_0 &= x \leq 10 \wedge x = y \\
Z_1 &= x \leq 10 \wedge y \leq 20 \wedge y - x = 10 \\
Z_2 &= x \leq 10 \wedge y \leq 30 \wedge y - x = 20 \\
Z_3 &= x \leq 10 \wedge y \leq 40 \wedge y - x = 30 \\
Z_4 &= x \leq 10 \wedge y \leq 50 \wedge y - x = 40 \\
&\vdots
\end{aligned}
$$

L'état $(s_0, Z_0)$ dénote que, initialement, l'automate peut rester dans l'état $s_0$ 10 unités de temps. L'état $(s_1, Z_1)$ est accessible à partir de $(s_0, Z_0)$ : $Z_1$ représente l'enseble de valuations accessibles dans l'état $s_0$ aprés l'éxecution de la transition $a$ (et laisser le temps passer tant que l'invariant le permet). L'état $s_0$ est contraint par l'invariant $x \leq 10$ sur l'horloge $x$. Cependant, aprés l'éxecution de $a$, la valeur de $y$ etla différence entre $x$ et $y$ augmentent : cela est parceque $a$ réinitialise $x$ et $y$ n'est pas controlée par un invariant. Par conséquent, une infinité de zones $Z_0, Z_1, \cdots$ sont accessibles, et ainsi l'algoritme d'accessibilité peut ne pas terminer (dans le cas, par exemple, où la propriété à vérifier nessécite l'exploration de tout l'espace d'état).

Afin d'assurer la terminaison de l'algorithme, une idée naturelle consiste à limiter les calculs à un nombre fini de zones. Nous pouvons donc utiliser une abstraction identifiant certaines zones. Pour cela, nous nous inspirons de la construction du graphe des régions qui tient essentiellement compte des valeurs inférieures à la constante maximale du système.

### Zone $k$-bornée et $k$-approximation

Une zone $Z$ est dite $k$-bornée si elle est définie par une contrainte d'horloges qui est $k$-bornée.

[13] Cet example est inspiré de [BY04]

La $k$-approximation de $Z$ est alors définie par :

$$Approx_k(Z) = \bigcap_{\text{zones k-bornées } Z' | Z \subseteq Z'} Z'$$


FIG. 3.11 – Exemple d'une 2-approximation d'une zone

Donc, pour obtenir la zone $Approx_k(Z)$ il suffit de suivre les étapes suivantes :

- Enlever toutes les contraintes atomiques de la forme $x \sim c$ et $x - y \sim c$ t.q $\sim \in \{<, \leq\}$ et $c > c_{\max}(\mathcal{A})$ (i.e., les bornes supérieures qui sont plus grandes que $c_{\max}(\mathcal{A})$ sont éliminées),
- Remplacer toutes les contraintes atomiques de la forme $x \sim c$ et $x - y \sim c$ t.q $\sim \in \{>, \geq\}$ et $c > c_{\max}(\mathcal{A})$ par $x > c_{\max}$ et $x - y > c_{\max}$ t.q $\sim \in \{>, \geq\}$ et $c > c_{\max}(\mathcal{A})$, respectivement (i.e., les bornes inferieures qui sont plus grandes que $c_{\max}(\mathcal{A})$ sont remplacer par $c_{\max}(\mathcal{A})$),
- Laisser le reste de contraintes atomiques.

Pour l'exemple de la Figure 3.10, les valeurs $v_2(y) \in [20, 30], v_3(y) \in [30, 40]$ et $v_4(y) \in [40, 50]$, t.q $v_2 \models Z_2, v_3 \models Z_3$ et $v_4 \models Z_4$. Il se trouve que la transition $b$ ne distingue pas entre toutes ces valeurs puisqu'elle est franchissable une fois $v(y) \geq 20$. Ainsi l'élimination de la contrainte $y \leq 30$ (respectivement $y \leq 40, y \leq 50$) de la zone $Z_2$ (respectivement $Z_3, Z_4$) produit une zone équivalente. De la même façon, remplacer les contraintes $y - x = 30$ et $y - x = 40$ par $y - x > 20$ dans $Z_3$ et $Z_4$ ne change pas ces zones. Nous avons, donc :

$Approx_{20}(Z_2) = x \leq 10 \wedge y - x = 20$

$Approx_{20}(Z_i) = x \leq 10 \wedge y > 20 \wedge y - x > 20, i \geq 3$

Le nouvel algorithme obtenu en utilisant cette abstraction est l'algorithme 3.2. Il consiste simplement, après chaque application de $Post$, à appliquer l'abstraction de sorte que les éléments stockés dans les listes "$Visités$" et "$En\_Attente$" soient en nombre fini. Remarquons que dans notre présentation de l'algorithme, nous avons placé la constante $k$ d'extrapolation en entrée, mais dans les outils, celle-ci est définie à partir de l'automate temporisé $\mathcal{A}$ pris en entrée, par exemple en prenant la constante maximale apparaissant dans $\mathcal{A}$.

**Algorithme 3.2** *Analyse en avant à la volée utilisant la k-approximation*

***Entrée*** *: $\mathcal{A}$ un automate temporisé et $k$ la constante d'extrapolation*

***Sortie*** *: $L(\mathcal{A}) = \varnothing$?*

$Visités := \varnothing$;
$En\_Attente := \{(s_0, Approx_k(Z_0))\}$ ;
***Répéter***
*Piocher $(s, Z)$ dans $En\_Attente$*;
***Si*** *$s$ est un état final* ***Alors***
*Retourner "Un état final est accessible." ;*
***Sinon***
***Si*** *il n'existe pas $(s, Z') \in Visités$ tel que $Z \subseteq Z'$* ***Alors***
$Visités := Visités \cup \{(s, Z)\}$ ;
$Successeurs := \{(s', Approx_k(Post(Z, e)))|e$ *transition de* $s$ *à* $s'\}$;
$En\_Attente := En\_Attente \cup Successeurs$;
***Fin Si***
***Fin Si***
***Jusqu'à*** $En\_Attente = \varnothing$;
*Retourner "Aucun état final n'est accessible." .*

### Structure de données pour l'analyse des systèmes temporisés

Les algorithmes vus jusqu'à présent utilisent tous la représentation symbolique des zones.

Dans ce qui suit, nous allons présenter une structure de données permettant d'implémenter facilement les zones. Nous vérifierons ensuite que toutes les opérations qui nous sont nécessaires sont aisément implémentables.

**Zones et DBMs** DBM. Une matrice à différences bornées (DBM) pour $n$ horloges est une matrice carrée de dimension $(n+1) \times (n+1)$ contenant des paires :

$$(\prec, m) \in \mathbb{V} = (\{<, \leq\}) \times \mathbb{Z} \cup \{(<, \infty)\}$$

Nous noterons les $(n+1)$ horloges $x_0, \cdots, x_n$. L'horloge $x_0$ sert seulement de référence, i.e.que c'est une "fausse" horloge dont la valeur est supposée toujours être 0. A une DBM $M = (\prec_{i,j}, m_{i,j})_{0 \leq i,i \leq n}$ est associé le sous-ensemble suivant de $\mathbb{R}^{n_H}_{0+}$[14] qui constitue une zone (avec la convention $v(x_0) = 0$) :

$$[\![M]\!] = \{v : \{x_1, \cdots, x_n\} \rightarrow \mathbb{R} \forall 0 \leq i, j \leq n, v(x_i) - v(x_j) \prec_{i,j} m_{i,j}\}$$

[14] Rappelons qu'une zone $Z \in Z(n_H)$ ,où $n_H$ est le nombre d'horloges,est un (peut être non bornée) polyhedre dans $\mathbb{R}^{n_H}_{0+}$ défini par une contrainte d'horloge

**Example 4** *La zone définie par les équations* $x_1 > 3 \wedge x_2 \leq 5 \wedge x_1 - x_2 < 4$ *peut être par exemple représentée par les deux DBM suivantes :*

$$\begin{pmatrix} (\leq,0) & (<,-3) & (<,\infty) \\ (<,\infty) & (\leq,0) & (<,4) \\ (\leq,5) & (<,\infty) & (\leq,0) \end{pmatrix} \; et \; \begin{pmatrix} (\leq,\infty) & (<,-3) & (<,\infty) \\ (<,\infty) & (\leq,\infty) & (<,4) \\ (\leq,5) & (<,\infty) & (\leq,0) \end{pmatrix}$$

**Forme normale**. Avec cette définition, plusieurs DBMs peuvent définir la même zone. Ceci pose problème car il n'est alors pas possible, étant données deux DBMs $M_1$et $M_2$, de tester de manière syntaxique si $[\![M_1]\!] = [\![M_2]\!]$. Une forme normale a donc été introduite afin de représenter sans ambiguïté les zones par des DBMs. La forme normale d'une DBM donnée est simplement l'unique DBM qui, parmi celles qui représentent la même zone, a les coefficients les plus petits. Cette DBM "minimale" représente alors les contraintes les plus fortes. Une fois définis un ordre et une addition sur les éléments de $\mathbb{V}$, nous pouvons utiliser un algorithme classique de plus court chemin, par exemple l'algorithme de Floyd, pour calculer la forme normale.

Ordre sur $\mathbb{V}$ : Si $(\prec, m), (\prec', m') \in \mathbb{V}$, alors

$$(\prec, m) \leq (\prec', m') \Leftrightarrow \begin{cases} m < m' \\ \text{ou} \\ m = m' \text{ et } \prec = \prec' \text{ ou } \prec = \leq \end{cases}$$

addition sur $\mathbb{V}$ : Si $(\prec, m), (\prec', m') \in \mathbb{V}$, alors $(\prec, m) + (\prec', m') = (\prec", m")$

$$\text{avec } m" = m + m', \prec " = \begin{cases} \leq \text{ si } \prec = \prec' = \leq \\ < \text{ sinon} \end{cases}$$

**Algorithme 3.3** *Algorithme de Floyd permettant de calculer la forme normale*

***Entrée :***$M = (M_{i,j})_{0 \leq i,j \leq n}$ *une DBM.*

***Sortie :***$M$ *en forme normale*

```
Pour i = 0 à n Faire
    Pour j = 0 à n Faire
        Pour k = 0 à n Faire
            M_{i,j} := min(M_{i,j}, M_{i,k} + M_{k,j})
        Fin Pour
    Fin Pour
Fin Pour
```

**Propriétés de la forme normale.** Nous noterons dans la suite $\Phi(M)$ la DBM sous forme normale associée à la DBM $M$. Elle vérifie les propriétés suivantes :

*Correction :*$[\![\Phi(M)]\!] = [\![M]\!]$

*Unicité :*$[\![M]\!] = [\![M']\!] \Rightarrow \Phi(M) = \Phi(M')$

*La forme normale représente les contraintes les plus fortes :*

$$\Phi(M) \leq M, \text{ i.e.} \forall 0 \leq i,j \leq n, \Phi(M)_{i,j} \leq M_{i,j}$$

*Inclusion :* $[\![M]\!] \subseteq [\![M']\!] \Leftrightarrow \Phi(M) \leq M' \Leftrightarrow \Phi(M) \leq \Phi(M')$

**Test du vide.** Nous disposons de la propriété suivante :

Soit $M = (\prec_{i,j}, m_{i,j})_{0 \leq i,j \leq n}$ une DBM (pas nécesserement sous forme normale), alors les propriétés suivantes sont équivalentes :

1. $[\![M]\!] = \varnothing$.
2. Il existe un cycle strictement négatif dans $M$, i.e. il existe une séquence d'indices distincts $(i_1, \cdots, i_{l-1})$ telle que (en posant $i_l = i_1$ )

$$(\prec_{i_1,i_2}, m_{i_1,i_2}) + \cdots + (\prec_{l-1,l}, m_{l-1,l}) < (\leq, 0)$$

3. $\Phi(M)$ contient un terme $(\prec, m)$ avec $m < 0$ ou $m = 0 \wedge \prec = <$ sur la diagonale.

**Opérations utilisées par l'algorithme** Nous devons enfin nous assurer que toutes les opérations dont a besoin l'algorithme sont aisément implémentables grâce aux DBMs.

**Futur.** Soit $M = (\prec_{i,j}, m_{i,j})_{0 \leq i,j \leq n}$ par :

$$\begin{cases} \left(\prec'_{i,j}, m'_{i,j}\right) = (\prec_{i,j}, m_{i,j}) \text{ si } j \neq 0 \\ \left(\prec'_{i,j}, m'_{i,j}\right) = (<, \infty) \text{ sinon} \end{cases}$$

Alors nous avons $[\![\overrightarrow{M}]\!] = \overrightarrow{[\![M]\!]}$ et de plus la DBM $\overrightarrow{M}$[15] est sous forme normale.

**Intersection.** Soient $M = (\prec_{i,j}, m_{i,j})_{0 \leq i,j \leq n}$ et $M' = \left(\prec'_{i,j}, m'_{i,j}\right)_{0 \leq i,j \leq n}$ deux DBMs (pas nécessairement sous forme normale).Définissons $M" = (\prec "_{i,j}, m"_{i,j})_{0 \leq i,j \leq n}$ par :

$$\forall i, j, (\prec "_{i,j}, m"_{i,j}) = \min\left((\prec_{i,j}, m_{i,j}), \left(\prec'_{i,j}, m'_{i,j}\right)\right).$$

Alors nous avouns $[\![M'']\!] = [\![M]\!] \cap [\![M']\!]$. Remarquons que nous ne pouvons pas assurer que $M"$ soit sous forme normale, même si $M$ et $M'$ l'étaient.

**Remise à zéro.** Soit $M = (\prec_{i,j}, m_{i,j})_{0 \leq i,j \leq n}$ une DBM sous forme normale. Définissons la DBM $M_{x_k:=0} = \left(\prec'_{i,j}, m'_{i,j}\right)_{0 \leq i,j \leq n}$ par :

$$\begin{cases} \left(\prec'_{i,j}, m'_{i,j}\right) = (\prec_{i,j}, m_{i,j}) \text{ si } i, j \neq k \\ \left(\prec'_{k,k}, m'_{k,k}\right) = \left(\prec'_{k,0}, m'_{k,0}\right) = \left(\prec'_{0,k}, m'_{0,k}\right) = (\leq, 0) \\ \left(\prec'_{i,k}, m'_{i,k}\right) = (\prec_{i,0}, m_{i,0}) \text{ si } i \neq k \\ \left(\prec'_{k,i}, m'_{k,i}\right) = (\prec_{0,i}, m_{0,i}) \text{ si } i \neq k \end{cases}$$

Alors nous avons $[\![M_{x_k:=0}]\!] = [x_k \leftarrow 0][\![M]\!]$ et de plus la DBM $M_{x_k:=0}$ est sous forme normale.

[15] Future de $M$ est noté par $\overrightarrow{M}$

**$k$-approximation.** Soit $M = (\prec_{i,j}, m_{i,j})_{0 \leq i,j \leq n}$ une DBM sous forme normale. Définissons la DBM $\overline{M}^k = \left(\prec'_{i,j}, m'_{i,j}\right)_{0 \leq i,j \leq n}$ par :

$$\begin{cases} \left(\prec'_{i,j}, m'_{i,j}\right) = (\prec_{i,j}, m_{i,j}) \text{ si } |m_{i,j}| \leq k \\ \left(\prec'_{i,j}, m'_{i,j}\right) = (<, \infty) \text{ si } m_{i,j} > k \\ \left(\prec'_{i,j}, m'_{i,j}\right) = (<, -k) \text{ si } m_{i,j} < -k \end{cases}$$

Alors nous avons $[\![\overline{M}^k]\!] = Approx_k([\![M]\!])$ mais la DBM $[\![\overline{M}^k]\!]$ n'est pas sous forme normale.

Nous avons ainsi tous les outils pour implémenter les algorithmes décrits en utilisant les DBMs pour représenter les zones. Les DBMs sous forme normale constituent donc une structure de données très satisfaisante. C'est d'ailleurs la structure de données de base utilisée dans la plupart des outils sur les systèmes temporisés.

## 3.4 Quelques environnements existants

Les méthodes décrites sont implémentées dans plusieurs environnements déjà utilisés avec succès dans l'industrie. Voici un panorama de certains environnements existants et de leurs fonctionnalités :

### 3.4.1 KRONOS

- Disponible à l'URL http : //www-verimag.imag.fr/TEMPORISE/Kronos.
- Développé par une équipe animée par Serge Yovine au laboratoire VERIMAG (Grenoble-France) dirigé par Joseph Sifakis.
- Permet de spécifier un réseau d'automates temporisés, puis de vérifier des propriétés de TCTL sur cet automate. C'est l'un des rares outils permettant de spécifier et vérifier des propriétés temporelles qui ne se ramènent pas à une propriété d'atteignabilité, comme la vivacité.
- Implémente l'analyse en avant et en arrière.

### 3.4.2 HYTECH

- Disponible à l'URL http : //www.eecs.berkeley.edu/ tah/Hytech.
- Développé par une équipe dirigée par Tom Henzinger à l'université de Berkeley (USA).
- Permet de spécifier des réseaux synchronisés de systèmes hybrides linéaires, pour lesquels la loi d'évolution de certaines variables réelles est régie par une équation différentielle linéaire. Il est le seul outil permettant aujourd'hui d'analyser des systèmes paramétrés aussi bien qu'hybrides. Les analyses sont limitées à des propriétés d'atteignabilité.

- Implémente l'analyse en avant et en arrière.

### 3.4.3 UPPAAL

- Disponible à l'URL http : //www.docs.uu.se/docs/rtmv/uppaal.
- Développé par P. Pettersson (université d'Uppsala en Suède) et Kim Larsen (université d'Alborg au Danemark), d'où son nom.
- Permet de spécifier un réseau d'automates temporisés synchronisés par messages, et en vérifier des propriétés d'atteignabilité. Son interface est de bonne qualité, comportant des outils graphiques.
- Implémente l'analyse en avant.

### 3.4.4 CMC

- Disponible à l'URL http : //www.lsv.ens-cachan.fr/∼fl.
- Développé par François Larroussinie (ENS-Cachan-France).
- Permet d'utiliser la structuration des spécification en un réseau d'automates temporisés communiquants pour éviter l'explosion du logiciel.Utilise la logique modale $L_\nu$.
- Implémente l'analyse en avant.

### 3.4.5 Notre choix : UPPAAL

Dans notre travail, nous nous intéressons particulièrement à UPPAAL comme environnement de description, simulation et vérification des systèmes temporisés.

Notre choix s'est porté sur UPPAAL [16] pour les raisons et caractéristiques suivantes :

- Cet outil est un logiciel libre.
- Convivial par son interface graphique.
- Facile à exploiter.
- Performance prouvée par multiples études de cas et exemples de référence [JBY96]. En terme de complexité, le Protocole Philips Contrôle d'audio avec collision est l'étude du cas la plus sérieuse où UPPAAL a été utilisé pour détecter et corriger plusieurs erreurs [BWDGY96].

[16] Notre choix sur UPPAAL n'exclue en aucun cas la possibiolité d'utilsation des autres environnements suscités et qui restent d'une performance relativement équivalente avec quelques variabilités dans les fonctionnalités proposées.

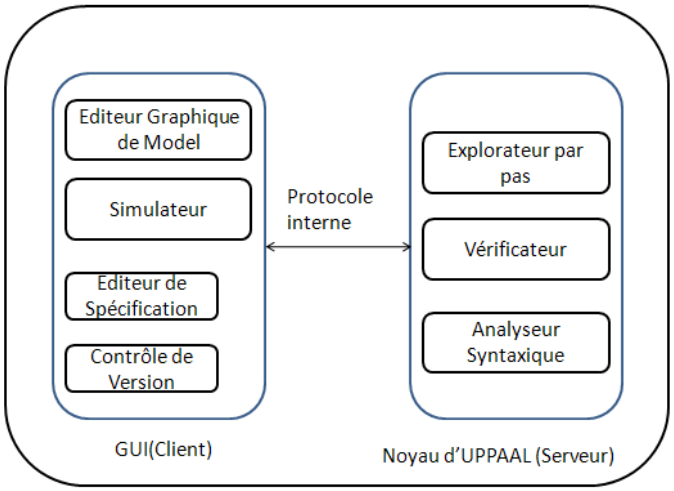


FIG. 3.12 – Vu d'ensemble d'UPPAAL

**Présentation**

UPPAAL [JBW98] est un ensemble d'outils pour la vérification automatique des propriétés de sûreté et de vivacité bornée, des systèmes temps réel. Il est construit avec l'architecture Client/Serveur (Figure 3.12). La machine (ou noyau) UPPAAL est le serveur, elle est développée en C++. L'interface graphique (GUI) est le client, développé en JAVA. La communication s'effectue via des protocoles internes.

Ainsi, la conception d'UPPAAL offre la possibilité d'exécuter le serveur et l'interface GUI sur deux machines différentes.

**Description** UPPAAL se compose de trois parties principales :

- Un éditeur,
- Un simulateur,
- Et un vérificateur de modèle.

L'éditeur permet la modélisation du système qui est défini comme un réseau d'automates temporisés, appelés processus, dans l'outil, mis en parallèle. Un processus est instancié d'une classe paramétrée (Template). L'éditeur est divisé en deux parties : un volet pour accéder aux différents Templates et déclarations et un canevas pour dessiner l'automate. L'arbre sur le côté gauche donne l'accès aux différentes parties de la description du système :

- Globale declaration : Contient des variables entières, des horloges, des canaux de synchronisation, et des constantes.
- Templates : Sont de différents automates teùmporisés paramétrés.

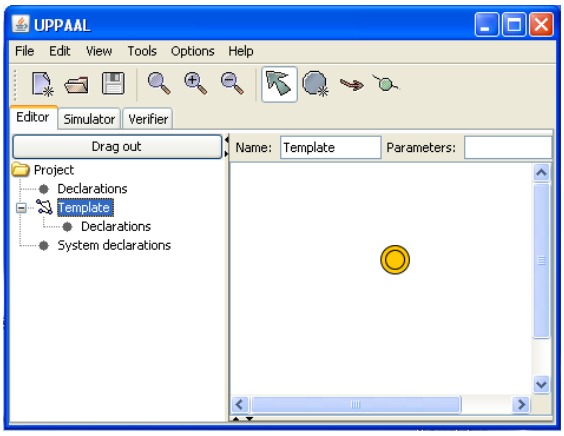


FIG. 3.13 – Editeur d'UPPAAL

Un Template peut avoir des déclarations locales (variables, canaux, et constantes).

- Process assignments : Les Templates sont instanciés en processus.
- System definition : La list des processus du system.

Le simulateur est un outil de validation qui permet l'examen dynamique des exécutions possibles d'un système au début de la conception (ou de modélisation). Il offre ainsi un moyen peu coûteux de détection des défauts avant la vérification exhaustive qui couvre le comportement dynamique complet du système.

Le simulateur peut être utilisé de trois façons : l'utilisateur peut exécuter le système manuellement et choisit la transition à prendre, le mode aléatoire où le système choisit les transitions à exécuter, ou l'utilisateur peut exécuter une trace (sauvgardée ou importée du vérificateur) pour voir comment certains états sont atteints.

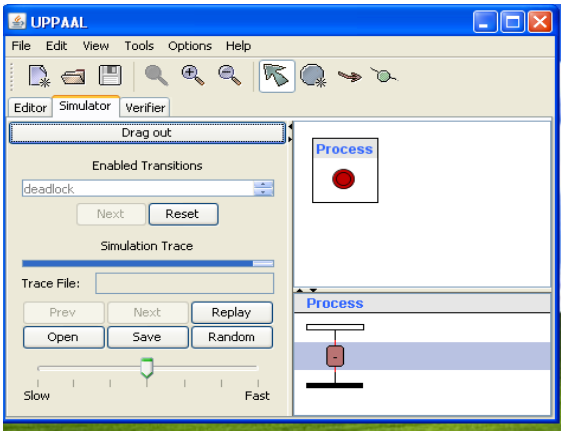


FIG. 3.14 – Simulateur d'Uppaal

Vérificateur : permet l'introduction d'une propriété CTL et la vérifie.

Quand l'option "génération de trace" est activée, elle sera importée au simulateur sous la demande de l'utilisateur. Les propriétés satisfaites sont marquées en vert et celles violées en rouge.

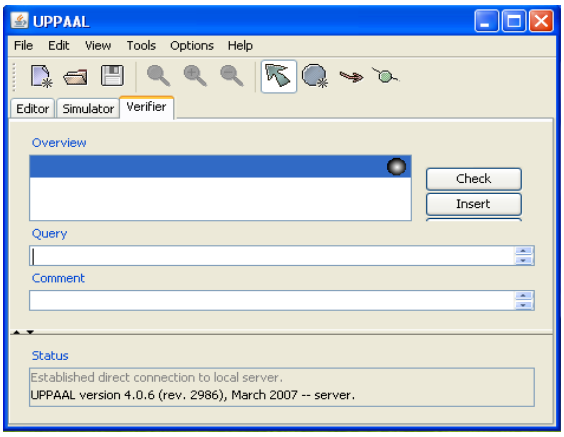


FIG. 3.15 – Vérificateur d'UPPAAL

**Architecture** Depuis son développement, UPPAAL [ADY03] a connu des améliorations en vue de supporter des fonctionnalités désirées qui sont de plus en plus complexes, et d'offrir la rapidité et la flexibilité. La nouvelle architecture est constituée par des couches qui peuvent communiquer entre elles.

**Fonctionnalité** La machine UPPAAL est constituée de plusieurs outils tel que checkta (Analyseur syntaxique) et verifyta (Vérificateur). L'interface graphique utilise des outils tel que atg2ta et hs2ta. La collaboration entre ces outils est illustrée par la Figure 3.17 [JBY95].

atg2ta : Ce programme est responsable de la transformation graphique des descriptions de Autograph [KGLY97] en format textuel.

Hs2ta : Ce programme est capable de transformer des systèmes hybrides simples en réseaux d'automates temporisés. Les systèmes considérés doivent être une sous classe de $LHS_{\neq 0}$, où la vitesse des variables est indépendante de l'état du système.

checkta : c'est un programme assurant la vérification syntaxique de la modélisation d'un système donnée en format textuel. La description doit être sous la forme d'automates temporisés. S'il s'agit de système hybride linéaire, il doit être d'abord transformé en système temporisé en utilisant hs2ta.

verifyta : Ce programme est le noyau de la vérification dans UPPAAL. Il reçoit en entrée la description du système et une propriété à vérifier. Il répond par "oui" ou "non". Il permet aussi de générer une trace qui confirme ou viole la propriété à vérifier.

Pour UPPAAL, un système temps réel typique est un réseau non-déterministe de processus séquentiels communiquant les uns avec les autres via des canaux. UPPAAL utilise les

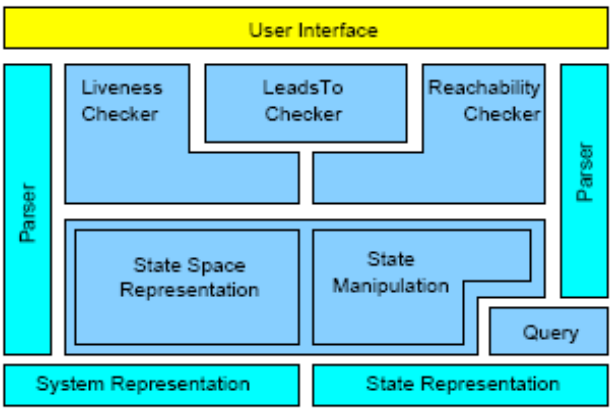


FIG. 3.16 – Architecture d'UPPAAL

automates à états-fini étendus avec des horloges et des variables de données pour décrire les processus et les réseaux d'automates relatifs au système temps réel traité.

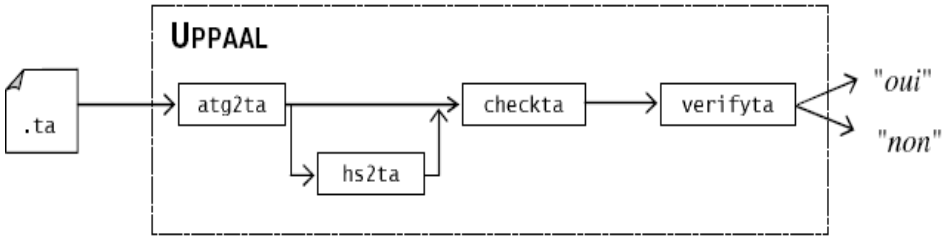


FIG. 3.17 – Fonctionnalités d'UPPAAL

### Syntaxe

La base du modèle d'UPPAAL est la notion des automates temporisés, développée par Alur et Dill [AD94], comme extension des automates à états-finis classiques avec les variables d'horloges et de données. Pour avoir une modélisation plus expressive et pour la rendre plus facile, les automates temporisés sont étendus avec des types plus généraux des variables de données tel que les variables booléennes et entières. Le but est le développement d'un langage de modélisation le plus proche possible d'un langage de programmation des systèmes temps réel de haut niveau. Clairement, ceci va créer des problèmes de décidabilité. Cependant, nous pouvons exiger que le domaine de valeurs des variables de données doive être fini afin de garantir la terminaison de la procédure de vérification.

Les transitions de l'automate temporisé, dans UPPAAL, sont étiquetées par trois types d'étiquettes (Figure 3.18) : (a) les gardes, exprimées sur les valeurs d'horloges et les variables entières qui doivent être satisfaites dans l'ordre pour les transitions qui vont être franchies ;

(b) une action de synchronisation qui est exécutée quand les transitions sont franchies ; et finalement (c) les attributions (assignement) des variables entières. Tous ces trois types sont optionnels.

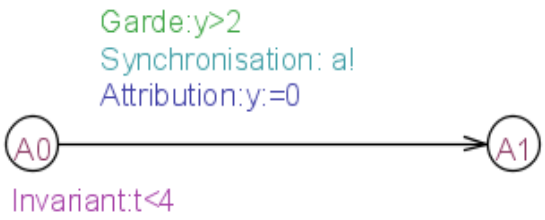


FIG. 3.18 – Les étiquettes dans UPPAAL

En plus, les noeuds de contrôles peuvent être étiquetés avec des invariants, qui sont des conditions exprimant des contraintes sur les valeurs d'horloges pour demeurer dans un noeud de contrôle particulier.

- Les gardes :

Les gardes expriment les conditions sur les variables d'horloges et des variables entières qui peuvent être satisfaites dans l'ordre de franchissement des transitions. Formellement, les gardes sont la conjonction des contraintes temporisées et des contraintes de données. Une contrainte d'horloge est de la forme : $x \tilde{} n$ ou $x - y \tilde{} n$, où $n$ est un nombre naturel et $\tilde{} \in \{\leq, \geq, =, >, <\}$. Une contrainte de données est de la même forme, $i \tilde{} j$ ou $i - j \tilde{} k$, sauf que $k$ est un entier arbitraire. La garde d'une transition est par défaut vraie.

Dans la Figure 3.18, la transition entre $A0$ et $A1$ n'est franchie que si la valeur de l'horloge est supérieure ou égale à 2.

- Opération de remise à zéro :

La remise à zéro d'une horloge ou d'une variable de données sur une transition est une affectation de la forme $w := e$, sachant que $w$ est une horloge ou variable de donnée et $e$ une expression.

L'opération de remise à zéro d'une horloge est de la forme $x := n$, où $n$ est un nombre entier. La remise à zéro des variables de données est de la forme $i := c * i + c\prime$, où $c, c\prime$ sont des constantes entières (sachant que$c$ et $c\prime$ peuvent être nuls ou négatifs).

- Canaux, synchronisation et urgence :

Un modèle UPPAAL consiste en un réseau d'automates temporisés (étendu). Les automates peuvent communiquer via des variables entières (dans UPPAAL elles sont globales) ou en utilisant des canaux de communication, permettant l'envoi et la réception d'un message. La

communication sur un canal apparaît comme étant la synchronisation entre deux processus.$a!$ (Envoie) et $a?$ (Réception) dénotent qu'un processus synchronise avec un autre. L'absence de l'action de synchronisation dénote une transition (de non synchronisation) interne. Pour empêcher un automate de s'attarder dans une situation, où deux composants sont capables de synchroniser, il faut déclarer le canal comme étant urgent. Pour des raisons d'efficacité, les transitions sont étiquetées avec des actions de synchronisation sur des canaux urgents, et elles doivent être privées des gardes sur les horloges pour que les transitions de deux processus puissent être franchies simultanément et ne pas bloquer un système synchronisé avec un autre.

- Emplacement comité (committed location) :

Soit un émetteur $S$ qui émet un message $m$ à deux destinataires $R1$ et $R2$ (schématisés par la Figure.3.19). La synchronisation entre ces trois processus ne peut pas être directement exprimée en UPPAAL. Elle ne peut être effectuée qu'entre deux processus uniquement. Cependant, l'émission sera modélisée sous la forme d'une séquence de synchronisation entre deux processus, où $S$ synchronise avec $R1$ sur $m1$ puis avec $R2$ sur $m2$. Pour assurer l'atomicité de la synchronisation. Nous marquons le noeud intermédiaire comme comité (indiqué par C).

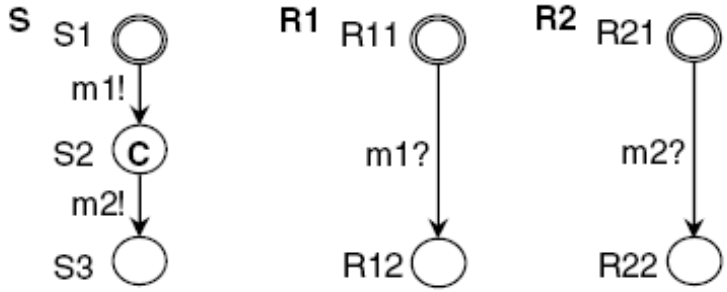


FIG. 3.19 – Transmission de communication et Emplacement comité

- Invariant :

Pour renforcer la progression dans le temps, les noeuds de contrôles doivent être étiquetés par des invariants, qui expriment dans l'ordre les contraintes sur les valeurs d'horloges pour le noeud de contrôle, afin qu'il puisse y résider.

**Sémantique**

Formellement, les états du modèle UPPAAL sont de la forme $(l, v)$, où pour chaque composant du réseau d'automate, $l$ est un vecteur de contrôle indiquant le noeud de contrôle en cours, et pour chaque horloge et variable entière $v$ est un assignement indiquant la valeur en cours.

- Transition de délai :

Le temps doit évoluer, sans l'affectation des vecteurs des noeuds de contrôles et en incrémentant les valeurs des horloges avec la durée du temps écoulé, aussi longtemps qu'aucun invariants des noeuds de contrôle n'est violé.

- Transition d'action :

Si deux transitions complémentaires de deux composants différents sont rendues actives, dans un état, alors elles peuvent se synchroniser.

- Canaux urgents :

Aucun délai n'est permis dans un état, où deux composants doivent synchroniser sur un canal urgent.

Ainsi, dans la Figure 3.20, si un canal $a$ est urgent, alors le temps ne doit pas s'écouler pendant 3,5 secondes à partir de l'état initial $((A0, B0), x = 0, y = 0, n = 0)$ puisque la synchronisation sur $a$ est possible dans l'état $((A0, B0), x = 3, y = 3, n = 0)$.

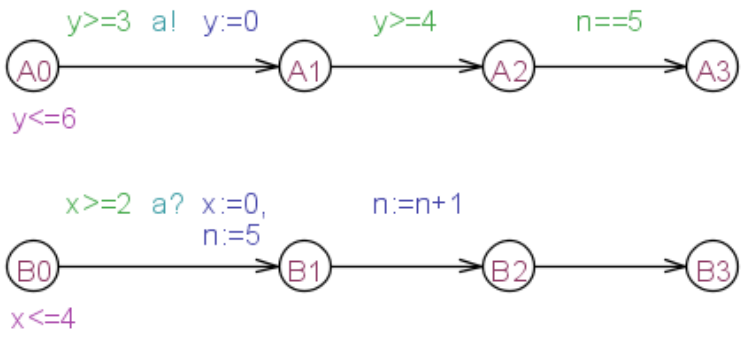


FIG. 3.20 – Exemple en UPPAAL : $x$ et $y$ sont des horloges, $n$ est un entier et $a$ est le canal.

### Vérification dans UPPAAL

**Logique** Le but principal d'un model checker est de vérifier sur le modèle une propriété . Comme le modèle, la spécification de la propriété doit être exprimée formellement dans un language de spécification . Plusieurs logiques existent dans la littérature ; UPPAAL utilise une version simplifiée de CTL.

Comme dans CTL, le langage comporte la notion de formule de chemin et formule d'état Les formules d'état décrivent des états individuels, alors que les formules de chemin concernent tous les chemins ou traces du modèle. Les formules de chemin peuvent être

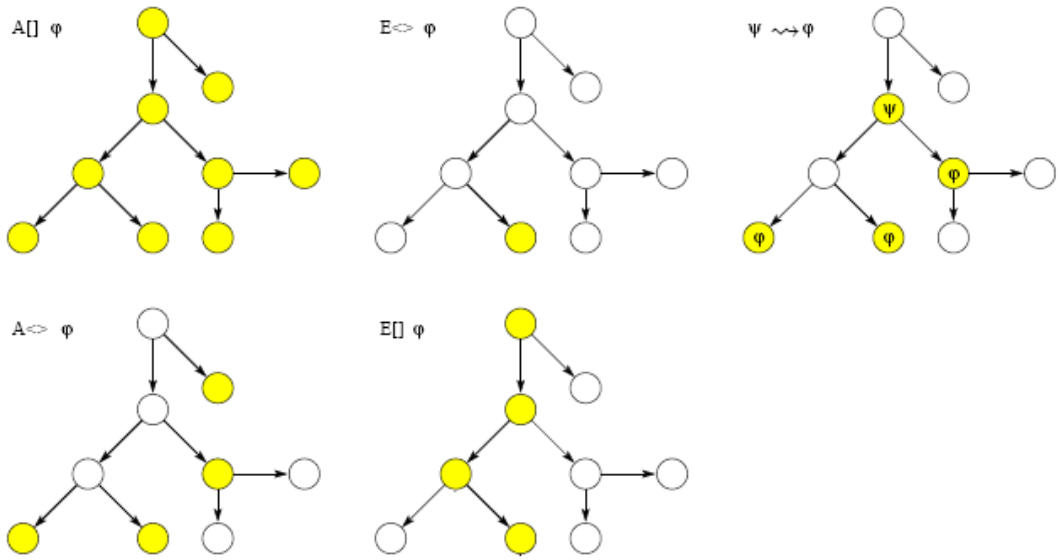


FIG. 3.21 – Les formules de chemins sont supportées par UPPAAL. Les états en jaune indiquent ceux satisfaisants une certaine formule $\phi$

classées en formules d'atteignabilité,de sureté et de vivacité. La Figure 3.21 illustre les différentes formules de chemin supportées par UPPAAL. Chaque type sera décrit ultérieurements dans 3.4.5.

Une formule d'état est une expression qui peut être évaluée sur un état sans regarder le comportement du modèle. Par exemple, l'expression simple $i == 7$, c'est vrai dans un état où $i$ égale à7. La syntaxe de formule d'état est un ensemble de gardes.

**Model-Checking** L'architecture d'UPPAAL a connu de nombreuses améliorations. L'algorithmre implémenté actuellement suppose l'unification des deux liste *Visitée* (Passed List) *Attendue* (Wait.List) en une seul list : PW.

**Algorithme 3.4** $Q = PW = \{(l_0, Z_0 \wedge I(l_0))\}$

*Tantque* $Q \neq 0$
  $(l, Z) = Q.popState()$
  *Si* $testProperty(l, Z)$ *Alors*
    *return vrai*
  $\forall (l', Z') : (l, Z) \Rightarrow (l', Z')$ *Faire*
    *Si* $\forall (l', Z') \in PW : Z' \nsubseteq Y'$ *Alors*
      $PW = PW \cup \{(l', Z')\}$
      $Q.append\ (l', Z')$
    *Fin Si*

*Fin Faire*
*Fin Tantque*
*return faux*

**Propriétés à tester** UPPAAL est capable de vérifier les propriétés d'atteignabilité, de sûreté, de vivacité simple, de vivacité bornée, et de non-blocage.

- Les propriétés d'atteignabilité sont la forme la plus simple de propriétés. Il s'agit de vérifier si une formule d'état donnée,$\varphi$, peut être satisfaite par un état accessible quelconque. Une autre façon de déclarer ceci est : "Existe-il un chemin partant de l'état initial, tel que $\varphi$ est éventuellement satisfaite sur ce chemin". Les propriétés d'atteignabilité sont souvent utilisées dans la conception d'un modèle pour la vérification de la correction (correctness). Par exemple, un protocole de communication comportant un émetteur et un récepteur, il est important de savoir si c'est possible pour l'émetteur d'envoyer un message à n'importe qu'elle récepteur ou si un message peut-être reçu. Ces propriétés ne peuvent pas garantir par eux-mêmes la corresction du protocole (c.-à-d. que tout message est éventuellement délivré ), mais ils valident le comportement de base du modèle.

  Nous exprimons que quelques états satisfont $\varphi$ devraient être accessibles en utilisant la formule de chemin $E \Diamond \varphi$. Dans UPPAAL, nous écrivons cette propriété en utilisant la syntaxe$E <> \varphi$.

- Les propriétés de sûreté sont de la forme : "quelque chose de mauvais n'arrivera jamais". Par exemple, dans un modèle d'une centrale électrique nucléaire, une propriété de sûreté peut éxiger que la température du fonctionnement est toujours sous un certain seuil (invariant), ou qu'une fusion ne se produira jamais. Une variante de cette propriété est "quelque chose, éventuelle, ne se passera jamais". Par exemple pour un jeu, un état sûr est un état dans lequel nous pouvons gagner encore le jeu, ainsi, éventuellement ,nous n'allons pas perdre.

  Dans UPPAAL ces propriétés sont formulées positivement, par exemple, quelque chose de bon est toujours vrai. Soit $\varphi$ une formule d'état. Nous exprimons que $\varphi$ devrait être vraie dans tout état accessible avec la formule de chemin $A\Box\varphi$, alors que $E\Box\varphi$ veut dire qu'il doit exister un chemin maximal [17]tel que $\varphi$ est toujours vraie Dans UPPAAL nous écrivons $A[]\varphi$ et $E[]\varphi$, respectivement.

- Les propriétés de vivacité sont de la forme : quelque chose arrivera finalement, par exemple, si nous appuyons sur le bouton de la télécommande, alors la télévision finira par s'allumer, ou dans un protocole de communication, tout message qui a été envoyé devrait être reçu.

[17]Un chemin maximal est un chemin qui est ou bien infini ou bien le dernier état n'a pas de transitions sortantes.

Dans sa forme simple, la vivacité est exprimée avec la formule de chemin $A\Diamond\varphi$, signifie $\varphi$ est satisfaite finalement. La forme la plus utile est la propriété de réponse, qui s'écrit $\varphi \rightsquigarrow \psi$[18] et lu : une fois $\varphi$ est satisfaite, alors $\psi$ le sera, par exemple toutes les fois qu'un message est envoyé, alors il finira par être reçu. Dans UPPAAL ces propriétés sont écrites ainsi, $A <> \varphi$ et $\varphi$ --→ $\psi$, respectivement.

- Les propriétés de vivacité bornée. Il s'agit de propriétés qui garantissent la vivacité tout en respectant une borne supérieur de temps. Les propriétés de la vivacité bornée sont essentiellement des propriétés de sûreté et donc souvent plus facile à vérifier. Donc se déplacer de la vivacité inconditionnelle vers celle à temps-borné permet, non seulement, de fournir une information supplémentaire, c.-à-d., si nous pouvons fournir une borne valide, mais également mènera aussi à une vérification plus efficace.

  Nous considérons deux variantes pour le temps-borné liées à l'opérateur $\varphi \rightsquigarrow_{\leq t} \psi$qui exprime une fois que la propriété d'état $\varphi$ soit vraie alors la propriété $\psi$ doit être vraie par la suite dans au plus $t$ unités de temps.

  Dans la première version nous utilisons une réduction simple à une formule non bornée. Premièrement, nous ajoutons une horloge supplémentaire $z$ qui est réinitialisée une fois que $\varphi$ soit vraie. La propriété à temps borné $\varphi \rightsquigarrow_{\leq t} \psi$ sera simplifiée en $\varphi \rightsquigarrow (\psi \wedge z \leq t)$

  Dans la seconde version , la plus efficace, nous utilisons la méthode proposée dans [MLY98] dans laquelle les propriétés à temps-borné sont réduites aux propriétés de sûreté simples. La formule soit étendue par une variable booléenne $b$ et une horloge supplémentaire $z$. La variable $b$ doit être initialisée à faux, une fois que $\varphi$ soit vraie nous associons vrai à $b$ et $z$ sera réinitialisée. Quand $\psi$ sera satisfaite $b$ prendra à nouveau la valeur faux. Donc la valeur de vérité de $b$ indique la satisfaction de $\psi$ dans le futur et $z$ mesure le temps accumulé jusqu'à la satisfaction de $\psi$. La vérification de la propriété à temps borné $\varphi \rightsquigarrow_{\leq t} \psi$ sera simplifiée en vérifiant la sureté de $A\Box(b \Rightarrow z \leq t)$.

  Une troisième méthode, que nous ne décrivons pas ici, est basée sur l'augmentation du modèle avec un test-automata [LAL98][LAL].

  Nous avons été délibérément quelque peu vagues au sujet de la nature exacte de l'augmentation exigée du modèle. Le cas le plus simple est quand les propriétés $\varphi$ et $\psi$ sont des localités (états) simples $l$ et $l'$ de l'automate. Dans ce cas,les variables $z$ et $b$ seront ajoutées à l'ensemble des transitions entrantes de $l$ et $l'$.

  Dans l'exemple du train-barrière [GBL04], une exigence naturelle est qu'un train aura le droit de traverser dans un temps borné (soit disons 100) après avoir signalé qu'il approche. En fait, la porte n'est pas uniquement responsable pour éviter des collisions sur la traversée mais aussi pour assurer l'équité et gérer les demandes à temps . Dans Figure.3.22 le Template de Train a été augmenté avec une variable booléenne locale $b$ et une horloge locale $z$. $b$ (initialisée à 0) est mis à 1 sur la transition vers la localité *Appr*

[18] Les experts dans CTL reconnaîtront que $\varphi \rightsquigarrow \psi$ est équivalente à $A\Box(\varphi \Rightarrow A\Diamond\varphi)$

et à 0 sur les deux transitions vers $Cross$. L'horloge $z$ est réinitialisée sur la transition vers $Appr$. Sur le modèle augmenté nous vérifions maintenant la propriété de la sûreté $A[](Train1.b == 1\ imply\ Train1.z <= 100)$. En fait, dû à la symétries évidentes dans le modèl, il suffit de vérifier la propriété pour un train, Train1 dans notre cas.

- L'interblocage. Dans UPPAAL, l'interblocage est exprimé en utilisant une formule d'état spéciale qui est simplement le mot-clé deadlock et est satisfaite pour tous les états de l'interblocage. Un état est un état d'interblocage s'il n'y a ni de transitions d'action sortantes ni de successeurs de délai.

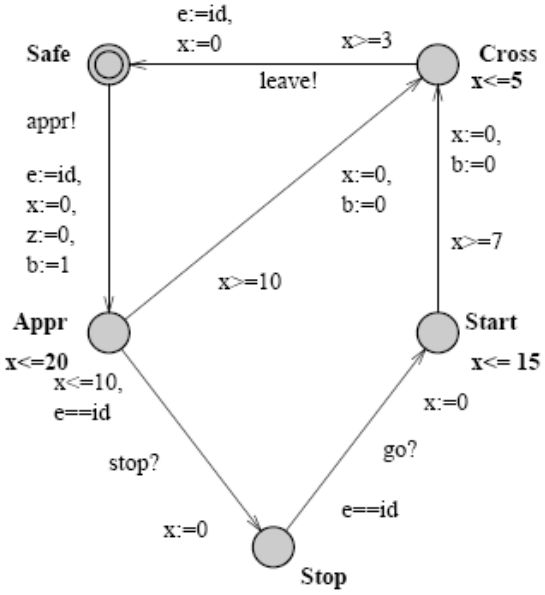


FIG. 3.22 – Le Train|-Barrière augmenté pour permettre la vérification de vivacité bornée.

## 3.5 Conclusion

Tout au long de ce chapitre, nous avons présenté le modèle des automates temporisés ainsi que les model-checkers proposés pour la vérification de propriétés quantitatives opérant sur un espace d'états symboliques généré à partir de l'espace concret infini. Notre objectif étant de choisir un Model-Checker puissant pour la vérification d'une spécification en TCTL. Nous avons retenu l'outil UPPAAL vu sa puissance et sa capacité de modélisation et de vérification des systèmes temps réel. D'autre part, UPPAAL permet d'exprimer le temps continu et des propriétés telles que la vivacité bornée, la sûreté, etc.ce qui a motivé notre choix.

Dans le chapitre suivant, nous tenons l'approche de vérification regroupant TCTL et DATA*, en se basant sur la formalisation du passage de DATA*'s vers les automates temporisés.

Deux étapes seront étudiées : Le passage proprement dit d'un DATA* vers un automate temporisé et l'exploitation de ce dernier pour la vérification de propriétés quantitatives en utilisant l'outil UPPAAL.

# Chapitre 4

# Vérification des DATA*'s à la UPPAAL

Notre objectif dans ce travail est de proposer une technique de vérification automatique bénéficiant du modèle des DATA*'s et la TCTL. Après une étude approfondie des différentes techniques utilisées, nous avons choisi de regrouper la logique TCTL, DLOTOS, les DATA*'s et le Model-Checker UPPAAL dans une approche de spécification et de vérification traitant le temps quantitativement et disposant de primitives pour le traitement des durées d'activités et de concepts temporels indispensables comme l'urgence. Le choix de TCTL et UPPAAL est dû au fait que la logique TCTL permet l'expression quantitative du temps et représente une extension assez naturelle des travaux de [SB05][SB06][Bel05] opérant sur la CTL et DATA*'s. Quand à UPPAAL il est connu comme étant le Model-Checker le plus facile à utiliser comparé à d'autres outils traitant les systèmes temps-réel temporisés.

L'approche que nous proposons consiste à :

- Spécifier le système avec DLOTOS,
- Générer le DATA* correspondant
- Transformer la modèle obtenu en automate temporisé pour UPPAAL,
- Décrire les propriétés du système avec TCTL,
- Transformer la spécification obtenue en une spécification UPPAAL,
- Utiliser le vérificateur d'UPPAAL pour la vérification.

## 4.1 Interprétation des DATA*'s par des automates temprisés

Considérons l'expression de comportement suivante $E := a; b; stop$, supposons que les deux actions $a$ et $b$ ont les durées respectives 10 et 12 et qu'elles sont offertes à l'environnement

dans les quantités de temps respectives 3 et 4 depuis leurs sensibilisations, le comportement global de $S$ est représenté par le DATA* de la Figure4.1

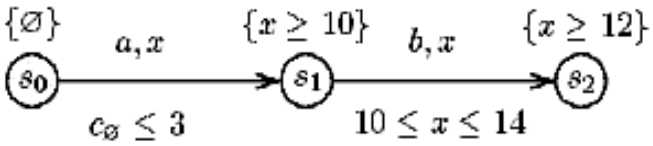


FIG. 4.1 – DATA* représentant l'exécution de deux actions successives

A partir de l'état initial $s_0$, l'action $a$ peut commencer son exécution tout en respectant la condition $c_\varnothing \leq 3$, qui veut dire que $a$ pourra toujours commencer son exécution tant qu'un délai de 3 unités depuis la sensibilisation du système [1]n'a pas encore été expiré. Le système passe à l'état $s_1$ et ne peut quitter avant la terminaison de l'action $a$, le même raisonnement s'applique pour l'action $b$.

Essayons maintenant d'exprimer se comportement avec un automate temporisé.

Le premier passage de l'état $s_0$ vers l'état $s_1$ peut être aisément décrit à l'aide de la transition $s_0 \xrightarrow{début(a), c_\varnothing \leq 3, x} s_1$(la transition de l'état $s_0$ à l'état $s_1$ commence l'action $a$, la condition $c_\varnothing \leq 3$ représente la garde et $x$ est l'horloges à réinitialiser à zéro par cette transition pour comptabiliser le temps d'exécution de $a$ ). Une fois le système est à l'état $s_1$ il ne pourra le quitter ni avant la terminaison de $a$ ni après l'expiration du délai d'offre de l'action $b$[2], l'action $b$ est sensibilisée à la fin de $a$ i.e. le délai d'offre de $b$ commence son expiration une fois $a$ est terminée. Là également, l'expression du passage de $s_1$ à $s_2$ pourra se faire à l'aide de la transition $s_0 \xrightarrow{début(b), 10 \leq x \leq 14, x} s_1$.

Il est claire qu'il est possible d'exprimer les durées sans être obliger à considérer chaque action comme deux évènements : début et fin. Ainsi, nous évitons l'affectation de l'automate en éclatant sa taille et son ensemble d'alphabet.

Donc, notre approche consiste à modéliser une action, qui a deux paramètres optionnels : un délai d'offre et une durée d'exécution, avec un automate temporisé comportant :

- Une transition, pour exprimer le début de l'action (avec une garde s'il existe un délai d'offre à respecter et une réinitialisation de l'horloge appropriée pour capturer l'instant de début de l'action),
- Un état, où le système réside au cours de l'exécution de l'action ( le système n'étant pas forcé à quitter cette état une fois l'action soit terminée).

La garde de la prochaine transition va nous servir à capturer, à la fois, la fin de l'action ($x \geq 10$) et le début de la prochaine action ($x \leq 14$) (l'horloge $x$ sera réinitialisée à 0

[1] Etant donné que $a$ est la première action à exécuter, sa sensibilisation correspond bien à la sensibilisation du système lui même.

[2] L'horloge $x$ sert à captuer la durée d'exécution de $a$ et le délai d'offre de $b$

encore une fois, nous disons qu'elle est réutilisée pour l'action $b$). Finalement nous obtenons l'automate représentant le comportement global de $S$ dans la Figure 4.2

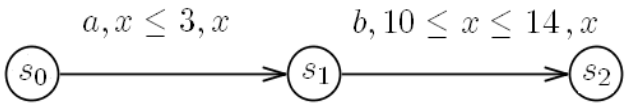


FIG. 4.2 – l'automate temporisé correspondant au DATA* de la Figure 4.1

En comparant les deux modèles illustrés par les deux Figures 4.1 et 4.2, nous pouvons constater que les deux expriment le même comportement sauf que pour l'automate temporisé nous avons perdu l'information sur la fin de l'action $b$.

Il est évident qu'il est impossible de capturer la fin de la toute dernière action sans ajouter une nouvelle transition. Pour cela nous considérons que tous les processus utilisent obligatoirement l'action particulière $\delta$ [3]pour marquer leurs fins. Et ainsi nous obtenons le comportement illustré par la Figure 4.3.

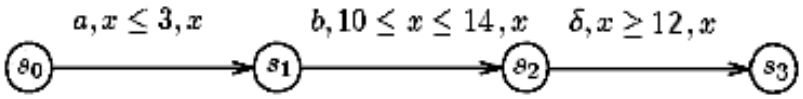


FIG. 4.3 – L'addition de l'action $\delta$

**Remarque 4.1** *Le temps pour une horloge dans un DATA* ne peut s'écouler avant l'affectation de cette horloge à une action, c'est pour cette raison là que le système est augmenté par l'horloge $c_\phi$ (initialisée dés la sensibilisation du système) pour comptabiliser le temps avant le déclanchement de toute action. Par contre, les horloges d'un automate temporisé commencent à comptabiliser le temps dés la sensibilisation du système. Ainsi, nous pouvons omettre l'horloge $c_\phi$ et la remplacer avec $x$ sans affecter la sémantique du modèle.*

Nous avons vu que le passage d'un DATA* vers un automate temporisé se fait d'une façon très naturelle dans le cas de processus séquentiels (ensemble d'action qui s'exécutent séquentiellement). Maintenant nous voulons savoir si cette approche supporte le parallélisme, l'urgence et l'autoconcurrence.

### 4.1.1 L'interprétation du parallélisme à l'aide d'un automate temporisé

Rappelons l'exemple du système $S$ qui se résume en deux sous-systèmes $S_1$ et $S_2$ s'exécutant en parallèle et se synchronisant sur une action $d$. Le sous-système $S_1$ exécute l'action $a$ suivie

[3]puisque l'action $\delta$ exprime naturellement la terminaison d'un processus

de $d$, tandis que $S_2$ exécute $b$ puis $d$. $S$ sera décrit par l'expression de comportement suivante $E := a; d; exit|||b; d; exit$, supposons que les actions $a$, $b$ et $d$ ont les durées respectives 10,12 et 4. La restriction temporelle du domaine de sensibilisation de l'action $d$ est de 5 et 4 unités de temps respectivement suivant la provenance de $d$ (de $S_1$ ou $S_2$), le comportement global de $S$ est représenté par le DATA* de la Figure 2.3

En appliquant le raisonnement précédent, nous obtenons l'automate temporisé illusté par la Figure 4.4

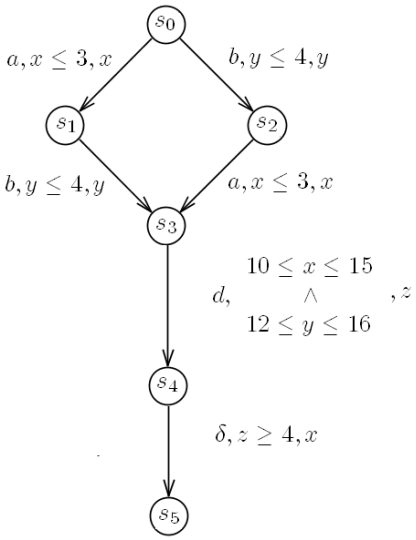


FIG. 4.4 – L'automate temporisé correspondant au DATA* de la Figure 2.3

### 4.1.2 L'interprétation de l'urgence à l'aide d'un automate temporisé :

Considérons toujours le même système mais cette fois ci avec l'hypothèse que toutes les actions sont urgentes (y compris $\delta$).

L'action est urgente, cela veut dire qu'elle pourra toujours avoir lieu tant que son délai d'offre soit respecté, et doit avoir lieu quand la borne supérieure de ce dernier soit atteinte (dans notre cas, c'est quand l'horloge $x$ (respectivement $y$) atteint la valeur 3 (respectivement 4) pour l'action $a$ (respectivement $b$)) quant à l'action $\delta$ dont l'intervalle d'offre est ouvert,elle devient urgente dés que son début soit possible (i.e.fin de $d$)

L'expression de l'urgence dans les automate temporisé ce fait à l'aide des invariants. Donc l'automate temporisé décrivant ce comportement sera comme dans la Figure 4.5).

### 4.1.3 L'interprétation de l'autoconcurrence à l'aide d'un automate temporisé

Il s'agit de lancer en concurrence des actions qui ont le même nom. Dans le but d'expression de la concurrence, l'attribution des horloges se fait par l'association, à chaque occurrence,

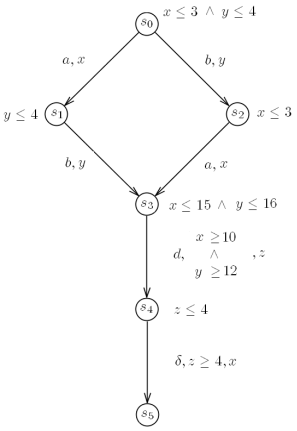


FIG. 4.5 – Automate temporisé exprimant l'urgence sur les actions

d'une horloge différente comme pour la concurrence de différentes actions. Donc le principe de l'interprétation du DATA* expliquée dans la section 4.1.1 reste également valable pour le cas de l'autoconcurrence.

### 4.1.4 Formalisation du passage d'un DATA* vers un automates temporisé

Soit $\mathcal{A} = (S, L_S, s_0, \mathcal{H}, T_D)$ un DATA* tel que :

- $S$ est un ensemble fini d'états,
- $L_S : S \rightarrow 2_{fn}^{\Phi_t(\mathcal{H})}$ 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.
- $T_D \subseteq S \times 2_{fn}^{\Phi_t(\mathcal{H})} \times 2_{fn}^{\Phi_t(\mathcal{H})} \times Act \times \mathcal{H} \times S$ 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 4.1** *Soit $\mathcal{A} = (S, L_S, s_0, \mathcal{H}, T_D)$ un DATA*. Soit $t = (s, G, D, a, x, s')$ une transition $\in T_D$. Nous pouvons écrire une contrainte temporelle $G$ (respectivememt $D$) comme suit*

**Définition 4.2** $\bigwedge d_i \sim x \sim d_s$ [4]*telque*: $d_i, d_s \in \mathbb{R}$ *et* $\sim \in \{<, \leq\}$*et* $d_i \sim x, x \sim d_s \in G$

[4] Quand l'intervalle est ouvert la contrainte sera donc de la forme $d_i \sim x < \infty$

*(respectivemem $D$).*

*Nous pouvons ainsi définir les fonctions suivantes : Pour une échéance $D$ ($D \neq \varnothing$) :*

$G\ /D = \bigwedge d_i \sim x \sim d_s$ *tel que* $\bigwedge d_i \sim x \sim d_s \models G \wedge \overline{D}$

$P_i(D) = \bigwedge d_i \sim x$

$P_s(D) = \bigwedge x \sim d_s$

**Remarque 4.2** *Une action urgente avec un intervalle d'offre non restreint est représentée dans $D$ par une contrainte de la forme $d \sim x \sim d$ au lieu de $d \sim x < \infty$ $d \in \mathbb{R}$ et $\sim \in \{<, \leq\}$.*

**L'automate temporisé** $\mathcal{TA}$ généré à partir du DATA* $\mathcal{A}$ sera le six-uplet $(Act, S, s_0, H, E, I)$ tel que :

- $Act$ est l'ensemble d'alphabet,
- $S$ est l'ensemble fini d'états,
- $s_0 \in S$ est l'états initial,
- $H$ est l'ensemble fini d'horloges, et
- $E \subseteq S \times S \times Act \times 2^H_{fn} \times \Phi(H)$ est l'ensemble des transitions tel que :
  - si $\langle s, G, D, a, x, s' \rangle \in T_D$ alors $\langle s, s', a, x, G\ /D \wedge P_i(D) \rangle \in E$ et $I(s) = P_s(D)$

Un arc $\langle s, s', a, \lambda, \delta \rangle$ de $\mathcal{TA}$ 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 $\delta$ est une contrainte temporelle sur $H$. $\langle s, s', a, \lambda, \delta \rangle$ peut être écrit $s \xrightarrow{a,\lambda,\delta} s'$.

## 4.2 Exploitation d'UPPAAL pour la vérification des DATA*'s

Dans ce qui suit nous nous intéressons uniquement à la vérification de la vivacité bornée.

Considérons le comportement décrit par l'expression $E = a; b; d; exit$, supposons que les actions $a$, $b$ et $d$ ont les durées respectives 10, 12 et 4 et qu'elles sont offertes pour les quantités de temps suivantes 3, 4 et 4.

L'automate temporisé généré à partir du DATA* soit décrit par la Figure 4.6

Afin de pouvoir exploiter l'approche décrite dans la sous-section 3.4.5, qui consiste à transformer la propriété de vivacité bornée en une simple propriéte de sureté, nous augmentons le modèle par deux variables :

- Une horloge supplémentaire, *time*, pour garder le cumul du temps.
- Une variable booléenne, $b$, initialisée à 0 (false).

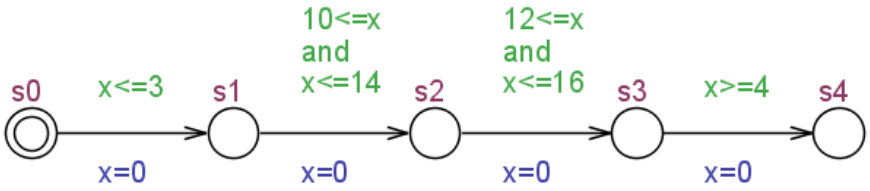


FIG. 4.6 – Exécution d'actions séquentielles

Cependant, pour savoir quand devons nous initialiser ces variables, nous devons d'abord examiner les zones de temps générées en utilisant l'outil UPPAAL.

Supposons que nous voulons vérifier que le temps entre les début des deux actions $a$ et $d$ respecte un certain intervalle $[i, j]$.

Pour ce faire, nous commençons par l'intialisation de l'horloge *time*. Si l'initialisation se fait sur la première transition de $s_0$ à $s_1$ i.e.au déclanchement de l'action $a$, la durée pour laquelle l'action $a$ est offerte ne sera pas repérer par l'horloge *time*, pour cela l'initialisation de *time* est faite avant le franchissemnt de la transition de $s_0$ vers $s_1$.

Les zones calculées par UPPAAL sont comme suit :

$$\left\{\begin{array}{l} time \geq 0 \\ x \geq 0 \\ time = x \end{array}\right. \xrightarrow{début(a)} \left\{\begin{array}{l} time \geq 0 \\ x \geq 0 \\ time - x \ in \ [0,3] \end{array}\right. \xrightarrow{début(b)} \left\{\begin{array}{l} time \geq 1à \\ x \geq 0 \\ time - x \ in \ [10,17] \end{array}\right. \xrightarrow{début(d)}$$
$$\left\{\begin{array}{l} time \geq 22 \\ x \geq 0 \\ time - x \ in \ [22,33] \end{array}\right. \xrightarrow{début(\delta)} \left\{\begin{array}{l} time \geq 26 \\ x \geq 0 \\ time - x \leq -26 \end{array}\right. \quad (4.1)$$

La zone $Z_0$ (correspond à l'état $s_0$) décrite par les contraintes $time \geq 0, x \geq 0$ qui permettent au système de rester indéfiniment ce qui correspond bien à la sémantique du modèle, quant à la contrainte $time = x$, elle exprime que le temps s'écoule de la même façon pour les deux horloges à savoir *time* et $x$. Voyons maintenant la contrainte $time - x \ in \ [0, 3]$ de $Z_1$, qui peut correspondre à l'intervalle d'offre de l'action $a$ i.e.$[0, 3]$. En observant les autres zones, l'intervalle donné par $time - x$ correspond toujours aux dates de début au plut tôt et au plus tard . Effectivement $b$ commence au plus tôt à l'instant 10 (fin de $a$) et au plus tard à l'instant $3 + 10 + 4 = 17$. L'action $d$ commence après la fin de $b$ donc au plus tôt après $10 + 12 = 22$ et au plus tard à $3 + 10 + 4 + 12 + 4 = 33$.

Cette information est de grande utilité pour la vérification, il suffit d'initialiser la variable $b$ par $1(true)$ sur la transition entrante à l'état $s_3$, déclanchant le début de $d$, et par $0$ $(false)$ sur la transition sortante Figure 4.7. Ainsi nous pouvons utiliser la formule $A[](b \ imply \ i \leq time - x \leq j \ )$ pour vérifier si le début de $d$ respecte un certain intervalle aprés le début de $a$.

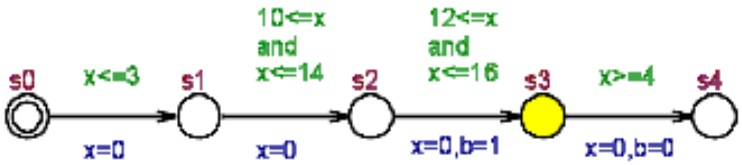


FIG. 4.7 – Augmentation du TA de la Figure 4.6 par *time* et $b$

Supposons maintemant que les deux processus $a; d; exit$ et $b; d; exit$ s'exécutent en parallèle puis se synchronisent sur $d$ avec $d$ offerte à l'environnement dans 5 et 4 dans chaque processus.

L'automate généré sera comme suit : Figure 4.8

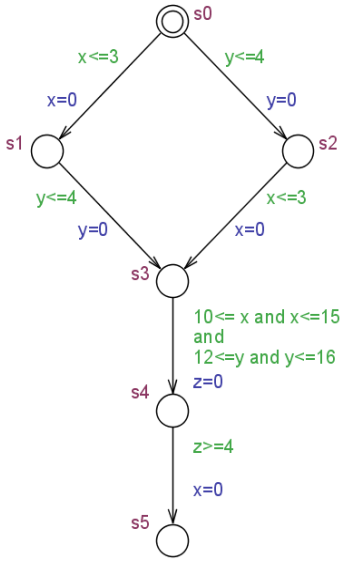


FIG. 4.8 – Comportement parallèle

L'automate augmenté par *time* et $b$ est illustré par la Figure 4.9

Et la zone correspondante à $s_4$ est :

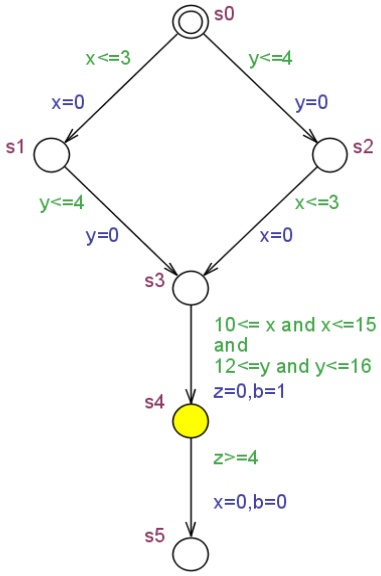


FIG. 4.9 – Augmentation du TA de la Figure 4.8

$b = 1$
$x \geq 12$
$y \geq 12$
$z \geq 0$
$time \geq 12$
$x - y \ in \ [0, 3]$
$x - z \ in \ [12, 15]$
$x - time \ in \ [-3, 0]$
$y - z \ in \ [12, 15]$
$y - time \ in \ [-4, 0]$
$time - z \ in \ [12, 18]$

Voyons maintenant si ces valeurs reflèlent correctement l'écoulement du temps pour toutes ces horloges (Voir Figure 4.10).

L'action $d$ ne peut commencer que si $a$ et $b$ ont terminé leurs exécutions, donc $d$ commense au plus tôt à l'instant 12. Calculons la date de début au plus tard de $d$.Dans le processus $b; d; exit$ , il reste possible de commencer l'action $d$ jusqu'à l'instant $4 + 12 + 4 = 20$ sauf que dans le processus $a; d; exit$ l'action $d$ ne peut commencer aprés un l'instant $3 + 10 + 5 = 18$ donc $d$ commence au plus tard à l'instant 18 ce qui correspond à la valeur de $time - z$.

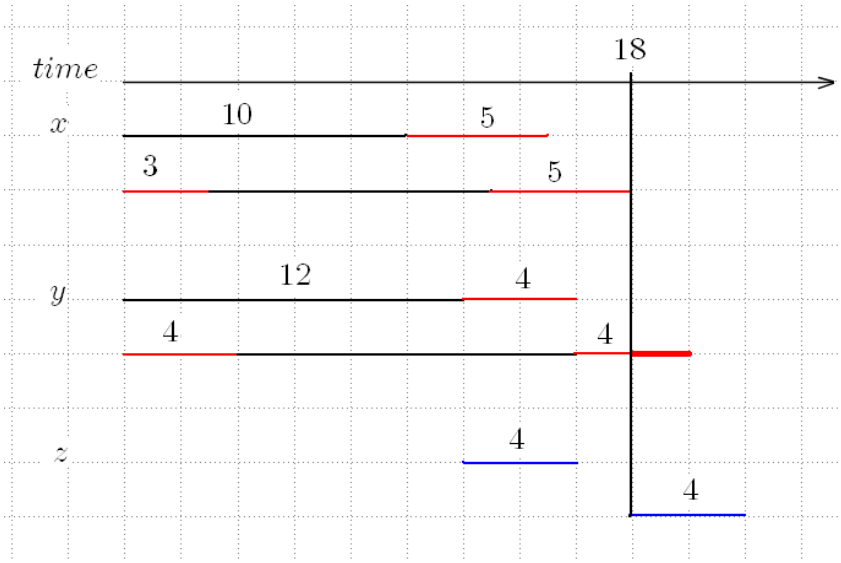


FIG. 4.10 – Ecoulement du temps

## 4.3 Etude de cas

Le but de cette section 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 quantitatives 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 4.11.

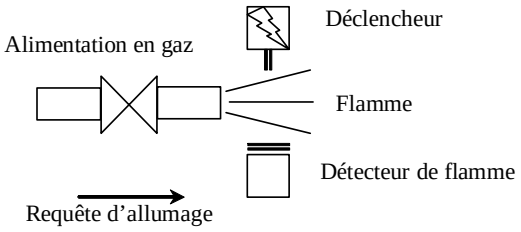


FIG. 4.11 – *Schéma primitif d'un brûleur à gaz*

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, et ainsi l'écoulement du gaz.

Le comportement simplifié du brûleur à gaz est illustré dans la Figure 4.12.

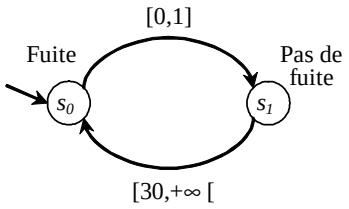


FIG. 4.12 – *Comportement simplifié d'un brûleur à gaz*

### 4.3.1 Spécification

Supposons que le brûleur à gaz a le comportement de la Figure 4.12, 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
(
  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 4.13.

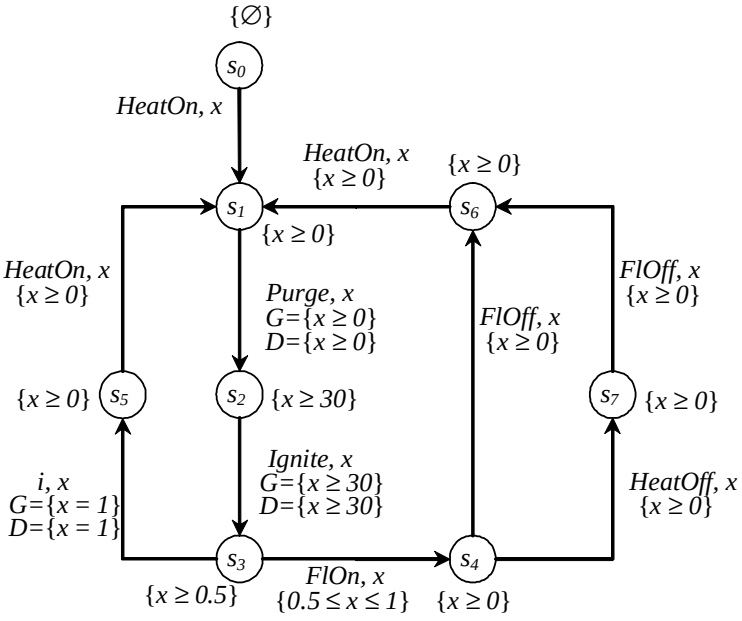


FIG. 4.13 – *DATA* du brûleur à gaz*

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 2.10. 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).

### 4.3.2 Vérification

Dans notre cas, nous adoptons une approche de vérification par model checking. Conformément à l'approche de vérification décrite dans la Section **??**, vérifier des propriétés quantitatives sur un système réactif revient à le modéliser par un DATA* puis le transformer en automate temporisé comme le décrit la figure 4.14 [5].

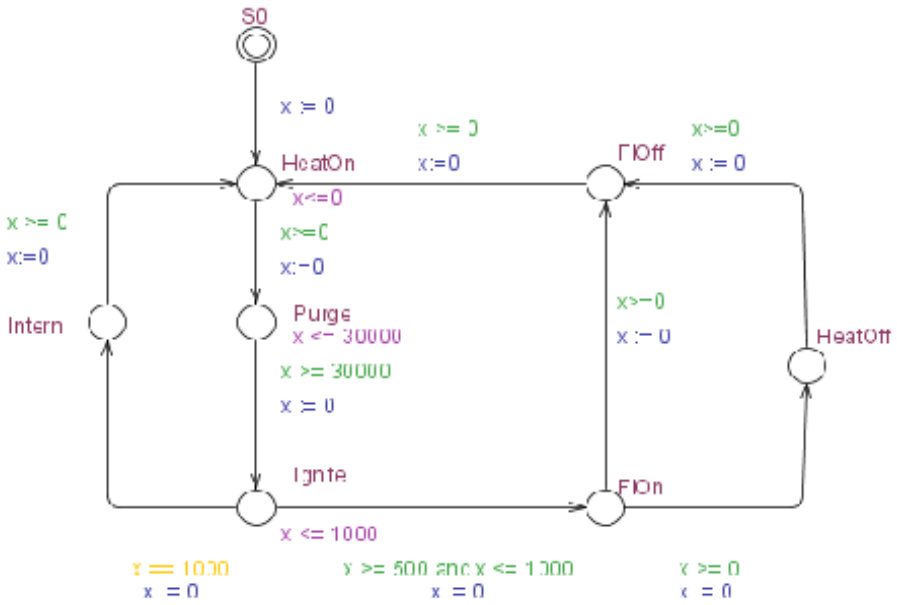


FIG. 4.14 – L'automate temporisé généré à partir du DATA* de la figure 4.13

#### Propriétés de vivacité bornée

1. «Le brûleur doit déclencher une étincelle après une requête d'allumage de l'utilisateur.dans 30 secondes». Nous commençons d'abord par l'augmentation de l'automate temporisé résultant par les deux variables *time* et *b* (figure 4.15).La propriété à vérifier sera donc : `A[](b imply time-x >=30000)`

2. «Une requête d'allumage a toujours la possibilité d'être satisfaite. dans un intervalle de 30.5 secondes à 31 secondes». Cette fois ci nous allons vérifier

[5] Les constantes dans Uppaal sont des entiers, ce qui nous a poussé à exprimer ces constantes en millisecondes, ainsi 0.5 secondes sera convertie en 500 millisecondes.

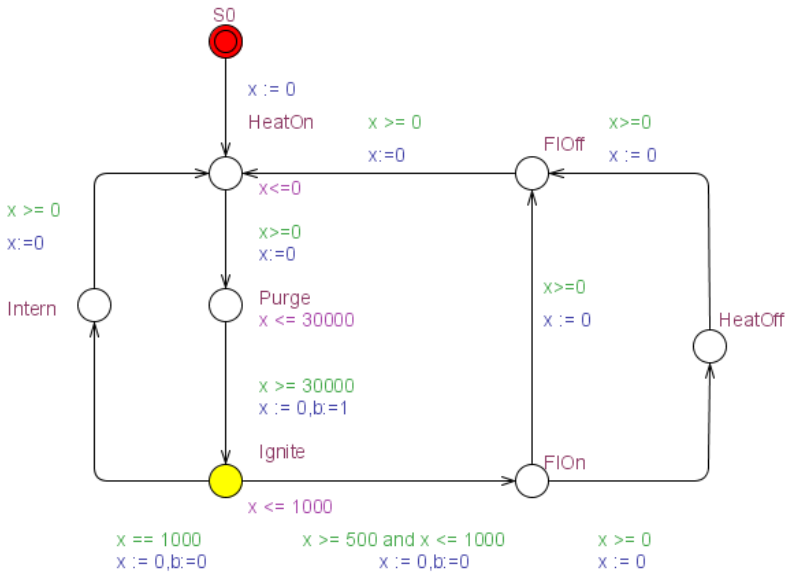


FIG. 4.15 – L'augmentation de l'automate temporisé par l'hrloge *time* et le booléen *b*

la propriété `A[](c imply 30500 <=time-x and time-x <=31000)` aprés l'augmentation de l'automate temporisé résultant par les deux variables *time* et *c*.(Figure.4.16)

Le résultat donné par le vérificateur est présenté par la figure 4.17.

## 4.4 Conclusion

Dans ce chapitre, nous avons présenté notre approche de vérification qui se base sur l'interprétation du DATA* exprimant le comportement du système en un automate temporisé. L'ordonnancement, la concurrence, l'urgence ainsi que l'autoconcurrence sont pris en considération.

Afin d'illustrer cette approche une étude d'un exemple de système à temps contraint avec durées explicites sur les actions est présentée. L'étude concerne principalement la spécification du système à l'aide du langage D-LOTOS puis sa traduction dans le modèle sémantique des DATA*'s et enfin l'interprétation du DATA* obtenu en un automate temporisé. Une dernière étape concerne la spécification des propriétés quantitatives attendues de ces systèmes pour être vérifiées ensuite à l'aide du vérificateur d'UPPAAL.

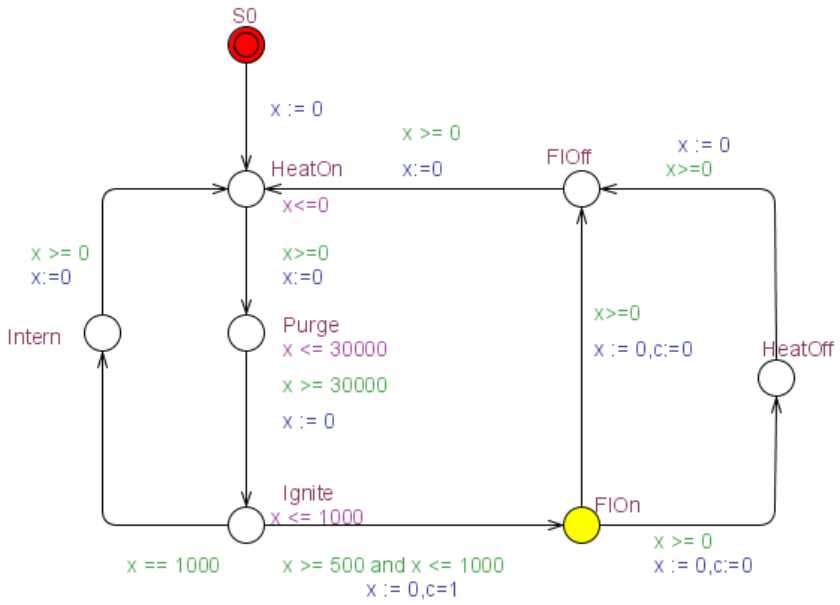


FIG. 4.16 – L'augmentation de l'automate temporisé par l'hrloge *time* et le booléen $c$

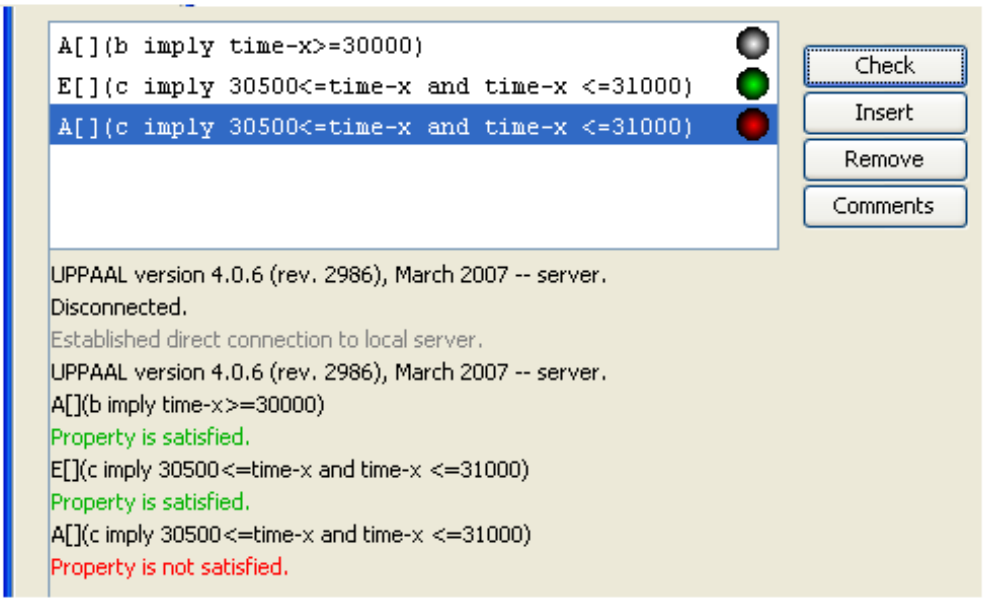


FIG. 4.17 – Résultat de la vérification par UPPAAL

# Chapitre 5

# Conclusion et perspectives

Au cours de ce travail, nous nous sommes intéressés à la spécification et la vérification des systèmes temps réel. Ces systèmes, qui deviennent de plus en plus complexes, peuvent avoir des pannes cataclysmiques causant des pertes économiques et mettant en jeu des vies humaines. Il est clair que la spécification et la vérification de tels systèmes est une étape fondamentale dans le cycle de développement des systèmes informatiques. En effet, l'utilisation des méthodes de spécification et de vérification contribue au développement de systèmes sûrs de fonctionnement.

Notre approche bénéficie d'une spécification tenant compte des spécificités des systèmes temps réel. L'utilisation d'une méthode formelle dans ce contexte apparaît comme l'une des solutions principales. En effet, elle permet de spécifier le système avec précision et de disposer d'un fondement mathématique qui permet de vérifier qu'une spécification obtenue satisfait les propriétés exigées par l'utilisateur. Il s'avère également que, les méthodes de vérification basées sur la technique du model-checking sont des méthodes disposant d'outils assez fiables et supportant des systèmes de taille importante.

Dans ce but, nous nous sommes focalisés sur une approche de spécification et de vérification de système temps réel regroupant un langage de spécification de haut niveau permettant l'expression explicite des durées ainsi que d'autres concepts essentiels à la spécification des systèmes à temps contraint, un modèle sémantique adapté au langage choisi et un Model-Checker pour la vérification.

Ainsi, nous nous sommes intéressés à deux étapes dans le cycle de développement des systèmes temps réels : la spécification et la vérification. Tout d'abord, nous avons étudié l'approche intégrant le langage D-LOTOS et la sémantique temporelle de maximalité dans le but de prendre en compte des contraintes temporelles et des durées d'actions (Chapitre 1). Par la suite, nous avons présenté le modèle des DATA*'s, qui fait l'objet de ce travail, comme modèle sémantique permettant l'expression explicite des durées ainsi que d'autres concepts essentiels à la spécification des systèmes à temps contraint notamment l'urgence (Chapitre 2). Egalement, nous avons introduit les model-checkers proposées pour la vérification de propriétés quantitatives opérant sur une logique temporelle quantitative (TCTL) et

le modèle des automates temporisés qui s'adaptent à la spécification de tels systèmes, ainsi que l'outil UPPAAL comme environnement de vérification (Chapitre 3). En fin, nous avons présenté notre approche de vérification de propriétés quantitatives sur des DATA* qui se base principalement sur la transformation des DATA*'s à des automates temporisés et l'exploitation de ces derniers pour la vérification en utilisant un outil approprié (UPPAAL). Les différentes étapes suivies sont abordées selon divers cas possibles à savoir l'ordonnancement, la concurrence, l'urgence et l'autoconcurrence. Une étude de cas est présentée : brûleur à gaz. Nous envisageons d'appliquer notre étude sur d'autres systèmes, tels que les protocoles de vérification et des exemples «académiques» comme le flux multimédia (Multimedia Stream). (Chapitre 4).

## 5.1 Perspectives

Pour conclure, notre travail peut être considéré comme un premier pas vers le rapprochement des deux modèles, à savoir le modèle des DATA*'s et le modèle des automates temporisés, malgré la considération de différentes hypothèses d'atomicité structurelle et temporelle. Seule l'utilisation de l'environnement UPPAAL est étudiée dans notre approche. En revanche, l'utilisation des autres environnements, comme KRONOS, reste envisageable afin de bénéficier d'autres contributions.

Cependant, le développement d'un environnement complet de spécification et de vérification des systèmes temps-réel, dont D-LOTOS est le langage de spécification, exploitant de préexistants environnements de vérification d'automates temporisés dépend de certains choix. Pour cela, la question qui nous semble fondamentale, étant donné que notre approche se base essentiellement sur un passage entre modèles, est de savoir si les automates temporisés peuvent remplacer les DATA*'s i.e. nous procédons à une génération directe d'un automate temporisé à partir d'une spécification D-LOTOS. C'est dans ce but, qu'une comparaison théorique entre les deux modèles s'imposera et que de multiples études de cas seraient indispensables avant de passer au développement d'un outil validant notre approche.

Egalement, dans la littérature, il y'a beaucoup d'autres logiques quantitatives à temps dense, outre que la logique TCTL, qui mériteront d'être étudiées et qui pourront être d'un grand intérêt pour le modèle des DATA*'s, nous citons à titre d'exemple la Duration Calculus.

# Bibliographie


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

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

[ADY03] K. G. Larsen A. David, G. Behrmann and W. Yi. A tool architecture for the next generation of uppaal. Technical report, Uppsala University, 2003.

[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.

[AH93] R. Alur and T. A. Henzinger. Real-time logics : Complexityand expressiveness. *Information and Computation*, 104(1):35–77, 1993.

[AH94] R. Alur and T. A. Henzinger. A really temporal logic. *ACM*, 41(1):181Ű203, 1994.

[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.

[BB91] J. C. M. Baeten and J. A. Bergstra. Real time process algebra. *Formal Aspects of Computing*, 3:142–188, 1991.

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

[Bel05] N. Belala. Formalisation des systèmes temps-réel avec durées d'Actions. Master's thesis, Université Mentouri, 25000 Constantine, Algérie, Juin 2005.

[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.

[Bol91] T. Bolognesi. LOTOS-like process algebras with urgent or timed interaction. In *FORTE*, pages 255–270. North Holland, 1991.

[BS05] N. Belala and D. E. Saïdouni. Non-atomicity in timed models. Research report, Computer Science Dept., University of Mentouri, 25000 Constantine, Algeria, February 2005.

[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.

[BWDGY96] J. Bengtsson, K. G. Larsen F. Larsson P. Pettersson W. D. Griffioen, K. J.Kristoffersen, and W. Yi. Verification of an audio protocol with bus collision using uppaal. In Rajeev Alur and Thomas A. Henzinger, editors, *CAV96*, number 1102 in LNCS, pages 244–256. Springer-Verlag, Jul 1996.

[BY04] Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In W. Reisig and G. Rozenberg, editors, *Lecture Notes on Concurrency and Petri Nets*, volume 3098 of *LNCS*. Springer-Verlag, 2004.

[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.

[CdO95] 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.

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

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

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

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

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

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

[CM05] P. Bouyer F. Chevalier and N. Markey. On the expressivenessof tptl and mtl. Technical Report 05, Laboratoire Specification et Vt̀erification, May 2005.

[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.

[DC96] I. Lee H. Xie et O. Sokolsky D. Clarke, H. Ben-Abdallah. XVERSA: An integrated graphical and textual toolset for the specification and analysis of resource-bound real-time systems. In *CAV'96*, volume 1102 of *LNCS*. Springer, 1996.

[eJMT96] P. Merino et J. M. Troya. EVP: Integration of FDTs for the analysis and verification of communication protocols. In *CAV'96*, volume 1102 of *LNCS*. Springer, 1996.

[eS96] R. Cleaveland et S.Sims. The NCSU concurrency workbench. In *CAV'96*, volume 1102 of *LNCS*. Springer, 1996.

[GBL04] A. David G. Behrmann and K. G. Larsen. A tutorial on uppaal. In Marco Bernardo and Flavio Corradini, editors, *Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004*, number 3185 in LNCS, pages 200–236. Springer-Verlag, 2004.

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

[JBW98] F. Larsson P. Pettersson W. Yi J. Bengtsson, K. G. Larsen and C. Weise. New generation of uppaal. In *Int. Workshop on Software Tools for Technology Transfer*, Jun 1998.

[JBY95] P. Pettersson J. Bengtsson, K. G. Larsen. F. Larsson and W. Yi. Uppaal a tool suite for automatic verification of real–time systems. In *Proc. Of Workshop on Verification and Control of Hybrid Systems III*, number 1066 in Lecture Notes in Computer Science, pages 232–243. Springer-Verlag, Oct 1995.

[JBY96] F. Larsson P. Pettersson J. Bengtsson, K. G. Larsen and Wang Yi. Uppaal in 1995. In *Proc. Of the 2nd Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, number 1055 in Lecture Notes in Computer Science, pages 431–434. Springer-Verlag, Mar 1996.

[JCF96] A. Kerbrat L. Mounier R. Mateescu et M. Siighireanu J. C. Fernandez, H. Garavel. CADP: A protocol validation and verification toolbox. In *CAV'96*, volume 1102 of *LNCS*, 1996.

[KGLY97] P. Pettersson K. G. Larsen and W. Yi. Uppaal in a nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, Oct 1997.

[LAL] A. Burgueno L. Aceto, P. Bouyer and K. G. Larsen. Thepower of reachability testing for timed automata. *Theoretical Computer Science,*, 1-3(300):411–475.

[LAL98] A. Burgueno L. Aceto and K. G. Larsen. Model checking via reachabilitytesting for timed automata. In Bernhard Steffen, editor, *TACASŠ98*, 1998.

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

[MLY98] P. Pettersson M. Lindahl and W. Yi. Formal design and analysis of a gearbox controller. In *Proc. Of the 4th Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, number 1384 in Lecture Notes in Computer Science, pages 281–297. Springer-Verlag, Mar 1998.

[NB96] E. Chang M. Colon A. Kapur Z. Manna H. B. Sipma et T. Uribe. N. Bjorner, A. Browne. STeP: Deductive algorithmic verification of reactive and real-time systems. In *CAV'96*, volume 1102 of *LNCS*. Springer, 1996.

[NS91] X. Nicollin and J. Sifakis. An overview and synthesis on timed process algebras. In K. G. Larsen and A. Skou, editors, *CAV'91*, volume 575 of *LNCS*, pages 376–398. Springer-Verlag, 1991.

[RC96] S. A. Smolka et O. Sokolsky. R. Cleaveland, P. M. Lewis. The concurrency factory: A development environment for concurrent systems. In *CAV'96*, volume 1102 of *LNCS*. Spriger, 1996.

[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.

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

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

[SC03a] D. E. Saïdouni and J. P. Courtiat. Prise en compte des durées d'action dans les algèbres de processus par l'utilisation de la sémantique de maximalité. In *CFIP'2003*. Hermes, France, 2003.

[SC03b] 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 (version étendue). Technical Report 03243, LAAS-CNRS, 7 avenue du colonel Roche, 31077, Toulouse Cedex France, may 2003.

[TAHY94] J. Sifakis T. A. Henzinger, X. Nicollin and S. Yovine. Symbolicmodel checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

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