



**HAL**  
open science

# Contribution à la modélisation réaliste et multi-échelles des systèmes bouclés temporisés

Matthieu Perin

► **To cite this version:**

Matthieu Perin. Contribution à la modélisation réaliste et multi-échelles des systèmes bouclés temporisés. Autre. École normale supérieure de Cachan - ENS Cachan, 2012. Français. NNT : 2012DENS0028 . tel-00753503

**HAL Id: tel-00753503**

**<https://theses.hal.science/tel-00753503>**

Submitted on 19 Nov 2012

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Contribution à la modélisation réaliste et multi-échelles des systèmes bouclés temporisés

## THÈSE

présentée et soutenue publiquement le 22 juin 2012

pour l'obtention du

Doctorat de l'École Normale Supérieure de Cachan

(spécialité Électronique/Électrotechnique/Automatique)

par

M. Matthieu PERIN

### Composition du jury

<i>Rapporteurs :</i>	Éric NIEL Éric RUTTEN	INSA Lyon, Ampère INRIA Grenoble
<i>Examineurs :</i>	Étienne CRAYE Alexandre PHILIPPOT	École Centrale de Lille, LAGIS Université Reims Champagne-Ardenne, CReSTIC
<i>Directeur de thèse :</i>	Jean-Marc FAURE	École Normale Supérieure de Cachan, LURPA
<i>Invité :</i>	M. François BICHET	Dassault Systèmes

Mis en page avec la classe thloria.

# Table des matières

<b>Introduction</b>	<b>1</b>
<b>Chapitre 1</b>	
<b>Contexte et problématique</b>	<b>5</b>
1.1 Contexte industriel . . . . .	6
1.1.1 Présentation de la suite DELMIA V6 . . . . .	6
1.1.2 Représentation d'une chaîne de production . . . . .	7
1.2 Problématique . . . . .	8
1.2.1 Caractéristiques du modèle détaillé . . . . .	8
1.2.2 Caractéristiques du modèle abstrait . . . . .	9
1.3 Objectifs de la thèse . . . . .	9
1.4 Présentation de l'exemple . . . . .	10
1.5 Organisation du mémoire . . . . .	14
<b>Chapitre 2</b>	
<b>Formalismes utilisés dans ces travaux</b>	<b>15</b>
2.1 Choix d'un formalisme . . . . .	17
2.1.1 Contraintes . . . . .	17
2.1.2 Brève présentation de quelques classes d'automates temporisés . . . . .	17
2.2 Formalisme de modélisation : les Automates Temporisés à Variables Dis- crètes (ATVD) . . . . .	20
2.2.1 Variables, expressions et contraintes . . . . .	20
2.2.2 Sémantique des ATVD sans urgence . . . . .	21
2.2.3 Ajout de la sémantique d'urgence pour les ATVD . . . . .	24
2.2.4 Réseau d'automates temporisés à variables discrètes . . . . .	26
2.2.5 Illustration des évolutions sur un exemple . . . . .	27
2.3 Passage des ATVD aux RATU . . . . .	28

2.3.1	Variables, expressions et localités . . . . .	28
2.3.2	Traduction des transitions non urgentes . . . . .	29
2.3.3	Traduction de la sémantique d’urgence des ATVD . . . . .	30
2.3.4	Traduction d’un réseau d’ATVD en un réseau d’automates temporels d’UPPAAL . . . . .	31
2.4	Conclusion . . . . .	31

### Chapitre 3

#### Construction du modèle détaillé à évolutions réalistes d’un système bouclé **33**

3.1	Construction du modèle de partie opérative . . . . .	35
3.1.1	État de l’art . . . . .	35
3.1.2	Principes de modélisation retenus . . . . .	36
3.1.3	Modélisation de la partie opérative du manipulateur . . . . .	44
3.1.4	Mise en évidence des concurrences internes . . . . .	48
3.1.5	Suppression des concurrences indésirables . . . . .	51
3.2	Construction du modèle du contrôleur . . . . .	52
3.2.1	Hypothèses . . . . .	52
3.2.2	État de l’art . . . . .	52
3.2.3	Modélisation d’un contrôleur non temporisé . . . . .	53
3.2.4	Modèle final, temporisé, du contrôleur . . . . .	59
3.3	Construction du modèle du système bouclé . . . . .	63
3.3.1	Prise en compte des phases de lecture et d’écriture des entrées/sorties de l’API . . . . .	64
3.3.2	Concurrence entre modèles du système contrôlé et du contrôleur . . . . .	65
3.3.3	Suppression des concurrences indésirables avec conservation de la réactivité du contrôleur . . . . .	69
3.3.4	Modélisation d’un contrôleur avec recherche de stabilité . . . . .	70
3.3.5	Synthèse des ajouts . . . . .	74
3.4	Conclusion . . . . .	75

### Chapitre 4

#### Vérification formelle de l’équivalence entre modèles détaillé et abstrait **77**

4.1	Choix d’une équivalence . . . . .	79
4.1.1	Rappel sur les STE . . . . .	80
4.1.2	Équivalences de la littérature . . . . .	81

4.1.3	Équivalence de traces d'ATVD . . . . .	86
4.1.4	Définition d'une équivalence de traces de deux ATVD . . . . .	89
4.2	Principe de la vérification . . . . .	92
4.2.1	Choix d'une technique de vérification formelle . . . . .	92
4.2.2	Vérification formelle à l'aide de l'outil UPPAAL . . . . .	92
4.2.3	Méthode des automates observateurs . . . . .	95
4.2.4	Principe des automates observateurs appliqué à la preuve d'équivalence . . . . .	96
4.3	Preuve d'une équivalence de traces . . . . .	98
4.3.1	Création d'un automate observateur-séquenceur non temporisé . . . . .	98
4.3.2	Équivalence de modèles temporisés . . . . .	104
4.3.3	Équivalence de modèles temporisés réagissant à des entrées . . . . .	112
4.4	Définition et vérification d'une équivalence de traces avec tolérance . . . . .	119
4.4.1	Motivations . . . . .	119
4.4.2	Équivalence de traces avec tolérance en valeur . . . . .	119
4.4.3	Définition d'une équivalence avec tolérance temporelle . . . . .	120
4.4.4	Proposition d'une équivalence de traces avec tolérance temporelle et en valeur . . . . .	122
4.4.5	Vérification d'une équivalence de traces avec tolérance . . . . .	123
4.5	Conclusion . . . . .	125
<b>Conclusions et perspectives</b>		<b>127</b>
<b>Bibliographie</b>		<b>129</b>
<b>Annexes</b>		<b>133</b>

<b>Annexe A Formalisme d'analyse : les Réseaux d'Automates Temporisés d'UPPAAL (RATU)</b>	<b>133</b>
A.1 Variables, expressions et contraintes . . . . .	134
A.2 Canaux de communication . . . . .	134
A.3 Sémantique d'un RATU sans urgence . . . . .	135
A.4 Exemple de RATU sans sémantique d'urgence . . . . .	137
A.5 Sémantique d'urgence d'un RATU . . . . .	138
A.6 Sémantique complète d'un RATU . . . . .	138
A.7 Illustration de la sémantique complète sur un RATU . . . . .	140

<b>Annexe B Modèles de partie opérative relatifs au chapitre 3</b>	<b>143</b>
B.1 Exemples de modèles d'une bibliothèque de composants génériques . . . . .	144
B.1.1 Actionneurs couplés à des pré-actionneurs . . . . .	144
B.1.2 Capteurs . . . . .	145
B.2 Modèles de la partie opérative du manipulateur . . . . .	146
<b>Annexe C Séquences d'évolutions relatives au chapitre 3</b>	<b>149</b>
C.1 Séquence d'évolutions avec les modèles de partie opérative modifiés . . . . .	150
C.2 Séquence d'évolutions après introduction des variables EVOL_PL et END_TI	152
C.3 Séquence d'évolutions après ajout de recherche de stabilité dans le contrôleur	154
<b>Annexe D Équations algébriques du Grafset et modèles du contrôleur du manipulateur</b>	<b>157</b>
D.1 Équations algébriques associées au Grafset du manipulateur . . . . .	158
D.2 Première version du modèle du contrôleur du manipulateur . . . . .	162
D.3 Modèle final avec recherche de stabilité du contrôleur . . . . .	165
<b>Annexe E Séquences d'évolutions et traces relatifs à la vérification de l'équivalence de traces</b>	<b>167</b>
E.1 Séquence d'évolutions avec ajout de l'Automate Observateur Séquenceur .	168
E.2 Séquence d'évolutions avec un AOS mesurant les décalages temporels de traces . . . . .	170
E.3 Séquence d'évolutions avec modèles modifiés des entrées . . . . .	173
E.4 Modèles et traces servant d'exemple aux équivalences avec tolérances . . . . .	176
E.4.1 Modèles et traces d'exemple de l'équivalence avec tolérance en valeur	176
E.4.2 Modèles et traces d'exemple de l'équivalence avec tolérance temporelle et conservation de la chronologie . . . . .	177

# Introduction

Le milieu académique a produit de nombreux résultats ces dernières années dans le domaine des méthodes d'analyse de systèmes à événements discrets. Un exemple frappant est le progrès réalisé dans le domaine des méthodes de vérification formelle comme le model-checking. Cependant, ces méthodes sont au final assez peu utilisées dans l'industrie. Ce manque d'application est généralement dû aux difficultés à :

- Utiliser la méthode d'analyse (écriture des propriétés formelles, compréhension et analyse des résultats, particulièrement en cas de preuve négative).
- Produire des modèles utilisables par les méthodes d'analyse (taille, modélisation réaliste de systèmes physiques, ...).

Dans le cadre de ces travaux, nous traiterons la deuxième difficulté en nous focalisant sur des modèles représentant des systèmes de production manufacturiers, qui sont généralement des systèmes bouclés composés d'un contrôleur et d'une partie opérative.

Ces modèles de systèmes bouclés doivent impérativement contenir un modèle de partie opérative, nécessaire pour la preuve de propriétés de vivacité (comme démontré par [Machado \*et al.\* \(2006\)](#)). De nombreux travaux ont proposé des modèles de contrôleur ([Hanisch \*et al.\* \(1997\)](#); [Frey et Litz \(2000\)](#); [Gourcuff \*et al.\* \(2008\)](#)) et de partie opérative ([Rohée \*et al.\* \(2006\)](#); [Machado \(2006\)](#); [Philippot \*et al.\* \(2009\)](#)), mais ils sont non temporisés ou sans vision globale du couple contrôleur-partie opérative. L'aspect temporisé des modèles (et en particulier de celui représentant la partie opérative) est pourtant une nécessité dans le cadre des analyses temporelles (temps de réponse, temps de cycle, ...). La mise en parallèle des modèles du contrôleur et de la partie opérative peut être aussi à l'origine d'évolutions irréalistes qui impactent les résultats des analyses de manière néfaste. C'est pourquoi nous utiliserons un formalisme temporisé pour nos modèles et nous apporterons une attention particulière à la construction d'un modèle du système bouclé ne contenant pas d'évolutions irréalistes.

La taille des modèles étant aussi un frein important lors des analyses (phénomène d'explosion combinatoire, comme décrit par [Bérard \*et al.\* \(2001\)](#) et par [Gourcuff \*et al.\* \(2008\)](#)), il faut des modèles représentant le même système bouclé à des niveaux de granularité différents. Ce choix implique de fait que les modèles, à des niveaux de granularité différents, aient un comportement identique permettant de remplacer l'un par l'autre en fonction des besoins d'analyse, et ce, sans influencer les résultats relatifs à leur comportement en tant que boîtes noires. C'est pourquoi nous proposons dans ces travaux une méthode permettant de prouver que deux modèles, vu comme des boîtes noires, sont équivalents et peuvent être utilisés indifféremment lors d'une analyse.

Le premier chapitre débute par une présentation de l'entreprise DASSAULT SYSTÈMES et de la suite logicielle DELMIA. Ceci amène à définir la problématique et les objectifs de ces travaux. Enfin, la présentation de l'exemple utilisé tout au long de ce document, un préhenseur pneumatique, clôt ce chapitre.

Le second chapitre est consacré au choix du formalisme temporisé utilisé. Après une présentation de différents formalismes de SED temporisés, notre choix se porte sur les Automates Temporisés à Variables Discrètes. Comme ce formalisme n'est pas outillé à l'heure actuelle, une méthode de traduction en Réseaux d'Automates Temporisés d'UPPAAL<sup>1</sup> est fournie.

Le troisième chapitre traite de la construction d'un modèle à évolutions réalistes d'un système bouclé et se compose de trois parties :

- L'obtention d'un modèle de partie opérative issu d'instances de modèles génériques amène des problèmes de concurrence entre modèles produisant des évolutions irréalistes. En utilisant le mécanisme d'urgence fourni par le formalisme utilisé, nous supprimons ce comportement.
- En supposant que la commande du système bouclé est décrite sous forme d'une spécification Grafcet, nous construisons un modèle du contrôleur basé sur le cycle d'un Automate Programmable Industriel et sur des équations algébriques représentant le Grafcet. La prise en compte de l'aspect temporisé de la spécification nous pousse à utiliser des modèles annexes de temporisations. Ces modèles ajoutés au modèle de la partie opérative forment le modèle du système contrôlé, commandé par le modèle du contrôleur.
- La mise en parallèle de tous les modèles (contrôleur, partie opérative et temporisations) permet de construire le modèle du système bouclé mais apporte un comportement bloquant au niveau du modèle du système contrôlé. L'ajout de variables et la modification du modèle du contrôleur nous permettent de supprimer ce blocage. Cependant, ces modifications entraînent un manque de réactivité et un comportement anormal au niveau du modèle du contrôleur. Il convient donc de le modifier à nouveau pour lui permettre d'être conforme à la norme Grafcet en termes de stabilité et de réactivité. On obtient alors un modèle du système bouclé à évolutions réalistes.

Le quatrième chapitre présente une méthode permettant de prouver l'équivalence entre deux modèles de granularité différente. L'analyse de nos besoins et des équivalences de la littérature nous amène à définir et à utiliser une équivalence en trace pour les automates temporisés que nous avons choisis. Le principe des automates observateurs couplés à un model-checker, méthode retenue pour la vérification de l'équivalence, est ensuite présenté. Pour finir, deux parties importantes décrivent :

- La vérification de l'équivalence en trace de deux modèles. En se limitant initialement à des modèles non temporisés et émetteurs seulement, des problèmes de concurrence imposent que l'automate observateur initialement prévu soit transformé en un automate observateur-séquenceur capable de limiter les évolutions des modèles si nécessaire. Le passage à des modèles temporisés fait apparaître des problèmes de décalage temporel des traces à comparer, ce qui impose une modification de l'auto-

---

1. <http://www.uppaal.com>

---

mate observateur-séquenceur afin de permettre le gel d'un des modèles comparés. Enfin, l'utilisation de modèles temporisés et réagissant à des entrées comme modèles à comparer fait apparaître de nouvelles concurrences avec les modèles générant les entrées. Ceci nous impose l'ajout d'un automate contrôlant les modèles des entrées et une nouvelle modification de l'automate observateur-séquenceur. On obtient alors une méthode permettant de prouver l'équivalence en trace de deux modèles d'un système bouclé temporisé.

- La définition et la vérification d'une équivalence avec tolérance. Les difficultés rencontrées pour produire un modèle abstrait strictement équivalent au modèle détaillé d'un système bouclé nous incitent à proposer une définition d'une équivalence en trace avec une tolérance sur les valeurs des variables, une tolérance temporelle, puis une tolérance à la fois sur les valeurs et temporelle. Une modification de la méthode de vérification précédente permet alors de vérifier l'équivalence avec tolérance de deux modèles d'un système bouclé temporisé.

Le dernier chapitre conclut quant aux apports de ces travaux et propose des perspectives à court, moyen et long terme.



# Chapitre 1

## Contexte et problématique

### Sommaire

---

<b>1.1</b>	<b>Contexte industriel</b>	<b>6</b>
1.1.1	Présentation de la suite DELMIA V6	6
1.1.2	Représentation d'une chaîne de production	7
<b>1.2</b>	<b>Problématique</b>	<b>8</b>
1.2.1	Caractéristiques du modèle détaillé	8
1.2.2	Caractéristiques du modèle abstrait	9
<b>1.3</b>	<b>Objectifs de la thèse</b>	<b>9</b>
<b>1.4</b>	<b>Présentation de l'exemple</b>	<b>10</b>
<b>1.5</b>	<b>Organisation du mémoire</b>	<b>14</b>

---

## 1.1 Contexte industriel

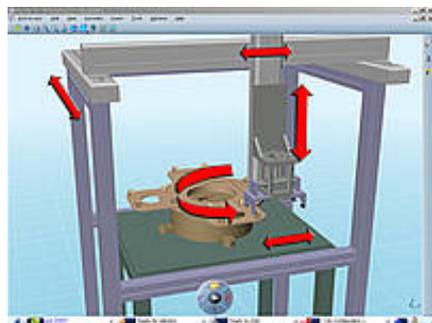
### 1.1.1 Présentation de la suite Delmia V6

La société DASSAULT SYSTÈMES est le leader mondial dans le domaine des logiciels d'aide à la conception, simulation et gestion de produits industriels. Dans son offre logicielle, la suite DELMIA est plus spécifiquement destinée à la conception et l'analyse de systèmes de production industriels. L'arrivée de la suite DELMIA V6 (figure 1.1) marque un tournant majeur dans la manière dont tous les acteurs de la production interagissent, amenant plus de simplicité et d'efficacité dans leur travail.

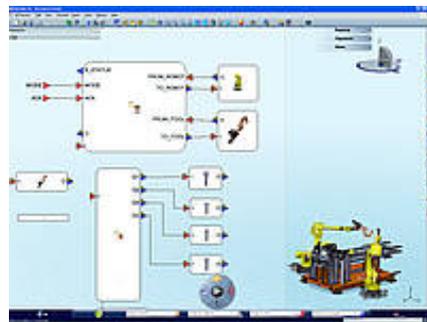
En effet, au travers de l'atelier DELMIA *Mechanical Device Builder* (figure 1.1a), l'utilisateur va construire un modèle détaillé et complet en 3D des processus utilisés dans une usine. Ce modèle est inclus dans un *Smart Device*, conteneur qui va concentrer tous les comportements (physique, logique, ...) relatifs à un objet technologique. Dans l'atelier DELMIA *Smart Device Builder* (figure 1.1b), le *Smart Device* va être enrichi par une logique représentant son comportement. C'est à cette occasion qu'un modèle de contrôleur se voit ajouter son code exécuté, par exemple, sous forme d'un Ladder Diagram ou d'un SFC<sup>2</sup>, entre autres.

Une fois les modèles des composants créés, on peut réaliser différentes analyses, comme la simulation grâce à l'atelier DELMIA *Virtual Controls Validation* (figure 1.1c), qui permet

2. Sequential Function Chart



(a) Atelier *Mechanical Device Builder*



(b) Atelier *Smart Device Builder*



(c) Atelier *Virtual Controls Validation*



(d) Atelier *Process Planning*

FIGURE 1.1 – Exemples d'ateliers DELMIA V6

de reconstituer un système bouclé : la commande est implémentée dans un Automate Programmable Industriel réel relié à un ordinateur utilisant le modèle virtuel du processus pour simuler la partie opérative.

Une autre possibilité consiste à associer plusieurs modèles de systèmes bouclés au travers de l'atelier DELMIA *Process Planning* (figure 1.1d) afin de fabriquer le modèle complet d'une chaîne de production. Les modèles simulés conjointement permettent alors de lancer des analyses temporelles.

### 1.1.2 Représentation d'une chaîne de production

La figure 1.2 montre schématiquement une chaîne de production manufacturière décrite dans l'atelier *Process Planning*. Elle est composée de trois stations (stations 1 à 3) elles-mêmes composées de postes agissant sur un produit, la station 2 étant en charge de l'assemblage de deux éléments. Tous les postes communiquent entre eux au travers d'un réseau de terrain. Nous considérons que les postes sont tous modélisés par des *Smart Devices* représentant le plus fidèlement possible le comportement de ceux-ci. Ces modèles peuvent à leur tour être composés de *Smart Devices* modélisant les différents composants du poste (moteurs, vérins pneumatiques, contrôleurs, ...).

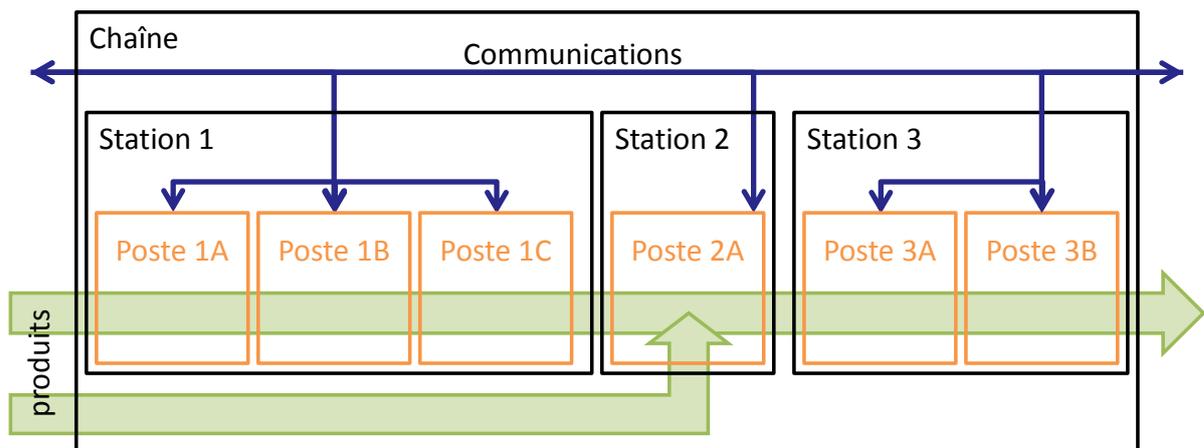


FIGURE 1.2 – Schéma d'une chaîne de production.

La figure 1.3 représente pour sa part le poste 1C de la chaîne de la figure 1.2 que l'on souhaite analyser en détail.

On remarque alors que, pour analyser le poste 1C, il est nécessaire d'utiliser les modèles des autres postes. En supposant que les modèles de ces derniers sont aussi détaillés que celui du poste étudié (composés de multiples *Smart Devices*), cette analyse va mobiliser un grand nombre de modèles à simuler conjointement.

DASSAULT SYSTÈMES est désireux de fournir à ses clients des solutions toujours plus performantes, traitant des usines toujours plus importantes avec un niveau de détail toujours plus élevé, ce qui a pour effet de démultiplier le nombre de modèles à utiliser conjointement.

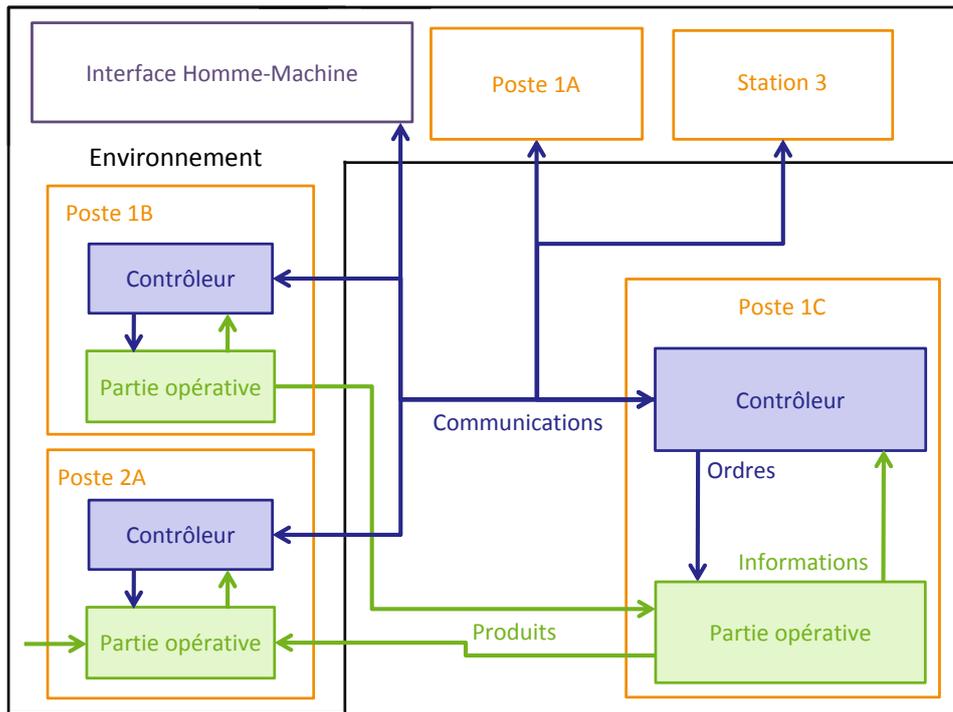


FIGURE 1.3 – Détail du poste 1C et de son environnement.

Pour garantir une utilisation de la suite DELMIA V6 sans soucis de performance, y compris sur des supports disposant de peu de puissance de calcul, il est donc nécessaire de pouvoir réduire le nombre et la taille des modèles à utiliser sans sacrifier le résultat des analyses.

## 1.2 Problématique

Le constat est le suivant : lors de l'analyse d'un poste d'une chaîne de production, l'utilisation d'un modèle détaillé pour le poste étudié est *indispensable*, mais *a contrario*, l'utilisation de modèles détaillés des postes de l'environnement pose de nombreuses difficultés.

La solution usuelle consiste alors à utiliser des modèles *abstrait*s des postes de l'environnement. On a donc pour chaque poste  $P_i$  de la chaîne de production deux modèles :

- Un détaillé, noté  $M_{P_i}^D$ , utilisé pour les analyses centrées sur le poste.
- Un abstrait, noté  $M_{P_i}^A$ , utilisé lorsque le comportement du poste à l'échelle de la chaîne est nécessaire mais que le poste n'est pas le centre de l'analyse.

### 1.2.1 Caractéristiques du modèle détaillé

Le modèle détaillé  $M_{P_i}^D$ , représenté dans la figure 1.4a, a les caractéristiques suivantes :

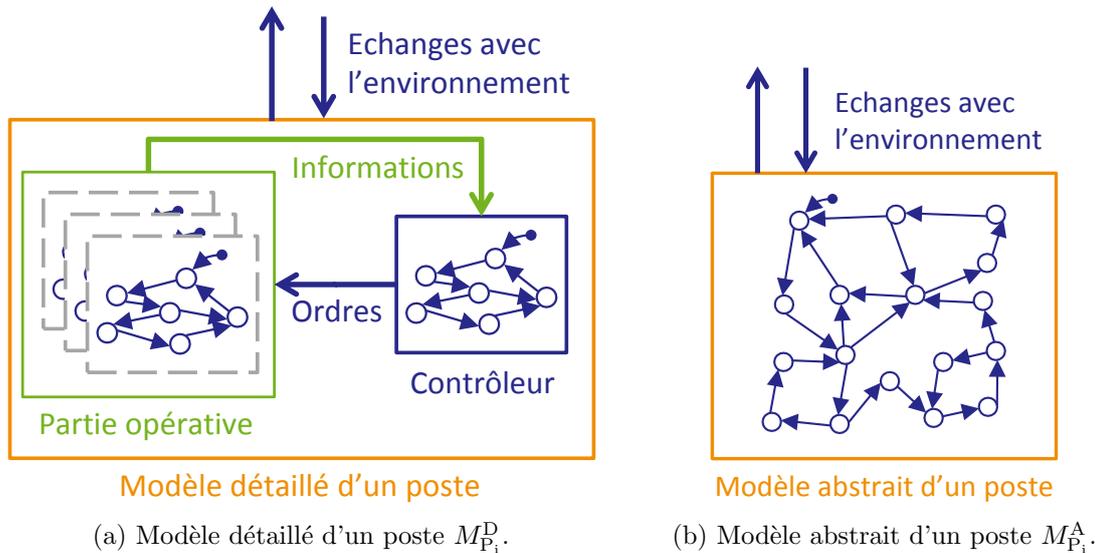


FIGURE 1.4 – Modèles détaillé et abstrait d'un poste.

- Ce modèle décrit un système bouclé, composé d'une partie opérative couplée à un contrôleur.
- Ce modèle est représenté sous la forme d'un Système à Événements Discrets (SED) car nous nous intéressons uniquement aux évolutions dues à des événements.
- Ce SED est générateur et accepteur puisqu'il doit communiquer avec d'autres SED modélisant les autres postes de la chaîne de production.
- Ce modèle est temporisé pour mieux représenter le système bouclé qu'il modélise.
- Ce modèle ne doit comporter que des évolutions et états *réalistes* pour garantir le résultat des analyses.

## 1.2.2 Caractéristiques du modèle abstrait

Le modèle abstrait  $M_{P_i}^A$ , représenté dans la figure 1.4b, a les caractéristiques suivantes :

- Ce modèle décrit le même système bouclé que le modèle détaillé  $M_{P_i}^D$  mais sans détailler tous les états et évolutions internes.
- Ce modèle est aussi représenté par un SED temporisé, générateur et accepteur afin de pouvoir être utilisé conjointement avec d'autres modèles détaillés  $M_{P_j}^D$  et abstraits  $M_{P_k}^A$ .
- Pour garantir les résultats d'une analyse utilisant ce modèle à la place du modèle détaillé  $M_{P_i}^D$ , il doit lui être équivalent au niveau de son comportement relatif aux entrées (aspect accepteur) et sorties (aspect générateur), comme montré dans la figure 1.5.

## 1.3 Objectifs de la thèse

Les apports escomptés de ces travaux sont donc :

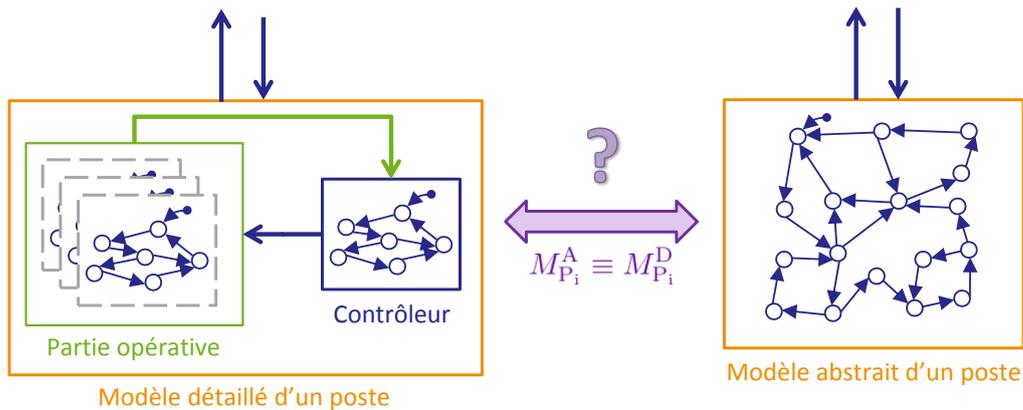


FIGURE 1.5 – Équivalence  $\equiv$  entre  $M_{P_i}^A$  et  $M_{P_i}^D$ .

### Une méthode de conception d'un modèle détaillé $M_{P_i}^D$ à évolutions réalistes.

Ce modèle étant celui d'un système bouclé, la méthode proposée considérera tout d'abord la conception du modèle de la partie opérative. Pour prendre en considération la réalité industrielle qui se base usuellement sur l'assemblage de composants, une approche modulaire sera retenue pour construire ce modèle. On s'attachera ensuite à construire le modèle du contrôleur. Il sera alors supposé que la spécification a été décrite en Grafset (norme [IEC 60848 \(2002\)](#)) et que ce contrôleur est implémenté dans un Automate Programmable Industriel à scrutation cyclique de ses entrées.

### Une méthode permettant de prouver l'équivalence entre les modèles détaillé $M_{P_i}^D$ et abstrait $M_{P_i}^A$ en terme d'entrées/sorties.

L'équivalence est définie après une étude des besoins et des équivalences proposées dans la littérature. La méthode permettant de prouver l'équivalence est basée sur la technique des automates observateurs associée à un outil de vérification formelle. Elle sera implantée en utilisant l'outil de vérification intégré à la suite UPPAAL<sup>3</sup>. De plus, une proposition d'équivalence avec tolérance sera faite pour pallier les difficultés rencontrées lors de la conception de modèles abstraits strictement équivalents aux modèles détaillés.

## 1.4 Présentation de l'exemple

Tout au long de cette thèse, la conception d'un modèle détaillé à évolutions réalistes ainsi que la preuve d'équivalence sont mises en application sur un cas d'étude qui doit répondre à certains impératifs :

- Sa taille doit être suffisante pour tester la capacité de passage à l'échelle des méthodes, sans être trop importante pour qu'il reste lisible et compréhensible comme cas d'étude.
- La spécification de sa commande doit contenir les structures classiques (parallélisme, sélection de séquence).

3. [www.uppaal.org](http://www.uppaal.org)

- Il doit avoir des comportements temporisés, au niveau de la partie opérative (consommation de temps lors des mouvements des pièces) comme dans son contrôleur (utilisation de temporisations dans la spécification).



FIGURE 1.6 – Exemple de manipulateur pneumatique.

Notre choix s'est porté sur un manipulateur de type portique dont un exemple est montré dans la figure 1.6. Ce manipulateur est chargé de distribuer des pièces aux autres postes de la station de test et d'usinage représentée dans la figure 1.7.

Cette station a pour rôle de tester des pièces issues d'une station précédente et de procéder à une série d'opérations d'usinage sur ces dernières avant de les transmettre à la station suivante. Elle est précisément constituée des postes suivants :

- Un poste de prise des pièces, interface avec la station précédente.
- Un poste de test des pièces, noté Testeur, qui permet de déterminer le type des pièces.
- Un centre d'usinage, noté M1, qui usine les pièces de type 1 avec une zone de prise des pièces et une zone de sortie des pièces.
- Un centre d'usinage, noté M2, qui usine les pièces de type 2 avec une zone de prise des pièces et une zone de sortie des pièces.
- Un manipulateur, chargé de la manutention des pièces entre les postes (non représenté sur la figure 1.7 par souci de lisibilité).
- Un poste de dépose, interface avec la station suivante.

Le manipulateur réalise les opérations suivantes :

1. Prise d'une pièce au poste de prise.
2. Déplacement de la pièce au poste de test.
3. En fonction du résultat :
  - (a) Déplacement puis dépose de la pièce à la zone de prise des pièces du centre d'usinage M1.
  - (b) Déplacement puis dépose de la pièce à la zone de prise des pièces du centre d'usinage M2.
4. Déplacement vers la zone de sortie des pièces du centre d'usinage correspondant puis attente de la sortie de la pièce.

5. Reprise de la pièce à la sortie du centre d'usinage correspondant.
6. Déplacement puis libération de la pièce au poste de dépose des pièces.

Le manipulateur est un portique permettant des déplacements dans deux directions (axes X et Y représentés sur la figure 1.7) au travers de vérins pneumatiques. Ce choix d'actionneur est motivé par le faible poids des pièces transportées et le faible nombre de positions d'arrêt associés à sa simplicité de mise en œuvre et à son faible coût d'achat. Sur chacun de ces axes, trois positions caractéristiques sont détectées par des capteurs, elles sont représentées par des pointillés sur la figure 1.7. Un troisième vérin, vertical selon l'axe Z, permet de faire descendre une ventouse associée à une pompe à vide chargée de maintenir la pièce transportée. Pour ce vérin, deux positions particulières sont détectées : haut et bas. La partie opérative est contrôlée au moyen d'un Automate Programmable Industriel (API) qui exécute un code déduit d'une spécification Grafacet.

La figure 1.8 montre toutes les entrées-sorties du contrôleur du manipulateur, et en particulier :

**Les entrées/sorties échangées avec la partie opérative** sont présentées dans la figure 1.8a :

- H\_X\_G\_OUT et H\_X\_G\_IN, ordres relatifs aux déplacements de sortie (vers la droite) et de rentrée (vers la gauche) de la tige du vérin de l'axe horizontal X.
- H\_X\_IN, H\_X\_MID et H\_X\_OUT, informations des trois capteurs de position associés aux positions de rentrée, de mi-course et de sortie de la tige du vérin de l'axe

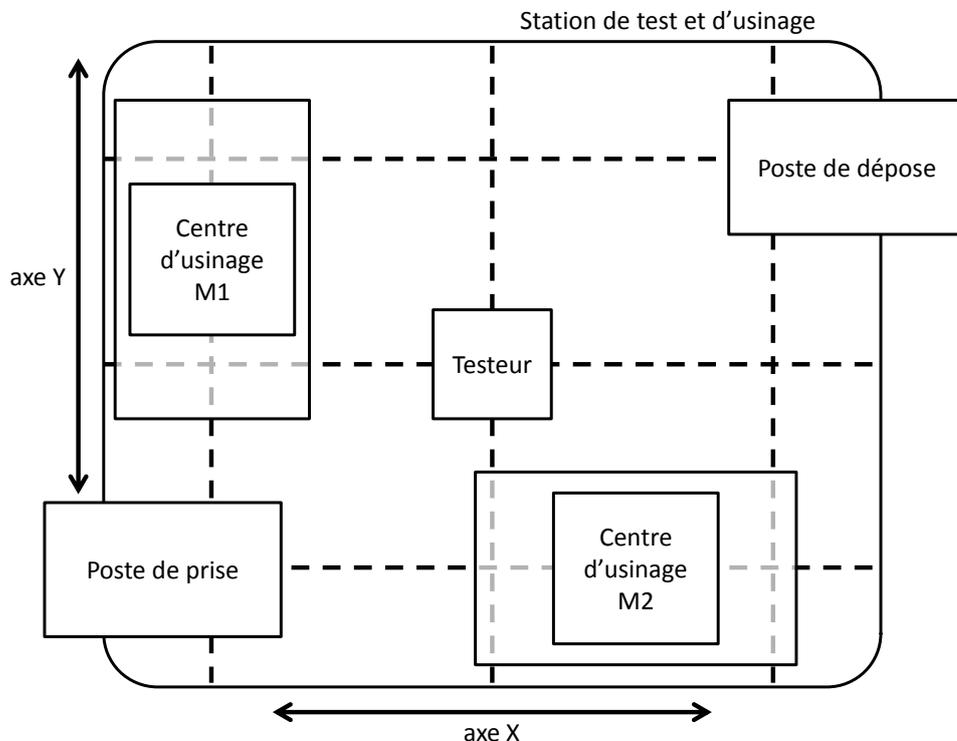


FIGURE 1.7 – Station de test et d'usinage, le manipulateur n'est pas représenté par souci de lisibilité.

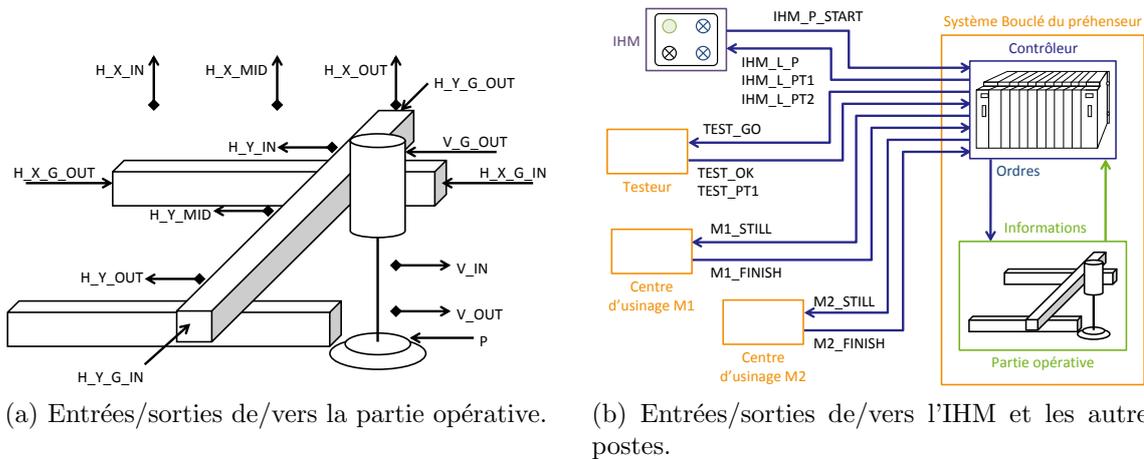


FIGURE 1.8 – Détail des Entrées et Sorties du contrôleur du manipulateur.

horizontal X.

- H\_Y\_G\_OUT et H\_Y\_G\_IN, ordres relatifs aux déplacements de sortie (vers l'avant) et de rentrée (vers l'arrière) de la tige du vérin de l'axe horizontal Y.
- H\_Y\_IN, H\_Y\_MID et H\_Y\_OUT, informations des trois capteurs de position associés aux positions de rentrée, de mi-course et de sortie de la tige du vérin de l'axe horizontal Y.
- V\_G\_OUT, ordre relatif au déplacement de sortie (vers le bas) de la tige du vérin de l'axe vertical Z. Le mouvement de rentrée de tige (vers le haut) est initié dès que l'ordre V\_G\_OUT cesse.
- V\_IN et V\_OUT, informations des deux capteurs de position associés aux positions de rentrée et de sortie de la tige du vérin de l'axe vertical Z.
- P, ordre de mise en route de la pompe à vide.

**Les entrées/sorties échangées avec l'Interface Homme-Machine (notée IHM) et les autres postes** sont décrites dans la figure 1.8b :

- IHM\_P\_START est la variable associée au bouton poussoir utilisé pour lancer le cycle API.
- IHM\_L\_P est la variable associée au voyant utilisé pour avertir du fonctionnement de la pompe à vide.
- IHM\_L\_PT1 et IHM\_L\_PT2 sont deux variables associées aux voyants utilisés après le test pour informer du type de pièce détecté (respectivement pièce de type 1 et pièce de type 2).
- TEST\_GO est une variable utilisée par le contrôleur pour autoriser le début de la séquence de test du Testeur.
- TEST\_OK et TEST\_PT1 sont deux variables utilisées par le Testeur. La première signifie au contrôleur du manipulateur que le test est terminé, et la seconde donne alors le verdict : si elle est vraie, alors la pièce est de type 1, sinon la pièce est de type 2.
- M1\_STILL est utilisée par le contrôleur du manipulateur pour interdire au centre

- d'usinage M1 de réaliser des actions dans ses zones de prise et de sortie de pièces car le manipulateur agit dans ces zones.
- M1\_FINISH est utilisée par le centre d'usinage M1 pour signifier la fin de l'usinage de la pièce en cours. Cette variable n'est remise à faux que lorsque le centre d'usinage est à nouveau disponible pour usiner une pièce.
  - M2\_STILL et M2\_FINISH ont des utilisations et comportements semblables mais relatifs au centre d'usinage M2.
  - Dans le cadre de ces travaux, nous ne considérerons pas les communications avec les postes de prise des pièces et de dépose des pièces afin de ne pas surcharger le cas d'étude.

## 1.5 Organisation du mémoire

Le chapitre 2 est consacré à la présentation du formalisme à événements discrets temporisé nécessaire à la réalisation du modèle détaillé  $M_{P_i}^D$ . Dans un premier temps, une étude des différentes familles de formalismes nous permet de définir la famille qui correspond le mieux à nos attentes et besoins. Dans un deuxième temps, une étude des différents formalismes de cette famille nous amène à choisir un formalisme de modélisation simple et efficace (les Automates Temporisés à Variables Discrètes) dont les sémantiques sont détaillées. Cependant, le manque d'outil pour ce formalisme nous oblige à utiliser la suite logicielle UPPAAL pour nos expérimentations. Des règles de traduction d'Automates Temporisés à Variables Discrètes en automates utilisés dans UPPAAL sont donc finalement données.

La conception du modèle détaillé  $M_{P_i}^D$  est précisée dans le chapitre 3. Pour commencer, la modélisation de la partie opérative est abordée et des problèmes de concurrence amenant à des évolutions irréalistes sont traités. Ensuite, la modélisation du contrôleur exécutant un code basé sur une spécification Grafcet est décrite, en particulier l'intégration des temporisations. Enfin, la mise en commun des modèles de la partie opérative et du contrôleur permet de mettre en avant de nouveaux problèmes de concurrence et de réactivité qui sont traités.

Le chapitre 4 traite de la vérification de l'équivalence entre le modèle détaillé  $M_{P_i}^D$  et le modèle abstrait  $M_{P_i}^A$ . En analysant les types d'équivalence existants et nos besoins, le choix se porte sur une équivalence en trace. Une méthode de vérification utilisant des automates observateurs est proposée afin de prouver l'équivalence entre les modèles détaillé et abstrait. L'étude des attentes des utilisateurs du modèle abstrait nous pousse ensuite à considérer le cas d'une équivalence avec tolérance et à en proposer une définition. L'adaptation de la méthode de vérification précédemment proposée à cette nouvelle équivalence avec tolérance conclut ce chapitre.

Enfin, le dernier chapitre présente une conclusion à ces travaux et propose d'autres voies à explorer.

## Chapitre 2

# Formalismes utilisés dans ces travaux

Ce chapitre est essentiellement consacré à la présentation du formalisme de description des modèles : les Automates Temporisés à Variables Discrètes (ATVD). Quel que soit l'intérêt de ce formalisme (possibilité de modéliser l'urgence et manipulation de variables entières, en particulier), il n'en demeure pas moins vrai qu'aucun outil d'analyse le supportant n'a été développé. Ceci explique que des règles de traduction des modèles en ATVD dans le formalisme de la suite logicielle UPPAAL, sans perte ni ajout de sémantique, doivent être proposées. Ces règles sont essentielles pour la validation de nos propositions (modélisation détaillée réaliste et preuve d'équivalence entre modèles détaillé et abstrait).

## Sommaire

---

<b>2.1</b>	<b>Choix d'un formalisme</b>	<b>17</b>
2.1.1	Contraintes	17
2.1.2	Breve présentation de quelques classes d'automates temporisés	17
<b>2.2</b>	<b>Formalisme de modélisation : les Automates Temporisés à Variables Discrètes (ATVD)</b>	<b>20</b>
2.2.1	Variables, expressions et contraintes	20
2.2.2	Sémantique des ATVD sans urgence	21
2.2.3	Ajout de la sémantique d'urgence pour les ATVD	24
2.2.4	Réseau d'automates temporisés à variables discrètes	26
2.2.5	Illustration des évolutions sur un exemple	27
<b>2.3</b>	<b>Passage des ATVD aux RATU</b>	<b>28</b>
2.3.1	Variables, expressions et localités	28
2.3.2	Traduction des transitions non urgentes	29
2.3.3	Traduction de la sémantique d'urgence des ATVD	30
2.3.4	Traduction d'un réseau d'ATVD en un réseau d'automates temporisés d'UPPAAL	31
<b>2.4</b>	<b>Conclusion</b>	<b>31</b>

---

## 2.1 Choix d'un formalisme

### 2.1.1 Contraintes

Nous avons déjà indiqué, au chapitre 1, que les modèles détaillés et abstraits doivent être temporisés. De plus, afin de modéliser l'évolution de grandeurs autres que le temps, ils doivent intégrer des variables numériques, entières ou réelles. Enfin ces travaux ne portent que sur des comportements sans défaillance, ce qui exclut les formalismes stochastiques.

Les formalismes hybrides possèdent toutes ces caractéristiques recherchées mais ils présentent de grandes difficultés de passage à l'échelle (Denis *et al.*, 2006; Juarez Orozco, 2008) et ne sont donc pas adaptés à des analyses portant sur des modèles importants, ce qui les exclut.

Les familles de formalismes restantes sont donc les automates et réseaux de Petri temporisés. Comme nous souhaitons, dans le modèle détaillé, décrire *explicitement* tous les états, seuls les automates temporisés conviennent.

### 2.1.2 Brève présentation de quelques classes d'automates temporisés

#### 2.1.2.1 Graphes temporisés

En 1990, Alur, Courcoubetis et Dill définissent les bases des systèmes temporisés au travers des graphes temporisés (Alur *et al.*, 1990) qui peuvent évoluer en suivant des arcs temporisés, utilisant des comparaisons entre une valeur d'horloge et des entiers comme gardes. De plus, il est possible de remettre à zéro (*reset*) les horloges lors du franchissement d'un arc. La figure 2.1 reprend un exemple de Alur *et al.* (1990).

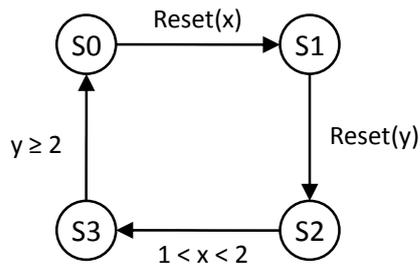


FIGURE 2.1 – Graphe temporisé avec deux horloges  $x$  et  $y$ .  $S_0$  est l'état initial.

#### 2.1.2.2 Automates temporisés

En faisant évoluer ce formalisme, Alur et Dill produiront en 1994 l'article fondateur sur le formalisme des automates temporisés (Alur, 1994) basé sur les systèmes de transitions temporisées (Henzinger *et al.*, 1992a). Ce formalisme reprend le principe des graphes temporisés en remplaçant le *reset* par une assignation ( $:=$ ) à zéro et ajoute des étiquettes sur les transitions, permettant un certain degré de communication entre différents automates

(tir synchrone de transitions). Cependant, ce formalisme pêche toujours par le fait que les évolutions de l'automate ne sont pas contraintes. En effet, sans invariants sur les états limitant les valeurs des horloges, rien n'interdit à l'automate de patienter *ad aeternam* dans un état actif. On peut donc attendre suffisamment longtemps pour arriver dans une situation de blocage (*deadlock*) où les valeurs des horloges ne permettent plus de franchir aucune transition sortante de l'état actif, figeant alors l'automate. La figure 2.2 montre un exemple d'automate temporisé où, si l'état actif est S2 et on attend 3 unités de temps, alors on arrive à un blocage (puisque'il n'y a plus aucune transition franchissable depuis S3).

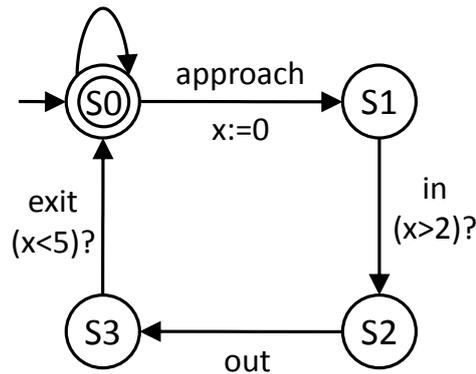


FIGURE 2.2 – Automate temporisé modélisant un train ; *approach*, *in*, *out* et *exit* sont des étiquettes,  $x$  est une horloge, les gardes sont notées à l'aide d'un « ? » en suffixe. S0 est l'état initial (arc entrant) ainsi que l'état marqué de l'automate (double cercle).

### 2.1.2.3 Automates de sécurité

En parallèle à ces évolutions, en 1992, [Henzinger et al.](#) proposent une extension des graphes temporisés/automates temporisés sous la forme d'automates temporisés de sécurité (*timed safety automata*) qui intègrent des invariants sur les états. Cet ajout permet de contraindre les évolutions de l'automate et réduit drastiquement les temps de vérification via model-checking tout en facilitant la modélisation. On note aussi l'apparition du concept de localité (*location*) qui remplace le concept d'état (*state*), l'état de l'automate n'étant plus simplement représenté par sa localité active mais aussi par les valeurs de ses horloges. La figure 2.3 est tirée de [Henzinger et al. \(1992b\)](#). On peut remarquer que la localité initiale n'est pas précisée sur le graphe et qu'il n'y a plus de localité marquée (contrairement à l'automate de la figure 2.2).

### 2.1.2.4 Réseaux d'Automates Temporisés d'UPPAAL (RATU)

Présenté dans la thèse de [Pettersson \(1999\)](#) et servant de formalisme de base pour le logiciel UPPAAL, ce formalisme a apporté plusieurs avancées au fil de son développement :

- Une sémantique utilisant des variables discrètes.
- Des invariants pouvant contenir des conditions booléennes.

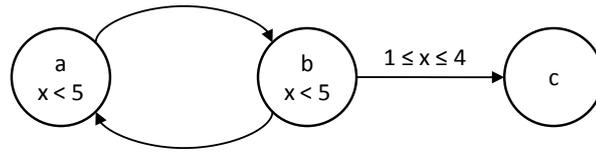


FIGURE 2.3 – Automate temporisé de sécurité avec l’horloge  $x$  et les localités  $a$  (initiale),  $b$  et  $c$ .

- Une sémantique de synchronisation par canaux de communication.
- Une sémantique d’urgence, qui peut être utile pour modéliser de façon réaliste des systèmes temporisés non triviaux comme il a été montré par [Perin et Faure \(2009\)](#), au travers des localités et des canaux de communication.

La figure 2.4 est tirée de [Pettersson \(1999\)](#) et montre un exemple d’automate d’un RATU. On peut y remarquer :

- Le nom des localités en italique (comme «  $b$  »), la localité initiale (ici, «  $a$  ») étant représentée par un double cercle.
- Les synchronisations au travers de canaux de communication (comme «  $UP?$  ») sont écrites en majuscules suivies d’un «  $!$  » pour les actions d’émission sur le canal ou d’un «  $?$  » pour celles de réception.
- Les gardes des transitions sont en police standard, telles que «  $Volt \geq 1$  », et les assignations (de variables ou d’horloges) en gras, comme «  $Volt := Volt - 1$  ».

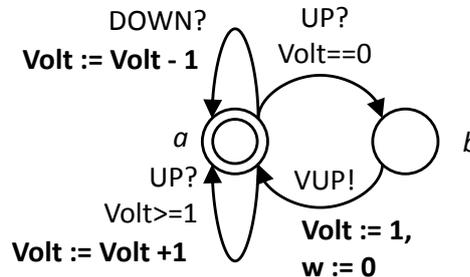


FIGURE 2.4 – Exemple d’automate extrait de [Pettersson \(1999\)](#).

Cependant, ce formalisme souffre d’un défaut important : au vu des nombreuses possibilités qu’il offre, la complexité des sémantiques mises en œuvre dans les RATU impose une attention toute particulière lors de la conception des modèles pour garantir leur cohérence. Or notre volonté est de proposer une méthode *simple* permettant la conception de modèles détaillés d’un système bouclé ; l’effort doit donc être porté sur la modélisation et non sur la maîtrise du formalisme utilisé. Compte tenu de la puissance du formalisme des RATU et des difficultés qui en découlent, nous n’utiliserons pas ce dernier pour la construction de modèles de systèmes bouclés.

### 2.1.2.5 Automates Temporisés à Variables Discrètes (ATVD)

Comme les RATU, ce formalisme décrit en détail par [Janowska et Janowski \(2006\)](#) permet :

- D'utiliser des variables discrètes,
- D'associer des invariants aux localités,
- D'utiliser des canaux de communications,
- De modéliser l'urgence.

Cette dernière possibilité n'est cependant possible qu'au moyen de transitions urgentes. Cette limitation ne constitue pas selon nous un inconvénient. Elle permet en effet, comme il sera montré dans le chapitre 3, de modéliser correctement toutes les évolutions des systèmes bouclés et évite les erreurs potentielles dues à l'utilisation de plusieurs mécanismes de modélisation de l'urgence comme le permettent les RATU. Ceci explique que nous avons retenu le formalisme des ATVD pour la construction de nos modèles.

Un exemple simple d'ATVD, sans urgence, est donné dans la figure 2.5, les conventions de représentation de ce formalisme sont majoritairement identiques à celles utilisées pour les RATU :

- Les noms des localités sont toujours en italique (comme « *s\_send* »), la localité initiale (« *s\_init* » dans l'exemple) est par contre représentée avec une transition source qui porte les valeurs initiales des variables comme assignation.
- Les synchronisations au travers de canaux de communication (comme « *rec\_ack?* ») sont suivies d'un « ! » pour les actions d'émission sur le canal ou d'un « ? » pour celles de réception.
- Les gardes, sur les variables discrètes, et contraintes temporelles, sur les horloges, des transitions sont en police standard, telles que « *s\_ack = s\_bit* » et « *x = T* », et les assignations (de variables ou d'horloges), en gras, comme « ***x := 0*** ».

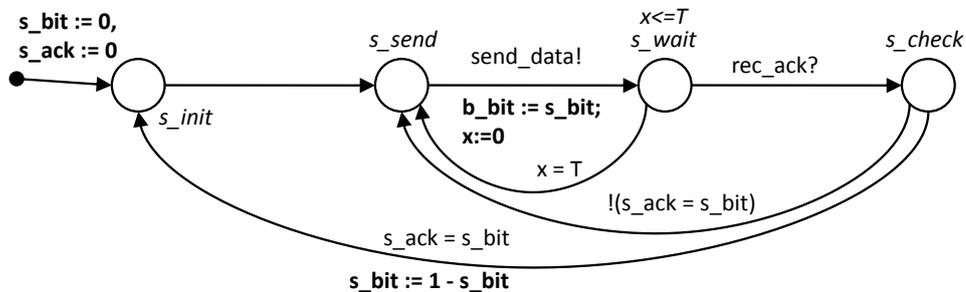


FIGURE 2.5 – Automate temporisé à variables discrètes, tiré de [Janowska et Janowski \(2006\)](#), « ! » est l'opérateur booléen  $\neg$  en préfixe dans les gardes.

## 2.2 Formalisme de modélisation : les Automates Temporisés à Variables Discrètes (ATVD)

### 2.2.1 Variables, expressions et contraintes

Soit  $V = V_b \cup V_z$  l'ensemble fini des variables discrètes associées à l'automate temporisé à variables discrètes (ATVD)  $A^{ATVD}$ , composé de l'ensemble  $V_b$  des variables booléennes, à valeurs dans  $\{true, false\}$ , et de l'ensemble  $V_z$  des variables entières, à valeurs dans  $\mathbb{Z}$ .

## 2.2. Formalisme de modélisation : les Automates Temporisés à Variables Discrètes (ATVD)

L'ensemble  $Expr(V_z)$  de toutes les expressions arithmétiques sur l'ensemble des variables entières  $V_z$  est alors défini comme :

$$\forall expr \in Expr(V_z), expr ::= z | v_z | expr \diamond expr | - expr | (expr) \quad (2.1)$$

avec  $z \in \mathbb{Z}$ ,  $v_z \in V_z$ , et  $\diamond \in \{+, -, \times, /\}$ , en prenant  $/$  comme la division euclidienne.

L'ensemble  $B(V)$  de toutes les expressions booléennes sur l'ensemble des variables discrètes (booléennes *et* entières)  $V$  est défini par :

$$\forall b \in B(V), b ::= true | expr \# expr | b \wedge b | b \vee b | \neg b | (b) | v_b \quad (2.2)$$

où  $expr \in Expr(V_z)$  est une expression sur les *variables entières*,  $v_b \in V_b$  et  $\# \in \{<, \leq, =, \geq, >\}$ .  $\neg true$  est aussi noté *false* pour plus de simplicité.

Remarque :  $b \in B(V)$  est une fonction sur  $V = V_b \cup V_z$  à valeurs dans  $\{true, false\}$ . L'ensemble  $A(V)$  de toutes les **actions** sur les variables discrètes  $V$  est défini par :

$$\forall a \in A(V), a ::= \epsilon | v_z := expr | v_b := b | a, a \quad (2.3)$$

où  $\epsilon$  est l'action nulle,  $v_z \in V_z$ ,  $v_b \in V_b$ ,  $b \in B(V)$  et  $expr \in Expr(V_z)$ . L'opérateur « , » correspond à un opérateur d'agrégation qui permet de grouper plusieurs actions en une seule. On remarque que l'assignation  $v_b := b$  avec  $b \in B(V)$  inclut les assignations particulières  $v_b := true$  et  $v_b := false$ .

De plus,  $C$  est défini comme l'ensemble des variables spéciales, appelées *horloges*, à valeurs dans  $\mathbb{R}^+$ . Ceci permet de définir l'ensemble des contraintes temporelles  $T(C)$  :

$$\forall t \in T(C), t ::= true | c \# n | t \wedge t \quad (2.4)$$

avec  $c \in C$ ,  $n \in \mathbb{N}$ , et  $\# \in \{<, \leq, =, \geq, >\}$ .

On remarque ici que l'on ne peut associer plusieurs contraintes temporelles qu'au travers de l'opérateur booléen de conjonction ( $\wedge$ ) ; ceci est dû à la volonté de garder un espace convexe pour les valeurs des horloges validant la contrainte temporelle associée à une transition.

Ce formalisme permet donc de manipuler **des booléens**, éléments de  $V_b$ , **des entiers**, éléments de  $V_z$ , et **des réels** sous forme d'horloges, éléments de  $C$ .

### 2.2.2 Sémantique des ATVD sans urgence

Les définitions précédentes permettent de définir un automate temporisé à variables discrètes comme un 7-uplet  $(L, l_0, V, \tilde{V}_0, C, E, I)$  avec :

- $L$  un ensemble fini de localités.
- $l_0 \in L$  la localité initiale.
- $V$  l'ensemble des variables discrètes,  $v \in V$  une variable de cet ensemble,  $\tilde{v}$  la valeur de cette variable et  $\tilde{V}$  l'ensemble des valeurs de toutes les variables de  $V$ .  $\tilde{V}_0$  est la valeur des variables à l'initialisation.

- $C$  un ensemble d’horloges,  $c \in C$  une horloge,  $\tilde{c} \in \mathbb{R}^+$  la valeur de l’horloge à un instant donné et  $\tilde{C}$  l’ensemble des valeurs de toutes les horloges. À l’initialisation, toutes les horloges sont à 0.
- $E \subseteq L \times B(V) \times T(C) \times A(V) \times 2^C \times L$  un ensemble de transitions partant d’une localité source  $l \in L$  vers une localité cible  $l' \in L$ , avec une garde  $g \in B(V)$  (formule 2.2), une contrainte temporelle  $t \in T(C)$  (formule 2.4) et une action  $a \in A(V)$  (formule 2.3). Le terme  $2^C$  est à relier à la remise à zéro des horloges : chaque horloge de  $C$  peut être remise à zéro ou non lors du franchissement de la transition.  $r \subseteq C$  est défini comme le sous-ensemble (potentiellement vide) d’horloges à remettre à zéro lors du tir de la transition.
- $I : L \rightarrow T(C)$  (formule 2.4) une fonction qui assigne un invariant à chaque localité.

La sémantique d’un ATVD est donnée ci-dessous en utilisant les notations supplémentaires suivantes :

- $\tilde{C} \models I(l)$  signifie que les valeurs des horloges vérifient l’invariant de la localité  $l$ .
- $\tilde{V} \models g$  signifie que les valeurs des variables vérifient la garde  $g$  associée à une transition  $e \in E$ .
- $\tilde{C} \models t$  signifie que les valeurs des horloges vérifient la contrainte temporelle  $t$  de la transition  $e$ .
- $[r \mapsto 0]\tilde{C}$  correspond aux modifications des valeurs des horloges telles que les horloges de  $r$  sont remises à zéro.
- $[V_a \xrightarrow{a} \mathbb{Z} \cup \{true, false\}]\tilde{V}$  correspond à la modification des valeurs des variables de  $V_a \subseteq V$  relatives à l’action  $a$ .

La sémantique d’un ATVD est alors définie par un système de transitions  $\langle S, s_0, \rightarrow \rangle$  avec l’ensemble des états  $S$ , un état  $s \in S$  étant un triplet  $(l, \tilde{V}, \tilde{C})$ ,  $s_0 = (l_0, \tilde{V}_0, \tilde{C}_0)$  décrivant l’état initial, et  $\rightarrow \subseteq S \times S$  la relation de transition telle que :

$$(l, \tilde{V}, \tilde{C}) \rightarrow (l, \tilde{V}, \tilde{C} + d) \text{ si} \tag{2.5}$$

$$\forall c \in C, \forall d \in \mathbb{N}, \forall d' \in [0; d], \tilde{c} + d' \models I(l)$$

$$(l, \tilde{V}, \tilde{C}) \rightarrow (l', \tilde{V}', \tilde{C}') \text{ si} \tag{2.6}$$

$$\exists e : l \xrightarrow{g, t, a, r} l', \tilde{V} \models g, \tilde{C} \models t, \tilde{C}' = [r \mapsto 0]\tilde{C}, \tilde{V}' = [V_a \xrightarrow{a} \mathbb{Z} \cup \{true, false\}]\tilde{V}$$

$$\text{et } \tilde{C}' \models I(l')$$

La sémantique 2.5 correspond à la sémantique temporelle des ATVD. Elle permet l’évolution de toutes les horloges, en accord avec les invariants de la localités active.

La sémantique 2.6 correspond quant à elle à la sémantique de franchissement d’une transition. Elle permet de changer de localité active, au regard de la garde et de la contrainte temporelle de la transition qui doivent toutes deux être vérifiées. De plus,

les valeurs des horloges doivent vérifier l'invariant de la localité d'arrivée. Ce franchissement permet aussi d'agir sur les variables au travers de l'action associée à la transition, et éventuellement de remettre à zéro des horloges.

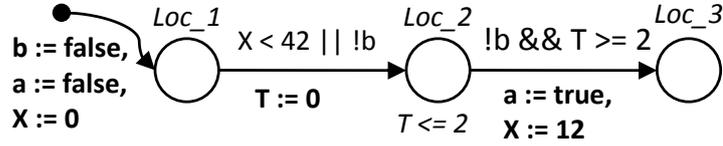


FIGURE 2.6 – Un automate temporisé à variables discrètes.

La figure 2.6 montre un exemple d'ATVD avec l'horloge  $T$ , deux variables booléennes  $a$  et  $b$ , et une variable entière  $X$ .

- Les localités (ici,  $Loc_1$ ,  $Loc_2$  et  $Loc_3$ ) sont représentées par des cercles, les noms étant en italique.
- La localité initiale (ici,  $Loc_1$ ) est représentée avec une transition source portant l'état initial des variables en gras (ici, les deux variables booléennes sont initialisées à faux, et la variable entière à 0). Les horloges ne sont pas explicitées lors de l'initialisation car elles sont toujours initialisées à 0. Dans une optique de lisibilité, sur des modèles comportant beaucoup de variables, celles initialisées à 0 ou faux ne seront pas notées sur la transition source.
- Les invariants attachés aux localités (comme  $T \leq 2$  pour la localité  $Loc_2$ ) sont en italique.
- Les transitions sont représentées par des flèches allant de la localité source à la localité cible.
- Les gardes et contraintes temporelles sont écrites près des transitions et sont combinées par une disjonction ou une conjonction notées respectivement «  $||$  » et «  $\&\&$  ». L'opérateur complémentaire booléen est noté «  $!$  ». La notation  $!b \ \&\& \ T \geq 2$  est un exemple de combinaison de garde et de contrainte temporelle.
- Les actions associées à une transition sont notées en utilisant l'opérateur d'assignation «  $:=$  » en gras. Les remises à zéro des horloges utilisent le même opérateur, en limitant l'assignation à la valeur 0. Un exemple est donné dans la transition allant de  $Loc_2$  à  $Loc_3$ .

Ces deux sémantiques 2.5 et 2.6 ne sont pas exclusives et peuvent donc être en concurrence, ce qui peut amener à des cas de non-déterminisme, qui peuvent aussi survenir à cause d'une concurrence entre deux évolutions utilisant la même sémantique.

La figure 2.7 montre un exemple d'un ATVD où des évolutions sont en concurrence. En supposant la localité initiale  $Loc_A$  active, on peut avoir les évolutions suivantes :

$Loc_A \rightarrow Loc_B$  : possible grâce à la sémantique 2.6 et à la garde *true* toujours vérifiée.

$Loc_A \rightarrow Loc_D$  : possible grâce à la même sémantique et à la garde *true* toujours vérifiée. À la différence de la précédente évolution, celle-ci a pour effet d'assigner la variable booléenne  $a$  à vrai.

$\mathbf{T} = \mathbf{0} \rightarrow \mathbf{T} = \mathbf{2}$ , **Loc\_A toujours active** : possible grâce à la sémantique 2.5 et à l'invariant de Loc\_A autorisant cette évolution.

La transition **Loc\_A**  $\rightarrow$  **Loc\_E** n'est par contre pas franchissable en raison de sa garde, fautive dans l'état initial.

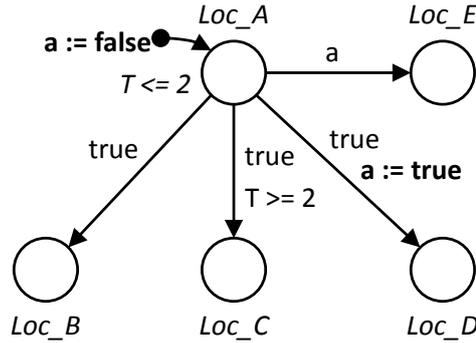


FIGURE 2.7 – Exemple d'ATVD avec trois évolutions concurrentes.

Le formalisme ne donne *a priori* aucune priorité sur une sémantique vis-à-vis d'une autre, ni même entre deux évolutions dues à la même sémantique.

### 2.2.3 Ajout de la sémantique d'urgence pour les ATVD

L'une des principales caractéristiques des ATVD est la possibilité d'ajout d'un comportement d'urgence sur les transitions. Ceci est réalisé au travers d'un attribut d'urgence  $u \in \{\text{true}, \text{false}\}$  ajouté à chaque transition. Si cet attribut est mis à vrai (*true*), alors la transition est nommée *transition urgente* et elle a priorité sur les évolutions des horloges. Si cet attribut est laissé à faux (*false*), la transition reste classique sans aucune différence de comportement avec la définition précédente.

La syntaxe des ATVD est alors modifiée comme suit : une transition devient un 7-uplet  $e = (l, g, t, u, a, r, l')$  avec deux localités  $l$  et  $l'$ , une garde  $g$ , une contrainte temporelle  $t$ , l'attribut d'urgence  $u$ , une action  $a$  et un ensemble d'horloges à remettre à zéro  $r$ . Lorsque l'attribut d'urgence est mis à vrai, la contrainte temporelle associée à la transition est toujours vérifiée ( $t = \text{true}$ ).

Avec cette modification, la sémantique complète d'un ATVD est définie par un système de transitions  $\langle S, s_0, \rightarrow \rangle$  avec l'ensemble des états  $S$ , un état  $s \in S$  étant un triplet  $(l, \tilde{V}, \tilde{C})$ ,  $s_0 = (l_0, \tilde{V}_0, \tilde{C}_0)$  décrivant l'état initial, et  $\rightarrow \subseteq S \times S$  la relation de transition telle que :

$$\begin{aligned}
 (l, \tilde{V}, \tilde{C}) &\rightarrow (l, \tilde{V}, \tilde{C} + d) \text{ si} \\
 &\forall c \in C, \forall d \in \mathbb{N}, \forall d' \in [0; d], \tilde{c} + d' \models I(l), \text{ et} \\
 &\forall e = (l, g, t, u, a, r, l'), (u = \text{false}) \vee ((u = \text{true}) \wedge ((\tilde{V} \not\models g) \vee (\tilde{C} \not\models I(l'))))
 \end{aligned} \tag{2.7}$$

$$\begin{aligned}
 (l, \tilde{V}, \tilde{C}) &\rightarrow (l', \tilde{V}', \tilde{C}') \text{ si} \\
 &\exists e : l \xrightarrow{g, t, u, a, r} l', \tilde{V} \models g, \tilde{C} \models t, \tilde{C}' = [r \mapsto 0]\tilde{C}, \tilde{V}' = [V_a \xrightarrow{a} \mathbb{Z} \cup \{\text{true}, \text{false}\}]\tilde{V} \\
 &\text{et } \tilde{C}' \models I(l')
 \end{aligned} \tag{2.8}$$

La sémantique temporelle 2.7 est fortement impactée par l'ajout de la sémantique d'urgence. En effet, l'évolution des horloges est interdite lorsqu'il existe une transition urgente franchissable ; il faut donc vérifier cette condition avant toute utilisation de cette sémantique. La sémantique de franchissement d'une transition 2.8 n'est pas impactée par l'ajout de la sémantique d'urgence.

Les concurrences possibles entre évolutions sont aussi impactées par l'ajout de l'attribut d'urgence : lorsqu'une transition urgente est franchissable (localité source active, invariant de la localité d'arrivée vérifié et garde vérifiée), la sémantique temporelle (2.7) est interdite, ce qui supprime la concurrence entre le tir d'une transition urgente et une évolution des horloges.

D'autres concurrences sont cependant encore possibles :

- Si aucune transition urgente n'est franchissable, on retrouve la concurrence entre les deux sémantiques : tir d'une transition non urgente (sémantique 2.8) et évolution des horloges (sémantique 2.7),
- La concurrence interne à la sémantique 2.8 est inchangée. Si plusieurs transitions, *urgentes ou non*, sont franchissables, elles sont en concurrence.

L'exemple de la figure 2.7 est repris avec la transformation d'une transition non urgente en transition urgente représentée par une flèche doublée. Les évolutions possibles sont alors les suivantes :

**Loc\_A** → **Loc\_B** : possible grâce à la sémantique 2.8 (qui concerne aussi les transitions urgentes) et à la garde *true* toujours vérifiée.

**Loc\_A** → **Loc\_D** : possible grâce à la même sémantique et à la garde *true* toujours vérifiée.

La transition **Loc\_A** → **Loc\_E** est toujours infranchissable (garde fausse) mais la transition **T = 0** → **T = 2**, **Loc\_A toujours active** devient aussi infranchissable car la transition **Loc\_A** → **Loc\_B** étant *franchissable et urgente*, les évolutions d'horloges (sémantique 2.7) sont interdites.

Les transitions **Loc\_A** → **Loc\_B** (urgente) et **Loc\_A** → **Loc\_D** (non urgente) utilisent toutes les deux la même sémantique 2.8 et sont par conséquent en concurrence : l'attribut d'urgence *ne donne pas* la priorité sur les transitions non urgentes.

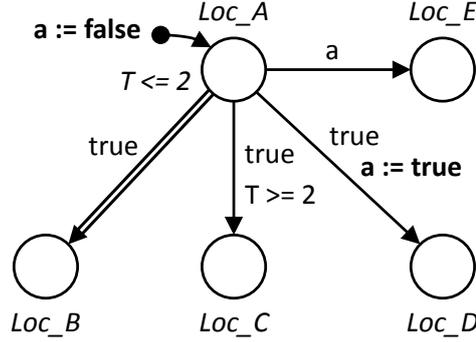


FIGURE 2.8 – Exemple d'ATVD avec une transition urgente et deux évolutions concurrentes.

## 2.2.4 Réseau d'automates temporisés à variables discrètes

Un réseau d'ATVD est un ensemble d'automates  $A_i^{\text{ATVD}} = (L_i, l_i^0, V, \tilde{V}_0, C, E_i, I_i)$  avec  $i \in \mathbb{N}^*$ . Les ensembles d'horloges ( $C$ ) et de variables ( $V$ ) sont communs à tous les automates. Le réseau  $\{A_i^{\text{ATVD}}\} = (\bar{L}, \bar{l}_0, V, \tilde{V}_0, C, E, I)$  est alors défini par :

- $\bar{L} = (L_1 \times L_2 \cdots \times L_n)$  l'ensemble de toutes les localités.
- $\bar{l} = (l_1, l_2, \dots, l_n)$  le vecteur des localités actives du réseau.
- $\bar{l}_0 = (l_1^0, l_2^0, \dots, l_n^0)$  le vecteur des localités initiales.
- $V$  et  $C$ , les ensembles communs d'horloges et de variables.  $\tilde{V}_0$  est l'ensemble des valeurs initiales des variables communes. Les horloges sont toujours initialisées à 0.
- $E = \bigcup_i E_i$  l'ensemble des transitions.
- $I(\bar{l}) = \bigwedge_j I_j(l_j)$ , la fonction d'invariant commune.
- $\bar{l}[l_i \mapsto l'_i]$  représente le changement de la  $i^{\text{ème}}$  valeur du vecteur  $\bar{l}$  de  $l_i$  à  $l'_i$ .

La sémantique d'un réseau d'automates temporisés à variables discrètes est alors définie par un système de transitions  $\langle S, s_0, \rightarrow \rangle$  où un état  $s \in S$  est un vecteur  $(\bar{l}, \tilde{V}, \tilde{C})$ , où  $s_0 = (\bar{l}_0, \tilde{V}_0, \tilde{C}_0)$  est l'état initial et  $\rightarrow \subseteq S \times S$  est la relation de transition telle que :

$$\begin{aligned}
 (\bar{l}, \tilde{V}, \tilde{C}) &\rightarrow (\bar{l}, \tilde{V}, \tilde{C} + d) \text{ si} \\
 &\forall c \in C, \forall d \text{ in } \mathbb{N}, \forall d' \in [0; d], \tilde{c} + d' \models I(\bar{l}), \text{ et} \\
 &\forall e = (\bar{l}, g, t, u, a, r, \bar{l}'), (u = \text{false}) \vee ((u = \text{true}) \wedge ((\tilde{V} \not\models g) \vee \tilde{C} \not\models I(l')))
 \end{aligned} \tag{2.9}$$

$$\begin{aligned}
 (\bar{l}, \tilde{V}, \tilde{C}) &\rightarrow (\bar{l}' = \bar{l}[l_i \mapsto l'_i], \tilde{V}', \tilde{C}') \text{ si} \\
 &\exists e_i : l_i \xrightarrow{g, t, u, a, r} l'_i, \tilde{V} \models g, \tilde{C} \models t, \tilde{C}' = [r \mapsto 0]\tilde{C}, \tilde{V}' = [V_a \xrightarrow{a} \mathbb{Z} \cup \{\text{true}, \text{false}\}]\tilde{V} \\
 &\text{et } \tilde{C}' \models I(\bar{l}')
 \end{aligned} \tag{2.10}$$

L'article de Janowska [Janowska et Janowski \(2006\)](#) propose une sémantique de tir synchrone de transitions de plusieurs automates. Cette possibilité n'a pas été retenue car inutile pour notre problème, comme il sera montré au chapitre 3. En effet, les communications entre automates se feront au travers de variables communes.

Les concurrences à l'intérieur d'un réseau d'ATVD sont semblables à celles rencontrées sur un seul ATVD :

- Si aucune transition urgente n'est franchissable (parmi toutes les transitions du réseau), une concurrence est possible entre la sémantique temporelle 2.9 et la sémantique 2.10 de tir d'une transition,
- Une concurrence est *toujours* possible au travers de la sémantique 2.10 entre les transitions franchissables du réseau, qu'elles soient urgentes ou non.

### 2.2.5 Illustration des évolutions sur un exemple

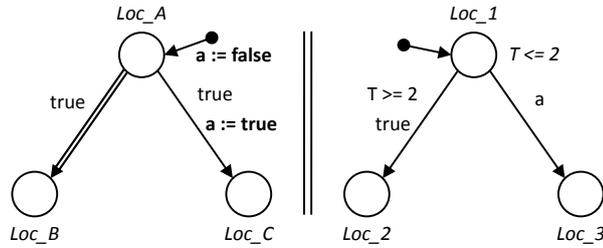


FIGURE 2.9 – Réseau composé de deux ATVD.

La figure 2.9 représente un réseau simple d'ATVD avec :

- $\bar{L} = \{Loc\_A, Loc\_B, Loc\_C\} \times \{Loc\_1, Loc\_2, Loc\_3\}$
- $\bar{l}_0 = (Loc\_A, Loc\_1)$
- $V = V_b \cup V_z = \{a\} \cup \emptyset$
- $\tilde{V}_0 = \{a = false\}$
- $C = \{T\}$
- $I(l) : Loc\_1 \mapsto T \leq 2$
- La transition urgente **Loc\_A**  $\rightarrow$  **Loc\_B** est représentée par une flèche doublée.

Dans cet exemple, les gardes toujours vérifiées ( $g = true$ ) ont été représentées. Dans le reste de ces travaux, ce ne sera plus le cas pour des raisons de lisibilité et de simplicité. De même, l'initialisation de « a » à faux est marquée, ce ne sera plus le cas sur les modèles suivants pour des raisons de lisibilité (seules les initialisations de variables à une autre valeur que faux ou 0 seront notées).

L'utilisation des sémantiques présentées dans la partie 2.2.4 permet de définir les évolutions possibles de ce réseau depuis l'état initial (Loc\_A et Loc\_1 actives, variable a à faux et horloge T à 0) :

**Loc\_A**  $\rightarrow$  **Loc\_B** est possible grâce à la garde toujours vraie (selon 2.10). Le prochain vecteur des localités actives sera alors (Loc\_B, Loc\_1).

**Loc\_A**  $\rightarrow$  **Loc\_C** est aussi possible pour la même raison (selon 2.10). Le prochain vecteur des localités actives sera alors (Loc\_C, Loc\_1) et la variable  $a$  sera mise à vrai (*true*).

On remarque alors que, comme précédemment noté, les transitions urgentes n'ont aucune priorité sur les transitions non urgentes.

Les autres évolutions ne sont pas possibles car :

**Loc\_1**  $\rightarrow$  **Loc\_2** nécessite une évolution du temps pour vérifier la contrainte temporelle sur T. Or comme la transition  $Loc_A \rightarrow Loc_B$  est franchissable et de type urgent, la sémantique d'évolution du temps est interdite. L'horloge T est donc bloquée à 0, rendant cette transition impossible à tirer.

**Loc\_1**  $\rightarrow$  **Loc\_3** requiert que la variable  $a$  soit vraie (*true*), ce qui n'est pas le cas dans l'état initial.

Si l'on suppose que la transition  $Loc_A \rightarrow Loc_C$  a été franchie, la variable  $a$  a été mise à vrai et le vecteur des localités actives est (Loc\_C, Loc\_1). Les évolutions possibles sont alors :

**Loc\_1**  $\rightarrow$  **Loc\_3** car la variable  $a$  est maintenant vraie.

**T=0**  $\rightarrow$  **T=2** car la transition urgente  $Loc_A \rightarrow Loc_B$  n'est plus franchissable. Cette évolution du temps permet ensuite de franchir la transition **Loc\_1**  $\rightarrow$  **Loc\_2** puisque la contrainte temporelle est alors vérifiée.

## 2.3 Passage des ATVD aux RATU

La modélisation se fait au travers des ATVD, choisis pour la simplicité de leur sémantique. Cependant, la volonté d'utiliser des outils de preuve formelle impose de passer par l'outil UPPAAL, et donc par son formalisme de RATU (détaillé dans l'annexe A). Il convient donc de définir clairement comment passer d'un ATVD à un RATU, sans perte ni ajout non maîtrisé de sémantique.

### 2.3.1 Variables, expressions et localités

Les définitions des variables (entières et booléennes) ainsi que celles des horloges des deux formalismes sont identiques.

**Règle 1 :** Les ensembles de variables et d'horloges d'un réseau d'ATVD sont directement traduits en leurs équivalents dans le RATU. De plus, les valeurs initiales des variables et horloges sont les mêmes.

Les définitions des expressions relatives aux variables entières et booléennes sont identiques et sont donc conservées.

Les localités d'un automate UPPAAL peuvent avoir des attributs d'urgence (urgent et obligatoire / *urgent* et *committed*) qui n'existent pas dans le formalisme des ATVD, d'où :

**Règle 2 :** Chaque localité d'un ATVD est traduite en une localité sans aucun attribut d'urgence dans l'automate UPPAAL équivalent. La localité initiale de chaque ATVD est traduite en une localité initiale dans l'automate UPPAAL équivalent.

Les invariants attachés aux localités d'un ATVD se limitent aux valeurs des horloges là où UPPAAL permet d'utiliser aussi les valeurs des variables, ce qui implique :

**Règle 3 :** Les invariants attachés aux localités d'un ATVD sont traduits directement dans UPPAAL.

On peut noter à cette occasion que l'ensemble des expressions autorisées dans un invariant d'un ATVD est inclus dans l'ensemble des expressions autorisées pour un invariant d'un RATU (voir définition de l'ensemble  $D$  dans l'équation A.1), ce qui permet une traduction simple et directe.

### 2.3.2 Traduction des transitions non urgentes

Les définitions des transitions diffèrent entre les deux formalismes :

- Les transitions non urgentes d'un ATVD sont des 7-uplets  $e = (l, g, t, u = false, a, r, l')$
- Les transitions d'un RATU sont des 6-uplets  $f = (l, g', \sigma, a, r, l')$

Les deux différences résident au niveau des gardes/contraintes temporelles des ATVD et des gardes des RATU ainsi qu'au niveau des actions de communication  $\sigma$  des transitions des RATU qui n'existent pas dans les transitions des ATVD.

La garde d'une transition d'un automate UPPAAL combine la garde et la contrainte temporelle d'un ATVD. On peut ainsi définir :

$$\forall (g, t), g' = g \wedge t \quad (2.11)$$

avec  $(g, t)$  un couple garde-contrainte temporelle d'une transition d'un ATVD, et  $g'$  la garde de la transition de l'automate équivalent du RATU.

De plus, les canaux de communication ne sont pas utilisés pour les transitions non urgentes. On peut alors définir la règle de traduction des transitions non urgentes :

**Règle 4a :** Pour toutes les transitions non urgentes  $e = (l_e, g, t, u = false, a_e, r_e, l'_e)$  d'un ATVD, on définit la transition équivalente  $f = (l_f, g', \sigma, a_f, r_f, l'_f)$  dans UPPAAL telle que :

- $l_f = l_e$
- $l'_f = l'_e$
- $g' = g \wedge t$
- $\sigma = \tau$
- $a_f = a_e$
- $r_f = r_e$

où  $\tau$  est l'absence de synchronisation dans la sémantique des RATU (voir l'annexe A.2).

### 2.3.3 Traduction de la sémantique d'urgence des ATVD

Les mécanismes d'urgence dans les RATU se basent sur deux concepts :

- L'activité de certaines localités (urgentes et obligatoires),
- L'aspect franchissable de transitions utilisant des canaux de communication urgents.

Le premier concept ne peut pas être utilisé pour traduire le mécanisme d'urgence proposée dans les ATVD qui s'appuie sur l'aspect franchissable des transitions urgentes, ainsi seule la seconde possibilité peut être envisagée.

Cette solution est présentée par Behrmann *et al.* (2004) :

- Un canal de communication urgent, nommé *FAST* dans nos travaux, va être ajouté si des transitions urgentes d'un ATVD doivent être traduites dans un RATU.
- Chaque transition d'un automate UPPAAL qui est la traduction d'une transition urgente d'un ATVD se voit ajouter une action de communication d'émission via ce canal de communication urgent.
- Pour que la sémantique d'urgence des canaux de communication d'UPPAAL puisse fonctionner, il faut un récepteur à cette action d'émission sur le canal *FAST*. Pour ce faire, un automate est ajouté : son seul rôle est de fournir une transition toujours disponible pour une action de réception sur le canal de communication urgent *FAST*.

L'automate ajouté est représenté dans la figure 2.10, il est nommé automate de priorisation et se compose d'une unique localité (logiquement initiale) et d'une unique transition bouclant sur celle-ci. Comme il doit fournir en permanence une transition franchissable avec une action de synchronisation sur le canal urgent *FAST*, cette transition a une garde toujours vérifiée. Afin de ne pas influencer le comportement des modèles, cette transition n'a pas d'action sur les variables ni sur les horloges.

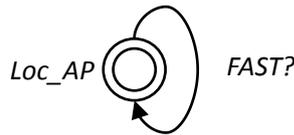


FIGURE 2.10 – Automate de priorisation utilisant le canal urgent *FAST*.

**Ainsi cet automate n'a pas d'influence sur les variables, il ne modifie pas le comportement initial du réseau, mais permet de traduire la sémantique d'urgence des ATVD dans un réseau d'automates temporisés d'UPPAAL.**

On définit donc la règle de traduction des transitions urgentes d'un ATVD :

**Règle 4b :** S'il existe au moins une transition urgente dans le réseau d'ATVD, on ajoute un automate de priorisation (figure 2.10) et, pour toutes les transitions urgentes  $e = (l_e, g, t = true, u = true, a_e, r_e, l'_e)$  d'un ATVD, on définit la transition équivalente  $f = (l_f, g', \sigma, a_f, r_f, l'_f)$  dans UPPAAL telle que :

- $l_f = l_e$
- $l'_f = l'_e$
- $g' = g$

- $\sigma = FAST!$
- $a_f = a_e$
- $r_f = r_e$

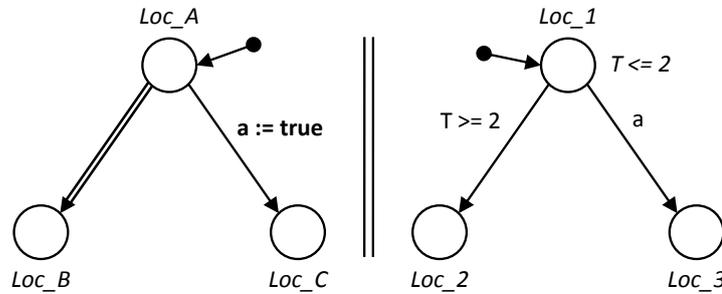
### 2.3.4 Traduction d'un réseau d'ATVD en un réseau d'automates temporisés d'UPPAAL

Pour des raisons de lisibilité, les règles 4a et 4b sont associées dans une règle unique de traduction des transitions :

**Règle 4 :** Chaque transition  $e = (l_e, g, t, u, a_e, r_e, l'_e)$  d'un ATVD est traduite en une transition  $f = (l_f, g', \sigma, a_f, r_f, l'_f)$  dans UPPAAL telle que :

- $l_f = l_e$
- $l'_f = l'_e$
- $g' = g \wedge t$
- $a_f = a_e$
- $r_f = r_e$
- S'il existe une transition urgente ( $\exists u = true$ ), alors on ajoute un automate de priorisation (figure 2.10)
- Si  $u = true$ ,  $\sigma = FAST!$
- Si  $u = false$ ,  $\sigma = \tau$  (non synchronisation au travers du canal FAST)

La figure 2.11a reprend l'exemple utilisé pour présenter la sémantique des ATVD. La figure 2.11b montre sa traduction en automates UPPAAL en utilisant les quatre règles précédemment données.



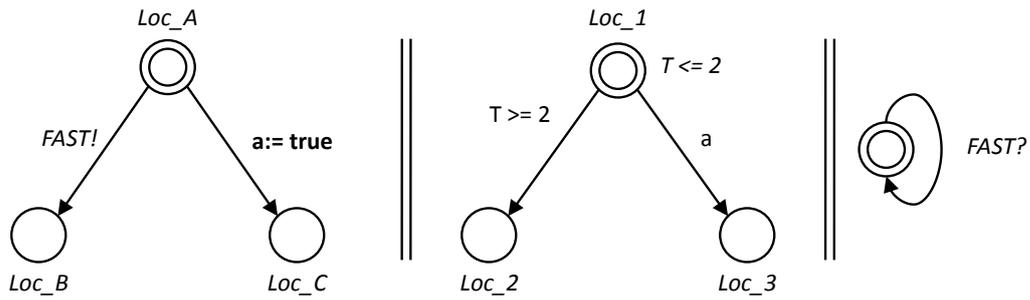
(a) Réseau d'ATVD à traduire

FIGURE 2.11 – Exemple de traduction d'un réseau d'ATVD en RATU(1).

On remarque alors que les seules évolutions possibles pour les ATVD ( $Loc\_A \rightarrow Loc\_B$  et  $Loc\_A \rightarrow Loc\_C$ ) sont aussi les seules possibles pour le RATU, les concurrences entre transitions urgentes et transitions sans contraintes d'horloge sont conservées.

## 2.4 Conclusion

Ce chapitre consacré aux formalismes utilisés a permis de choisir les automates temporisés comme formalisme de base pour nos travaux. Notre volonté de faciliter la phase



(b) Traduction en réseau d'automates temporisés d'UPPAAL

FIGURE 2.11 – Exemple de traduction d'un réseau d'ATVD en RATU(2).

de modélisation en choisissant un formalisme modélisant de manière simple le traitement de variables entières et l'urgence nous a poussés à choisir, dans la section 2.1, le formalisme des automates temporisés à variables discrètes (ATVD), présenté dans les travaux de [Janowska et Janowski \(2006\)](#) et détaillé dans la section 2.2. Cependant, l'absence d'outil pour ce formalisme nous pousse à utiliser UPPAAL et son formalisme, les RATU, détaillés dans l'annexe A, pour pouvoir utiliser des outils d'analyse formelle qui seront nécessaires dans le chapitre 4.

La section 2.3 permet de traduire simplement les modèles obtenus sous forme d'un réseau d'ATVD en un RATU analysable par la suite logicielle UPPAAL.

# Chapitre 3

## Construction du modèle détaillé à évolutions réalistes d'un système bouclé

Ce chapitre traite de la conception du modèle détaillé (noté  $M_{P_i}^D$ ) d'un système bouclé décrit au moyen d'Automates Temporisés à Variables Discrètes (ATVD). Au vu du rôle central tenu par ce modèle à plusieurs égards (phase de conception et de réglage de la commande du système bouclé  $P_i$  et référent de l'équivalence), nous attachons une importance particulière à sa capacité à ne produire que des évolutions reflétant celles du système réel.

La première section traite de la conception du modèle de la partie opérative qui est constitué d'un réseau d'ATVD étant des instances de modèles génériques de composants. Le mécanisme d'urgence des ATVD est ensuite utilisé pour supprimer des concurrences entre modèles amenant des évolutions irréalistes.

La conception du modèle du contrôleur est abordée dans la seconde section. Une spécification Grafset de la commande est implantée dans le modèle du contrôleur sous forme de jeux d'équations algébriques. Les temporisations sont alors ajoutées sous forme d'automates annexes.

Enfin, la troisième section traite de la conception d'un modèle du système bouclé. La mise en commun des différents modèles (partie opérative, contrôleur et temporisations) amène des blocages du modèle de la partie opérative et un comportement du modèle du contrôleur non satisfaisant. La modification de certains modèles et l'ajout de variables partagées permet finalement d'obtenir un modèle détaillé à évolutions réalistes du système bouclé.

## Sommaire

---

<b>3.1</b>	<b>Construction du modèle de partie opérative</b>	<b>35</b>
3.1.1	État de l'art	35
3.1.2	Principes de modélisation retenus	36
3.1.3	Modélisation de la partie opérative du manipulateur	44
3.1.4	Mise en évidence des concurrences internes	48
3.1.5	Suppression des concurrences indésirables	51
<b>3.2</b>	<b>Construction du modèle du contrôleur</b>	<b>52</b>
3.2.1	Hypothèses	52
3.2.2	État de l'art	52
3.2.3	Modélisation d'un contrôleur non temporisé	53
3.2.4	Modèle final, temporisé, du contrôleur	59
<b>3.3</b>	<b>Construction du modèle du système bouclé</b>	<b>63</b>
3.3.1	Prise en compte des phases de lecture et d'écriture des entrées/sorties de l'API	64
3.3.2	Concurrence entre modèles du système contrôlé et du contrôleur	65
3.3.3	Suppression des concurrences indésirables avec conservation de la réactivité du contrôleur	69
3.3.4	Modélisation d'un contrôleur avec recherche de stabilité	70
3.3.5	Synthèse des ajouts	74
<b>3.4</b>	<b>Conclusion</b>	<b>75</b>

---

Ce chapitre est consacré à la modélisation d'un système bouclé, composé d'un unique contrôleur et d'une partie opérative constituée par un assemblage de composants standards (moteurs, vérins, capteurs, ...), ce qui est une solution usuelle dans l'industrie.

Ceci explique que le modèle de la partie opérative est construit de manière modulaire par assemblage de modèles de composants, comme montré dans la figure 3.1. Ces derniers sont eux-mêmes des instances de modèles génériques représentant le comportement de ces classes de composants.

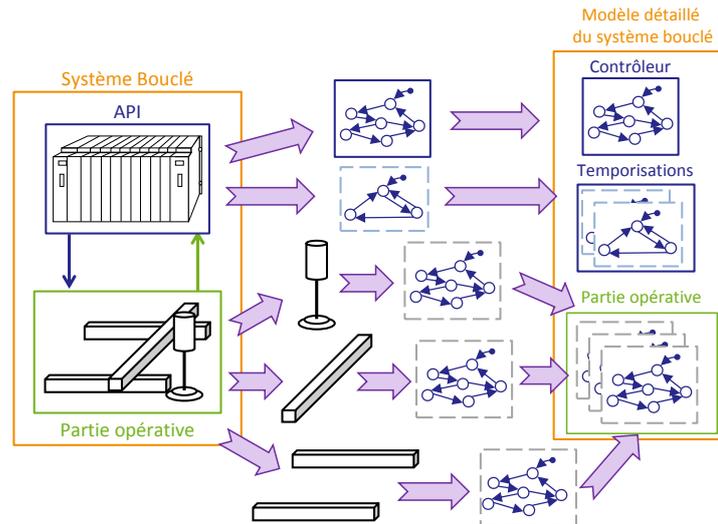


FIGURE 3.1 – Approche de modélisation modulaire.

## 3.1 Construction du modèle de partie opérative

### 3.1.1 État de l'art

Des travaux antérieurs traitent de la modélisation de la partie opérative. Nous nous intéressons en particulier aux travaux de Machado (2006) sur le découpage de la partie opérative, de Rohée *et al.* (2006) sur la modélisation des chaînes fonctionnelles et de Philippot *et al.* (2009) sur les modèles des actionneurs et de leurs pré-actionneurs associés.

#### 3.1.1.1 Découpage en chaînes fonctionnelles

Machado détaille (Machado (2006), Machado *et al.* (2006a)) une approche basée sur les chaînes fonctionnelles qui consiste à détailler les boucles contrôleur - actionneur - capteur - contrôleur en se basant sur les fonctions réalisées par ces dernières. Pour chaque membre de cette boucle, un modèle est ensuite créé. Cette proposition nous semble un guide très utile et incontournable lors de la phase de découpage de la partie opérative en composants et sera utilisée dans ces travaux.

Cependant, les modèles utilisés dans ces travaux étant non temporisés, ils ne conviennent pas à nos besoins et ne seront pas utilisés.

### 3.1.1.2 Modélisation de la chaîne fonctionnelle

Rohée *et al.* (2006) propose d'utiliser aussi une approche modulaire mais en se focalisant sur les modèles des pré-actionneurs. Le comportement des capteurs est déduit de ce modèle pour y être intégré sous forme d'une séquence. Cette approche présente l'avantage de produire des modèles très compacts des chaînes fonctionnelles intégrant le comportement du pré-actionneur, de l'actionneur et des capteurs associés.

Cependant, cette solution n'est pas très flexible : un changement au niveau des capteurs associés à un actionneur (leur nombre, par exemple) impose la reconstruction complète du modèle de la chaîne. Ce fait nous semble en contradiction avec un principe de modélisation modulaire où chaque modèle représente un composant standard de la partie opérative.

Ainsi nous choisirons d'utiliser plusieurs modèles pour une chaîne fonctionnelle, mais l'idée de produire des modèles compacts nous semble *a contrario* indispensable pour parvenir à une bonne compréhension de notre modèle final complet de partie opérative. Un unique modèle est prévu pour l'ensemble des capteurs associés à un actionneur, ce modèle sera appelé modèle du *groupe de capteurs* associé à l'actionneur.

### 3.1.1.3 Modélisation du couple pré-actionneur/actionneur

Enfin, Philippot *et al.* (2009) est un exemple intéressant de modélisation *temporisée* de partie opérative. La modélisation est là encore modulaire (au travers du concept de « Part of Plant ») et les modèles obtenus sont une sous-classe d'automates de Moore temporisés. Le modèle du pré-actionneur reçoit les ordres du contrôleur, son état actif est utilisé comme entrée par le modèle de l'actionneur pour évoluer. Ce choix de modéliser séparément l'actionneur et le pré-actionneur colle parfaitement à la réalité mais implique qu'un changement dans le modèle du pré-actionneur impose une modification du modèle de l'actionneur (car ses entrées sont modifiées), ce qui lie fortement ces deux modèles.

Quitte à produire un couple de modèles pour chaque couple pré-actionneur/actionneur, notre volonté de simplification nous pousse à ne produire qu'un seul modèle pour le couple pré-actionneur/actionneur. De plus, les modèles proposés par Philippot *et al.* n'utilisent pas de variables discrètes ce qui les empêche de modéliser les déplacements induits par l'action de l'actionneur et ne couvre donc pas toutes nos attentes.

## 3.1.2 Principes de modélisation retenus

### 3.1.2.1 Découpage en chaînes fonctionnelles

En se basant sur l'analyse de l'état de l'art, nous choisissons les principes de modélisation suivants :

- La partie opérative est découpée en chaînes fonctionnelles.
- Chaque couple pré-actionneur/actionneur est modélisé par un modèle issu d'une bibliothèque de modèles génériques.
- Chaque groupe de capteurs associé à un actionneur est modélisé par un modèle unique.

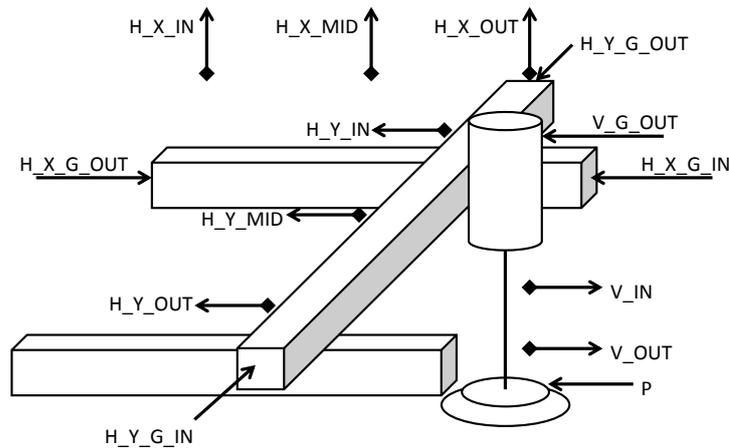


FIGURE 3.2 – Partie opérative du manipulateur avec ordres et informations.

Dans le cas de la partie opérative de notre exemple représentée sur la figure 3.2, on peut considérer les chaînes fonctionnelles suivantes :

- La chaîne fonctionnelle qui assure *le déplacement selon l'axe X*, composée dans la partie opérative :
  - Du vérin pneumatique horizontal d'axe X.
  - Des capteurs H\_X\_IN, H\_X\_MID et H\_X\_OUT pour les trois positions indiquées.
- La chaîne fonctionnelle qui assure *le déplacement selon l'axe Y*, composée dans la partie opérative :
  - Du vérin pneumatique horizontal d'axe Y.
  - Des capteurs H\_Y\_IN, H\_Y\_MID et H\_Y\_OUT pour les trois positions indiquées.
- La chaîne fonctionnelle qui assure *le déplacement selon l'axe Z*, composée dans la partie opérative :
  - Du vérin pneumatique vertical d'axe Z.
  - Des capteurs V\_IN et V\_OUT pour les deux positions indiquées.
- La chaîne fonctionnelle qui assure *la préhension de la pièce*, composée dans la partie opérative uniquement de la pompe et de la ventouse (sans capteur physique mais avec un capteur logiciel de type temporisation).

En utilisant cette décomposition, il convient donc de créer pour la partie opérative de la chaîne fonctionnelle de l'axe X présentée dans la figure 3.3 :

- Un modèle pour le vérin qui assigne une variable X, entière et interne au modèle de la partie opérative, modélisant la longueur de tige sortie du vérin en fonction des ordres reçus H\_X\_G\_OUT et H\_X\_G\_IN et du temps écoulé.
- Un (ou des) modèle(s) pour les capteurs de position qui, en fonction de la valeur de la variable X, émettent les informations H\_X\_IN, H\_X\_MID et H\_X\_OUT.

En utilisant le même procédé pour les autres chaînes fonctionnelles, on obtient l'ensemble des modèles à créer pour modéliser entièrement la partie opérative du manipulateur :

**Axe X :** – Un modèle du vérin, noté V\_H\_X, réagissant aux ordres H\_X\_G\_OUT et H\_X\_G\_IN et assignant la variable X.

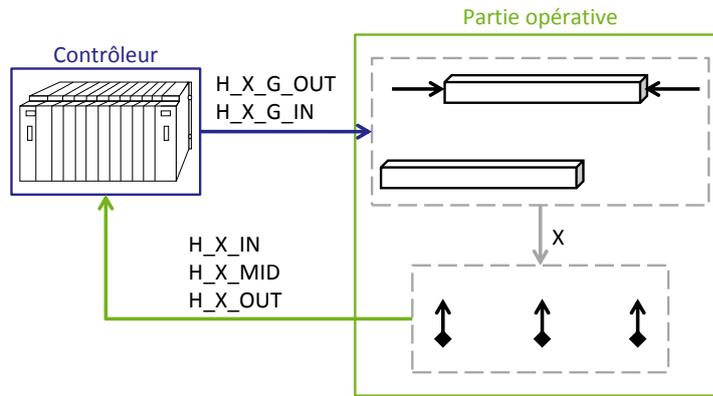


FIGURE 3.3 – Exemple de chaîne fonctionnelle sur l'axe X.

- Un modèle, noté S\_H\_X, pour les trois capteurs de position réagissant à la variable X et émettant les informations H\_X\_IN, H\_X\_MID et H\_X\_OUT.

**Axe Y :** – Un modèle du vérin, noté V\_H\_Y, réagissant aux ordres H\_Y\_G\_OUT et H\_Y\_G\_IN et assignant la variable Y.

- Un modèle, noté S\_H\_Y, pour les trois capteurs de position réagissant à la variable Y et émettant les informations H\_Y\_IN, H\_Y\_MID et H\_Y\_OUT.

**Axe Z :** – Un modèle du vérin, noté V\_V, réagissant à l'ordre V\_G\_OUT et assignant la variable Z.

- Un modèle, noté S\_V, pour les deux capteurs de position réagissant à la variable Z et émettant les informations V\_IN et V\_OUT.

La figure 3.4 montre l'ensemble des modèles nécessaires à la modélisation de la partie opérative du manipulateur.

Les modèles de la pompe à vide et de la ventouse n'ont pas été traités car l'information de cette chaîne fonctionnelle est issue d'un capteur logiciel *interne* au contrôleur.

### 3.1.2.2 Modélisation du couple pré-actionneur/actionneur

#### 3.1.2.2.1 Principe

En nous appuyant sur les principes proposés par Machado (2006) et les modèles temporisés proposés par Philippot *et al.* (2009), nous proposons les règles suivantes de modélisation des actionneurs (et de leurs pré-actionneurs associés) :

1. En considérant les variables de contrôle (ordres) issues du contrôleur déjà existantes :
  - Une variable discrète modélisant la grandeur physique modifiée par l'actionneur est ajoutée. Elle est nommée *grandeur physique* (modifiée par l'actionneur) et notée GP par la suite. Sa plage de variation peut éventuellement être restreinte par les valeurs limites GP\_min et GP\_max. Elle est supposée initialisée à la valeur GP\_init (dans l'intervalle de variation s'il est défini).

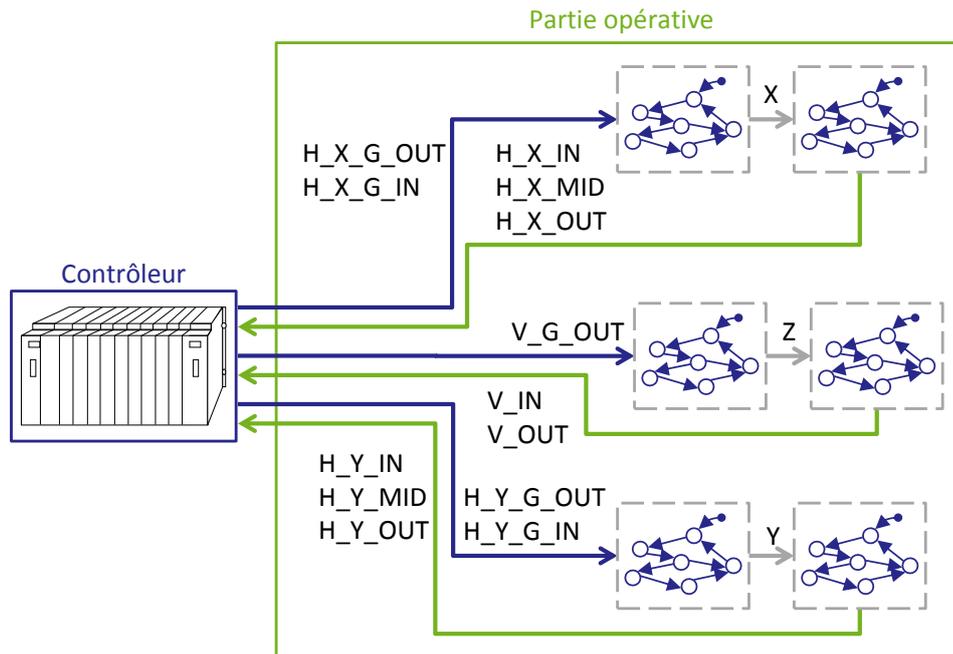


FIGURE 3.4 – Schéma d'ensemble des modèles de la partie opérative du manipulateur.

- Une horloge utilisée uniquement par ce modèle pour modéliser le comportement temporel de l'actionneur est ajoutée. Elle est nommée *horloge d'action* et notée  $T_{GP}$ . À chaque pas de temps, la *grandeur physique* est supposée être modifiée d'une quantité  $D_{GP}$  (hypothèse de variation linéaire).
2. L'état immobile (sans mouvement) de l'actionneur est modélisé par une localité, nommée *localité immobile*.
  3. Chaque mouvement de solide initié par l'actionneur est modélisé par une localité, nommée *localité de mouvement*.
  4. Sur les localités de mouvement, on ajoute :
    - Des transitions bouclées nommées *transitions de mouvement* avec :
      - Une assignation modifiant GP de la quantité  $D_{GP}$ .
      - Une contrainte temporelle portant sur  $T_{GP}$  pour modéliser la dynamique de l'actionneur.
      - Une remise à zéro de  $T_{GP}$ .
    - Un invariant sur  $T_{GP}$  pour contraindre le modèle à franchir les *transitions de mouvement*.
  5. Le comportement du pré-actionneur est ensuite intégré :
    - Sur les transitions existantes, les valeurs des variables de contrôle nécessaires au franchissement étant ajoutées à la garde.
    - Avec l'ajout de nouvelles transitions liant les localités entre elles selon les possibilités offertes par le pré-actionneur et au regard des valeurs des variables de contrôle nécessaires à de telles évolutions. Les transitions aboutissant dans une localité de mouvement doivent comporter une remise à zéro de  $T_{GP}$ .

6. Les contraintes technologiques de l'actionneur sont enfin ajoutées :
  - Sur les gardes des transitions existantes, en utilisant GP, D\_GP, et GP\_min ou GP\_max pour interdire une évolution qui ferait sortir GP de son domaine de variation et pour interdire les évolutions sans sens physique.
  - Au travers d'une transition liant chaque localité de mouvement à la localité immobile afin de modéliser l'arrivée aux limites de GP. Cette transition possède une garde relative à GP, D\_GP, et GP\_min ou GP\_max, et une contrainte temporelle relative à T\_GP, dont l'assignation impose à GP une des valeurs limites.
7. Le choix de la localité initiale est issu de la technologie du pré-actionneur et de la valeur initiale GP\_init précédemment définie. À défaut, la localité immobile est à privilégier.

Plusieurs remarques peuvent être faites au vu de ces règles :

- Le comportement décrit par ce modèle est sans défaillances.
- Nous considérons dans ces travaux que la grandeur D\_GP est une constante, par conséquent les mouvements issus de nos modèles se font à vitesse constante.
- Dans ces travaux, le tir de la *transition de mouvement* est fait systématiquement lorsque T\_GP arrive à 1.
- La modification de GP dans une *transition de mouvement* peut être simple, comme pour le cas d'un vérin :  $\mathbf{GP} := \mathbf{GP} + \mathbf{D\_GP}$ , ou complexe, comme pour le cas d'un moteur où GP modélise l'angle de rotation de l'arbre moteur en degrés :  $\mathbf{GP} := \mathbf{GP} + \mathbf{D\_GP} - (((\mathbf{GP} + \mathbf{D\_GP})/360) \times 360)$  (voir modèle du moteur en annexe, figure B.2). Ici, la grandeur physique doit rester entre 0 et 360 degrés, et passer de 360 à 0 degrés (ou vice versa). Pour ce faire, l'utilisation de la division euclidienne (« / ») est un atout indispensable.
- Les temps de commutation du pré-actionneur comme ceux de changement d'état de l'actionneur sont considérés dans ces travaux comme très petits devant le temps caractéristique d'action de l'actionneur sur la grandeur physique qu'il modifie. Ainsi, les transitions ne modifiant pas la grandeur physique sont supposées sans consommation de temps.

### 3.1.2.2.2 Exemple d'application

Cette section traite de la conception pas à pas du modèle générique d'un vérin pneumatique à double effet (figure 3.5a) couplé à un pré-actionneur de type distributeur 5\3 à centre fermé (figure 3.5b), tel qu'utilisé dans notre exemple. Le distributeur pneumatique, relié au vérin par des connecteurs pneumatiques, est commandé au travers de deux bobines électriques (avec retour par ressort) par deux ordres : G\_IN pour l'ordre de rentrée de la tige du vérin et G\_OUT pour l'ordre de sortie. Son comportement est le suivant :

- En l'absence d'ordre, le distributeur reste (ou revient) en position centrale (fermée) du fait des rappels par ressort. Il ne distribue aucune énergie : le vérin est à l'arrêt.
- En cas de commande simple (G\_IN ou G\_OUT de manière exclusive), le distributeur commute dans l'une des positions latérales et la tige du vérin se déplace.

- En cas de commande double (G\_IN et G\_OUT simultanément), on considère que le distributeur reste en position s'il est déjà commuté, sinon il reste en position centrale.

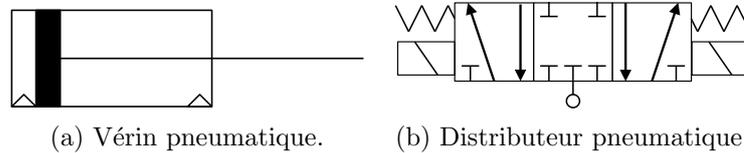


FIGURE 3.5 – Couple vérin pneumatique double effet et son pré-actionneur de type distributeur pneumatique 5\3 centre fermé à commande électrique.

La figure 3.6 montre les étapes de conception du modèle de ce couple pré-actionneur / actionneur :

**Règle 1 :** étant donné la nature de l'actionneur, la *grandeur physique* modifiée est la longueur de tige sortie que nous notons  $X$ . De par sa technologie, cette grandeur est limitée par les valeurs  $X_{\min}$  et  $X_{\max}$ , et supposée initialisée à la valeur  $X_{\text{init}}$ . L'horloge  $T_X$  est aussi ajoutée et la dynamique du vérin est modélisée au travers de  $D_X$  qui est ajouté ou retranché à  $X$  à chaque pas de temps de  $T_X$  lorsque la tige est en mouvement. Le vérin est commandé au travers de deux ordres G\_IN et G\_OUT.

**Règles 2 & 3 :** comme montré dans la figure 3.6a représente cette étape, la localité immobile est ajoutée (notée  $0$ ) ainsi qu'une localité de mouvement pour chaque mouvement : une pour la sortie de tige (notée  $1$ ) et une pour sa rentrée (notée  $2$ ).

**Règle 4 :** le comportement dynamique de l'actionneur est ajouté au travers des deux transitions de mouvement bouclant sur les localités de mouvement et de l'invariant ajouté sur celles-ci, comme le montre la figure 3.6b.

**Règle 5 :** le comportement du pré-actionneur est ajouté, comme le montre la figure 3.6c :

- passage de la localité immobile à une localité de mouvement si l'ordre adéquat est donné (avec remise à zéro de l'horloge  $T_X$ ),
- interdiction de mouvement (retour à la localité immobile) si l'ordre n'est plus émis,
- tir de la transition de mouvement uniquement si l'ordre reste émis dans une localité de mouvement,
- en cas de double commande, la localité active ne change pas.

**Règles 6 & 7 :** On intègre maintenant les limites technologiques au modèle :

- les transitions allant vers les localités de mouvement sont interdites si l'ordre reçu fait bouger la tige dans le sens où elle est déjà en butée.
- les gardes des transitions de mouvement sont modifiées afin de ne pas dépasser  $X_{\min}$  ou  $X_{\max}$ .
- pour modéliser l'arrivée en butée, une transition partant de la localité de mouvement et arrivant à la localité immobile est ajoutée pour chaque mouvement. Celle-ci se substitue à la transition de mouvement pour la dernière évolution

amenant en butée : lorsque le temps est écoulé et si l'ordre est maintenu, si la valeur future de X dépasse l'une des limites, alors X est assignée à la place à la valeur limite.

Puis l'état initial est ajouté : la localité 0 est choisie comme localité initiale et la variable X est initialisée à X\_init. Toutes ces modifications sont montrées sur la figure 3.6d.

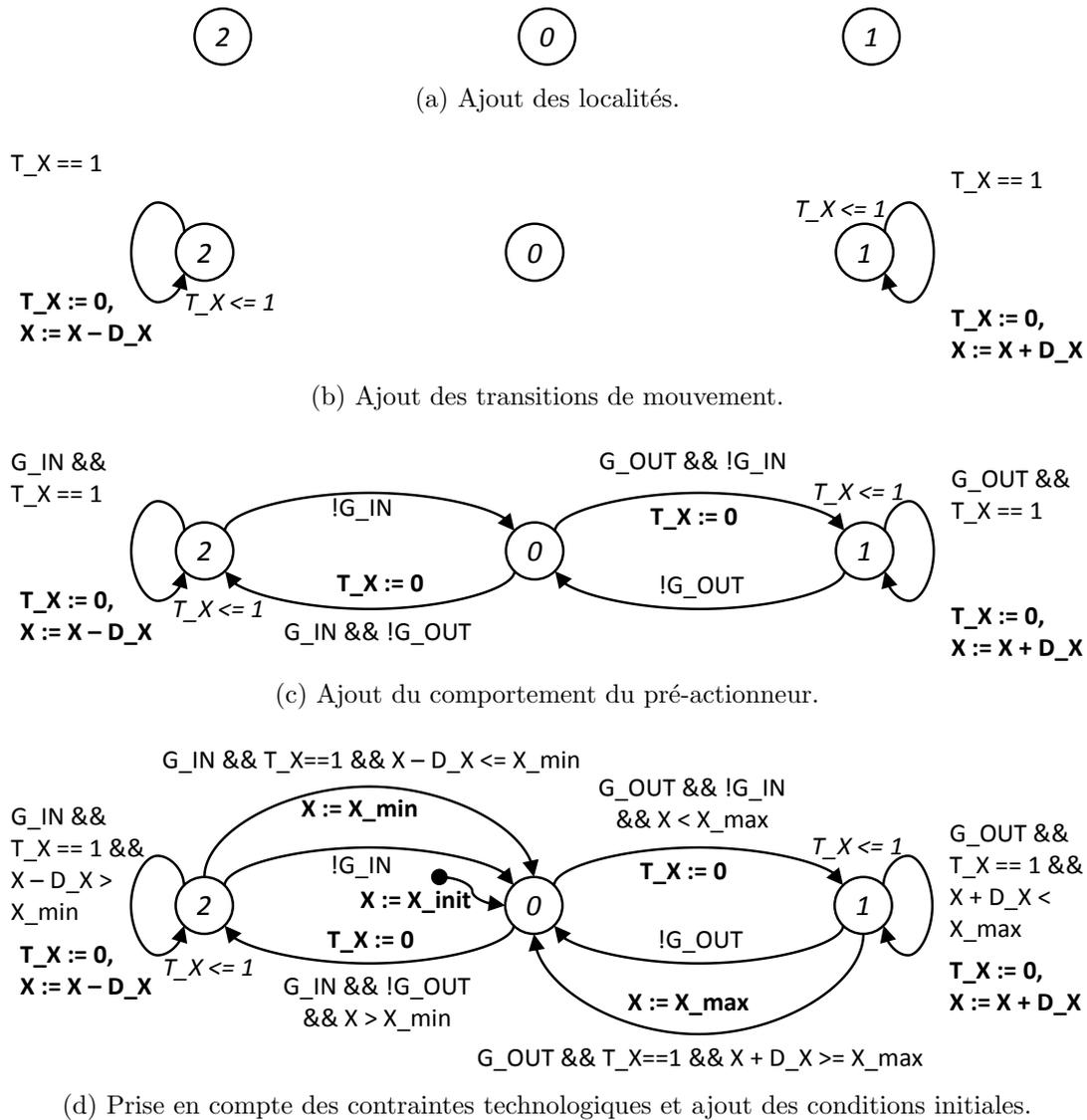


FIGURE 3.6 – Étapes de modélisation du couple pré-actionneur / actionneur de la figure 3.5.

### 3.1.2.3 Modélisation des capteurs

#### 3.1.2.3.1 Principes

En nous appuyant sur les principes de modélisation proposés par Machado (2006) et sur le choix de compacité des modèles de capteurs issus de l'analyse des travaux de Rohée *et al.* (2006), nous proposons les règles de modélisation d'un groupe de capteurs suivantes :

1. Tous les capteurs du groupe doivent être sensibles à la même grandeur physique GP modifiée par un unique actionneur.
2. Chaque position détectable via les sorties logiques associées au groupe de capteurs est représentée par une localité.
3. Les états intermédiaires entre ces positions détectées sont modélisés par des localités.
4. Des transitions assignant les informations des capteurs sont ajoutées entre les localités en fonction des possibilités d'évolution des capteurs.
5. L'initialisation d'un tel modèle repose sur la valeur initiale de la grandeur physique mesurée. La localité initiale sera donc définie au moment de l'instanciation.

On peut faire les remarques suivantes sur les règles énoncées :

- Nous nous limitons dans ces travaux au cas des capteurs logiques.
- Seul un comportement sans défaillances est modélisé.
- Le temps de détection des capteurs est considéré très petit face au temps caractéristique des actions sur la grandeur physique qu'ils surveillent. Ainsi toutes les transitions sont sans consommation de temps.

### 3.1.2.3.2 Exemple d'application

L'application des règles données pour la conception du modèle d'un *groupe de capteurs* va nous permettre de construire le modèle du groupe de capteurs associé au vérin précédemment modélisé dans la section 3.1.2.2.2. Ce groupe de capteurs comporte trois capteurs logiques émettant les informations X\_IN, X\_MID et X\_OUT pour signifier que la tige du vérin est respectivement rentrée, dans une position intermédiaire au milieu, ou sortie. La figure 3.7 montre un exemple de zones de répartition de trois capteurs le long du trajet de la tige d'un vérin (axe X). On peut y lire les informations émises ainsi que les paramètres de limites de détection des capteurs. On suppose à cet égard que les limites extrêmes de détection (tige totalement rentrée ou sortie) sont les limites mécaniques (dues à la technologie de l'actionneur associé) du mouvement de la tige (respectivement X\_min et X\_max).

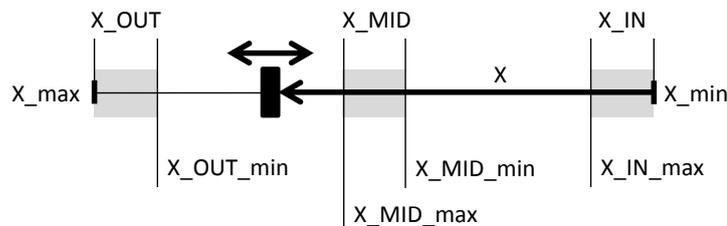


FIGURE 3.7 – Exemple de limites de détection d'un groupe de trois capteurs.

Le comportement attendu du modèle final est alors :

- Si  $X$  est inférieure à  $X\_IN\_max$ , l'information  $X\_IN$  est émise.
- Si  $X$  est comprise entre  $X\_MID\_min$  et  $X\_MID\_max$ , l'information  $X\_MID$  est émise.
- Si  $X$  est supérieure à  $X\_OUT\_min$ , l'information  $X\_OUT$  est émise.
- Dans tous les autres cas de figure, aucune information n'est émise.

La figure 3.8 montre les étapes de modélisation détaillée de ce groupe de capteurs :

**Règle 1 :** tous les capteurs de ce groupe observent la position de la tige du vérin et sont donc en relation avec la même grandeur physique. Cette dernière n'est supposée être modifiée que par le vérin, donc on peut faire un modèle de *groupe de capteurs*.

**Règles 2 & 3 :** les localités associées aux positions détectables sont ajoutées (localités 0, 2 et 4 sur la figure 3.8a), puis les localités représentant les positions situées entre les plages de détections des capteurs (localités 1 et 3 sur la figure), en accord avec le comportement attendu du groupe de capteur.

**Règle 4 :** la figure 3.8b montre les transitions ajoutées : elles assignent les variables d'information émises par les capteurs en fonction de leurs localités de départ/arrivée.

**Règle 5 :** l'initialisation du modèle n'est pas faite, car elle dépend de  $X\_init$ .

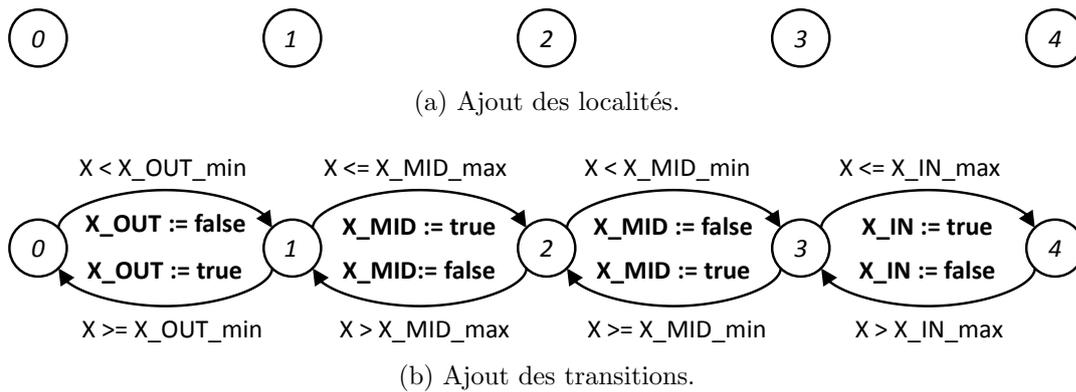


FIGURE 3.8 – Étapes de modélisation d'un groupe de 3 capteurs associé au vérin précédemment modélisé.

### 3.1.3 Modélisation de la partie opérative du manipulateur

#### 3.1.3.1 Modèles de vérins horizontaux et verticaux

Après avoir produit les modèles génériques, nous souhaitons maintenant produire les différents modèles de vérins de la partie opérative du manipulateur, comme montré sur la figure 3.4.

Il convient donc de produire les modèles  $V\_H\_X$  et  $V\_H\_Y$  des vérins horizontaux (et de leurs pré-actionneurs associés) et  $V\_V$  du vérin vertical.

Les choix technologiques associés à ces vérins sont les suivants :

**Vérins horizontaux :** vérins pneumatiques double effet couplés à des pré-actionneurs de type distributeur pneumatique 5\3 centre fermé à commande électrique. La figure 3.5 montre les représentations associées à ces choix technologiques et la figure 3.6d, le modèle générique obtenu.

**Vérin vertical :** vérin pneumatique double effet couplé à un pré-actionneur de type distributeur pneumatique 5\2 monostable à commande électrique et rappel par ressort. Ce choix est motivé par le besoin de garantir qu'en cas de coupure d'énergie, la position tige rentrée (pièce en haut) est privilégiée.

Le modèle obtenu pour un vérin pneumatique double effet associé à un distributeur pneumatique 5\2 monostable à commande électrique et rappel par ressort est présenté dans la figure 3.9.

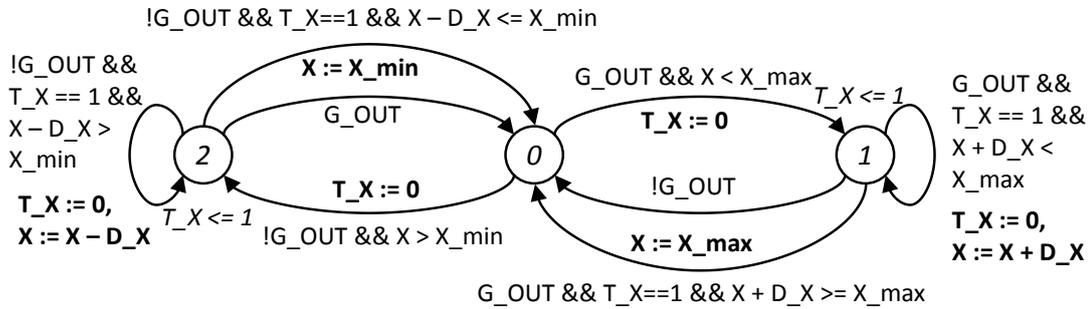


FIGURE 3.9 – Modèle d'un vérin pneumatique double effet couplé à un distributeur 5\2 monostable à commande électrique et rappel par ressort.

Les différences avec le modèle présenté dans la figure 3.6d sont issues de l'aspect monostable du distributeur, et sont les suivantes :

- La localité initiale est *a priori* inconnue : localité de rentrée de tige (localité 2) du vérin si cette dernière n'est pas complètement rentrée, localité immobile (localité 0) si la tige est déjà rentrée.
- La localité immobile 0 ne peut rester active que si la tige est rentrée en butée avec absence d'ordre ou sortie en butée avec l'ordre G\_OUT maintenu.
- Les localités de mouvement sont actives en fonction de l'unique ordre et des limites technologiques du vérin.
- Les transitions de mouvement sont franchissables en fonction de l'unique ordre (ou de son absence) et des limites technologiques du vérin.

On peut maintenant obtenir les modèles des vérins du manipulateur présentés dans la figure 3.10 en instanciant les modèles génériques :

- **V\_H\_X**, présenté dans la figure 3.10a, est obtenu en instanciant le modèle générique de la figure 3.6d avec :
  - Les ordres H\_X\_G\_OUT et H\_X\_G\_IN à la place de G\_OUT et G\_IN.
  - La grandeur physique X pour la longueur de tige de vérin sortie.
  - L'horloge T\_X utilisée pour la dynamique du modèle.
  - Les limites technologiques  $X_{min} = 0$  et  $X_{max} = 20$  pour les limites du mouvement de la tige.

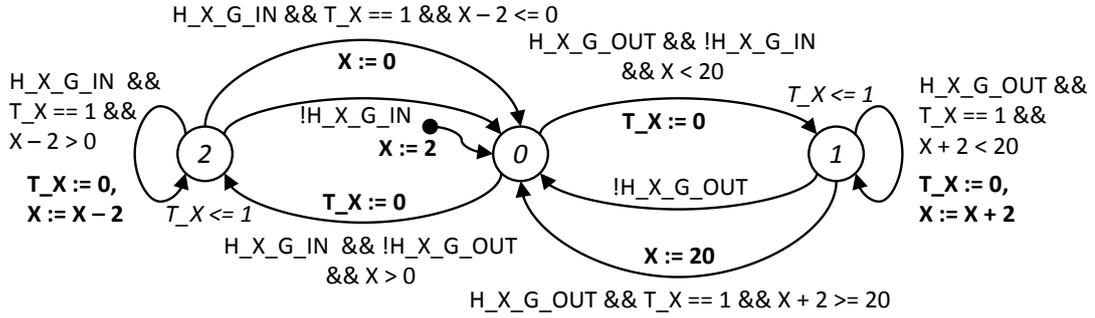
- La donnée dynamique  $D_X = 2$ , ce qui correspond à un changement de 2 unités sur  $X$  à chaque pas de temps de  $T_X$ .
- L'initialisation est choisie par rapport aux limites de détection du groupe de capteurs associé au vérin, précisées dans la section suivante (en position rentrée donc entre 0 et 2).
- $V_{H_Y}$ , présenté dans la figure 3.10b, est obtenu en instanciant le modèle générique de la figure 3.6d avec :
  - Les ordres  $H_Y\_G\_OUT$  et  $H_Y\_G\_IN$  à la place de  $G\_OUT$  et  $G\_IN$ .
  - La grandeur physique  $Y$  pour la longueur de tige de vérin sortie.
  - L'horloge  $T_Y$  utilisée pour la dynamique du modèle.
  - Les limites technologiques  $Y_{min} = 0$  et  $Y_{max} = 20$  pour les limites du mouvement de la tige.
  - La donnée dynamique  $D_Y = 2$ , ce qui correspond à un changement de 2 unités sur  $Y$  à chaque pas de temps de  $T_Y$ .
  - L'initialisation est choisie par rapport aux limites de détection du groupe de capteurs associé au vérin, précisées dans la section suivante (en position rentrée donc entre 0 et 2).
- $V_V$ , présenté dans la figure 3.10c, est obtenu en instanciant le modèle générique de la figure 3.9 avec :
  - L'ordre  $V\_G\_OUT$  à la place de  $G\_OUT$ .
  - La grandeur physique  $Z$  pour la longueur de tige de vérin sortie.
  - L'horloge  $T_Z$  utilisée pour la dynamique du modèle.
  - Les limites technologiques  $Z_{min} = 0$  et  $Z_{max} = 10$  pour les limites du mouvement de la tige.
  - La donnée dynamique  $D_Z = 1$ , ce qui correspond à un changement d'une unité sur  $Z$  à chaque pas de temps de  $T_Z$ .
  - L'initialisation à  $Z_{init} = 0$  est imposée a cause du choix de pré-actionneur, qui impose aussi que la localité initiale est celle immobile (localité  $\emptyset$ ).

### 3.1.3.2 Modèles des capteurs

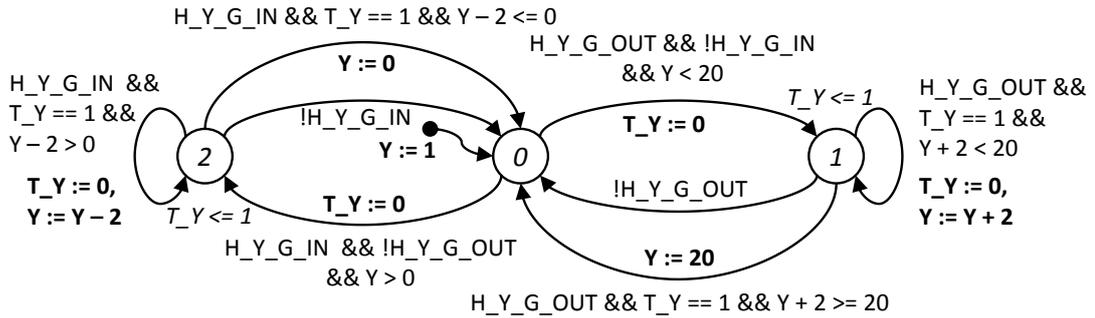
Les trois capteurs associés à chaque vérin horizontal sont modélisés par une unique instance du modèle présenté sur la figure 3.8b. Lors de cette instanciation, les différentes valeurs de seuils de détection  $X\_IN\_max$ ,  $X\_MID\_min$ ,  $X\_MID\_max$  et  $X\_OUT\_min$  sont choisies respectivement égales à 2, 9, 11 et 18. Les variables assignées (signaux des capteurs) seront  $H_X\_OUT$ ,  $H_X\_MID$  et  $H_X\_IN$  pour les capteurs associés au vérin  $X$ , et  $H_Y\_OUT$ ,  $H_Y\_MID$  et  $H_Y\_IN$  pour les capteurs associés au vérin  $Y$ .

Les vérins sont initialisés en position de détection de seuil tige rentrée, à savoir  $X$  et  $Y$  égales à 2. Les variables  $H_X\_IN$  et  $H_Y\_IN$  sont donc initialisées à vrai (*true*) et la localité initiale est la localité  $\emptyset$ .

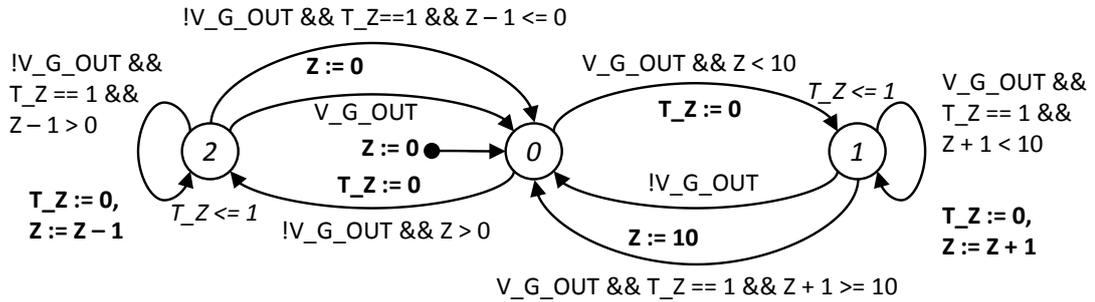
Les figures 3.11a et 3.11b montrent les modèles des capteurs pour les vérins horizontaux. Le modèle des deux capteurs associés au vérin vertical  $Z$  (figure 3.11c) utilise quant à lui seulement deux seuils (1 et 9) et assigne les variables booléennes  $V\_IN$  et  $V\_OUT$ . Comme le vérin est initialisé en position tige totalement rentrée ( $Z$  égale à 0), la variable



(a) Modèle du vérin horizontal X de l'exemple.



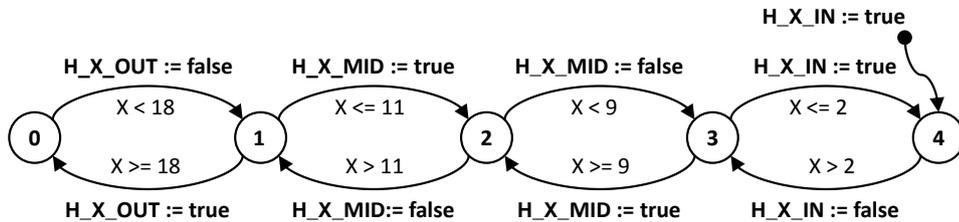
(b) Modèle du vérin horizontal Y de l'exemple.



(c) Modèle du vérin vertical Z de l'exemple.

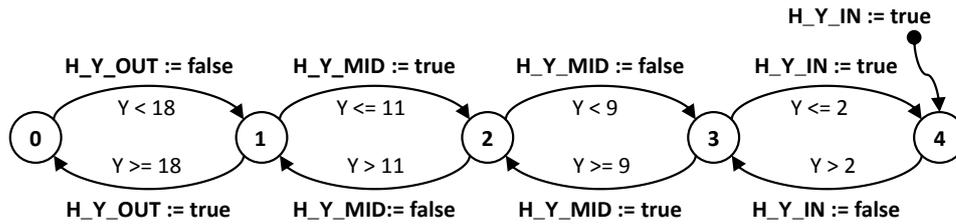
FIGURE 3.10 – Modèles des vérins de l'exemple.

V\_IN est initialisée à vrai et la localité 2 est initiale.

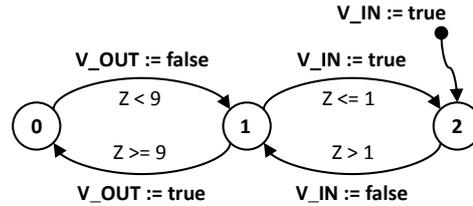


(a) Modèle des capteurs associés au vérin horizontal X de l'exemple.

FIGURE 3.11 – Modèles des capteurs associés aux vérins de l'exemple.



(b) Modèle des capteurs associés au vérin horizontal Y de l'exemple.



(c) Modèle des capteurs associés au vérin vertical Z de l'exemple.

FIGURE 3.11 – Modèles des capteurs associés aux vérins de l'exemple.

### 3.1.4 Mise en évidence des concurrences internes

#### 3.1.4.1 Mise en évidence du problème sur une séquence d'évolutions

Une des difficultés principales lors de la conception d'un modèle de partie opérative en utilisant des modèles génériques instanciés provient du nombre de modèles évoluant simultanément. Ce problème, intrinsèque à la méthode de conception choisie, impose de porter une attention toute particulière aux concurrences d'évolutions lors de l'utilisation des modèles.

Lors des premiers tests sur les modèles de l'exemple, un cas de figure mettant en avant ces concurrences et leurs effets néfastes est apparu ; il est détaillé ci-dessous.

Cet exemple utilise le modèle du vérin horizontal X ainsi que le modèle des capteurs associés. La modélisation du contrôleur sera abordée plus tard (section 3.2) mais, pour l'exemple, la commande H\_X\_G\_OUT sera mise à vrai pour forcer un mouvement du vérin de la position rentrée ( $X = 2$ ) jusqu'à la position intermédiaire. Ainsi, lorsque le capteur central détecte la tige (au travers de la variable H\_X\_MID passant à vrai), la commande est passée à faux afin de laisser le manipulateur en position intermédiaire sur l'axe X.

Comme le mouvement ne concerne que la sortie de la tige, le modèle du vérin sera réduit à cette partie. De même, comme la détection de la position de la tige du vérin ne concerne que les positions rentrée (*IN*) et intermédiaire (*MID*), le reste du modèle des capteurs sera omis.

La figure 3.12a représente la position initiale : vérin rentré ( $X$  égale à 2) et à l'arrêt (localité 0 active), capteurs détectant une position rentrée (H\_X\_IN à vrai, H\_X\_MID et H\_X\_OUT à faux). La commande de sortie du vérin (H\_X\_G\_OUT) est pour l'instant à faux.

La figure 3.12b représente la première évolution : la commande H\_X\_G\_OUT passe à vrai. En réaction, le modèle du vérin évolue vers la localité 1 qui modélise le mouvement

de sortie de la tige du vérin.

Après un pas de temps,  $T\_X$  passe à 1 et permet le tir de la transition modélisant la sortie de la tige, comme présenté sur la figure 3.12c. Ce tir est obligatoire dû à l'invariant de la localité 1. La variable  $X$  passe de 2 à 4 et l'horloge  $T\_X$  est ré-initialisée.

Ensuite, les sémantiques des ATVD permettent deux évolutions :

- Le comportement attendu par le concepteur est représenté sur la figure 3.12d. Le modèle du capteur évolue ( $4 \rightarrow 3$ ) pour représenter l'absence de détection du capteur de position rentrée. La variable  $H\_X\_IN$  passe à faux.
- Un comportement irréaliste est cependant possible (figure 3.12e). Comme il n'y a pas de priorité entre les sémantiques, le temps peut encore évoluer et, lorsque  $T\_X$  atteint à nouveau 1, l'arc modélisant la sortie de la tige peut à nouveau être franchi, amenant la variable  $X$  de 4 à 6.

Ce comportement irréaliste et néfaste peut être reconduit jusqu'à atteindre la limite physique du vérin, à savoir  $X$  égale à 20.

Le modèle du groupe de capteurs a alors un retard conséquent : il n'a pas encore mis à jour la valeur de la variable modélisant le signal du capteur de position rentrée alors que la tige du vérin est totalement sortie !

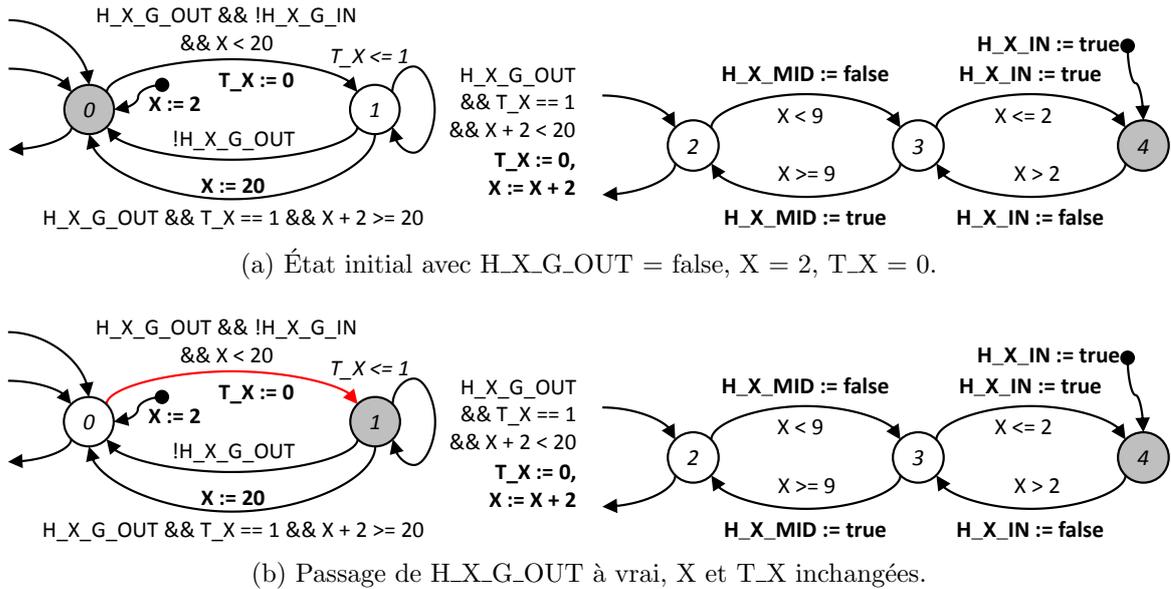
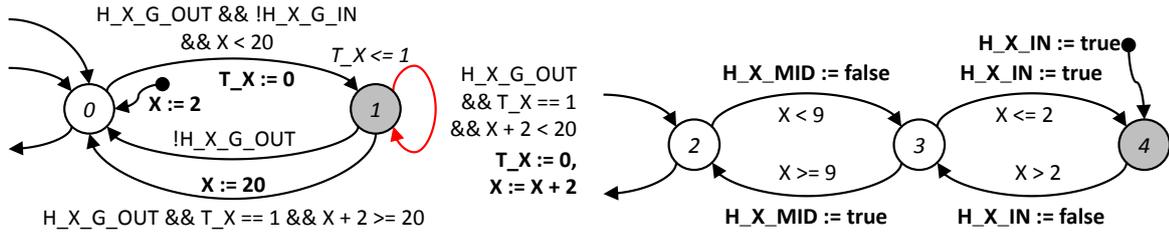


FIGURE 3.12 – Séquence d'évolutions potentiellement réaliste et irréaliste.

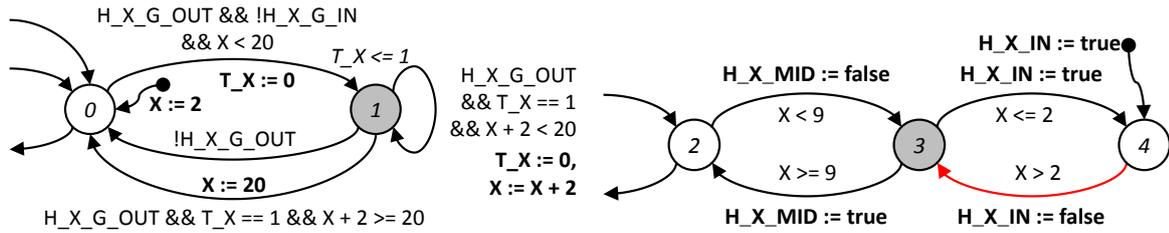
### 3.1.4.2 Analyse du comportement irréaliste

Le principal souci dans l'exemple précédent est que l'idée de départ du concepteur de modéliser des évolutions ne consommant pas de temps par des transitions non urgentes sans contraintes temporelles est une erreur.

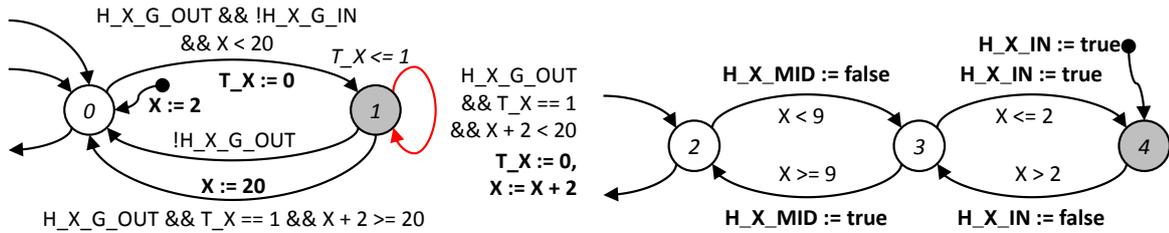
En effet, il pose comme hypothèse de départ que les transitions ne consommant pas de temps seront traitées de manière prioritaire, ce qui n'est pas le cas, aucune priorité n'étant



(c) Évolution de l'horloge  $T_X$  à 1, puis tir de la transition de mouvement.  $T_X$  finit donc à 0 et  $X$  passe de 2 à 4.



(d) Évolution *souhaitée* avec  $X$  à 4 et  $H\_X\_IN$  passant à faux.



(e) Évolution irréaliste possible avec  $X$  passant à 6 (après passage de  $T_X$  à 1 et tir de la transition de mouvement) et  $H\_X\_IN$  restant à vrai!

FIGURE 3.12 – Séquence d'évolutions potentiellement réaliste et irréaliste.

définie entre les sémantiques (comme présenté dans la section 2.2.3) pour les ATVD.

Lors du processus de modélisation, les concepteurs font généralement face à deux types d'évolutions :

- Des évolutions *lentes* qui représentent généralement des mouvements physiques ou des temporisations.
- Des évolutions *rapides* qui représentent des modifications des valeurs de capteurs ou les changements d'état des pré-actionneurs.

Or ces deux types d'évolutions ont des échelles de temps très différentes.

Deux solutions sont donc possibles :

- On peut choisir une échelle de temps pour l'ensemble des modèles qui est faible. On peut alors modéliser toutes les évolutions au travers d'évolutions temporisées. Cette solution implique que les évolutions lentes nécessiteront de nombreux pas de temps pour être franchissables, ce qui est très pénalisant pour les performances en termes de vérification et de simulation.
- On peut choisir de négliger la consommation de temps physique des évolutions rapides, les considérant alors comme *instantanées* vis-à-vis des évolutions lentes.

Dans le cadre de ces travaux, la deuxième solution sera retenue car les défauts de la première sont incompatibles avec les besoins d'analyse des modèles souhaités.

Ainsi, on définit alors deux types d'évolutions :

- Les *évolutions temporisées* modélisent des évolutions *lentes* qui consomment du temps physique dans le système réel. Ces évolutions utiliseront des transitions avec des contraintes temporelles et des invariants sur la localité de départ.
- Les *évolutions instantanées* modélisent des évolutions *rapides* qui consomment très peu de temps physique dans le système réel. Elles sont considérées à temps nul et sont modélisées par des transitions qui devront avoir une priorité sur les évolutions temporisées : des *transitions urgentes*.

### 3.1.5 Suppression des concurrences indésirables

Lors de la conception des modèles génériques, une démarche de réflexion sur les temps caractéristiques des processus physiques modélisés devient obligatoire. Cette dernière doit permettre de définir si l'évolution réelle modélisée consomme du temps physique ou peut être considérée comme instantanée.

En reprenant l'exemple d'un vérin pneumatique double effet couplé à un pré-actionneur 5\3 à centre fermé, on peut noter que :

- Les mouvements de tige (rentrée et sortie) sont des évolutions qui consomment du temps physique, elles restent donc toujours modélisées par des transitions avec une contrainte temporelle associée à un invariant sur la localité source.
- Les évolutions du pré-actionneur, modélisées par les transitions reliant les localités entre elles, peuvent être considérées comme instantanées. En effet, le temps de commutation d'un pré-actionneur est négligeable devant le temps caractéristique de déplacement de la tige. Ainsi, les transitions reliant les localités deviennent des transitions urgentes.

La figure 3.13 présente le nouveau modèle générique d'un vérin pneumatique double effet relié à un pré-actionneur 5\3 à centre fermé.

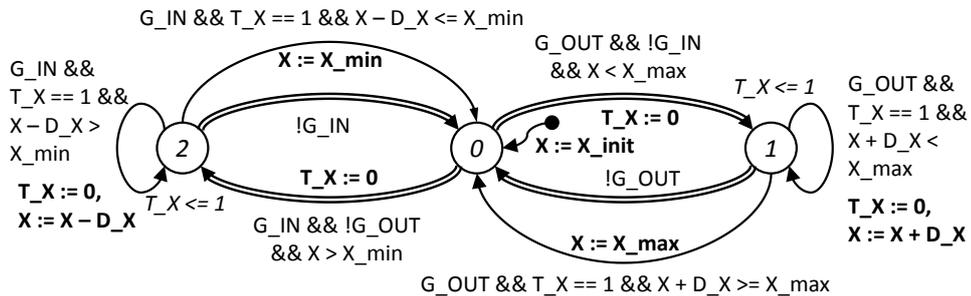


FIGURE 3.13 – Modèle générique modifié d'un vérin pneumatique double effet couplé à un pré-actionneur 5\3 à centre fermé.

En ce qui concerne les modèles des capteurs, les temps caractéristiques de commutation de ces derniers sont très petits devant les temps physiques mis en œuvre lors du déplacement des tiges, et toutes leurs transitions deviennent urgentes.

La figure 3.14 montre une version modifiée du modèle générique d'un groupe de trois capteurs.

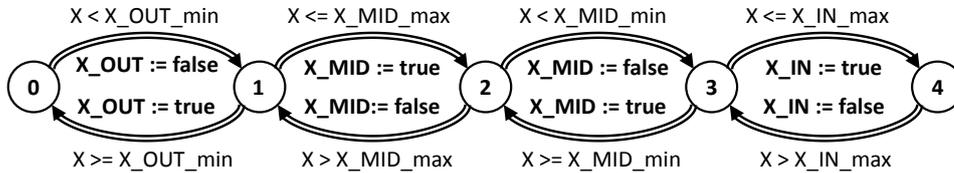


FIGURE 3.14 – Modèle modifié d'un groupe de trois capteurs.

La séquence utilisée pour la figure 3.12 est reprise avec les modèles modifiés. Pour des raisons de place, elle est décrite dans l'annexe C.1 : la concurrence n'existe plus et seule la sémantique réaliste est conservée.

## 3.2 Construction du modèle du contrôleur

### 3.2.1 Hypothèses

Dans le cadre de ces travaux, le contrôleur du système bouclé est supposé être un automate programmable industriel à scrutation cyclique des entrées. Ce contrôleur exécute un code spécifié au travers d'une spécification Grafcet (norme IEC 60848 (2002)).

Ce choix est motivé par la grande utilisation de ce type de matériel dans l'industrie, et plus particulièrement dans l'industrie manufacturière. De plus, la génération du code à partir d'une spécification dans un langage normalisé améliore la traçabilité et garantit la conformité de ce code aux besoins exprimés.

### 3.2.2 État de l'art

De nombreux travaux traitent de la conception d'un modèle du contrôleur. En particulier, Bauer *et al.* (2004) et Stursberg et Lohmann (2005) proposent de traduire des SFC (langage d'implémentation de la norme IEC 61131 (1999) proche du Grafcet) en automates. Machado *et al.* (2006b) et Provost *et al.* (2011) se basent quant à eux directement sur la spécification Grafcet pour produire un modèle algébrique qui peut alors être utilisé dans des automates ou pour produire des machines de Mealy.

#### 3.2.2.1 Traduction du code implanté en automate UPPAAL

Bauer *et al.* (2004) propose de traduire un code implémenté en SFC en automates d'UPPAAL. La traduction consiste à décrire les séquences d'un SFC (une série d'alternances étape-transition qui sont proches de celles existant dans un Grafcet) dans un automate d'UPPAAL. Un automate de coordination est utilisé pour définir quels automates de séquences doivent être lancés. Le modèle final est constitué d'une multitude d'automates (le coordinateur, un par séquence, un par qualificateur d'action (comme P

pour Pulse) utilisé, ...) qui sont liés par des canaux de communication et des variables partagées.

Mais cette approche utilise grandement des primitives fortes d'UPPAAL (localités urgentes et obligatoires, communication de type broadcast) que nous avons choisi de ne pas utiliser, ce qui nous pousse à chercher une autre méthode de traduction.

### 3.2.2.2 Traduction du code implanté via un méta-langage

Les travaux de [Stursberg et Lohmann \(2005\)](#) portent sur une alternative intéressante qui propose un méta-langage pour décrire un SFC. À l'aide d'opérateurs particuliers ( $\blacktriangledown$  pour une ouverture de séquence parallèle,  $\blacktriangle$  pour sa fermeture), ils décrivent un SFC sous une forme textuelle. Après un travail de réduction de cette forme, elle est traduite en automates temporisés.

Cette méthode produit à nouveau de nombreux modèles dont les interactions sont complexes. Faute d'un exemple de passage à l'échelle, la faisabilité de la méthode sur des SFC de grandes dimensions n'est pas encore démontrée. L'effort à fournir pour proposer cette méthode sur une spécification Grafcet à la place d'un code SFC ainsi que la démonstration de son passage à l'échelle nous semblent des tâches difficiles qui devraient constituer le cœur d'un travail de recherche.

### 3.2.2.3 Traduction d'une spécification Grafcet en système d'équations algébriques

[Machado \*et al.\* \(2006b\)](#) propose de décrire un Grafcet en utilisant des équations algébriques qui permettent de définir quelles transitions sont franchissables (validées et réceptivités associées vraies), puis de définir les prochaines étapes actives (en fonction des transitions franchissables et des étapes déjà actives) pour finir par l'émission des variables de sortie (en fonction des étapes actives). Cette méthode est mise en œuvre sur un cas simple (avec une spécification Grafcet non temporisée) pour comparer les méthodes de vérification avec et sans modèle de la partie opérative.

Ces travaux constituent une base solide pour traduire la spécification Grafcet en un jeu d'équations algébriques qui seront utilisées dans un automate modélisant le contrôleur. Cependant, le cas des temporisations n'étant pas traité, il conviendra de se pencher sur ce sujet, ce qui est réalisé dans la section [3.2.4.2](#).

## 3.2.3 Modélisation d'un contrôleur non temporisé

### 3.2.3.1 Bref rappel sur la norme de spécification Grafcet

Le Grafcet est un langage de spécification de comportement. Il permet donc de décrire précisément le comportement de n'importe quel système logique et est défini dans la norme [IEC 60848 \(2002\)](#) :

Cette norme est destinée principalement aux utilisateurs (concepteurs, réalisateurs, agents de maintenance, etc.) qui ont besoin de spécifier le comportement

d'un système (commande d'une machine automatique, composant de sûreté, etc.). Ce langage de spécification peut également servir de moyen de communication entre les concepteurs et les utilisateurs de systèmes automatisés.

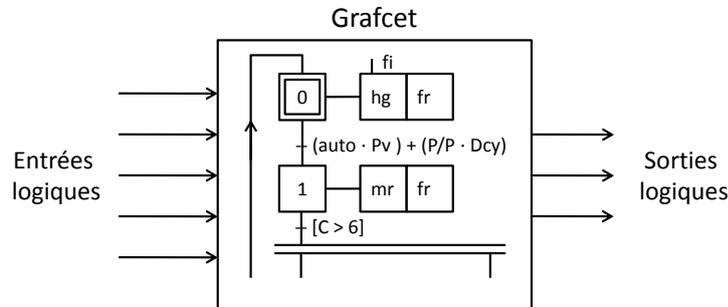


FIGURE 3.15 – Principe d'une spécification Grafcet

Il faut prendre garde à ne pas confondre le Grafcet, qui est un langage de *spécification*, et le SFC (norme IEC 61131 (1999)), qui lui est destiné à l'*implantation* de la spécification dans un contrôleur industriel.

### 3.2.3.1.1 Vocabulaire associé au Grafcet

Un bref rappel du vocabulaire du Grafcet est donné ci-dessous ; la définition complète est disponible dans la norme IEC 60848 (2002).

Le Grafcet est un graphe composé d'*étapes* (représentées par des carrés) qui sont liées à des *transitions* (représentées par des traits horizontaux) au moyen d'arcs orientés. L'alternance étape - transition ne peut en aucun cas être brisée.

Le sens de parcours du graphe est par défaut du haut vers le bas. Pour des raisons de lisibilité, les arcs suivant ce sens ne portent pas de flèche, mais les arcs montants se voient adjoindre une flèche pour éviter tout risque d'erreur.

Chaque *étape* représente une partie de l'état du système et peut être active ou inactive, une variable booléenne (nommée variable d'activité d'étape) est liée à l'activité de chaque étape et peut être utilisée dans le graphe pour modéliser des rendez-vous. L'ensemble des étapes actives forme la *situation* du Grafcet, représentant l'état de celui-ci. Des *actions* peuvent être associées aux étapes d'un Grafcet : elles sont appliquées lorsque l'étape est active et agissent sur les variables de sortie du Grafcet (agissant sur la partie opérative).

chaque *transition* comporte une *réceptivité* : cette condition est une expression booléenne qui est utilisée pour définir si la transition est franchissable. Une réceptivité peut contenir des variables d'entrée (émises par la partie opérative), des variables d'activité d'étapes et des expressions fonctions du temps.

Le principe d'un Grafcet est de faire évoluer une *situation* en changeant d'*étape* active au moyen des *transitions*. À chaque situation correspond une configuration des variables de sortie du Grafcet, qui sont utilisées par la partie opérative, tandis que les évolutions au travers des transitions sont sensibles aux variables d'entrée du Grafcet, générées par la partie opérative, et aux variables d'activité d'étapes.

### 3.2.3.1.2 Règles d'évolution d'un Grafcet

Le comportement d'un Grafcet est défini par les cinq règles suivantes :

1. À l'instant initial, toutes les étapes initiales (définies par le concepteur au moyen d'étapes à double carré) sont actives. Les autres étapes sont inactives.
2. Une transition est dite *validée* lorsque toutes les étapes amont (qui la précèdent directement au sens des liens orientés) sont actives. Une transition est dite *franchissable* lorsqu'elle est validée et que sa réceptivité associée est vraie. De plus, *toute transition franchissable est immédiatement franchie*.
3. Le franchissement d'une transition provoque simultanément l'activation de toutes les étapes aval et la désactivation de toutes les étapes amont.
4. Si plusieurs transitions sont franchissables, elles sont simultanément franchies.
5. Si une étape est simultanément activée et désactivée, elle reste active.

### 3.2.3.2 Modélisation algébrique d'un Grafcet non temporisé

#### 3.2.3.2.1 Limitations des travaux

La norme Grafcet permet de définir des comportements en utilisant des primitives fortes et complexes. Pour des raisons de temps principalement, dans le cadre de ces travaux, les sémantiques suivantes ne sont pas retenues :

- Le comportement événementiel sur les variables dans les réceptivités (front montant / front descendant).
- Le comportement événementiel sur les émissions des variables de sortie (pas d'affectation des variables de sortie).
- Le comportement temporisé sur les variables d'entrée dans les réceptivités.

L'ensemble des Grafcets traités est toutefois susceptible de contenir :

- Des divergences / convergences en OU structurelles (choix de séquences).
- Des divergences / convergences en ET structurelles (séquences parallèles).
- Des assignations conditionnelles.
- Des temporisations associées à des variables d'étape dans les réceptivités.

#### 3.2.3.2.2 Description formelle d'un Grafcet

En se basant sur la définition proposée par [Provost et al. \(2011\)](#), on définit un Grafcet comme un 6-uplet  $(I_g, O_g, S, S_{init}, T, A)$  avec :

- $I_g$  l'ensemble des variables booléennes d'entrée.
- $O_g$  l'ensemble des variables booléennes de sortie.
- $S$  l'ensemble des étapes (*Steps*),  $s \in S$  une étape et  $X_s$  la variable booléenne liée à l'activité de l'étape  $s$ .
- $S_{init} \subseteq S$  l'ensemble des étapes initiales du graphe.

- $T$  l'ensemble des transitions  $t$  de la forme  $t = (Up_t, Dn_t, TC_t)$  avec :
  - $Up_t \subset S$  l'ensemble des étapes amont de la transition  $t$ .
  - $Dn_t \subset S$  l'ensemble des étapes aval de la transition  $t$ .
  - $TC_t \in B_g$  (défini ci-dessous) la réceptivité associée à la transition  $t$ .
- $A$  l'ensemble des actions  $a$  de la forme  $a = (s, o, cond)$  avec :
  - $s \in S$  l'étape à laquelle est attachée l'action.
  - $o \in O_g$  la variable concernée par l'action.
  - $cond \in B_g$  (défini ci-dessous) la condition associée à l'action. Si  $cond = \underline{1}$ , alors l'action n'est pas conditionnelle.

avec  $B_g$  l'ensemble des expressions booléennes  $b_g$  d'un Grafcet définies comme :

$$b_g ::= \underline{1} \mid b_g \cdot b_g \mid b_g + b_g \mid \overline{b_g} \mid (b_g) \mid i \mid X_s \quad (3.1)$$

où  $\underline{1}$  représente la réceptivité toujours vérifiée,  $\overline{b_g}$  le complément de  $b_g$ ,  $i \in I_g$  et  $X_s$  est la variable d'activité de l'étape  $s$ .

On définit aussi, pour des questions de simplicité d'utilisations futures :

- $Pre_s \subset T$  l'ensemble des transitions amont de l'étape  $s$ .
- $Post_s \subset T$  l'ensemble des transitions aval de l'étape  $s$ .
- $FC_t$  la variable booléenne associée à la transition  $t$  qui représente sa possibilité d'être tirée. Cette variable passe à vrai lorsque la transition est franchissable, à faux sinon.

### 3.2.3.2.3 Représentation algébrique du Grafcet sans temporisations

Cette traduction se fera au travers de trois jeux d'équations, comme définies par [Machado et al. \(2006b\)](#) :

**Calcul des conditions de tir** Pour chaque transition  $t \in T$ , on calcule la condition de tir  $FC_t$  telle que :

$$FC_t = \left( \bigwedge_{s \in Up_t} X_s \right) \wedge TC_t \quad (3.2)$$

**Calcul des étapes actives** Pour chaque étape  $s \in S$ , on calcule la nouvelle valeur de la variable d'activité associée  $X_s$  telle que :

$$X_s = \left( \bigvee_{t \in Pre_s} FC_t \right) \vee \left( X_s \wedge \left( \bigwedge_{u \in Post_s} (\neg FC_u) \right) \right) \quad (3.3)$$

**Calcul des sorties** Enfin, pour chaque sortie booléenne  $o \in O_g$ , on calcule la valeur telle que :

$$o = \bigvee (X_s \wedge cond_s) \quad (3.4)$$

où  $X_s$  est la variable d'activité de l'étape  $s$  telle qu'il existe une action  $a \in A$  ( $a = (s, o, cond_s)$ ) où la variable  $o$  est assignée dans l'étape  $s$ . Lorsque  $cond_s = \underline{1}$ , l'expression  $\wedge cond_s$  sera supprimée, car inutile.

À l'initialisation, les étapes initiales voient leur variable d'activité mise à vrai, les autres étant initialisées à faux. De même, les variables concernées par des actions associées aux étapes initiales sont mises à vrai à l'initialisation, contrairement aux autres qui sont laissées à faux.

#### 3.2.3.2.4 Application à un Grafcet sans temporisations

La figure 3.16 montre une spécification Grafcet de la commande d'une porte de sécurité, les trois jeux d'équations algébriques qui lui correspondent sont donnés figure 3.17. Le comportement de la porte est le suivant :

- La position de la porte est détectée par deux capteurs, porte ouverte (OUT à vrai et IN à faux) et porte fermée (IN à vrai et OUT à faux).
- Lors de l'appui sur le bouton START, la porte se ferme (CLOSE assigné) ou s'ouvre (OPEN assigné) selon sa position actuelle.
- En réponse à l'appui sur le bouton START, une lampe de contrôle (variable LIGHT) s'allume pour signifier le lancement de la procédure.
- Durant les mouvements de la porte, la lampe de contrôle reste allumée (variable LIGHT) et une sonnerie retentit (variable BUZZER).
- Lorsque la porte est totalement ouverte ou fermée, sa position est verrouillée (variable LOCK à vrai) pour prévenir tout mouvement intempestif.

Les notations suivantes sont utilisées dans les conditions des transitions pour la spécification Grafcet :

- + est l'opérateur de disjonction,
- · est l'opérateur de conjonction,
- $\bar{x}$  est le complément de  $x$ .

### 3.2.3.3 Modélisation d'un Automate Programmable Industriel exécutant un Grafcet non temporisé

#### 3.2.3.3.1 Cycle d'un Automate Programmable Industriel

Comme précisé précédemment, nous nous intéressons dans cette étude à un contrôleur de type API<sup>4</sup> à scrutation cyclique des entrées. Ce type de matériel suit un cycle d'exécution bien connu, représenté sur la figure 3.18 :

- Les entrées sont copiées dans une mémoire interne.
- Le code du contrôleur est exécuté, les sorties sont calculées.
- Les sorties du contrôleur sont émises au niveau des bornes de sortie.

Le code exécuté ici doit se comporter comme spécifié dans le Grafcet. Ainsi, la phase d'exécution du code correspond au calcul des trois jeux d'équations précédemment définis.

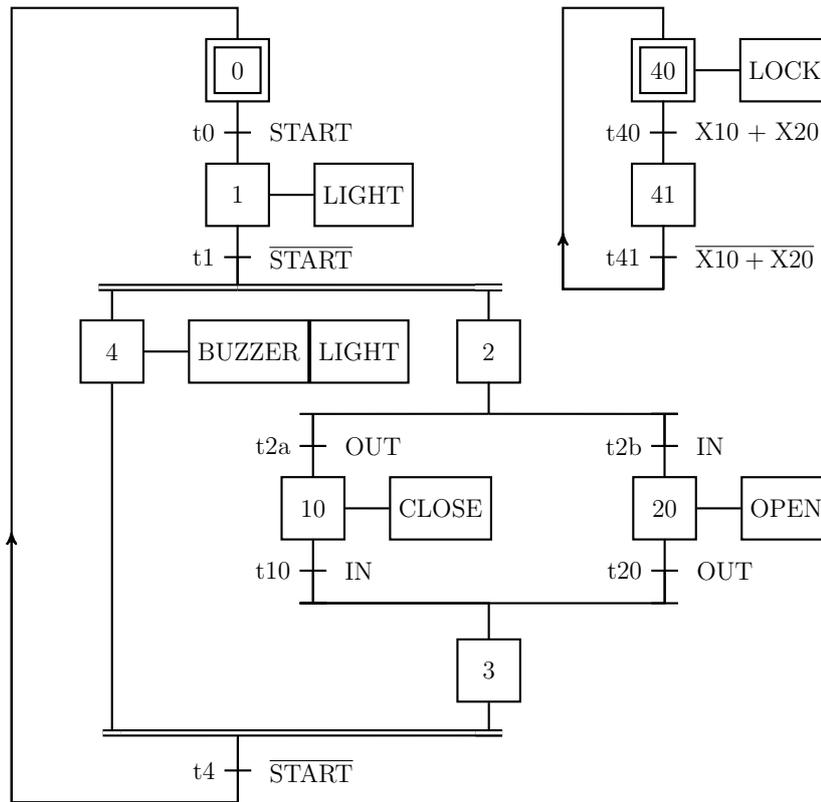


FIGURE 3.16 – Spécification Grafcet de la commande de la porte de sécurité.

### 3.2.3.3.2 Modèle d'un API exécutant un Grafcet non temporisé en ATVD

Le premier modèle construit pour représenter le contrôleur est présenté dans la figure 3.19. Il comprend les trois jeux d'équations sur trois transitions avec trois localités.

Les phases de copie des entrées et d'émission des sorties sont abordées dans la section suivante car elles traitent toutes deux d'interactions entre le contrôleur et son environnement.

La phase d'exécution du code est donc modélisée par trois transitions sans gardes. La première transition ( $0 \rightarrow 1$ ) comporte dans ses actions le jeu d'équations du calcul des conditions de tir. La seconde ( $1 \rightarrow 2$ ) permet de mettre à jour les étapes actives. La dernière ( $2 \rightarrow 3$ ) s'occupe de calculer les variables de sortie.

Comme le temps d'exécution du cycle API est très faible devant les temps physiques mis en jeu dans la partie opérative, ces évolutions sont considérées comme instantanées et par conséquent modélisées par des transitions urgentes.

Ce choix aura des conséquences importantes qui seront discutées dans la section suivante.

$FC_{t0} = X_0 \wedge START$ $FC_{t1} = X_1 \wedge (\neg START)$ $FC_{t2a} = X_2 \wedge OUT$ $FC_{t2b} = X_2 \wedge IN$ $FC_{t4} = (X_4 \wedge X_3) \wedge (\neg START)$ $FC_{t10} = X_{10} \wedge IN$ $FC_{t20} = X_{20} \wedge OUT$ $FC_{t40} = X_{40} \wedge (X_{10} \vee X_{20})$ $FC_{t41} = X_{41} \wedge (\neg(X_{10} \vee X_{20}))$	$X_0 = FC_{t4} \vee (X_0 \wedge (\neg FC_{t0}))$ $X_1 = FC_{t0} \vee (X_1 \wedge (\neg FC_{t1}))$ $X_2 = FC_{t1} \vee (X_2 \wedge (\neg FC_{t2a}) \wedge (\neg FC_{t2b}))$ $X_3 = (FC_{t10} \vee FC_{t20}) \vee (X_3 \wedge (\neg FC_{t4}))$ $X_4 = FC_{t1} \vee (X_4 \wedge (\neg FC_{t4}))$ $X_{10} = FC_{t2a} \vee (X_{10} \wedge (\neg FC_{t10}))$ $X_{20} = FC_{t2b} \vee (X_{20} \wedge (\neg FC_{t20}))$ $X_{40} = FC_{t41} \vee (X_{40} \wedge (\neg FC_{t40}))$ $X_{41} = FC_{t40} \vee (X_{41} \wedge (\neg FC_{t41}))$
(a) Calcul des conditions de tir.	(b) Calcul des variables d'activité.

$$LIGHT = X_1 \vee X_4$$

$$BUZZER = X_4$$

$$OPEN = X_{20}$$

$$CLOSE = X_{10}$$

$$LOCK = X_{40}$$

(c) Calcul des valeurs des sorties.

FIGURE 3.17 – Spécification Grafcet d'une porte de sécurité accompagnée de ses trois jeux d'équations algébriques.

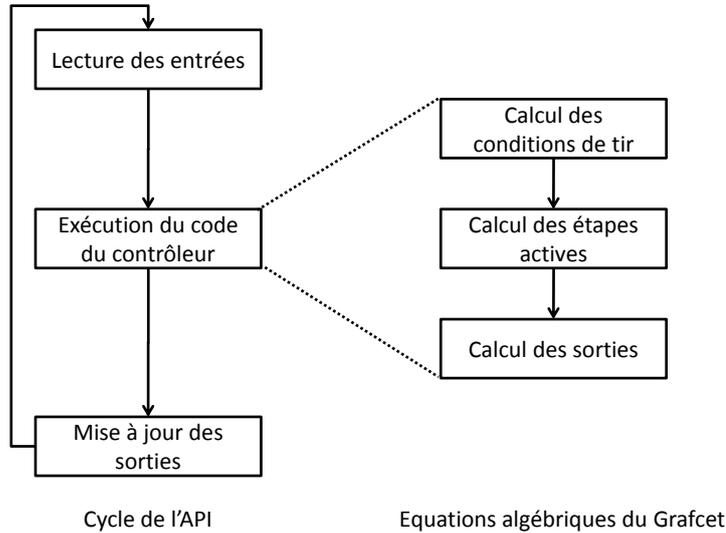


FIGURE 3.18 – Cycle d'un API.

### 3.2.4 Modèle final, temporisé, du contrôleur

L'exemple de la spécification de la porte de sécurité est simple et dépourvu de temporisations. Mais les spécifications de systèmes d'automatisation plus complexes contiennent généralement des temporisations qu'il va falloir intégrer à nos modèles. Le Grafcet du manipulateur est un bon exemple de spécification qui contient des temporisations (pour la

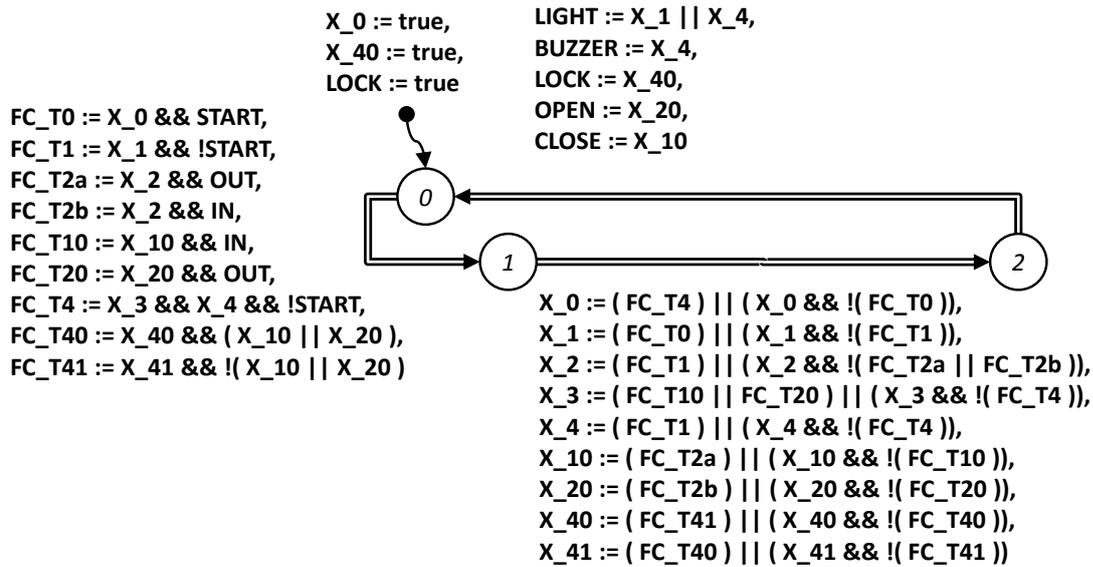


FIGURE 3.19 – Première version du modèle du contrôleur pour le Grafset de la figure 3.16.

gestion de la ventouse). Il faut donc traduire ce comportement dans les jeux d'équations algébriques, et ensuite l'intégrer au modèle du contrôleur.

### 3.2.4.1 Spécification Grafset du contrôleur du préhenseur

Le comportement attendu du contrôleur de l'exemple a été décrit succinctement dans le chapitre 1. La figure 3.20 représente ce comportement spécifié à l'aide de la norme Grafset. Par rapport aux règles de représentation énoncées plus tôt, le principal ajout concerne les temporisations sur les réceptivités. Les temporisations sont de la forme  $Ns/X_i$  et représentent le fait que la condition devient valide  $N$  secondes après l'activation de l'étape  $i$ .

Les entrées (informations des capteurs) et sorties (ordres) de ce graphe sont respectivement les sorties et les entrées de la partie opérative.

### 3.2.4.2 Traduction d'un Grafset avec des temporisations

Les équations précédemment décrites ne prennent pas en compte le temps physique, et donc par extension ne prennent pas en compte les temporisations.

Afin de pouvoir simplement intégrer le comportement temporisé souhaité, la solution retenue se rapproche de celle proposée dans la norme Grafset pour introduire des tests sur des variables réelles (à valeurs dans  $\mathbb{R}$ ) comme montré sur la figure 3.21a : un bloc indépendant nommé test réalise la comparaison entre la grandeur  $C$  est le seuil 6, fournissant le résultat (variable booléenne) au Grafset.

Pour les temporisations, on considère donc une temporisation externe comme montré sur la figure 3.21b. Elle est contrôlée par le Grafset au moyen d'une variable booléenne de la forme  $T_{X_s}Ns$  avec  $X_s$  la variable d'activité de l'étape  $s$  liée à la temporisation

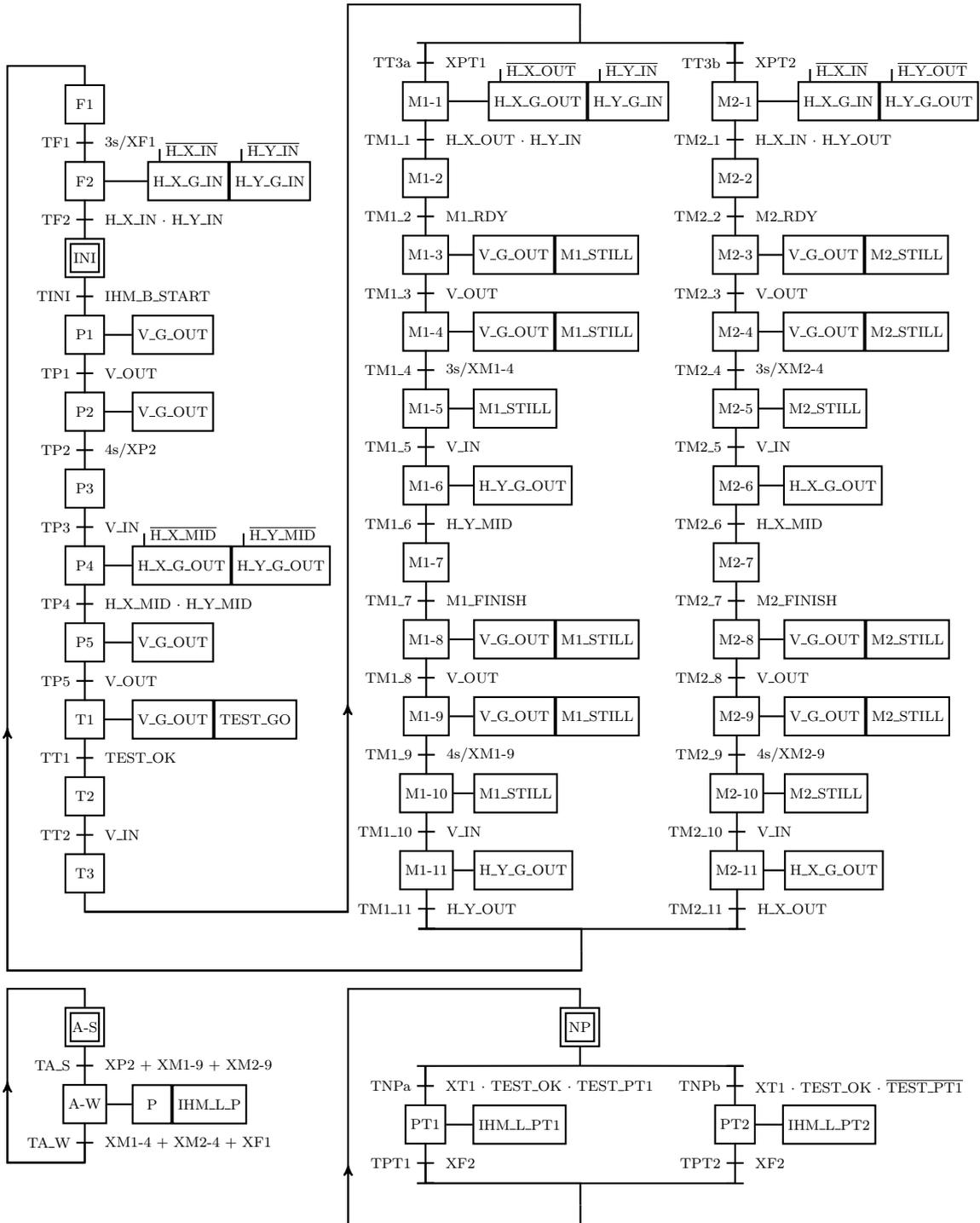
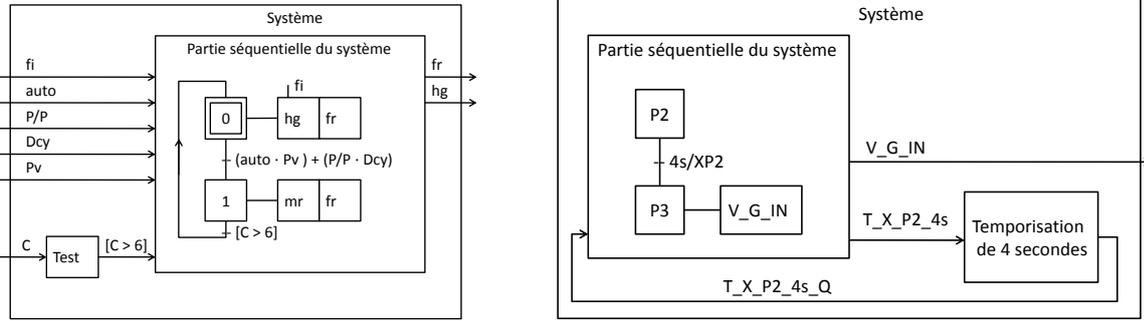


FIGURE 3.20 – Spécification Grafcet de l'exemple.

et  $N_s$  le temps d'attente (ici en secondes). Lorsque le temps est écoulé, la temporisation met à vrai la variable booléenne  $T\_X\_s\_N_s\_Q$  qui signifie que le délai souhaité est passé. Cette modélisation est possible car nous limitons ici aux seules temporisations sur

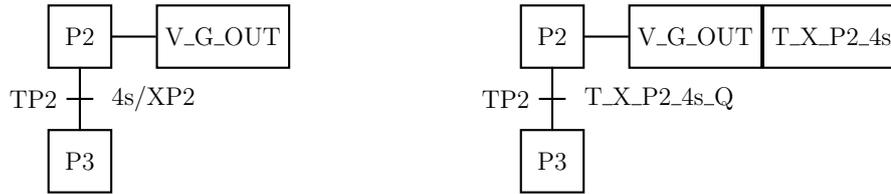
les transitions, donc relatives aux variables d'activité des étapes.



(a) Exemple d'utilisation d'une variable de type réel avec un Grafcet. (b) Proposition d'inclusion de la sémantique de temporisation.

FIGURE 3.21 – Ajout d'un comportement non séquentiel dans un Grafcet.

La figure 3.22 montre une partie du Grafcet de l'exemple avec une temporisation (figure 3.22a), sa traduction avec apparition explicite des variables liées à la temporisation (figure 3.22b) et enfin les parties des équations y faisant référence (figure 3.22c).



(a) Partie du Grafcet de l'exemple contenant une temporisation.

(b) Grafcet avec variables de contrôle de la temporisation explicitées.

$$FC_{TP2} = X_{P2} \wedge T\_X\_P2\_4s\_Q$$

$$X_{P2} = FC_{TP1} \vee (X_{P2} \wedge (\neg FC_{TP2}))$$

$$T\_X\_P2\_4s = X_{P2}$$

(c) Équations algébriques liées à la temporisation.

FIGURE 3.22 – Variables de contrôle des temporisations.

### 3.2.4.3 Ajout des temporisations dans le modèle

Il faut maintenant modifier le modèle afin qu'il mesure du temps physique lors de l'utilisation des variables de contrôle précédemment ajoutées dans le Grafcet.

Ainsi, un modèle de temporisation est ajouté au modèle de l'API précédemment décrit pour chaque temporisation. Ce modèle est inspiré des modèles de *timer* proposés par Mader et Wupper (1999) et Rossi et Schnoebelen (2000). La figure 3.23 montre un modèle générique de temporisation :

- Lorsque la localité initiale est active, la temporisation est désactivée (localité *OUT* active). Quand la variable de contrôle (assignée par le Grafcet)  $T\_X\_s\_Ns$  passe à vrai, la localité active passe à *RUN* : la temporisation est lancée (horloge  $T\_S$  mise à zéro).
- Depuis cette localité, deux évolutions sont possibles. La temporisation peut être désactivée à nouveau, si la variable de contrôle repasse à faux. Sinon, lorsque l'horloge arrive à la valeur de seuil ( $T\_S = N$ , liée à la temporisation du Grafcet), la localité active passe à *OK*, signifiant que le temps demandé s'est écoulé. La variable  $T\_X\_s\_Ns\_Q$  passe alors à vrai, ce qui peut permettre à une condition de tir de devenir vérifiée.
- Lorsque la temporisation est en *OK*, la désactivation de la variable de contrôle permet la ré-initialisation du modèle, ré-initialisant par là même la variable de fin de temporisation  $T\_X\_s\_Ns\_Q$ .

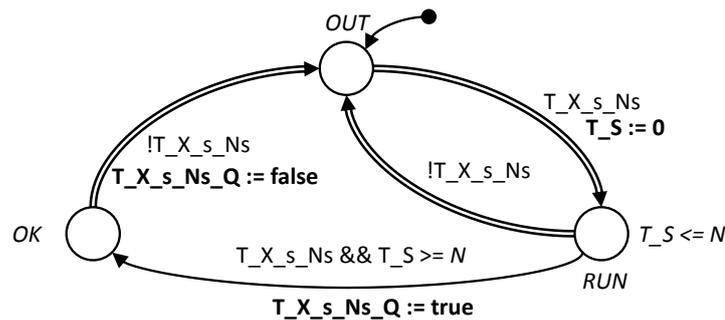


FIGURE 3.23 – Modèle d'une temporisation associée à l'étape  $s$  (donc liée à la variable  $X_s$ ) de  $N$  secondes.

Ainsi, pour chaque temporisation du Grafcet, un automate de temporisation lié aux variables de contrôle (assignées par le Grafcet) et assignant la variable de fin de temporisation (utilisée par le Grafcet dans les conditions de tir) est ajouté.

Le modèle obtenu pour l'API du manipulateur est détaillé dans l'annexe D, il comprend un modèle du contrôleur (avec les équations associées à la spécification Grafcet et les variables ajoutées pour les temporisations détaillées dans l'annexe D.1) ainsi que les modèles des temporisations associées aux étapes P2, M1-4, M2-4, M1-9, M2-9 et F1.

### 3.3 Construction du modèle du système bouclé

Nous avons jusqu'à présent élaboré :

- Le modèle de la partie opérative, réseau d'ATVD communiquant par variables partagées,

- Le modèle de l'API, constitué d'un modèle du contrôleur dont la structure repose sur le cycle d'exécution de l'API, et d'un ensemble de modèles de temporisations.

Dans ce qui suit, nous considérerons d'une part le modèle du contrôleur, et d'autre part le modèle du *système contrôlé* comme montré dans la figure 3.24. La partie opérative est naturellement intégrée à ce modèle du système contrôlé, cependant nous choisissons aussi d'y intégrer les modèles des temporisations car elles sont réalisées au travers de fonctions opératives internes à l'API mais extérieures au cycle de scrutation cyclique des entrées modélisé par le modèle du contrôleur.

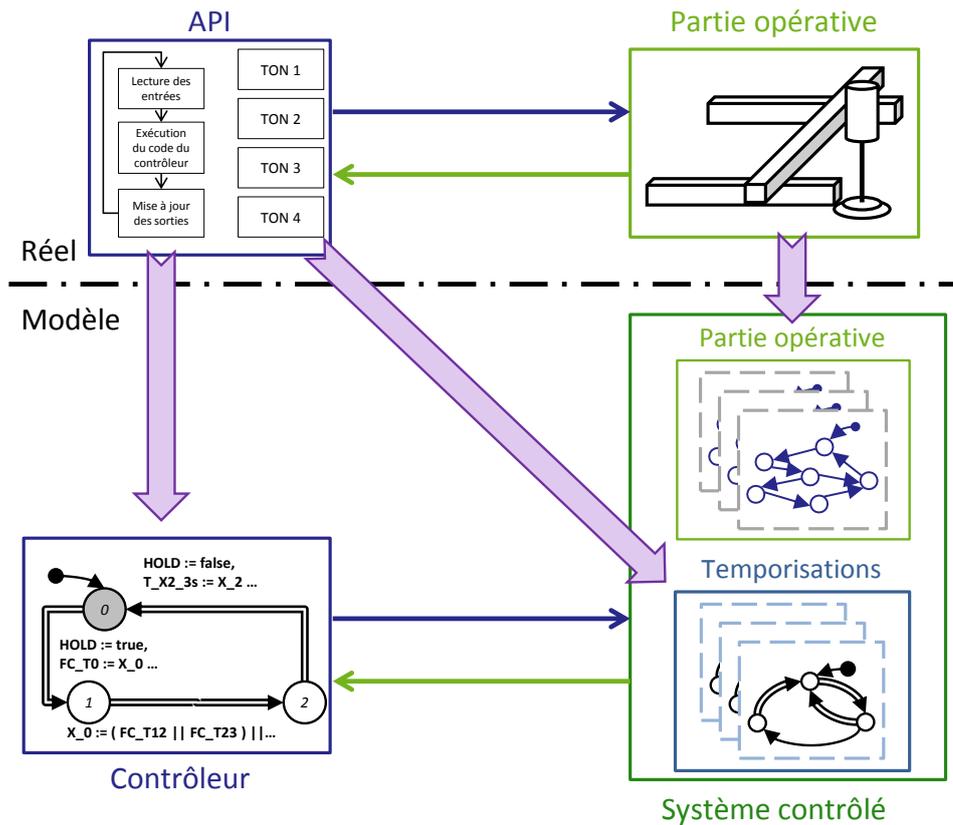


FIGURE 3.24 – Présentation du couple contrôleur / système contrôlé.

Ces deux modèles (contrôleur et système contrôlé) communiquent au travers de variables partagées. Il importe à présent d'étudier leurs évolutions, en particulier afin de s'assurer qu'aucun comportement irréaliste ne se produit dans le modèle du système bouclé.

### 3.3.1 Prise en compte des phases de lecture et d'écriture des entrées/sorties de l'API

Lors de l'étape de modélisation d'un cycle d'un API (figure 3.18), les phases de copie des entrées et d'émission des sorties n'ont pas été traitées. Maintenant que l'on s'intéresse au système bouclé, il faut considérer ces phases.

Un API réel utilise une copie des valeurs de ses entrées pour exécuter son cycle. Cette solution peut être directement reprise. En effet, en dédoublant toutes les variables d'entrée de l'API (avec un suffixe `_mem`, par exemple), on pourrait travailler sur des copies des variables d'entrée. Cette solution est cependant très coûteuse. En effet, sur un modèle industriel de grande dimension, la copie des variables d'entrée de l'API revient à ajouter un grand nombre de variables, ce qui peut être un frein important lors de l'utilisation d'outils d'analyse formelle.

Ainsi, nous choisissons dans ces travaux de partir sur une autre idée : si l'on ne souhaite pas mémoriser les variables d'entrée, alors il suffit de les forcer à rester inchangées durant chaque cycle.

Pour ce faire, une variable booléenne notée `HOLD` est ajoutée ; son comportement est le suivant :

- Cette variable est assignée uniquement par le modèle du contrôleur, mais est utilisée par un ensemble de modèles comprenant les modèles des capteurs et ceux des temporisations.
- Elle est mise à vrai lors du franchissement de la première transition du modèle du contrôleur (calcul des conditions de tir).
- Elle est remise à faux lors du tir de la dernière transition du modèle du contrôleur (émission des sorties). Ainsi elle reste à vrai durant tout le cycle du contrôleur.
- Afin de geler les entrées de l'API, *toutes* les transitions comportant une assignation d'une variable d'entrée de l'API (modèles de groupes de capteurs et modèles de temporisations) voient leur garde modifiée par l'ajout d'un « `&&!HOLD` », interdisant de fait le franchissement si `HOLD` est à vrai.

Avec l'ajout de cette variable, lorsque le contrôleur a commencé un cycle et tant qu'il ne l'a pas terminé, les évolutions agissant sur les variables d'entrée sont interdites. On garantit donc que les variables d'entrée utilisées tout au long du cycle du contrôleur restent inchangées durant tout ce cycle.

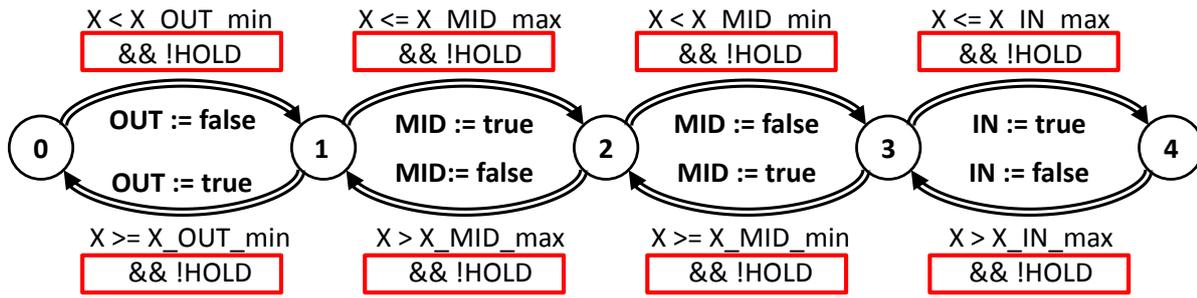
À la fin du cycle du contrôleur, la variable `HOLD` est repassée à faux afin de laisser ces évolutions se produire.

La figure 3.25a montre une version modifiée d'un modèle générique de groupe de trois capteurs. La figure 3.25b montre pour sa part une version modifiée d'un modèle de temporisation.

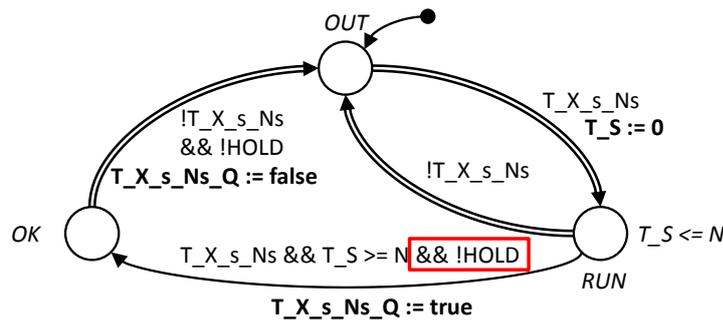
En ce qui concerne l'étape d'émission des sorties, il n'y a pas de difficultés. En effet, l'utilisation des variables partagées pour la communication entre les modèles implique que les sorties calculées lors de l'assignation par la dernière transition du modèle du contrôleur sont directement utilisées par le modèle de la partie opérative. La figure 3.26 montre l'ajout de la variable `HOLD` sur le modèle du contrôleur de l'exemple.

### 3.3.2 Concurrence entre modèles du système contrôlé et du contrôleur

Lors des premières simulations du système bouclé, un problème de concurrence entre le modèle du contrôleur et celui du système contrôlé est apparu.



(a) Modèle générique d'un groupe de capteur modifié.



(b) Modèle générique d'une temporisation modifié.

FIGURE 3.25 – Modèles modifiés pour prendre en compte la variable HOLD.

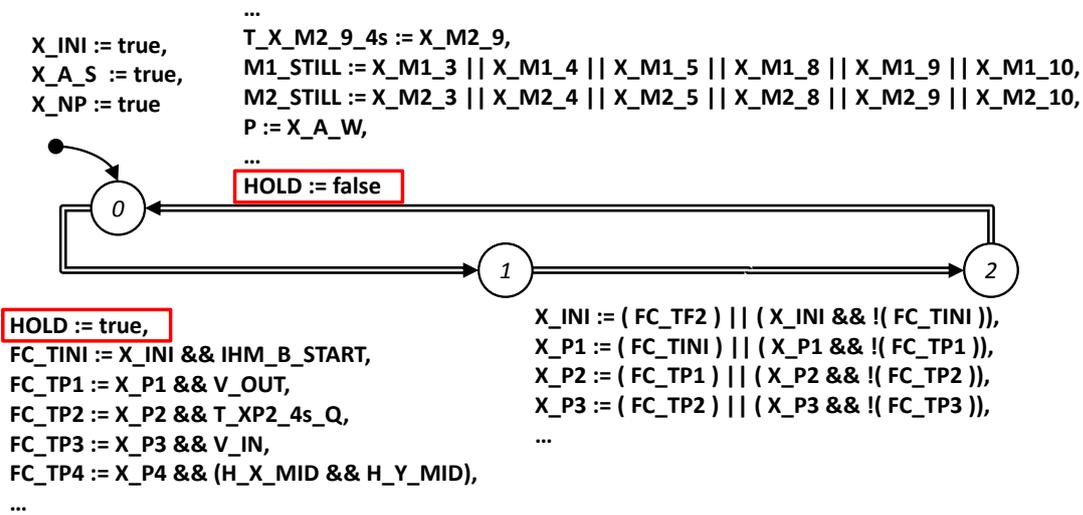


FIGURE 3.26 – Modèle partiel du contrôleur de l'exemple avec l'ajout de HOLD (encadré).

### 3.3.2.1 Mise en évidence au travers d'une séquence d'évolutions de l'exemple

La figure 3.27 montre une séquence d'évolutions possible du modèle du système bouclé. Pour des raisons de lisibilité, l'ensemble des modèles de l'exemple n'a pas été représenté. Seules des représentations partielles du modèle du vérin pneumatique vertical (V\_V) et du modèle du contrôleur (Cont.) sont présentées. Afin de faciliter la compréhension, une

partie de la spécification Grafcet est aussi montrée. Dans cette dernière, les étapes actives sont grisées pour une plus grande lisibilité, et les transitions qui doivent être tirées sont aussi grisées.

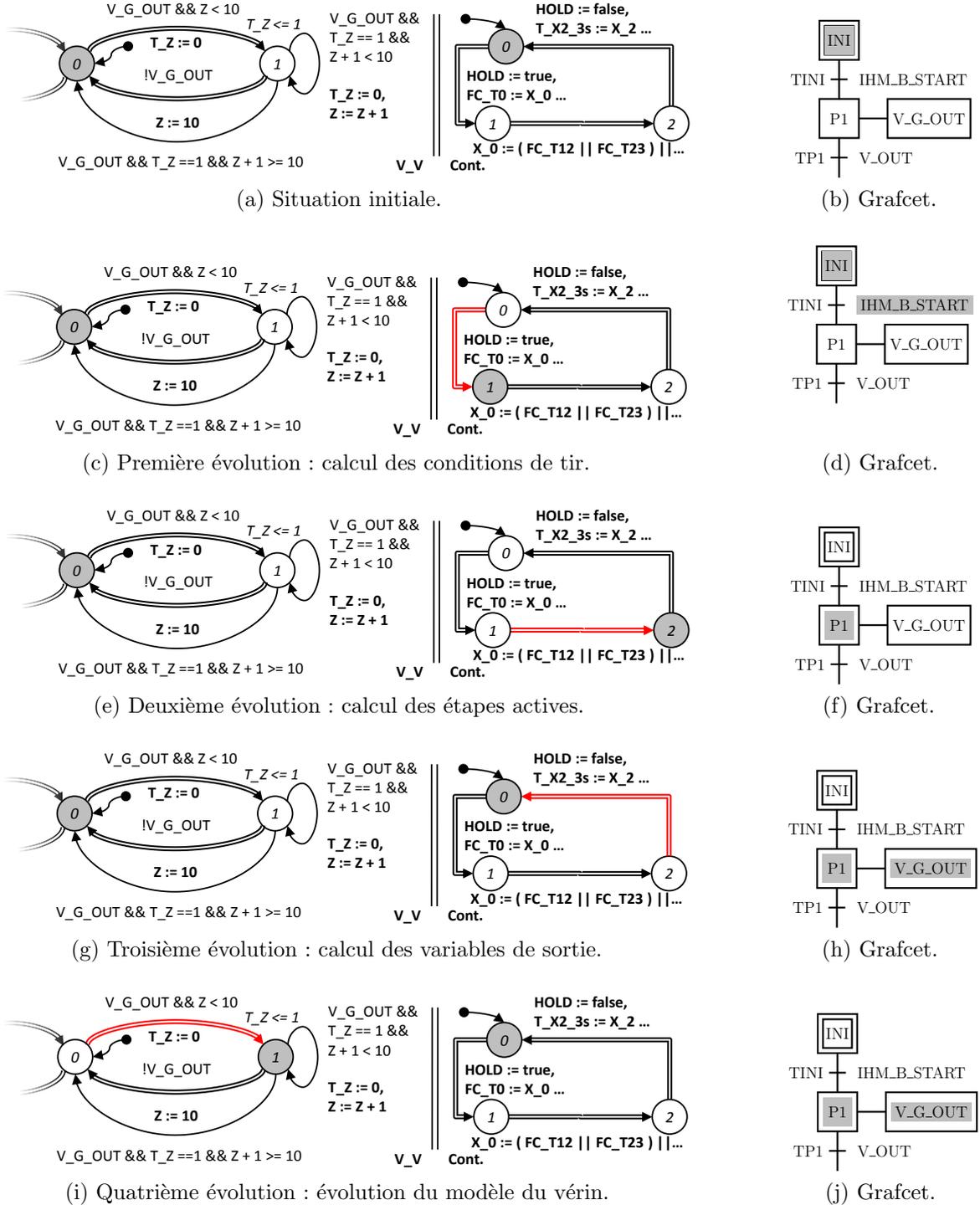


FIGURE 3.27 – Séquence d'évolutions du système bouclé.

À l'initialisation (figure 3.27a), les localités initiales des deux modèles (V\_V et API)

sont actives ( $X\_INI$  est donc à vrai). On considère que la variable booléenne  $IHM\_B\_START$  est à vrai, les autres variables booléennes étant à faux. La tige du vérin vertical étant considérée rentrée, on considère que  $Z$  vaut 0.

- La première évolution concerne le modèle du contrôleur (figure 3.27c) car le modèle du vérin ne peut pas évoluer ( $V\_G\_OUT$  est à faux). Cette évolution correspond au calcul des conditions de tir et définit que  $FC\_TINI$  est vraie. De plus,  $HOLD$  est passée à vrai.
- Le modèle du contrôleur continue d'évoluer. Le calcul des nouvelles étapes actives conduit à désactiver l'étape  $INI$  et à activer l'étape  $P1$  (figure 3.27e) car la condition de tir  $FC\_TINI$  est vraie.
- La troisième évolution (figure 3.27g) du modèle du contrôleur met à jour les sorties en mettant la variable  $V\_G\_OUT$  à vrai. La variable  $HOLD$  repasse à faux.
- Ensuite, deux transitions urgentes sont franchissables : une du modèle du contrôleur représentant le départ d'un nouveau cycle et une autre modélisant le changement d'état du vérin vertical. Les cycles successifs du modèle du contrôleur n'amenant aucune modification des étapes actives et des sorties émises, nous choisissons de tirer la transition urgente du modèle  $V\_V$ , comme le montre la figure 3.27i. Ce dernier évolue donc vers la localité 1 modélisant la sortie de la tige.

Cette dernière évolution mène à **un blocage du modèle de la partie opérative**. En effet, les transitions urgentes du modèle du contrôleur étant toujours franchissables, la sémantique temporelle des ATVD est interdite : l'horloge  $T\_V$  associée au modèle du vérin ne pourra jamais évoluer. La transition bouclée modélisant la sortie de la tige ne pourra donc *jamais* être tirée (puisque la contrainte temporelle sera toujours fausse) : la tige du vérin ne pourra *jamais* sortir malgré l'ordre donné par le contrôleur. On notera que la transition urgente  $1 \rightarrow 0$  du modèle  $V\_V$  n'est pas franchissable car  $V\_G\_OUT$  est à vrai.

Ce comportement est fortement irréaliste dans le cadre d'une partie opérative sans défaillances !

### 3.3.2.2 Généralisation et analyse

Le problème mis en évidence dans la partie précédente est dû à une concurrence entre le modèle du contrôleur et celui du système contrôlé. De manière plus générale, le modèle du contrôleur utilise des transitions urgentes car son temps de cycle est très court devant les temps caractéristique de la partie opérative. Un API réel démarre sans pause un nouveau cycle à la fin du précédent, ainsi initialement son modèle ne contient aucune garde sur ses transitions. Ceci équivaut pourtant à laisser dans le modèle du contrôleur une boucle ininterrompue de transitions urgentes sans aucune garde et donc toujours franchissables, ce qui a pour conséquence un gel *permanent* de la sémantique temporelle. Ceci ne laisse aucune possibilité d'évolution pour le modèle de la partie opérative qui, modélisant des évolutions matérielles, doit impérativement utiliser du temps physique et donc la sémantique temporelle des ATVD.

Comme l'a montré la section 3.1.4, l'utilisation de transitions urgentes est obligatoire pour garantir un comportement réaliste du modèle de la partie opérative et l'utilisation

de transitions non urgentes pour modéliser le cycle d'un API n'a pas de sens : le modèle du contrôleur ne serait alors plus prioritaire devant les évolutions consommant du temps physique !

La solution au problème de blocage du modèle de la partie opérative consiste à brider les évolutions du modèle du contrôleur en réfléchissant aux événements qui provoquent une évolution de ce modèle.

### 3.3.3 Suppression des concurrences indésirables avec conservation de la réactivité du contrôleur

La solution proposée pour éviter ce blocage consiste en une limitation des évolutions du modèle du contrôleur aux *seuls* cas de figure où c'est nécessaire. On cherche à ne laisser le modèle du contrôleur évoluer que lorsque c'est utile, c'est-à-dire quand cette évolution doit permettre à ce modèle du contrôleur de *réagir* à un changement dans le modèle du système contrôlé, en changeant éventuellement d'étapes actives et de sorties émises. À nos yeux, il n'y a que deux types d'événements qui correspondent à nos critères : un changement dans les variables d'entrée, qui est le mode de fonctionnement le plus utilisé lors du contrôle de la partie opérative, ou l'arrivée à échéance d'une temporisation.

Deux variables sont alors ajoutées pour détecter ces événements :

- EVOL\_PL est une variable booléenne représentant une assignation d'une variable d'entrée du contrôleur par la partie opérative. Cette variable est mise à vrai à chaque fois qu'une transition de la partie opérative met à jour une variable d'entrée du contrôleur au moyen d'une action. Ceci est réalisé en ajoutant `EVOL_PL := true` à l'ensemble des actions de cette transition. Cette variable permet le départ d'un cycle du modèle du contrôleur, qui la remet à zéro lors du début du cycle afin que cette dernière puisse être utilisée à nouveau. La figure 3.28 montre une version modifiée du modèle S\_V du groupe de capteurs associé au mouvement de la tige du vérin vertical comprenant l'ajout de la variable EVOL\_PL dans les actions.
- END\_TI est une variable booléenne qui représente la fin d'une temporisation. Cette variable est mise à vrai lorsqu'une temporisation arrive à échéance et permet aussi un cycle du modèle du contrôleur. Elle est remise à zéro au début du cycle afin de la rendre disponible pour une autre utilisation. La figure 3.29 montre le modèle modifié de la temporisation associée à l'étape P2, avec l'ajout de l'action `END_TI := true` sur la transition modélisant la fin de la temporisation.

Ces deux variables fonctionnent sur le principe des sémaphores : assignées à vrai par un ensemble de modèles et remises à zéro par un autre modèle ne faisant pas partie de l'ensemble précédent.

On peut noter que **l'introduction de ces deux variables ne modifie pas les hypothèses de réactivité et de synchronisme de la norme Grafset**. En effet, tout événement d'entrée est perçu dès son occurrence (passage de EVOL\_PL ou END\_TI à vrai autorisant un cycle du modèle du contrôleur) et traité dans un temps physique nul (les évolutions du contrôleur restant de type urgent, aucune évolution temporisée n'est permise tant que le modèle du contrôleur a des transitions franchissables).

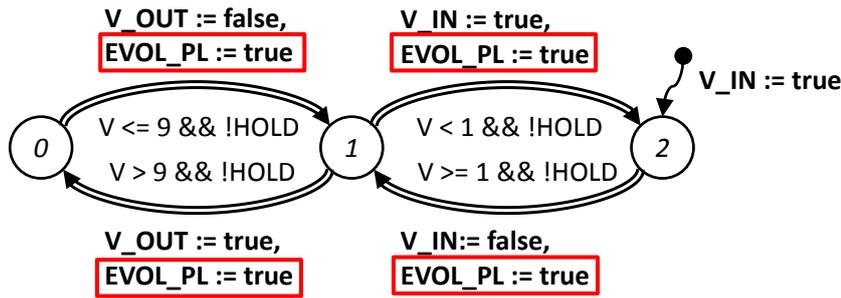


FIGURE 3.28 – Version finale du modèle des capteurs associés au vérin vertical.

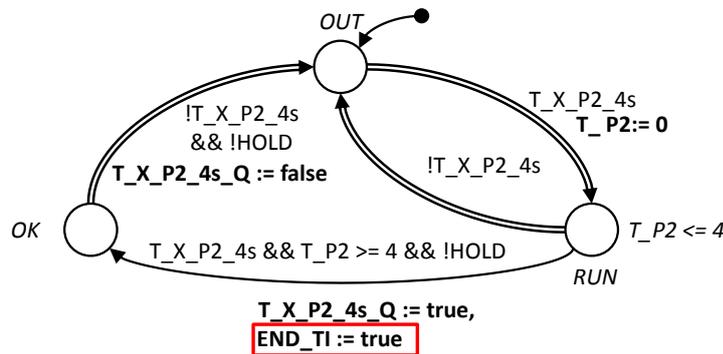


FIGURE 3.29 – Version finale du modèle de la temporisation associée à l'étape P2.

Une fois ces deux variables ajoutées dans les modèles de la partie opérative (représentés dans l'annexe B.2, figure B.5) et des temporisations (représentés dans l'annexe D.3, figure D.6), le modèle du contrôleur, présenté dans la figure 3.30, doit lui aussi être modifié :

- Une localité initiale d'attente, notée  $w$ , est ajoutée.
- Une transition urgente  $w \rightarrow 0$  est ajoutée, elle modélise le départ du cycle du contrôleur. Elle a comme garde  $EVOL\_PL \parallel END\_TI$  pour laisser le franchissement possible si l'une des variables (au moins) est à vrai. Ses actions consistent en la mise à faux des deux variables (ré-initialisation pour une utilisation future) ainsi qu'en la mise à vrai de la variable HOLD (car c'est cette nouvelle transition qui représente le début du cycle et donc, par conséquent, le début du gel des variables d'entrée).

Avec ces modifications, la séquence précédente (figure 3.27) n'est plus possible. La nouvelle séquence est présentée dans l'annexe C.2.

### 3.3.4 Modélisation d'un contrôleur avec recherche de stabilité

Les modifications introduites dans les modèles du système contrôlé et du contrôleur ont permis de supprimer des blocages dans le premier modèle. Cependant, ces modifications peuvent amener à leur tour un comportement irréaliste au niveau du modèle du contrôleur, en ce qui concerne la prise en compte du concept de stabilité.

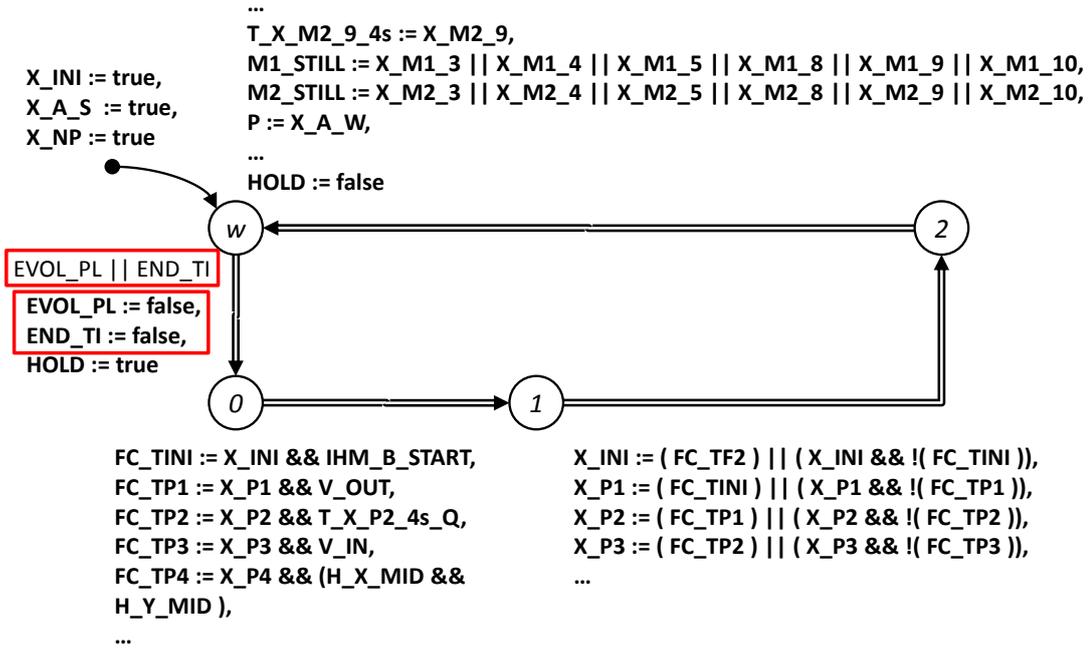


FIGURE 3.30 – Modèle du contrôleur avec prise en compte de EVOL\_PL et END\_TI.

### 3.3.4.1 Mise en évidence sur une séquence

La spécification Grafcet est dans une situation (i.e. un ensemble d'étapes actives) dite stable lorsque aucune condition de tir n'est validée. *A contrario*, la situation est dite instable lorsqu'au moins une condition de tir est validée (voir la norme IEC 60848 (2002) et les travaux de Provost *et al.* (2011) pour plus de détails). Or après le franchissement d'une transition, il se peut que la nouvelle situation soit instable car l'évolution permet par exemple un autre tir de transition. Dans ce cas, notre modèle du contrôleur sera stoppé, laissant le Grafcet dans une situation instable qui ne représente pas le comportement attendu.

La figure 3.31 montre une suite d'évolutions mettant en avant ce problème : le tableau de gauche donne les valeurs des variables utilisées pour la démonstration, le modèle du contrôleur est donné au centre et une représentation partielle (sans explicitation des variables des temporisations) de la spécification Grafcet est rappelée à droite (avec à nouveau les étapes actives et les transitions franchissables grisées).

- Dans la situation initiale (figure 3.31a) de la spécification Grafcet, les étapes P1 et A-S sont actives. La tige du vérin vertical est en train de descendre (V\_G\_OUT est à vrai) mais n'est pas encore arrivée en fin de course (V\_OUT à faux). EVOL\_PL et END\_TI sont à faux.
- La tige du vérin arrive en fin de course et le capteur réagit en passant V\_OUT à vrai, comme le montre la figure 3.31b. De même, la variable EVOL\_PL passe à vrai car une variable d'entrée du contrôleur a été assignée.
- Le modèle du contrôleur réagit à ce changement de variable et exécute un cycle (figure 3.31c) : EVOL\_PL repasse à faux, la condition de tir FC\_TP1 est vérifiée,

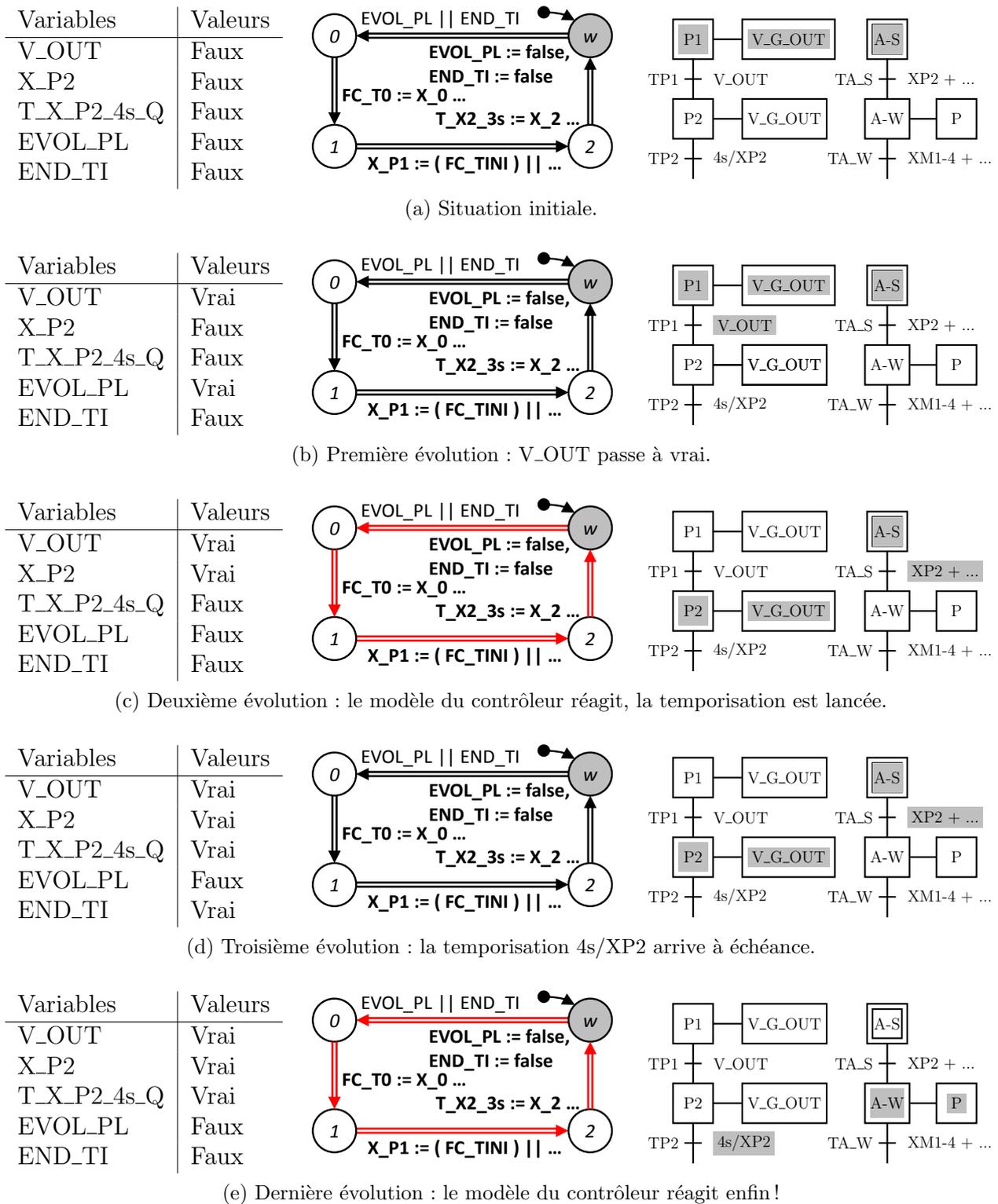


FIGURE 3.31 – Séquence d'évolutions du système bouclé avec la version modifiée du modèle du contrôleur.

- l'étape P1 est désactivée et l'étape P2 activée, et V\_G\_OUT reste à vrai. La temporisation liée à l'étape P2 est aussi lancée (au travers de la variable T\_X\_P2\_4s).
- On remarque alors que la condition de tir FC\_TA\_S est vérifiée. Mais comme EVOL\_PL a été remise à faux, le modèle du contrôleur ne peut plus évoluer. Le Grafcet reste donc dans une situation instable, ce qui n'est pas conforme à sa sémantique. Il faut attendre l'échéance de la temporisation (T\_X\_P2\_4s\_Q passe à vrai) pour que END\_TI passe à vrai et permette un nouveau cycle du modèle du contrôleur (figure 3.31d).
  - Enfin, le modèle du contrôleur peut évoluer (figure 3.31e) grâce à END\_TI, mais le calcul des conditions de tir se basant sur une situation logiquement impossible du Grafcet produit des activations d'étapes non conformes à la volonté du concepteur : l'étape A-W devient active alors que l'étape P2 se désactive (alors que la spécification demandait le contraire!). En pratique, cela revient à ce que l'aspiration (variable P à vrai) devant permettre de saisir la pièce ne se déclenche que lorsque le manipulateur remonte (action associée à l'étape P3, non représentée) et pas au début de la temporisation de l'étape P2 pourtant prévue pour l'établissement du vide!

### 3.3.4.2 Généralisation

La norme Grafcet suppose que les sorties ne sont pas émises lorsque la situation n'est pas stable, mais les limitations imposées au modèle du contrôleur dans la section précédentes ont rendu ce comportement possible.

Il faut donc à nouveau modifier le modèle du contrôleur pour que ce dernier soit contraint d'atteindre une situation stable avant d'arrêter ses évolutions et d'émettre ses sorties. De plus, dans la norme Grafcet, les actions d'assignation qui sont utilisées ici ne doivent pas être émises dans les situations instables, le modèle du contrôleur doit donc être modifié pour prendre en compte cette hypothèse.

Il convient donc :

- D'autoriser des cycles du modèle du contrôleur tant que la situation atteinte n'est pas stable.
- De n'émettre les sorties qu'une fois une situation stable atteinte.

### 3.3.4.3 Ajout d'un critère de stabilité dans le modèle du contrôleur

Le problème précédent est résolu en ajoutant une variable booléenne nommée *stable*. Cette dernière est une variable qui n'est utilisée que par le modèle du contrôleur : elle n'est mise à vrai que lorsque le calcul des conditions de tir a montré qu'aucune transition n'est franchissable :

$$stable = \bigwedge (\neg FC_t) \quad (3.5)$$

Cette variable est ajoutée à la garde du départ de cycle pour permettre un nouveau cycle tant qu'elle est fausse. De plus, le calcul des sorties n'est désormais plus réalisé

que lorsque la variable *stable* est vraie, amenant alors le modèle du contrôleur en localité initiale *w* et libérant les évolutions du modèle de la partie opérative (HOLD remis à faux). Le modèle final du contrôleur avec l'ajout de *stable* et de la transition urgente évitant le calcul des sorties est présenté dans la figure 3.32.

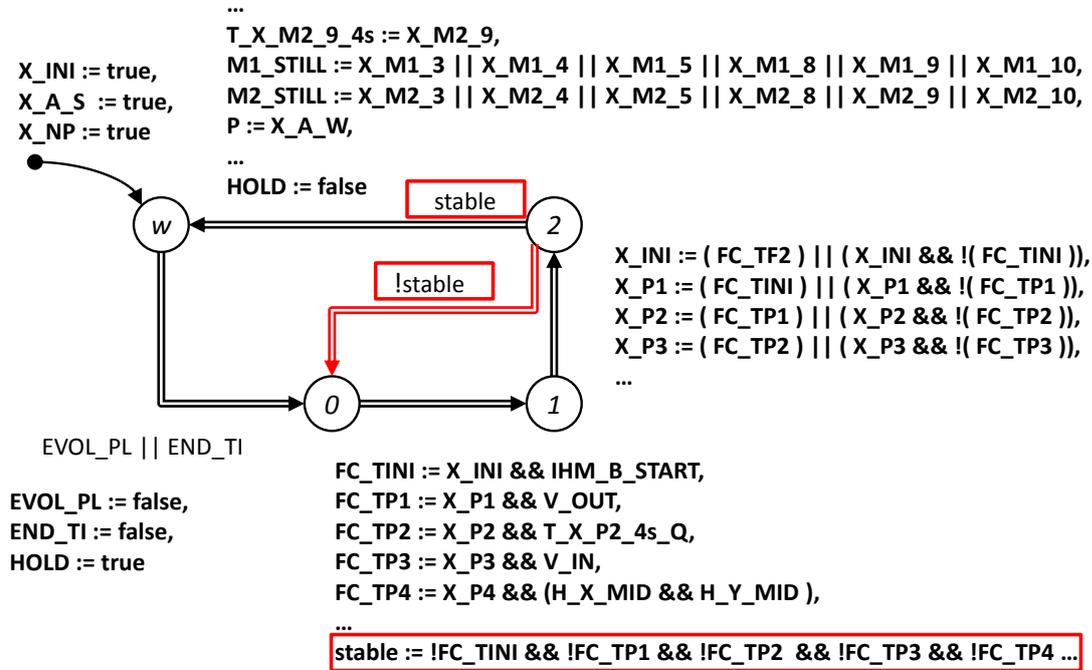


FIGURE 3.32 – Modèle final du contrôleur avec l'ajout de la variable *stable*.

Les modifications permettent au modèle du contrôleur d'atteindre une situation stable, et les sorties ne sont émises que lorsqu'une situation stable est atteinte.

La présence d'une séquence infinie de situations instables est une éventualité qui peut bloquer à nouveau la partie opérative. Cependant une telle spécification n'a que peu de chances de se présenter, qui plus est elle représente un comportement peu sauf. Dans tous les cas, nous considérons nos spécifications Grafcet exemptes de telles séquences, les travaux de Roussel et Lesage (1996) et Lamperiere-Couffin et Lesage (2000) pouvant être utilisés pour garantir cette hypothèse.

Le modèle complet final du contrôleur relatif à l'exemple est donné en annexe, section D.3, figure D.7. De plus, la séquence précédente (figure 3.31) reprise avec ce modèle modifié du contrôleur du manipulateur est décrite dans l'annexe C.3 pour des raisons de place. Le comportement irréaliste n'y apparaît plus.

### 3.3.5 Synthèse des ajouts

Nous avons dans cette section introduit trois jeux de variables :

- **HOLD** pour garantir l'absence de modification des valeurs des variables d'entrée du contrôleur durant son cycle, par blocage des évolutions assignant les variables d'entrée des modèles du système contrôlé.

- **EVOL\_PL** et **END\_TI**, pour éviter les blocages du modèle du système contrôlé, en limitant le départ du cycle du contrôleur sans restreindre sa réactivité.
- **stable** pour assurer des évolutions du modèle du contrôleur avec recherche de stabilité, sans calcul des sorties en cas de situations instables.

Ces variables assurent que le modèle du système bouclé ne comporte pas de blocage et que le modèle du contrôleur est infiniment réactif (il réagit à tous les changements de variables, et ce sans consommation de temps physique) et avec recherche de stabilité. La figure 3.33 reprend une suite d'évolutions après ajout des variables.

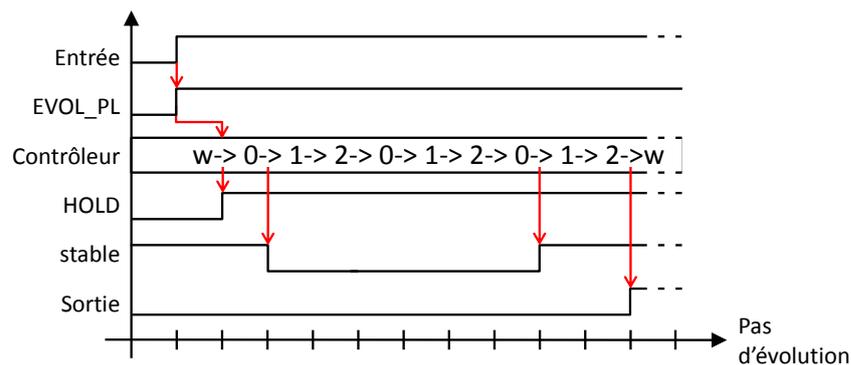


FIGURE 3.33 – Exemple d'évolutions.

## 3.4 Conclusion

Ce chapitre comporte une méthode de conception permettant d'obtenir un modèle détaillé à évolutions réalistes, ce qui remplit notre premier objectif.

Nous avons montré que la construction d'un modèle de partie opérative exige l'utilisation d'un mécanisme d'urgence afin de supprimer des évolutions irréalistes.

Le modèle du contrôleur quant à lui est issu d'une spécification Grafcet traduite en jeux d'équations algébriques, le comportement temporisé étant traité au travers d'automates annexes intégrés dans le modèle du système contrôlé.

Enfin, la réalisation d'un modèle du système bouclé (contrôleur + système contrôlé) à évolutions réalistes impose l'utilisation de plusieurs variables (**HOLD**, **EVOL\_PL**, **END\_TI** et **stable**) afin d'éviter les blocages du modèle du système contrôlé tout en garantissant la réactivité et un comportement conforme à la spécification du modèle du contrôleur.

Notre publication [Perin et Faure \(2009\)](#) traite du problème de concurrence interne à la partie opérative, le blocage du modèle de la partie opérative par le modèle contrôleur est détaillée dans une publication soumise à la conférence *Emerging Technologies and Factory Automation* de 2012 et enfin un article à paraître en 2012 dans la revue *Control Engineering Practice* ([Perin et Faure \(2012\)](#)) traite de la méthode complète de construction d'un modèle de système bouclé à évolution réaliste.



## Chapitre 4

# Vérification formelle de l'équivalence entre modèles détaillé et abstrait

Dans ce chapitre, nous considérons que le modèle détaillé  $M_{P_i}^D$  a été produit en utilisant la méthode d'obtention de modèles de systèmes bouclés détaillés dans le chapitre 3.

Dans le cas où un expert a construit le modèle abstrait  $M_{P_i}^A$ , il est nécessaire de prouver l'équivalence comportementale entre les modèles abstrait et détaillé d'un même poste pour garantir les résultats des analyses utilisant des modèles détaillés et abstraits conjointement.

Ce chapitre débute par l'étude des équivalences proposées dans la littérature, en commençant par un bref rappel sur le formalisme des systèmes de transitions utilisé dans ces publications, afin de choisir l'équivalence qui correspond à nos besoins : une équivalence de traces. Elle impose cependant que les modèles analysés soient déterministes.

Ensuite, une étude de la littérature sur les méthodes de vérifications formelles nous permet de choisir la méthode la plus à même de vérifier l'équivalence précédemment choisie. La technique des automates observateurs, associés à un model-checker, est ensuite décrite en détail.

La technique de vérification de l'équivalence de traces des modèles abstrait et détaillé peut alors être expliquée, en particulier l'automate observateur-séquenceur utilisé, qui constitue le cœur de cette technique. L'aspect temporel des modèles est traité dans un second temps. Le problème de concurrence entre les modèles analysés et ceux générant la trace d'entrée est alors mis en avant et traité par la modification de ces derniers.

Les modèles à comparer étant des modèles de systèmes bouclés, l'hypothèse de déterminisme faite précédemment peut être assez limitante. Pour s'en affranchir nous proposons une définition d'une équivalence de traces avec une tolérance en valeur, temporelle puis des deux. Cette nouvelle équivalence implique des modifications dans la méthode de preuve qui sont précisées.

## Sommaire

---

<b>4.1</b>	<b>Choix d'une équivalence</b>	<b>79</b>
4.1.1	Rappel sur les STE	80
4.1.2	Équivalences de la littérature	81
4.1.3	Équivalence de traces d'ATVD	86
4.1.4	Définition d'une équivalence de traces de deux ATVD	89
<b>4.2</b>	<b>Principe de la vérification</b>	<b>92</b>
4.2.1	Choix d'une technique de vérification formelle	92
4.2.2	Vérification formelle à l'aide de l'outil UPPAAL	92
4.2.3	Méthode des automates observateurs	95
4.2.4	Principe des automates observateurs appliqué à la preuve d'équivalence	96
<b>4.3</b>	<b>Preuve d'une équivalence de traces</b>	<b>98</b>
4.3.1	Création d'un automate observateur-séquenceur non temporisé	98
4.3.2	Équivalence de modèles temporisés	104
4.3.3	Équivalence de modèles temporisés réagissant à des entrées	112
<b>4.4</b>	<b>Définition et vérification d'une équivalence de traces avec tolérance</b>	<b>119</b>
4.4.1	Motivations	119
4.4.2	Équivalence de traces avec tolérance en valeur	119
4.4.3	Définition d'une équivalence avec tolérance temporelle	120
4.4.4	Proposition d'une équivalence de traces avec tolérance temporelle et en valeur	122
4.4.5	Vérification d'une équivalence de traces avec tolérance	123
<b>4.5</b>	<b>Conclusion</b>	<b>125</b>

---

Ce chapitre est consacré à la définition, puis la mise en pratique d'une technique de vérification de l'équivalence, comme montré dans la figure 4.1, permettant de prouver qu'un modèle abstrait peut être utilisé à la place d'un modèle détaillé lors d'une analyse.

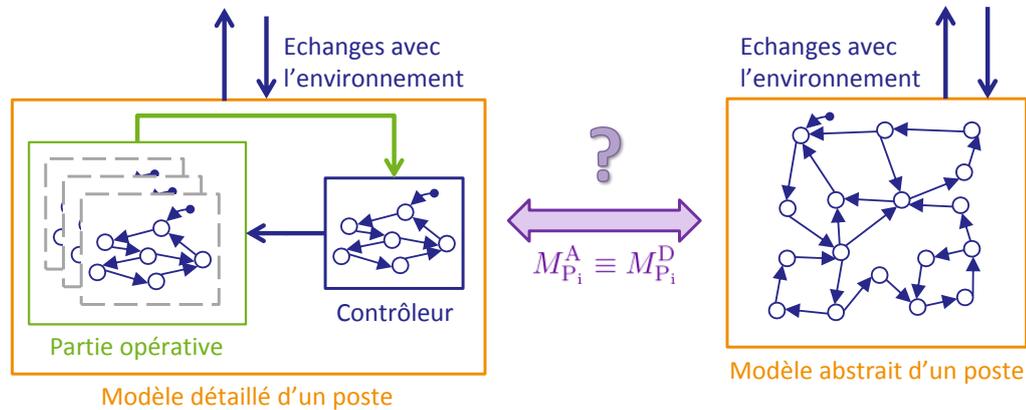


FIGURE 4.1 – principe de l'équivalence

Dans le cadre de l'utilisation des modèles abstrait et détaillé, c'est au niveau des échanges avec l'environnement que les modèles doivent être équivalents. Nous supposons dans ce chapitre que **le modèle détaillé d'un poste est construit en utilisant les résultat du chapitre 3**. Cic qui implique que les échanges avec l'environnement sont, pour les deux modèles à comparer, sous la forme de variables partagées :

- **Les entrées** du modèle du système bouclé sont des variables utilisées par ce modèle mais assignées par l'environnement.
- **Les sorties** du modèle du système bouclés sont des variables assignées par ce modèle et utilisées par l'environnement.

Ainsi, lors du remplacement du modèle détaillé d'un système bouclé par son modèle abstrait correspondant (et vice-versa) on souhaite conserver le même comportement, à savoir les mêmes sorties assignées pour les mêmes configurations d'entrées le tout aux mêmes dates. Ceci afin d'obtenir les mêmes résultats d'analyse, le modèle étant considéré alors comme une boîte noire.

## 4.1 Choix d'une équivalence

Les équivalences sont généralement décrites sur des Systèmes de Transitions à Étiquettes (STE). Ainsi nous débutons par un bref rappel sur ce formalisme. Ensuite, une étude de quelques équivalences de la littérature nous permet de choisir celle qui convient pour notre problématique.

### 4.1.1 Rappel sur les STE

#### 4.1.1.1 Définition d'un STE

La définition d'un système de transition à étiquettes présentée ci-dessous est proposée par [Rabinovich \(1997\)](#).

Un système de transitions à étiquettes (*STE*)  $\langle S, s_0, \Sigma, \longrightarrow \rangle$  est défini par :

- Un ensemble d'états  $S$ .
- Un état initial  $s_0 \in S$ .
- Un alphabet d'actions  $\Sigma$  et une action spéciale, invisible,  $\tau \notin \Sigma$ .
- Une relation d'évolution  $\longrightarrow \subset S \times \{\Sigma \cup \{\tau\}\} \times S$ .

La notation  $s \xrightarrow{\sigma} s'$  correspond à la transition allant de l'état  $s \in S$  à l'état  $s' \in S$  avec l'action  $\sigma \in \{\Sigma \cup \{\tau\}\}$ .

La figure 4.2 montre une représentation graphique d'un système de transitions à étiquettes avec :

- L'ensemble des états  $S = \{S0, S1, S2\}$ ;
- L'état initial  $S0$ ;
- L'alphabet  $\Sigma = \{a, b, c\}$ ;
- Les évolutions associées à l'action invisible  $\tau$  sont représentées par des transitions sans étiquette (transition bouclant sur l'état  $S1$  par exemple).

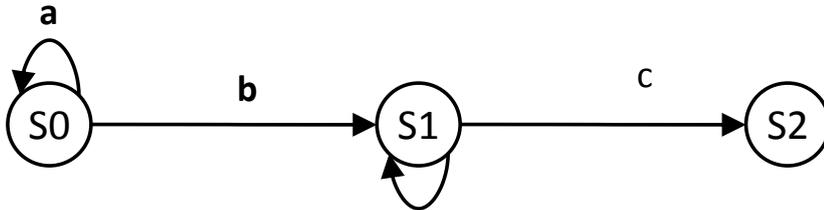


FIGURE 4.2 – Représentation graphique d'un système de transitions à étiquettes.

#### 4.1.1.2 Définition d'une trace d'un STE

Une séquence est définie comme l'alternance d'états et d'actions du système de transitions, comme montré dans l'équation 4.1.

$$Seq = s_0, a_0, s_1, a_1, s_2, a_2, s_3, \dots \quad (4.1)$$

Une trace est issue d'une séquence d'où les états et les actions invisibles  $\tau$  ont été retirés, soit une suite :

$$Tra = a_0, a_1, \dots \quad (4.2)$$

avec  $\forall i \in \mathbb{N}, a_i \in \Sigma$  et où  $i$  est l'indice de trace courant.

En reprenant l'exemple de la figure 4.2, on peut définir la séquence  $Seq = S0, a, S0, a, S0, b, S1, \tau, S1, \tau, S1, c, S2$ . On peut déduire de cette séquence la trace suivante :  $Tra = a, a, b, c$ .

## 4.1.2 Équivalences de la littérature

### 4.1.2.1 Types d'équivalences

De nombreuses équivalences ont été définies dans la littérature, comme le montre la figure 4.3, tirée des travaux de van Glabbeek (2001). La plus exigeante des équivalences est la bi-simulation, qui impose un comportement identique au niveau des évolutions possibles depuis des états liés des deux modèles. *A contrario*, l'équivalence de traces est moins exigeante car elle ne concerne que les traces des deux modèles.

Afin de choisir l'équivalence qui convient le mieux pour comparer deux modèles du même système à des échelles différentes (modèles abstrait et détaillé du même système bouclé), nous allons dans un premier temps définir et étudier ces deux équivalences *limites*.

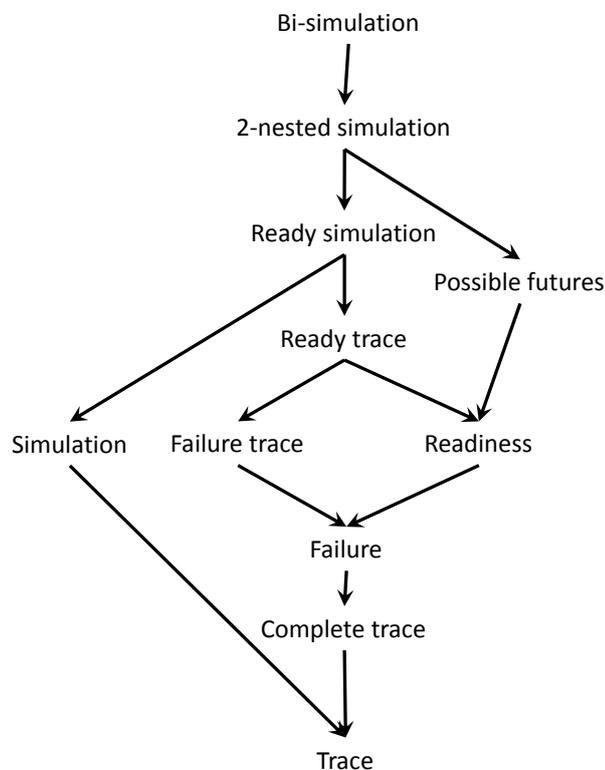


FIGURE 4.3 – Vue synthétique des équivalences existantes, de la plus contraignante à la plus relâchée.

### 4.1.2.2 Bi-simulation des systèmes de transitions à étiquettes

#### 4.1.2.2.1 Définition

La bi-simulation est la relation la plus forte qui peut lier deux modèles. On dit que  $A$  simule  $B$  (noté plus bas  $A \succ B$ ) s'il existe une relation  $R$  liant les états de  $A$  à ceux de  $B$  et ce quelle que soit l'évolution choisie.

Soit deux systèmes de transitions à étiquettes  $A$  et  $B$  avec le même alphabet  $\Sigma$ . On a :  $A = \langle S^A, s_0^A, \Sigma, \longrightarrow^A \rangle$  et  $B = \langle S^B, s_0^B, \Sigma, \longrightarrow^B \rangle$ . On définit une relation  $R$  telle que  $s^A R s^B$  liant les états de  $B$  à ceux de  $A$ .

$$\begin{aligned}
 A \succ B &\Rightarrow \exists R : s_A \mapsto s_B \text{ tel que :} \\
 &s_0^A R s_0^B \text{ et} \\
 &\forall s^A, s'^A \in S^A, \forall s^B, s'^B \in S^B \text{ tel que } s^A R s^B, \text{ alors :} \\
 &\forall \sigma \in \Sigma \text{ tel que } s^A \xrightarrow{\sigma}^A s'^A, \exists s^B \xrightarrow{\sigma}^B s'^B \text{ avec } s'^A R s'^B
 \end{aligned} \tag{4.3}$$

Cette définition n'est pas symétrique puisque seules les évolutions du modèle  $A$  sont parcourues. Si la réciproque est vraie ( $B$  simule  $A$ ), on peut alors parler de bi-simulation définie par :

$$\begin{aligned}
 A \prec\succ B &\Rightarrow \exists R : s_A \mapsto s_B \text{ tel que :} \\
 &s_0^A R s_0^B \text{ et} \\
 &\forall s^A, s'^A \in S^A, \forall s^B, s'^B \in S^B \text{ tel que } s^A R s^B, \text{ alors :} \\
 &\forall \sigma \in \Sigma \text{ tel que } s^A \xrightarrow{\sigma}^A s'^A, \exists s^B \xrightarrow{\sigma}^B s'^B \text{ avec } s'^A R s'^B \text{ et} \\
 &\forall \sigma \in \Sigma \text{ tel que } s^B \xrightarrow{\sigma}^B s'^B, \exists s^A \xrightarrow{\sigma}^A s'^A \text{ avec } s'^A R s'^B
 \end{aligned} \tag{4.4}$$

#### 4.1.2.2 Exemple de simulation partielle

La figure 4.4 est un exemple de deux STE tel que  $A \succ B$  (mais pas  $B \succ A$ ). On a les modèles suivants :

**Système de transitions A :**

- L'ensembles des états  $S^A = \{S0, S1, S2\}$  ;
- L'état initial  $S0$  ;
- L'alphabet  $\Sigma^A = \{a, b, c\}$ .

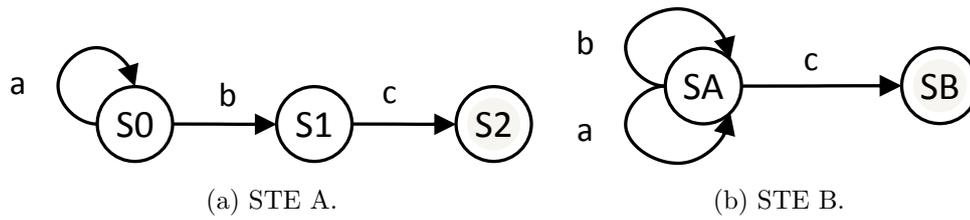
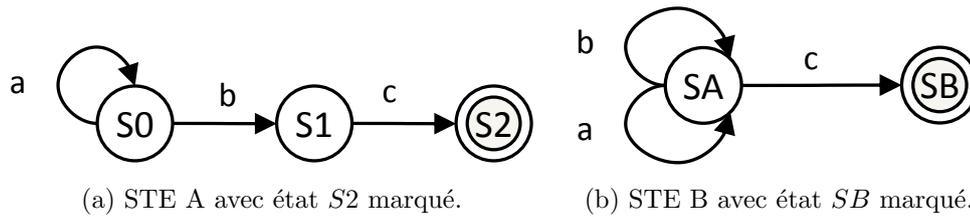
**Système de transitions B :**

- L'ensemble des états  $S^B = \{SA, SB\}$  ;
- L'état initial  $SA$  ;
- L'alphabet  $\Sigma^B = \Sigma^A = \{a, b, c\}$ .

En utilisant la relation  $R$  liant  $\{S0, S1\}$  à  $SA$  et  $S2$  à  $SB$ , on remarque que toutes les évolutions possibles du modèle A le sont aussi pour le modèle B, B simule donc A. Cependant, il n'existe pas de relation liant les états de B à ceux de A de manière identique,  $SA$  posant problème à cause de l'évolution  $c$ .

yo

La figure 4.5 montre un parallèle avec la théorie des langages : en marquant (cercles doublés) les états  $S2$  et  $SB$ , le langage accepté de A  $L_m^A = a^*bc$  et celui de B  $L_m^B = (a+b)^*c$  vérifient  $L_m^A \subset L_m^B$  mais  $L_m^B \not\subset L_m^A$ , le mot  $c \in L_m^B$  étant un exemple de la non-inclusion.

FIGURE 4.4 – Exemple où  $STE A \succ STE B$ .FIGURE 4.5 – Exemple où  $STE A \succ STE B$ , avec certains états marqués.

#### 4.1.2.2.3 Exemple de bi-simulation

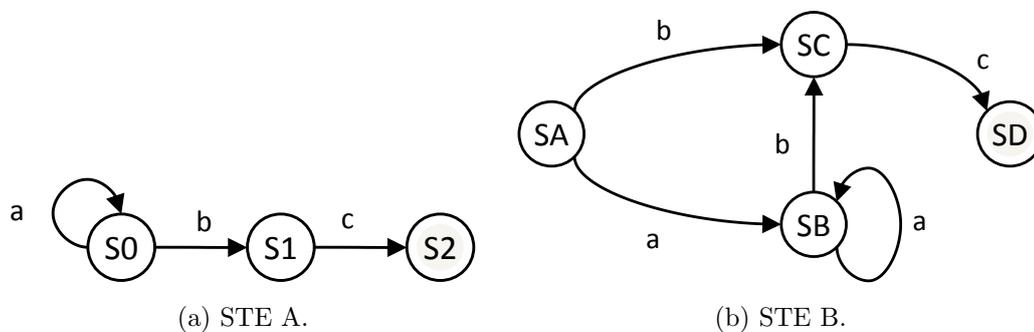
Le second exemple décrit par la figure 4.6 montre un cas de bi-simulation. On a les modèles suivants :

**Système de transitions A** (inchangé) :

- L'ensemble des états  $S^A = \{S0, S1, S2\}$  ;
- L'état initial  $S0$  ;
- L'alphabet  $\Sigma^A = \{a, b, c\}$ .

**Système de transitions B** :

- L'ensemble des états  $S^B = \{SA, SB, SC, SD\}$  ;
- L'état initial  $SA$  ;
- L'alphabet  $\Sigma^B = \Sigma^A = \{a, b, c\}$ .

FIGURE 4.6 – Exemple de bi-simulation où  $STE A \prec\sim STE B$ .

Ici, la relation  $R$  liant  $S0$  à  $\{SA, SB\}$ ,  $S1$  à  $SC$  et  $S2$  à  $SD$  est une relation de bi-simulation.

De nouveau, en marquant les états  $S2$  et  $SD$  et en travaillant avec la théorie des langages, figure 4.7, on obtient le langage de A  $L_m^A = a^*bc$  et celui de B  $L_m^B = aa^*bc + bc$  qui vérifient  $L_m^A \equiv L_m^B$ .

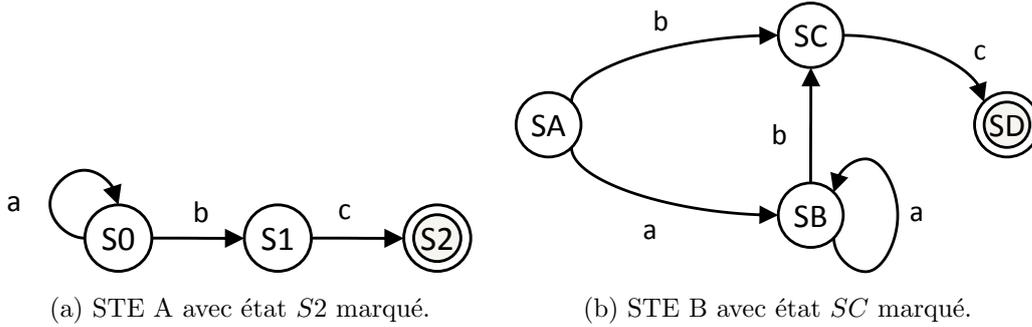


FIGURE 4.7 – Exemple de bi-simulation où  $STE A \prec\!\succ\! STE B$ , avec certains états marqués.

#### 4.1.2.2.4 Utilité pour notre usage

Une relation de bi-simulation lie les états des deux modèles comparés, ce qui revient à lier des localités dans le cas des ATVD. L'idée de l'équivalence souhaitée est cependant de fonctionner avec des modèles de type boîte noire dont l'état interne *n'est pas connu*. Ainsi cette équivalence est bien trop contraignante pour notre utilisation et n'est donc pas retenue.

#### 4.1.2.3 Équivalence de traces de systèmes de transitions à étiquettes

##### 4.1.2.3.1 Définition

Cette équivalence ne concerne pas l'état interne du système de transitions. Dans ce cas, seules les traces des deux STE sont comparées.

Soit deux STE  $A$  et  $B$  avec le même alphabet  $\Sigma$ . On a  $A = \langle S^A, s_0^A, \Sigma, \rightarrow^A \rangle$  et  $B = \langle S^B, s_0^B, \Sigma, \rightarrow^B \rangle$ . On définit alors l'ensemble des traces des deux STE  $Traces^A$  et  $Traces^B$ .

Les deux STE sont équivalents en trace si  $Traces^A = Traces^B$ , c'est-à-dire :

$$\begin{aligned} \forall Tra^A \in Traces^A, \exists Tra^B \in Traces^B \text{ tel que } \forall i \in \mathbb{N}, \forall a_i^A \in Tra^A \text{ et } a_i^B \in Tra^B, a_i^A = a_i^B \text{ et} \\ \forall Tra^B \in Traces^B, \exists Tra^A \in Traces^A \text{ tel que } \forall i \in \mathbb{N}, \forall a_i^B \in Tra^B \text{ et } a_i^A \in Tra^A, a_i^B = a_i^A \end{aligned} \quad (4.5)$$

## 4.1.2.3.2 Exemple

La figure 4.8 montre un cas d'équivalence de traces pour deux STE A et B :

**Système de transitions A :**

- Les états  $S^A = \{S0, S1, S2\}$  ;
- L'état initial  $S0$  ;
- L'alphabet  $\Sigma^A = \{a, b, c\}$  ;
- Les évolutions associées à l'action invisible  $\tau$  sont représentées par des arcs sans étiquette.

**Système de transitions B :**

- Les états  $S^B = \{SA, SB, SC, SD, SE\}$  ;
- L'état initial  $SA$  ;
- L'alphabet  $\Sigma^B = \Sigma^A = \{a, b, c\}$  ;
- Les évolutions associées à l'action invisible  $\tau$  sont représentées par des arcs sans étiquette.

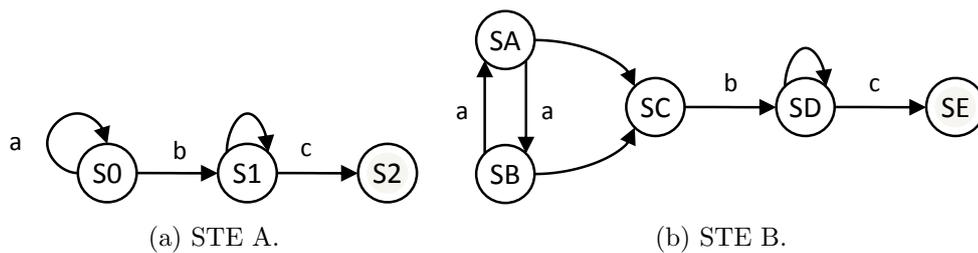


FIGURE 4.8 – Exemple d'équivalence de traces entre deux systèmes de transitions.

Là où la relation de simulation s'intéresse aux évolutions possibles et aux états des modèles, l'équivalence de traces n'utilise que les traces, à savoir les étiquettes. Sur l'exemple ci dessus, les traces issues des deux modèles sont identiques mais les deux modèles n'ont pas de relation de simulation. Ceci est dû aux évolutions non observables en rapport avec l'action  $\tau$  : elles ne sont pas prises en compte par l'équivalence de traces mais sont considérées par la notion de simulation. Les transitions  $SA \rightarrow SC$  et  $SB \rightarrow SC$  sont donc invisibles d'un point de vue de l'équivalence de traces, mais discriminantes pour la notion de simulation, car les évolutions possibles en  $SC$  sont différentes des celles possibles en  $SA$  ou  $SB$ .

## 4.1.2.3.3 Utilité pour notre usage

L'équivalence de traces se focalise sur les traces générées par les systèmes de transitions comparés, sans prendre compte l'état interne de ce système. L'idée de départ de modèles de type boîtes noires et alors totalement compatible avec ce type d'équivalence, qui permet de garantir que deux modèles se comportent comme deux *générateurs* identiques.

Cette équivalence convient donc parfaitement à nos besoins. Cependant les ATVD sont des modèles générateurs *et* récepteurs. Il convient donc d'adapter la définition de la trace utilisée et celle de l'équivalence elle-même aux ATVD.

### 4.1.3 Équivalence de traces d'ATVD

#### 4.1.3.1 Séquences et traces d'un ATVD

##### 4.1.3.1.1 Séquences d'un ATVD

La sémantique complète des ATVD, issue des travaux de [Janowska et Janowski \(2006\)](#), est définie dans la section 2.2.

Pour rappel, les deux sémantiques possibles sont :

- Une évolution des horloges, en accord avec les invariants des localités actives et au regard des transitions urgentes franchissables.
- Un changement de la localité active, avec le tir d'une transition au regard de sa garde, de sa contrainte temporelle et de l'invariant de la localité de destination.

La définition d'une séquence est reprise du papier présentant le formalisme : pour un automate  $(L, l_0, V, C, E, I)$ , une séquence est définie comme la suite :

$$Seq = s_0 \xrightarrow{\delta_0} s_1 \xrightarrow{\delta_1} s_2 \xrightarrow{\delta_2} s_3 \xrightarrow{\delta_3} s_4 \dots \quad (4.6)$$

avec :

- $s_i = (l, \tilde{V}_i, \tilde{C}_i) \in S$  un état du système de transitions ;
- $\delta_i$  une évolution du système de transitions associé à l'ATVD :
  - soit  $e_i \in E$ , la transition de l'automate qui est franchie,
  - soit  $d_i \in \mathbb{N}^*$ , la durée passée telle que  $\tilde{C}_{(i+1)} = \tilde{C}_i + d_i$ .

La figure 4.9 représente un ATVD qui est utilisé comme exemple pour les définitions des séquences et de traces. Il assigne :

- Les variables booléennes a, b et c,
- Les variables entières x, y et z,
- L'horloge Temp.

Une séquence de cette automate est la suivante :

$$Seq = \begin{array}{cccccccccc} \frac{L0}{F} & \frac{L1}{F} & \frac{L1}{F} & \frac{L2}{T} & \frac{L2}{T} & \frac{L3}{T} & \frac{L3}{T} & \frac{L4}{T} & \frac{L4}{T} \\ F & T & T & T & T & F & F & F & F \\ F \xrightarrow{L0 \rightarrow L1} T & T \xrightarrow{5} T & T \xrightarrow{L1 \rightarrow L2} T & T \xrightarrow{8} T & T \xrightarrow{L2 \rightarrow L3} T & T \xrightarrow{L3 \rightarrow L3} F & F \xrightarrow{L3 \rightarrow L4} F & F \xrightarrow{13} F & \dots \\ 0 & 1 & 1 & 2 & 2 & 4 & 4 & 0 & 0 \\ 0 & 4 & 4 & -5 & -5 & 8 & 8 & 24 & 24 \\ 42 & 42 & 42 & 3 & 3 & 3 & 8 & 8 & 8 \\ \frac{0}{0} & \frac{0}{0} & \frac{5}{5} & \frac{5}{5} & \frac{13}{13} & \frac{13}{13} & \frac{0}{0} & \frac{0}{0} & \frac{13}{13} \end{array}$$

Avec le vecteur d'état  $Loc|a b c x y z|Temp$  où les variables booléenne on comme valeurs F (False, fausse) ou T (True, vraie).

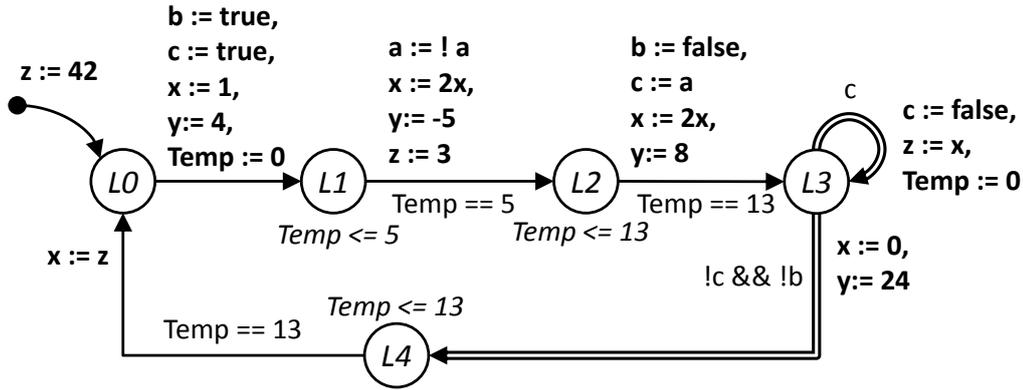


FIGURE 4.9 – ATVD servant d'exemple pour les définitions de séquences et de traces.

#### 4.1.3.1.2 Traces d'un ATVD

En se référant à la définition de la séquence fournie dans l'équation 4.6, on obtient une trace d'un ATVD de la forme :

$$Tra = \delta_0, \delta_1, \delta_2, \delta_3, \dots \quad (4.7)$$

Pour notre exemple d'ATVD de la figure 4.9, on obtient donc la trace :

$$Tra = L0 \rightarrow L1, 5, L1 \rightarrow L2, 8, L2 \rightarrow L3, L3 \rightarrow L3, L3 \rightarrow L4, 13, \dots$$

Cette définition ne permet cependant pas de connaître directement les valeurs des variables et horloges qui sont pourtant essentielles pour la vérification de l'équivalence.

Nous proposons donc une version de la trace qui supprime toujours les localités actives mais explicite les valeurs des variables et des horloges, soit une trace étendue :

$$Tra' = (\tilde{V}_0, \tilde{C}_0) \xrightarrow{\delta_0} (\tilde{V}_1, \tilde{C}_1) \xrightarrow{\delta_1} (\tilde{V}_2, \tilde{C}_2) \xrightarrow{\delta_2} (\tilde{V}_3, \tilde{C}_3) \xrightarrow{\delta_3} (\tilde{V}_4, \tilde{C}_4) \dots \quad (4.8)$$

où  $\delta_i$  représente une évolution du système de transitions utilisant la même notation que précédemment (soit  $e_i$ , soit  $d_i$ ) avec  $i \in \mathbb{N}$  l'indice de la trace courant.

Pour notre exemple d'ATVD, on obtient donc une trace étendue :

	$F$	$F$	$F$	$T$	$T$	$T$	$T$	$T$	$T$
	$F$	$T$	$T$	$T$	$T$	$F$	$F$	$F$	$F$
	$F$	$T$	$T$	$T$	$T$	$T$	$F$	$F$	$F$
$Tra' =$	$0$	$1$	$1$	$2$	$2$	$4$	$4$	$0$	$0 \dots$
	$L0 \rightarrow L1$	$5 \rightarrow$	$L1 \rightarrow L2$	$8 \rightarrow$	$L2 \rightarrow L3$	$L3 \rightarrow L3$	$L3 \rightarrow L4$	$13 \rightarrow$	
	$0$	$4$	$4$	$-5$	$-5$	$8$	$8$	$24$	$24$
	$42$	$42$	$42$	$3$	$3$	$3$	$8$	$8$	$8$
	$0$	$0$	$5$	$5$	$13$	$13$	$0$	$0$	$13$

Avec le vecteur  $abcxyz|Temp$  où les variables booléennes ont comme valeurs F (False, fausse) ou T (True, vraie).

### 4.1.3.2 Définition d'une trace adaptée à la vérification de l'équivalence entre ATVD

#### 4.1.3.2.1 Traces relatives à un sous-ensemble de variables

La définition 4.8 porte sur la trace d'un seul ATVD. Dans un réseau d'ATVD, les variables (ensemble  $V$ ) comme les horloges (ensemble  $C$ ) sont communes (décrit section 2.2.4) à l'ensemble des automates. Cependant la comparaison des traces produites par deux modèles ne peut se concevoir que sur l'ensemble des variables du réseau assignées par les deux modèles (leurs variables de sorties communes). Ainsi, nous définissons une trace relative à un sous-ensemble de variables  $W \subseteq V$  de la forme :

$$Tra'(W) = (\tilde{W}_0, \tilde{C}_0) \xrightarrow{\alpha_0} (\tilde{W}_1, \tilde{C}_1) \xrightarrow{\alpha_1} (\tilde{W}_2, \tilde{C}_2) \xrightarrow{\alpha_2} (\tilde{W}_3, \tilde{C}_3) \xrightarrow{\alpha_3} (\tilde{W}_4, \tilde{C}_4) \dots \quad (4.9)$$

avec  $i \in \mathbb{N}$  l'indice de trace courant et  $\alpha_i$  une évolution de la trace :

- soit  $\epsilon_i$ , une action sur les variables telle que  $\tilde{W}_i \xrightarrow{\epsilon_i} \tilde{W}_{(i+1)}$  et  $\tilde{C}_i \xrightarrow{\epsilon_i} \tilde{C}_{(i+1)}$  au vu des horloges à remettre à zéro  $r_i$  ;
- soit  $d_i \in \mathbb{N}^*$ , un passage du temps tel que  $\tilde{C}_{(i+1)} = \tilde{C}_i + d_i$  et  $\tilde{W}_{(i+1)} = \tilde{W}_i$  ;
- soit  $\tau$ , une évolution invisible telle que les variables de  $W$  restent inchangées :  $\tilde{W}_{(i+1)} = \tilde{W}_i$  mais les horloges subissent des remises à zéro :  $\tilde{C}_i \xrightarrow{\tau} \tilde{C}_{(i+1)}$  au vu des horloges à remettre à zéro  $r_i$  de la transition franchie.

En reprenant notre exemple, et en posant  $W = \{a, b, x, y\}$  on obtient la trace relative à ce sous-ensemble :

$$Tra'(W) = \begin{array}{cccccccccc} & F & F & F & T & T & T & T & T & T \\ & F & T & T & T & T & F & F & F & F \\ 0 & \xrightarrow{L0 \rightarrow L1} & 1 & \xrightarrow{5} & 1 & \xrightarrow{L1 \rightarrow L2} & 2 & \xrightarrow{8} & 2 & \xrightarrow{L2 \rightarrow L3} & 4 & \xrightarrow{\tau} & 4 & \xrightarrow{L3 \rightarrow L4} & 0 & \xrightarrow{13} & 0 & \dots \\ 0 & & 4 & & 4 & & -5 & & -5 & & 8 & & 8 & & 24 & & 24 \\ 0 & & 0 & & 5 & & 5 & & 13 & & 13 & & 0 & & 0 & & 13 \end{array}$$

Avec le vecteur  $abxy|Temp$  où les variables booléennes ont comme valeurs F (False, fausse) ou T (True, vraie). L'évolution  $L3 \rightarrow L3$  devient une évolution invisible  $\tau$  puisqu'elle ne modifie aucune des valeurs des variables de  $W$ . Elle est cependant conservée à cause de la mise à zéro de l'horloge Temp.

#### 4.1.3.2.2 Traces temporisées d'un ATVD

Dans une trace, les évolutions invisibles (n'affectant pas les variables de sorties communes de  $W$ ) sont normalement supprimées. Cependant, à cause des remises à zéro potentielles des horloges, ce n'est pas possible dans notre cas. Pour simplifier encore la trace, nous adoptons la formalisation d'une trace temporisée proche de celle proposée dans Alur (1994) qui utilise une horloge globale initialisée à zéro mais jamais remise à zéro comme marqueur temporel :

$$Tra_t(W) = (\tilde{W}_0, 0) \xrightarrow{\alpha_0} (\tilde{W}_1, t_1) \xrightarrow{\alpha_1} (\tilde{W}_2, t_2) \xrightarrow{\alpha_2} (\tilde{W}_3, t_3) \xrightarrow{\alpha_3} (\tilde{W}_4, t_4) \dots \quad (4.10)$$

avec :

- $(\tilde{W}_i, t_i)$  une valuation des variables au temps  $t$  absolu, mis à zéro au départ de la trace.
- pour tout indice de trace  $i \in \mathbb{N}$ ,  $\alpha_i$  une évolution de la trace :
  - soit  $\epsilon_i$ , une action sur les variables telle que  $\tilde{W}_i \xrightarrow{\epsilon_i} \tilde{W}_{(i+1)}$  et  $t_{(i+1)} = t_i$ .
  - soit  $d_i \in \mathbb{N}$ , un passage du temps tel que  $t_{(i+1)} = t_i + d_i$  et  $\tilde{W}_{(i+1)} = \tilde{W}_i$ .

Par la suite, la notion de trace associée aux ATVD fera toujours référence à une trace temporisée relative à un sous-ensemble de variables, et sera notée  $Tra(W)$  où  $W$  est le sous-ensemble de variables considéré.

La trace temporisée, relative aux variables  $a, b, x$  et  $y$ , de notre exemple d'ATVD de la figure 4.9 est donc :

$$Tra(W) = \begin{array}{cccccccc} & F & F & F & T & T & T & T & T \\ & F & T & T & T & T & F & F & F \\ 0 & \xrightarrow{L0 \rightarrow L1} & 1 & \xrightarrow{5} & 1 & \xrightarrow{L1 \rightarrow L2} & 2 & \xrightarrow{8} & 2 & \xrightarrow{L2 \rightarrow L3} & 4 & \xrightarrow{L3 \rightarrow L4} & 0 & \xrightarrow{13} & 0 & \dots \\ \frac{0}{0} & & \frac{4}{0} & & \frac{4}{5} & & \frac{-5}{5} & & \frac{-5}{13} & & \frac{8}{13} & & \frac{24}{13} & & \frac{24}{26} \end{array}$$

Avec le vecteur  $abxy|t$  où les variables booléenne on comme valeurs F (False, fausse) ou T (True, vraie). L'évolution invisible  $\tau$ , ex  $L3 \rightarrow L3$ , a été supprimée. L'horloge Temp n'apparaît plus car l'horloge globale  $t$  remplace toutes les horloges.

Cette trace peut aussi être représentée simplement au travers d'un tableau, comme le montre le tableau 4.1 où est représentée la trace temporisée relative aux variables de sorties de  $W$  de l'ATVD de l'exemple de la figure 4.9. Pour des raisons de place, lorsqu'une variable booléenne est vraie on la marque égale à 1, de manière similaire on la note à 0 lorsqu'elle est à faus.

$a$	0	0	0	1	1	1	1	1
$b$	0	1	1	1	1	0	0	0
$x$	0	1	1	2	2	4	0	0
$y$	0	4	4	-5	-5	8	24	24
$t$	0	0	5	5	13	13	13	26

TABLE 4.1 – Exemple de représentation d'une trace d'un ATVD.

#### 4.1.4 Définition d'une équivalence de traces de deux ATVD

Il convient maintenant d'adapter la définition d'une équivalence de traces d'un système de transitions à étiquettes au cadre des automates temporisés à variables discrètes.

#### 4.1.4.1 Notations

Nous commençons par définir les notations suivantes :

- $V_e^A$  l'ensemble des variables d'entrée de l'ATVD A,
- $V_e^B$  l'ensemble des variables d'entrée de l'ATVD B,
- $V_e = V_e^A \cap V_e^B$  l'ensemble des variables d'entrée communes aux deux ATVD A et B,
- $V_s^A$  l'ensemble des variables de sortie de l'ATVD A,
- $V_s^B$  l'ensemble des variables de sortie de l'ATVD B,
- $V_s = V_s^A \cap V_s^B$  l'ensemble des variables de sortie communes aux deux ATVD A et B.

où les variables d'entrée d'un modèle sont des variables non assignées par ce modèle (donc assignées par l'environnement) mais utilisées dans au moins une garde d'une transition de ce modèle, et les variables de sorties d'un modèle sont quand à elle assignées par ce modèle et utilisées par l'environnement.

On peut donc définir aussi :

- $Tra(V_e)$ , une trace relative à l'ensemble des variables d'entrée communes des modèles A et B, appelée trace d'entrée, et  $Traces(V_e)$  l'ensemble de ces traces.
- $Tra^A(V_s)$ , une trace relative à l'ensemble des variables de sorties communes du modèle A, appelée trace de sortie de A, et  $Traces^A(V_s)$  l'ensemble de ces traces.
- $Tra^B(V_s)$ , la trace relative à l'ensemble des variables de sorties communes du modèle B, appelée trace de sortie de B, et  $Traces^B(V_s)$  l'ensemble de ces traces.

#### 4.1.4.2 Principe

Dans le cadre des modèles de systèmes bouclés à comparer, il convient de comparer les traces de sortie produites par ces deux modèles. Comme ces modèles sont réactifs, réagissant à des changements des variables d'entrée, il convient de comparer leurs traces de sortie pour des évolutions des entrées identiques.

La comparaison des traces de sortie n'a de sens que pour un ensemble de variables de sorties communes  $V_s$  et pour une trace d'entrée ne considérant que les variables d'entrée communes  $V_e$  aux deux modèles.

Le principe de l'équivalence de traces appliqué à deux ATVD A et B est montré dans la figure 4.10. Une trace d'entrée  $Tra(V_e)$ , reflétant les évolutions des valeurs de variables d'entrée communes aux deux modèles à comparer, est donnée aux deux modèles A et B. Les traces de sortie qu'ils génèrent ( $Tra^A(V_s)$  et  $Tra^B(V_s)$ ) sont comparées. Si les deux modèles produisent la même trace de sortie pour toutes les traces d'entrée possibles, alors ils sont dits équivalents en trace.

#### 4.1.4.3 Définition

Soit deux ATVD A et B, ces deux ATVD sont dits équivalents de traces si, pour toute trace d'entrée  $Tra(V_e) \in Traces(V_e)$  donnée aux deux automates, les traces de sortie  $Tra(V_s)$  des deux automates sont identiques.

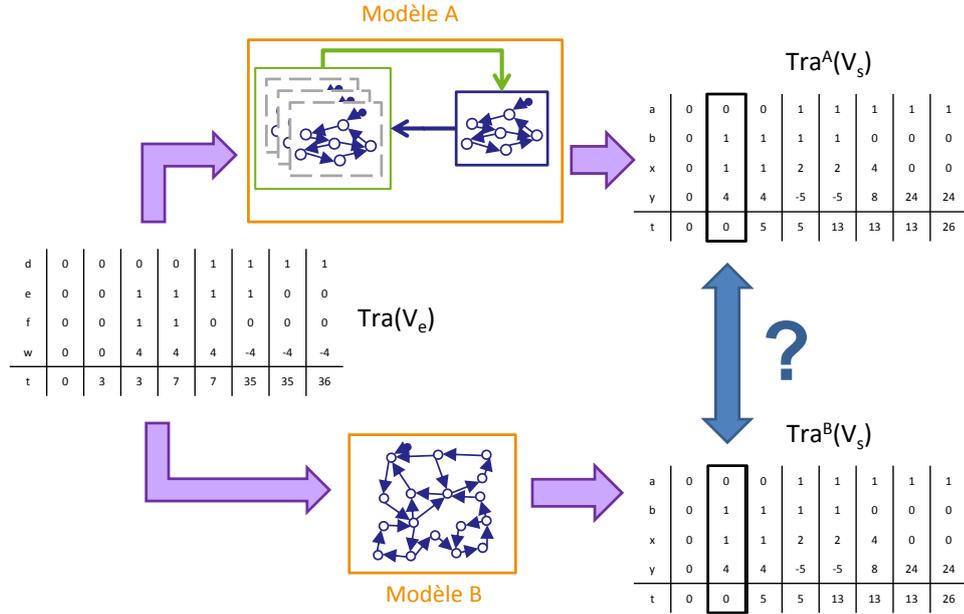


FIGURE 4.10 – Principe de la vérification de l'équivalence de traces entre deux ATVD A et B, avec  $V_e = \{d, e, f, w\}$  et  $V_s = \{a, b, x, y\}$ .

$$\begin{aligned}
 A \equiv B \Rightarrow \\
 \forall \text{Tra}(V_e), \forall i \in \mathbb{N}, (\tilde{V}_i^A, t_i^A) \in \text{Tra}^A(V_s) \text{ et } (\tilde{V}_i^B, t_i^B) \in \text{Tra}^B(V_s), \\
 (\tilde{V}_i^A, t_i^A) = (\tilde{V}_i^B, t_i^B)
 \end{aligned} \tag{4.11}$$

On peut donc affirmer que deux ATVD équivalents en trace ont un comportement identique du point de vue de leurs variables d'entrée/sortie : ils produisent la même trace de sortie (sur l'ensemble des variables communes considérées) pour toutes les traces d'entrée (scénarios) possibles.

**Remarque** : Nous pensons qu'il est important pour une relation d'équivalence de vérifier un postulat d'auto équivalence, ainsi un modèle doit être équivalent à lui-même. De ce fait, pour qu'un ATVD soit équivalent en trace à lui-même, il est nécessaire qu'une trace d'entrée ne puisse produire qu'une seule et unique trace de sortie.

**Hypothèse** : L'utilisation d'un ATVD dans le cadre d'une preuve d'équivalence implique donc qu'il est déterministe au niveau des traces de sortie produites.

Cette hypothèse forte est discutée plus tard dans ce chapitre, principalement à cause de l'existence de modèles de systèmes bouclés non déterministes (à cause de modèles de partie opérative non déterministes par exemple).

## 4.2 Principe de la vérification

### 4.2.1 Choix d'une technique de vérification formelle

Les deux techniques majeures d'analyse formelle sont le model-checking et le theorem proving. [Berezin \(2002\)](#) compare ces deux techniques et fait un récapitulatif des atouts et inconvénients de chacune :

#### Theorem-proving :

- L'atout principal est que la méthode est entièrement symbolique.
- Les inconvénients majeurs sont la difficulté de construction des modèles (écrits en langage symbolique) ainsi que l'utilisation de la technique qui, même supportée par un outil logiciel, nécessite l'intervention d'un utilisateur formé et chevronné.

#### Model-checking :

- L'atout principal est que la méthode est très bien supportée par des outils logiciels (NuSMV, SPIN, UPPAAL) et permet de traiter des cas de grande dimension.
- Les inconvénients majeurs sont l'écriture des propriétés, la construction de modèles sans comportements indésirables et l'analyse des résultats de preuve.

Comme le theorem-proving nécessite un utilisateur formé, son utilisation dans le milieu industriel par des ingénieurs apparait comme difficile. De plus une des difficultés classiques de la technique de model-checking à déjà été résolue dans le chapitre 3 (construction de modèles sans comportement indésirable).

Le model-checking apparait donc comme la technique à utiliser pour prouver l'équivalence entre deux modèles d'un même poste. Il reste cependant à définir les propriétés à fournir au model-checker, ce qui sera traité dans ce chapitre, pour le cas de la vérification de l'équivalence de traces.

### 4.2.2 Vérification formelle à l'aide de l'outil UPPAAL

#### 4.2.2.1 Principe du model-checking

Le principe de base du model-checking consiste à explorer de manière exhaustive l'ensemble des états accessibles d'un modèle à états fini afin de vérifier une propriété.

Ce principe est expliqué dans cette section et de plus amples informations sont disponibles dans le livre de [Bérard \*et al.\* \(2001\)](#).

Cette technique de vérification a été utilisée dans plusieurs domaines comme la vérification de contrôleur ([Lindahl \*et al.\*, 1998](#); [Rossi et Schnoebelen, 2000](#); [Gourcuff \*et al.\*, 2008](#)), l'évaluation de performances temporelles ([Ruel, 2009](#)) et l'analyse prévisionnelle des fautes ([Perin et Faure, 2008](#)).

La figure 4.11 montre de manière schématique les principales étapes d'utilisation d'un model-checker :

- Un modèle décrit dans un langage formel est donné au model-checker.
- Une propriété formelle décrite en logique temporelle est donnée au model-checker.

- Le model-checker, en travaillant sur l'espace d'états du modèle, va définir et prouver que la propriété est vraie ou fausse et, le cas échéant, un exemple ou un contre-exemple peut être fourni.

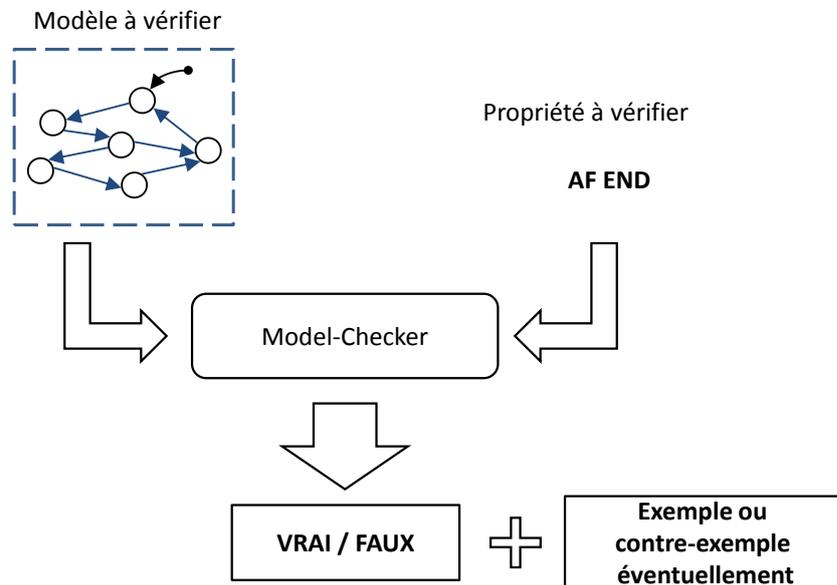


FIGURE 4.11 – Principe de base du model-checking.

#### 4.2.2.2 Logiques temporelles

Les logiques temporelles permettent de décrire une propriété relative à une proposition logique. Cette dernière est une expression booléenne utilisant des localités actives des modèles et des conditions sur des variables ou sur des horloges (expressions arithmétiques). Cette proposition logique est donc vérifiée (*i.e.* évaluée à vraie) ou non selon l'état du modèle analysé.

Il existe deux principales logiques temporelles de description de la propriété :

**CTL** pour Computation Tree Logic permet de définir une propriété dans le cadre de l'exploration d'un futur décrit sous la forme d'un arbre, comme montré dans la figure 4.12. Une propriété écrite dans ce langage utilise des couples de quantificateurs (un de chemin et un d'état) et une proposition logique à vérifier ;

**LTL** pour Linear Temporal Logic permet de définir une propriété le long d'un chemin, comme montré dans la figure 4.13. Une propriété écrite dans ce langage utilise uniquement des quantificateurs d'état et une proposition logique à vérifier.

On note dans la figure 4.12 l'apparition des opérateurs de chemin et d'état qui signifient :

**A** pour *All*, opérateur de chemin décrivant tous les chemins,

**E** pour *Exists*, opérateur de chemin décrivant au moins un chemin,

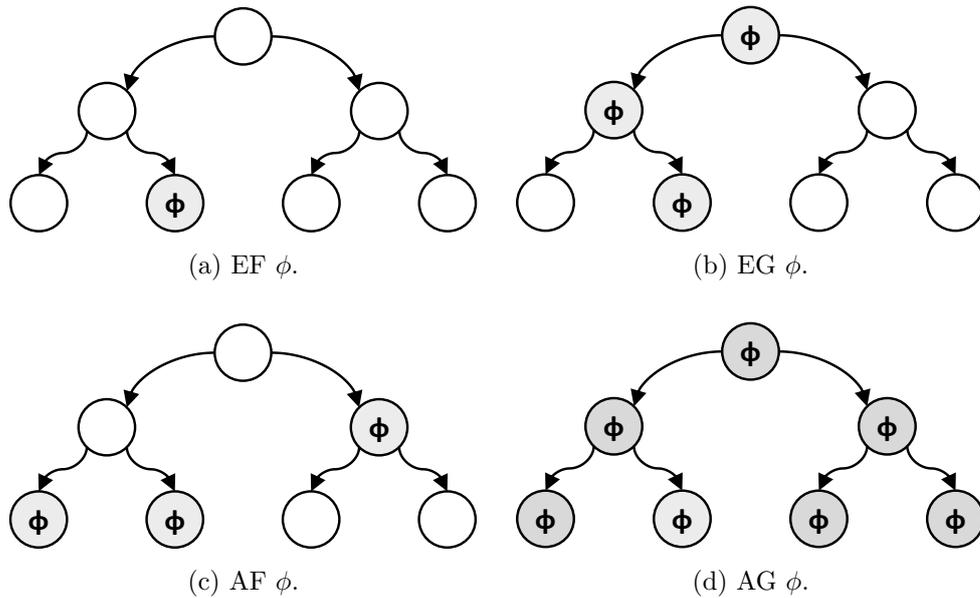


FIGURE 4.12 – Exemples de propriétés CTL sur la proposition logique  $\phi$  et de modèles les vérifiant.

**F** pour *Finally*, opérateur temporel décrivant un état du futur,

**G** pour *Globally*, opérateur temporel décrivant tous les états futurs.

Pour la logique LTL, on retrouve les opérateurs temporels F et G ainsi qu'un nouveau : l'opérateur X, pour *neXt*, qui correspond au prochain état.

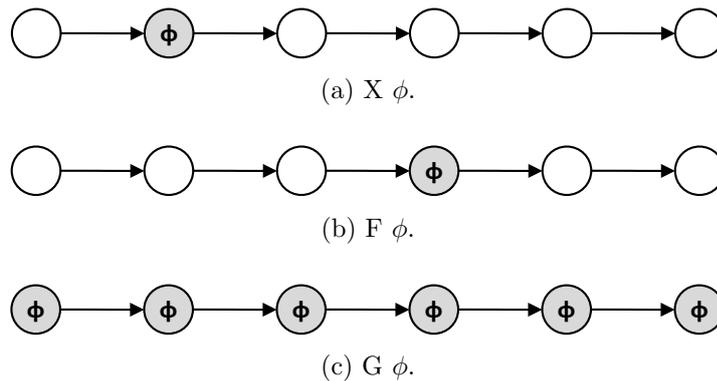


FIGURE 4.13 – Exemples de propriétés LTL sur la proposition logique  $\phi$  et de modèles les vérifiant.

Selon le model-checker utilisé, on peut définir la propriété formelle en CTL et/ou LTL et on peut construire des propriétés complexes en imbriquant plusieurs quantificateurs d'un même langage.

### 4.2.2.3 Le model-checker d'UPPAAL

Le manque d'outillage du formalisme des ATVD nous a poussé à utiliser la suite logicielle UPPAAL dans le chapitre 2. Il convient donc de présenter le model-checker de cette suite logicielle.

Ce model-checker possède de nombreuses qualités (accessibilité, taille des modèles traités) mais possède aussi des limitations, il utilise en effet une sous-classe de la logique CTL limitée à seulement 5 types de propriétés.

Les propositions logiques utilisables dans UPPAAL sont des expressions booléennes (opérateurs  $\&\&$ ,  $\|\|$  et  $!$ ) sur des localités actives des modèles (de la forme `automate.localité`) et des conditions sur des variables ou sur des horloges (expressions arithmétiques utilisant les opérateurs  $<$ ,  $>$ ,  $<=$ ,  $>=$ ,  $==$  et  $!=$ ).

Si on définit  $p$  et  $q$  deux propositions logiques d'UPPAAL, il est possible de définir les 5 propriétés supportées par UUPPAAL en utilisant les quantificateurs de chemins (A et E) et d'états ( $\langle\rangle$  pour F et  $\llbracket$  pour G) :

**Possibly** : La propriété  $E\langle\rangle p$  est vrai si et seulement s'il existe au moins une séquence d'évolutions partant de l'état initial et atteignant un état où la proposition  $p$  est vérifiée.

**Invariantly** : La propriété  $A\llbracket p$  est vrai si et seulement si tous les états atteignables depuis l'état initial vérifient  $p$ .

**Potentially always** : La propriété  $E\llbracket p$  est vraie si et seulement s'il existe au moins une séquence d'évolutions partant de l'état initial pour laquelle  $p$  est vérifiée pour tout les états. Cette séquence doit en plus être infinie ou aboutissant sur un blocage.

**Eventually** : La propriété  $A\langle\rangle p$  est vraie si et seulement si toutes les séquences d'évolution possible partant de l'état initial atteignent un état vérifiant  $p$ .

**Leads to** : La syntaxe  $p \rightarrow q$  décrit une propriété signifiant que si  $p$  est vérifiée, alors  $q$  devra être vérifiée dans le futur, pour toutes les séquences possibles depuis l'état où  $p$  a été vérifiée. C'est une écriture condensée de la formule  $A\llbracket p \text{ imply } A\langle\rangle q$ . L'opérateur `imply` a un comportement tel que  $a \text{ imply } b$  est équivalent à  $!a \|\| (a \&\& b)$ .

## 4.2.3 Méthode des automates observateurs

Une des principales difficultés du model-checking consiste en l'écriture des propriétés à vérifier (Bérard *et al.* (2001); Campos et Machado (2009)). En effet, le langage formel utilisé est très contraignant et il peut devenir extrêmement difficile d'écrire des propriétés complexes imbriquant plusieurs quantificateurs.

De plus, l'outil utilisé (UPPAAL) ne permet pas d'écrire des propriétés en logique LTL ou d'imbriquer des quantificateurs.

Dans ce cas, on a recours à la méthode dite des automates observateurs, qui consiste à ajouter des automates qui se limitent à l'observation du modèle analysés et dont les états et transitions sont utilisés pour vérifier la propriété.

La figure 4.14 montre l'exemple d'un automate observateur (figure 4.14a) utilisé pour remplacer une propriété de sécurité complexe qui consiste à vérifier qu'un mécanisme ne démarre pas après une alarme sans avoir subi de réparation. L'activation de la localité *KO* signifie que le comportement à éviter est apparu; nous associons donc la propriété de la figure 4.14b qui consiste à demander à ce qu'il n'existe aucun chemin tel que la localité *KO* de l'automate OBS soit atteinte. Ainsi, si la propriété de non-atteignabilité est vérifiée, le comportement interdit n'est pas possible et la propriété globale est vérifiée, sinon un contre-exemple est donné.

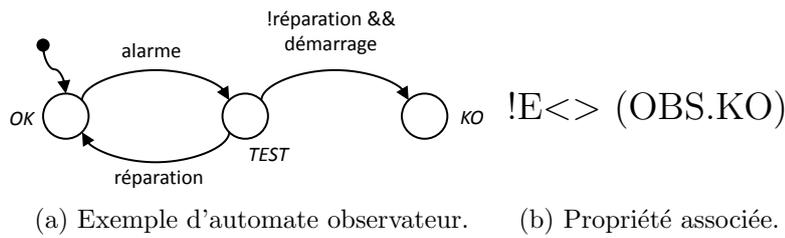


FIGURE 4.14 – Exemple d'un automate observateur et de sa propriété associée.

Cet ajout change quelque peu l'utilisation du model-checker, la figure 4.15 montre schématiquement l'utilisation d'un model-checker lorsqu'on introduit un automate observateur.

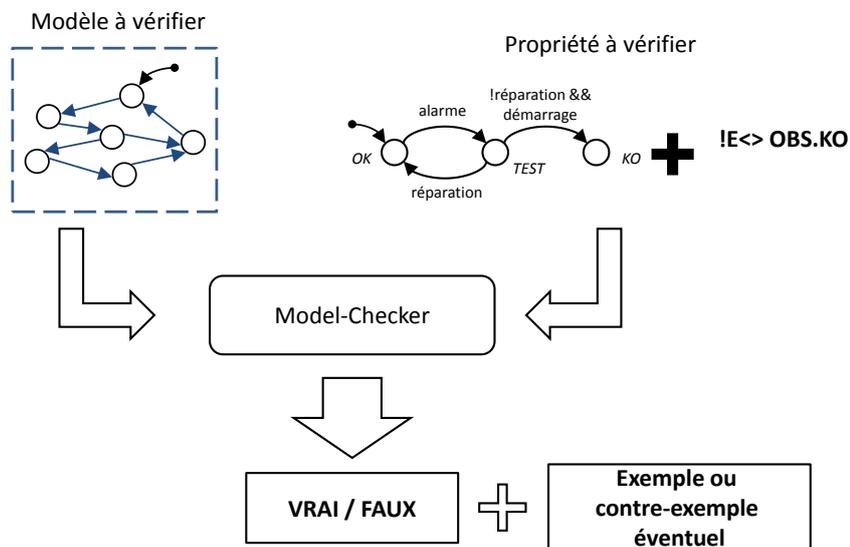


FIGURE 4.15 – Principe du model-checking avec observateur(s).

#### 4.2.4 Principe des automates observateurs appliqué à la preuve d'équivalence

On souhaite vérifier que les traces de sorties des deux modèles soient identiques. Une solution simple consiste à donner les deux modèles au model-checker et à utiliser un

automate observateur qui va observer les variables de sortie des deux modèles et évoluer en cas de différence. L'utilisation conjointe des deux modèles pose cependant un problème simple ; si ils assignent les mêmes variables on ne peut plus distinguer quel modèle a fait l'assignation.

Pour pallier à cela, nous définissons toujours deux modèles A et B à comparer, et les variable communes utilisées pour l'équivalence sont renommée en ajoutant `_A` pour le modèle A et `_B` pour le modèle B.

Ainsi on utilise avec le model-checker :

**Modèles A et B :** Les deux modèles à comparer sont donnés à l'outil de vérification.

**Automate observateur :** Un automate observateur est ajouté. Il observe les sorties des deux modèles à comparer. Cet automate peut éventuellement se présenter sous la forme de plusieurs automates (un par variable à surveiller) pour une plus grande lisibilité.

**Une propriété formelle :** Une propriété d'atteignabilité sur un état spécifique de l'observateur (ici `KO`), ou plusieurs propriétés dans le cas où l'on décide de mettre un observateur par sortie.

Dans l'exemple de la figure 4.16, pour que les modèles A et B soient équivalents, il faut que les sorties du modèle A (`OUT1_A` et `OUT2_A`) soient identiques à leurs homologues du modèle B (`OUT1_B` et `OUT2_B`) tout au long de la trace. L'automate observateur vérifie cette égalité à l'aide de la garde `OUT1_A != OUT1_B || OUT2_A != OUT2_B` permettant d'atteindre l'état `KO`. La propriété formelle est donc une non-atteignabilité de l'état `KO` de l'observateur.

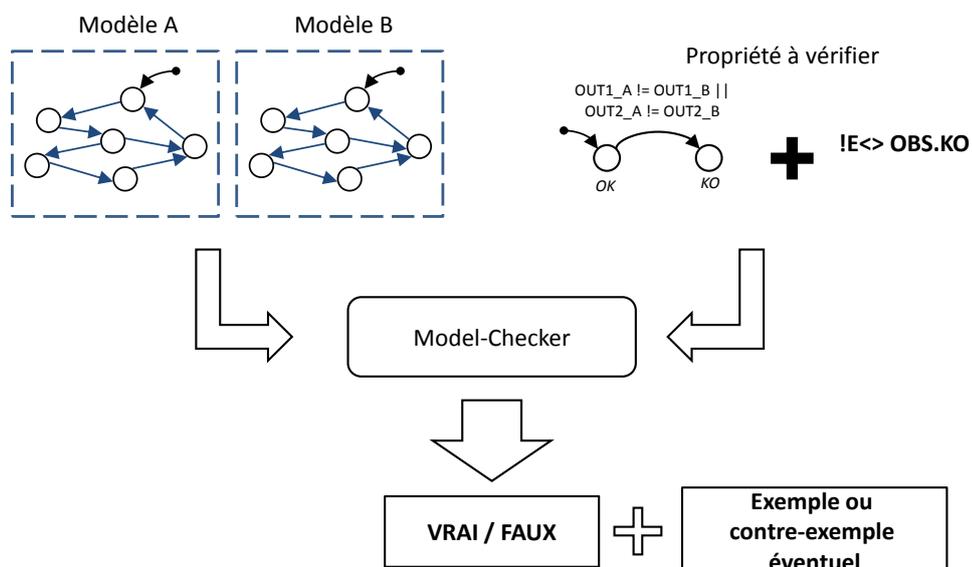


FIGURE 4.16 – Schéma de vérification d'une équivalence de traces par automate observateur et model-checking.

## 4.3 Preuve d'une équivalence de traces

Cette section traite de la création d'un automate chargé de vérifier l'équivalence de traces de deux modèles A et B. On débute par des modèles A et B non temporisés sans variables d'entrée ( $V_e^A = V_e^B = \emptyset$ ), puis on utilise des modèles temporisés pour finir par des modèles temporisés réagissant à des variables d'entrée.

### 4.3.1 Création d'un automate observateur-séquenceur non temporisé

Nous nous intéressons maintenant au cas des modèles non-temporisés uniquement générateurs comme présenté sur la figure 4.17.

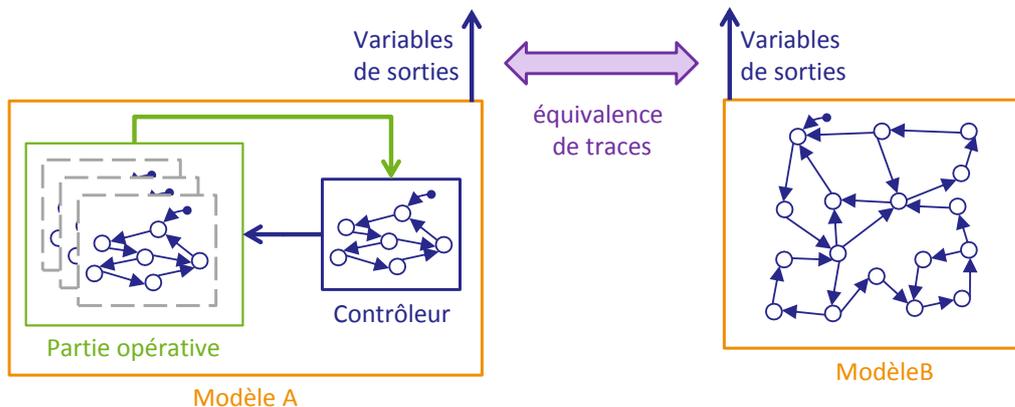


FIGURE 4.17 – Équivalence de traces de modèles non-temporisés uniquement générateurs.

#### 4.3.1.1 Concurrences entre les modèles à tester et l'automate observateur

La figure 4.18 montre une séquence mettant en avant ces concurrences. Pour l'exemple, des modèles simples A et B identiques assignant les variables OUT\_A et OUT\_B alternativement à vrai ou faux sont utilisés.

Un automate observateur (noté OBS\_OUT) est chargé de surveiller les deux sorties : ce dernier passe dans un état *KO* représentant un comportement non identique si, à un moment donné, OUT\_A est différent de OUT\_B.

La figure 4.18a montre les automates dans leurs états initiaux.

La figure 4.18b montre une première évolution du modèle A.

Puis, sans laisser l'opportunité au modèle B d'évoluer, l'automate observateur évolue car il détecte un comportement non identique, comme montré dans la figure 4.18c.

On remarque donc que l'observateur détecte une non-équivalence alors que les deux modèles sont identiques et *a fortiori* équivalents.

La définition de l'équivalence retenue suppose que les sorties des modèles sont comparées à un même indice de trace, or dans l'exemple précédent la trace du modèle A n'a pas le même indice que celle du modèle B.

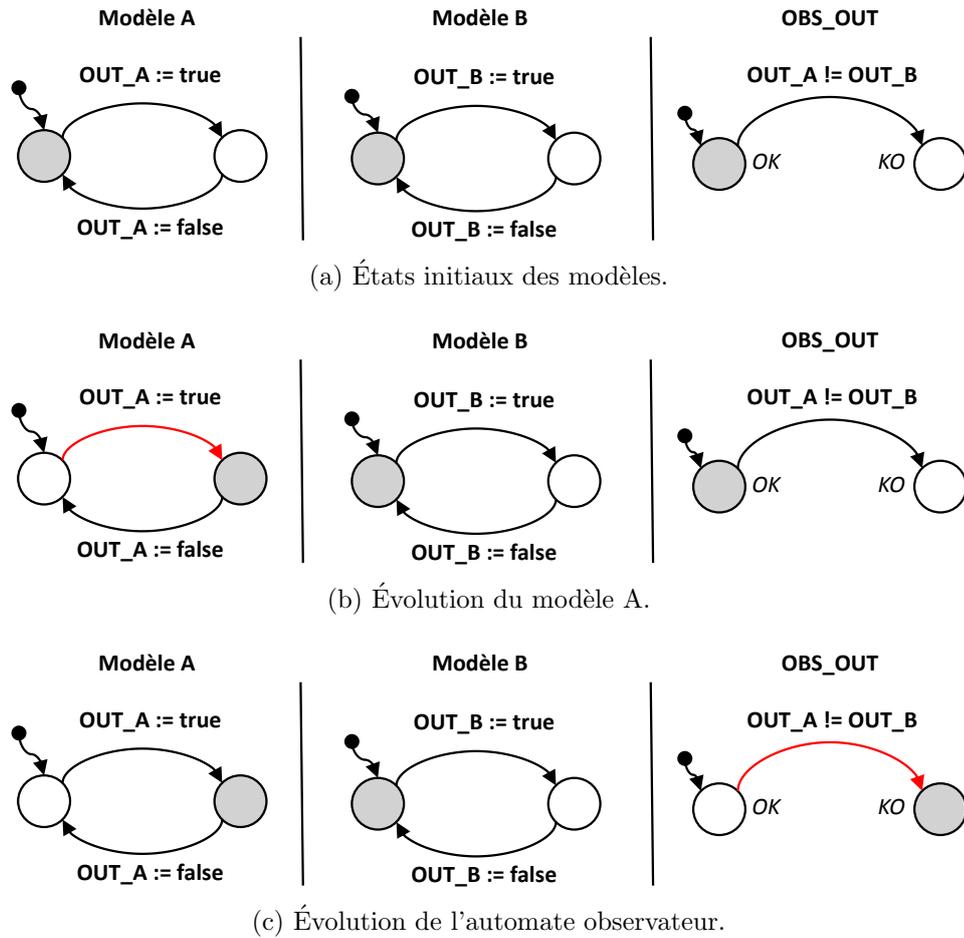


FIGURE 4.18 – Séquence d'exemple.

Notre automate observateur doit donc être modifié afin :

- De compter l'indice courant des traces des modèles à comparer.
- D'autoriser la comparaison des sorties uniquement lorsque les indices des traces sont identiques.

Cet automate est appelé un Automate Observateur-Séquenceur (AOS), car il observe les évolutions des modèles à comparer pour définir l'indice courant des traces des deux modèles (aspect observateur) tout en interdisant le test d'équivalence lorsque les traces ne sont pas au même indice (aspect séquenceur).

#### 4.3.1.2 Ajout d'un automate observateur-séquenceur

L'Automate Observateur-Séquenceur (AOS) va donc être une version plus complexe qu'un simple observateur et va utiliser les variables suivantes :

$i\_A$  et  $i\_B$  sont des variables entières qui représentent les indices des traces des modèles A et B.

**FREEZ** est une variable booléenne qui permet la communication entre les modèles à tester et l'AOS :

- Elle est mise à vrai lorsqu'une évolution d'un des modèles (A ou B) assigne une variable prise en compte dans l'équivalence.
- Son complément est ajouté dans *toutes* les gardes des modèles A et B. Une fois mise à vrai, aucune évolution des modèles A et B n'est possible afin de laisser la priorité à l'AOS pour qu'il mette à jour les valeurs des indices de trace.
- Une fois les indices mis à jour, l'AOS repasse cette variable à faux pour laisser les modèles A et B continuer leurs évolutions.

**OUT\_A\_MEM et OUT\_B\_MEM** : Variables qui mémorisent les valeurs des variables OUT\_A et OUT\_B pour l'indice courant. Lorsque FREEZ passe à vrai, on compare ces valeurs à celles actuelles afin de déterminer si la trace a évolué ou si seule une évolution invisible est arrivée, comme une assignation d'une variable à sa valeur actuelle. S'il y a une différence entre la valeur mémorisée et la valeur actuelle, l'AOS fait évoluer la variable d'indice de trace et met à jour les valeurs de OUT\_A\_MEM et OUT\_B\_MEM pour les comparaisons futures. De manière plus générique, pour chaque variable à comparer pour l'équivalence, une variable de même type (booléenne ou entière) est créée avec le même nom mais avec le suffixe \_MEM et est utilisée pour déterminer si la trace a évolué. Cette variable est initialisée à la même valeur que la variable dont elle sert de mémoire (ici, pour OUT\_A et OUT\_B, initialisée à faux donc non représentée sur la figure).

Afin de garantir le comportement vis-à-vis de la variable FREEZ, les modifications suivantes sont apportées aux modèles A et B :

- Si une transition assigne une variable de sortie comparée dans l'équivalence, on ajoute une assignation de FREEZ à vrai ( $\text{FREEZ} := \text{true}$ ) à son action.
- Dans toutes les gardes des transitions des modèles A et B, une condition sur FREEZ est ajoutée : l'évolution n'est possible que si FREEZ est fausse. Ainsi, dès qu'un modèle assigne une variable à comparer, il bloque les évolutions des deux modèles A et B.

Pour sa part, l'AOS prend la forme de l'automate montré dans la figure 4.19 :

Son comportement est le suivant : la localité initiale est quittée lorsque FREEZ devient vraie, on passe alors dans la localité 1.

D'ici, trois situations sont possibles :

- Si le modèle A a assigné FREEZ et que OUT\_A est différente de OUT\_A\_MEM, la localité active devient 3, on fait avancer l'indice de trace ( $i_A := i_A + 1$ ) et la valeur de OUT\_A\_MEM est mise à jour ( $\text{OUT\_A\_MEM} := \text{OUT\_A}$ ). Ensuite, en revenant à la localité initiale 0, la variable FREEZ est remise à faux afin de laisser les modèles A et B poursuivre leurs évolutions.
- Si le modèle B a assigné FREEZ et que OUT\_B est différente de OUT\_B\_MEM, la localité active devient 2. Ce faisant, on fait évoluer l'indice de trace  $i_B$  et on met à jour OUT\_B\_MEM. Enfin, en revenant à la localité 0, on assigne FREEZ à faux.
- Si les sorties des deux modèles A et B sont identiques aux valeurs mémorisées, alors l'assignation est due à une action invisible et les indices de trace ne doivent pas évoluer. On revient directement à la localité 0 en assignant FREEZ à faux.

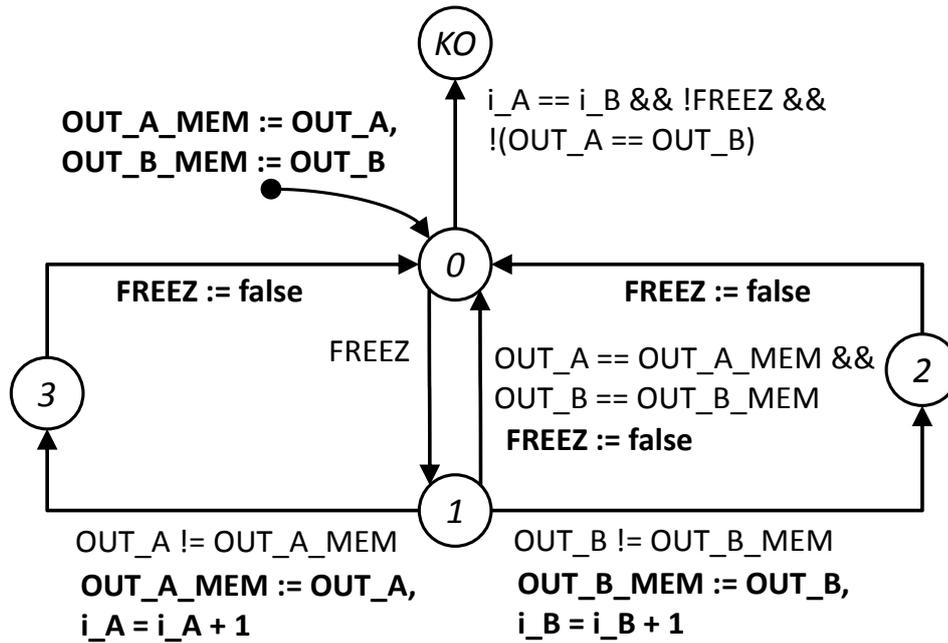
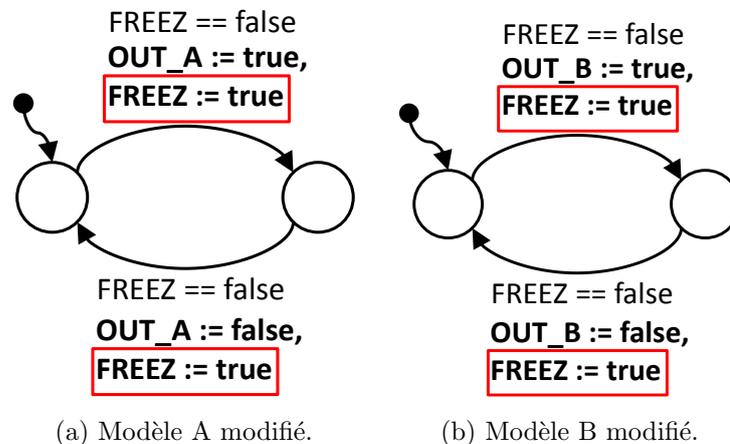


FIGURE 4.19 – Première version de l'AOS.

Enfin, à tout moment, si les indices de trace sont identiques ( $i\_A == i\_B$ ), si une évolution d'une trace n'est pas en cours ( $FREEZ$  à faux) et si les valeurs de sortie des deux modèles ne sont pas identiques, on peut passer dans la localité  $KO$  qui signifie que les deux modèles ne sont pas équivalents.

Nous nous intéressons dans cette section à des ATVD à comparer non temporisés, ainsi toutes les transitions sont non urgentes dans un premier temps.

Les figures 4.20a et 4.20b montrent les modèles A et B précédents modifiés pour prendre en compte la variable  $FREEZ$ .

FIGURE 4.20 – Modèles modifiés pour prendre en compte la variable  $FREEZ$ .

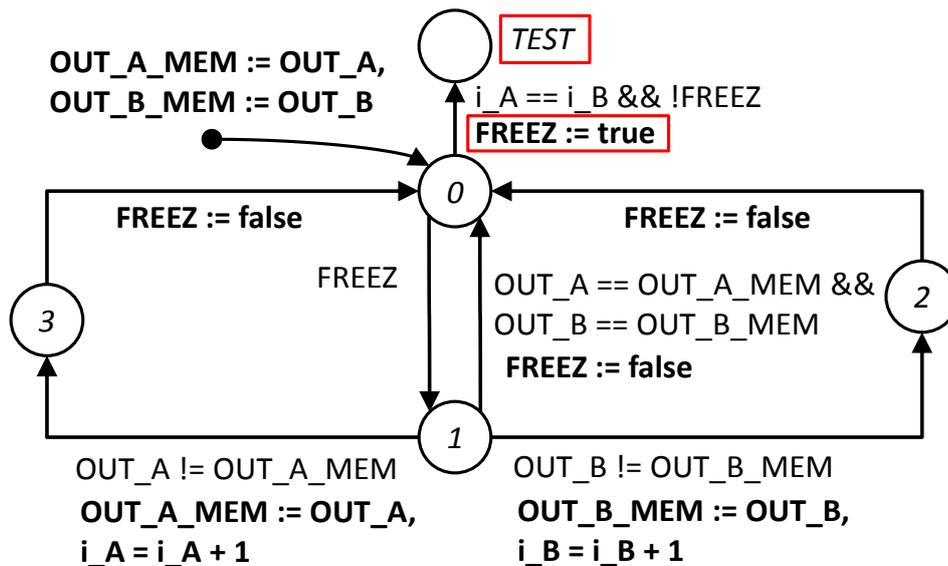
L'exemple précédent de la figure 4.18 est repris avec le nouvel AOS de la figure 4.19. Cette séquence d'évolutions est décrite avec précision dans l'annexe E.1 pour des raisons de place mais on constate alors que le test est concluant : le model-checker vérifie que les deux modèles sont équivalents.

#### 4.3.1.3 Évolution de l'AOS pour un gain en efficacité

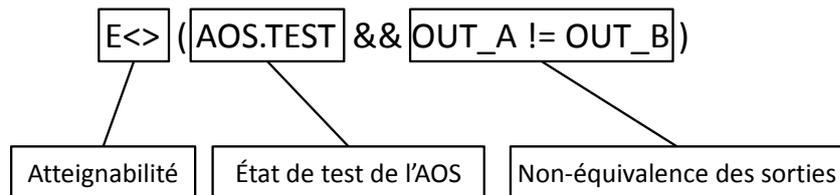
Le modèle d'AOS présenté dans la figure 4.19 permet de vérifier simplement l'équivalence de deux ATVD non temporisés et non réactifs (ne réagissant pas à des entrées) produisant une trace. Dans le cadre d'un passage à l'échelle sur des modèles à comparer produisant plusieurs variables de sortie (OUT1\_A, OUT2\_A, ...), l'automate proposé passe dans l'état *KO* quel que soit le couple de variables qui ont une valeur différente. Une vérification supplémentaire excluant le couple trouvé doit alors être relancée pour vérifier l'équivalence de comportement sur les autres variables.

Pour gagner en efficacité, nous avons choisi de faire apparaître les variables à contrôler au niveau de la *propriété formelle* et non plus au niveau de l'AOS. Le nouvel AOS et sa propriété associée pour l'exemple de la figure 4.18 sont représentés dans la figure 4.21. On peut y remarquer que la localité *KO* est remplacée par une localité *TEST* qui correspond à une configuration où l'équivalence peut être testée. La transition permettant d'accéder à cette localité *peut* être franchie à chaque fois que les traces des modèles A et B sont au même indice et que FREEZ est faux. Afin de garantir que l'indice de trace des modèles comparés reste égal à l'indice de trace de comparaison, leurs évolutions sont bloquées par le passage de FREEZ à vrai. Comme le model-checker vérifie *toutes* les évolutions possibles, il testera l'équivalence à chaque fois que c'est possible (dès que la transition est franchissable).

Ainsi, pour chaque couple de variables (une du modèle A et une du modèle B) à comparer, une propriété formelle est ajoutée permettant d'obtenir un résultat d'équivalence par variable. Les deux modèles sont dits équivalents si *aucune des propriétés formelles n'est vérifiée*, à savoir si l'on ne se trouve jamais dans la localité *TEST* avec des variables de valeurs différentes entre les modèles A et B.



(a) Nouvel AOS avec la localité *TEST* utilisée pour tester les valeurs des sorties au même indice de trace.



(b) Propriété formelle détaillée relative aux variables *OUT\_A* et *OUT\_B*.

FIGURE 4.21 – Nouvel AOS avec la localité *TEST* et la propriété formelle associée aux variables de l'exemple de la figure 4.18.

### 4.3.2 Équivalence de modèles temporisés

Cette section traite des modifications à apporter à l'AOS si les modèles à comparer sont temporisés.

#### 4.3.2.1 Modification de l'AOS pour limiter les évolutions d'horloges

Lorsque les modèles à comparer sont temporisés, les évolutions des horloges sont possibles et la variable FREEZ n'est plus suffisante seule pour bloquer les évolutions des modèles A et B. L'AOS doit donc interdire la sémantique temporelle dès que la variable FREEZ passe à vrai : pour ce faire, toutes les transitions de cet automate deviennent des transitions urgentes. *La seule exception* à cette règle concerne la transition allant vers la localité *TEST* car une transition urgente ici interdirait toute évolution des horloges dès que les deux traces auraient le même indice et que FREEZ serait faux, bloquant par là même les évolutions temporisées des modèles A et B en permanence (blocage déjà rencontré dans la section 3.3.2).

La version modifiée de l'AOS est présentée dans la figure 4.22.

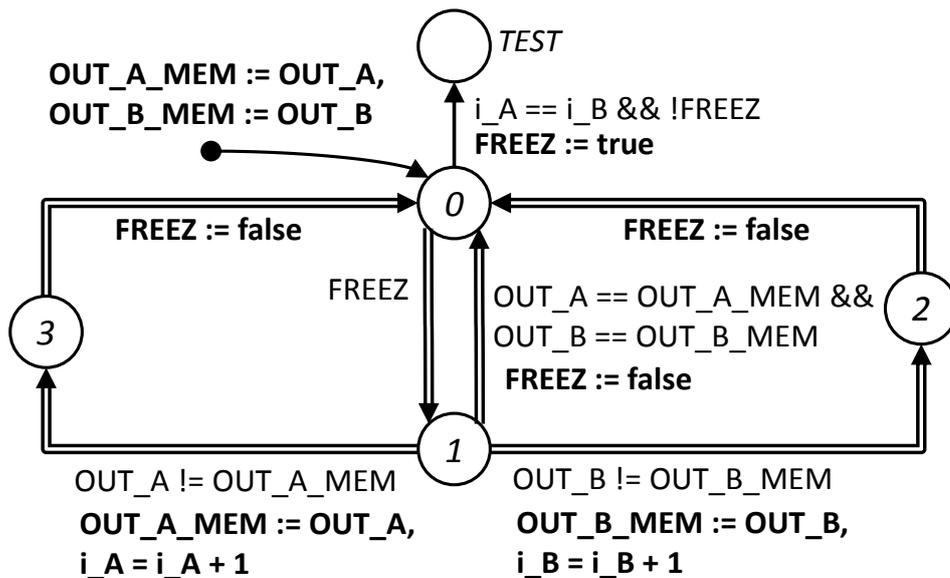
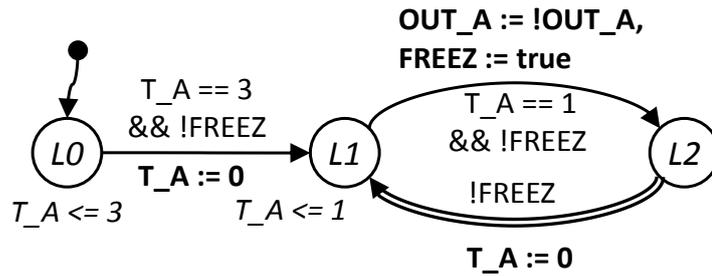


FIGURE 4.22 – Adaptation de l'AOS aux modèles temporisés.

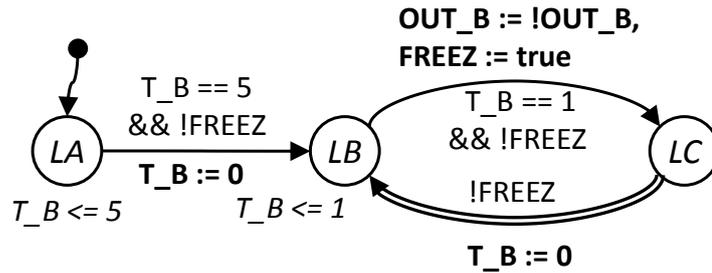
#### 4.3.2.2 Mise en évidence du problème de décalage temporel des traces

Lors de l'utilisation de modèles à comparer temporisés, un problème de décalage temporel peut amener à ne pas détecter des modèles non équivalents.

Pour mettre en évidence ce comportement, les modèles A et B précédemment utilisés (figure 4.20) ont été modifiés pour intégrer des comportements temporisés. Ils sont détaillés dans la figure 4.23.



(a) Version temporisée du modèle A (avec variable FREEZ ajoutée).



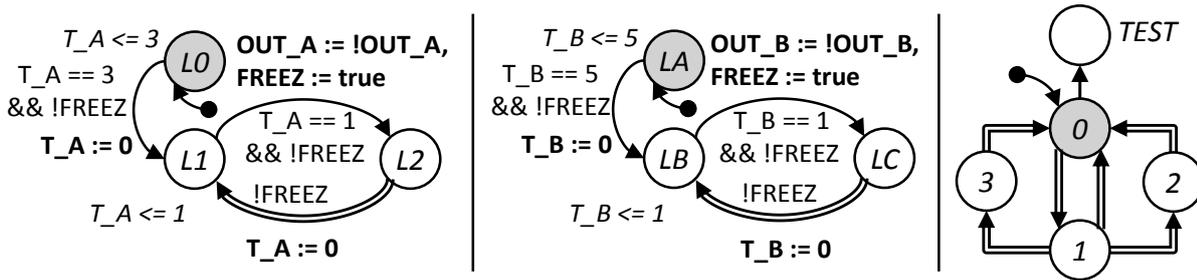
(b) Version temporisée du modèle B (avec variable FREEZ ajoutée).

FIGURE 4.23 – Version temporisée des modèles A et B.

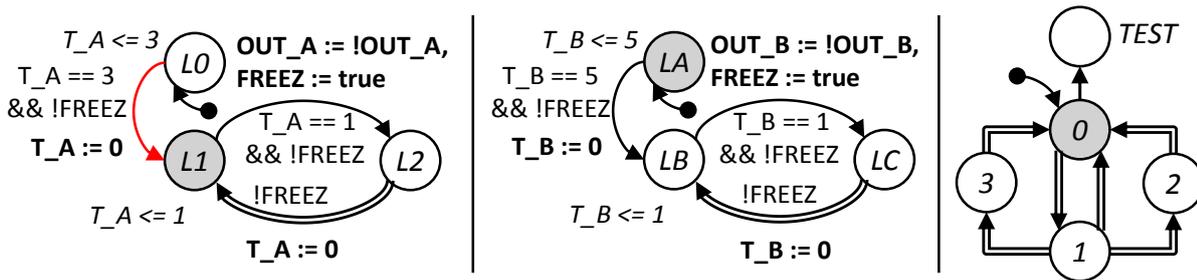
Outre l'ajout de contraintes temporelles, d'invariants et d'une transition urgente, les modèles sont maintenant *non équivalents* du fait de la première transition, et de l'invariant de la localité initiale du modèle B qui utilise une contrainte temporelle différente : l'horloge du modèle B doit avoir atteint la valeur 5 (au lieu de 3) pour que la transition soit franchissable. Les deux traces sont alors décalées dans le temps et donc non équivalentes.

La figure 4.24 montre une suite d'évolutions mettant en avant le problème de décalage temporel :

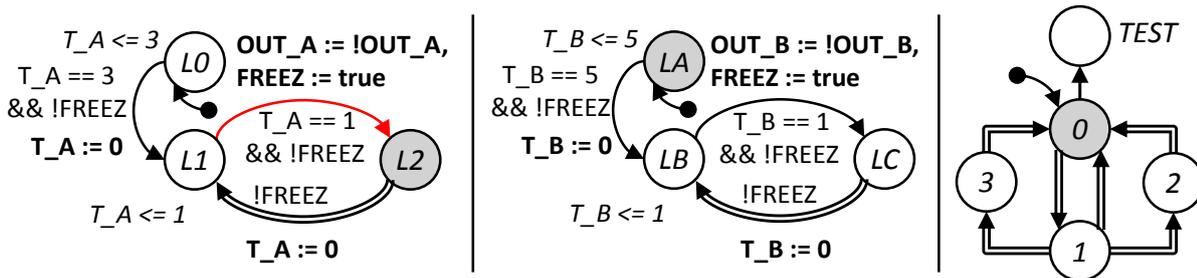
- La figure 4.24a représente les modèles (A, B et de l'AOS) dans leur situation initiale.
- Après 3 unités de temps, le modèle A évolue et passe dans la localité  $L1$  (figure 4.24b). Comme cette évolution n'agit pas sur  $OUT\_A$ , elle ne comporte pas d'assignation concernant  $FREEZ$  et l'AOS n'y réagit pas (évolution interne au modèle ne modifiant pas la trace).
- Une unité de temps plus tard, le même modèle A évolue vers  $L2$  (figure 4.24c) et assigne  $OUT\_A$  (initialisée à faux) à vrai. Cette évolution agit sur une variable de la trace de sortie et comporte donc une assignation sur  $FREEZ$ .
- La figure 4.24d montre la réaction de l'AOS qui passe  $i\_A$  de 0 à 1 (évolution de l'indice de trace).
- Ensuite, comme  $FREEZ$  est repassée à faux par l'AOS, le modèle A évolue via sa transition urgente et revient en  $L1$  (figure 4.24e).
- Une unité de temps plus tard, deux transitions sont franchissables : celle du modèle A (passage de  $L1$  en  $L2$ ) et celle du modèle B (passage de  $LA$  en  $LB$ ). Nous choisissons de laisser B évoluer et la figure 4.24f montre cette évolution : passage de  $LA$  en  $LB$ . Comme cette évolution ne concerne pas la variable  $OUT\_B$  observée par l'AOS, la



(a) Situation initiale des modèles A, B et de l'AOS.



(b) Évolution du modèle A après 3 unités de temps.



(c) Évolution, après une unité de temps, du modèle A modifiant la trace.

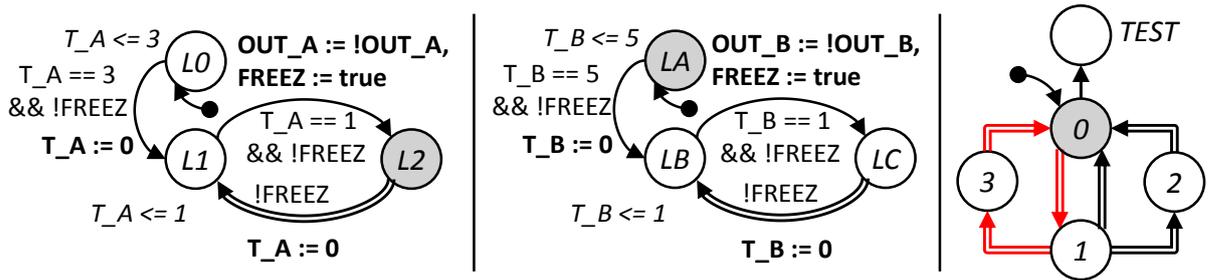
FIGURE 4.24 – Suite d'évolutions mettant en évidence le problème de décalage temporel.(1)

variable `FREEZ` n'est pas assignée et l'AOS ne réagit pas à cette évolution.

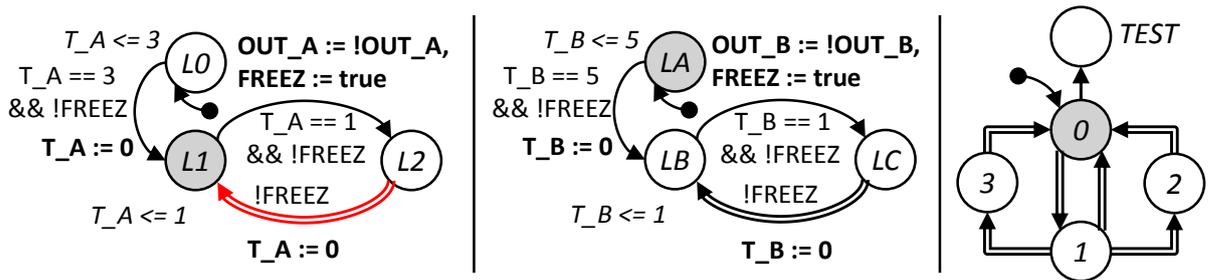
Il convient maintenant d'étudier la situation :

- L'évolution du modèle A est en suspens mais ce modèle va de toute manière continuer d'évoluer avec un changement de valeur de `OUT_A` toutes les unités de temps.
- Le modèle B, quant à lui, commence à peine le cycle d'inversion de sa variable `OUT_B` à chaque unité de temps.

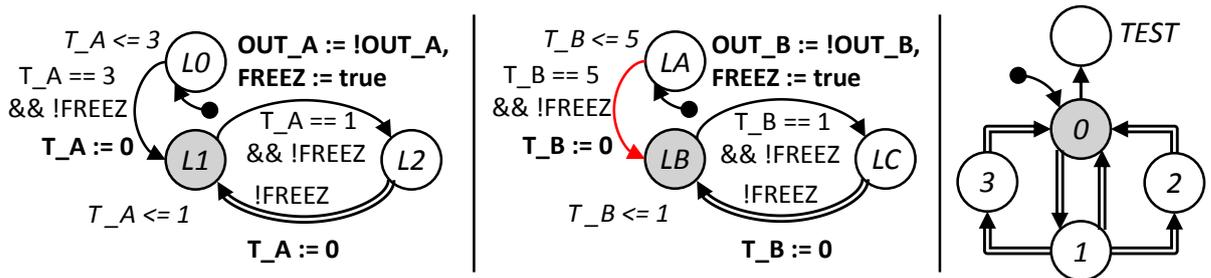
L'indice de trace `i_A` du modèle A va donc bientôt passer à 2 alors que celui du modèle B est toujours à 0, et cet écart va perdurer tout au long des évolutions car le modèle A a « pris de l'avance » sur le modèle B. Cet écart a une conséquence importante sur le test d'équivalence : en effet, il n'y aura *jamais plus* `i_A` égal à `i_B` car les deux modèles sont forcés à évoluer par leurs invariants. La localité `TEST` n'est donc *plus jamais* accessible et la comparaison ne peut plus être faite qu'à l'initialisation où `i_A = i_B = 0` et où les sorties des deux modèles sont identiques.



(d) Réaction de l'AOS,  $i_A$  passe de 0 à 1.



(e) Évolution urgente du modèle A.



(f) Choix : évolution du modèle B.

FIGURE 4.24 – Suite d'évolutions mettant en évidence le problème de décalage temporel.(2)

La réponse à la propriété formelle est donc négative (on n'atteint jamais la localité *TEST* avec  $OUT_A$  différente de  $OUT_B$ ) : *les modèles apparaissent comme équivalents!*

### 4.3.2.3 Solution au problème de décalage temporel des traces

Pour résoudre ce problème, il existe deux solutions :

- Mémoriser les valeurs des variables pour chaque indice puis comparer les traces indice par indice.
- Geler le modèle avec l'indice de trace le plus élevé afin de permettre au modèle en retard d'arriver au même indice de trace pour effectuer la comparaison et détecter l'éventuelle non-équivalence.

La première solution pose un problème évident de mémoire car, dans le cadre de modèles industriels avec de nombreuses variables, il est coûteux de mémoriser ainsi les valeurs pour de nombreux indices de trace.

La seconde solution est donc retenue. On souhaite le comportement suivant :

- L'indice de trace le plus grand ( $i_A$  ou  $i_B$ ) peut à tout moment être choisi comme indice de comparaison.
- Lorsque ce choix est fait, le modèle correspondant à l'indice le plus grand est gelé à l'aide d'une variable spéciale ( $FREEZ\_A$  ou  $FREEZ\_B$  selon le modèle considéré), ses évolutions sont bloquées afin de conserver la valeur de ses sorties de manière identique à celle utilisée pour  $FREEZ$  (ajout dans les gardes, précisé plus loin).
- Le second modèle peut alors « rattraper » le premier au niveau des indices de trace. Pour permettre ces évolutions, les invariants du premier modèle (en avance) doivent être désactivés afin de ne pas bloquer les évolutions des horloges. Pour ce faire, une localité spéciale est ajoutée et toute localité avec un invariant a la possibilité d'évoluer vers cette localité puits lorsque le modèle est gelé.

Nous rappelons que le model-checker va parcourir l'ensemble des possibilités, ainsi il teste obligatoirement tous les indices possibles, que ce soit  $i_A$  ou  $i_B$  le plus grand (modèle A ou B en avance). De plus, bloquer ainsi un modèle n'est pas gênant car le model-checker va jusqu'au test et, s'il est négatif, reprend *une autre possibilité d'évolution*, à savoir le non-blocage du modèle.

La figure 4.25 montre schématiquement les variables utilisées pour les communications entre les modèles A et B et l'AOS :

- **FREEZ**, assignée par les modèles A et B lors d'une évolution observable et par l'AOS lors du test, est toujours utilisée pour bloquer les *deux* modèles lorsque l'AOS évolue ;
- **FREEZ\_A**, assignée *uniquement* par l'AOS, est utilisée par l'AOS pour geler le modèle A ;
- **FREEZ\_B**, assignée *uniquement* par l'AOS, est utilisée par l'AOS pour geler le modèle B.

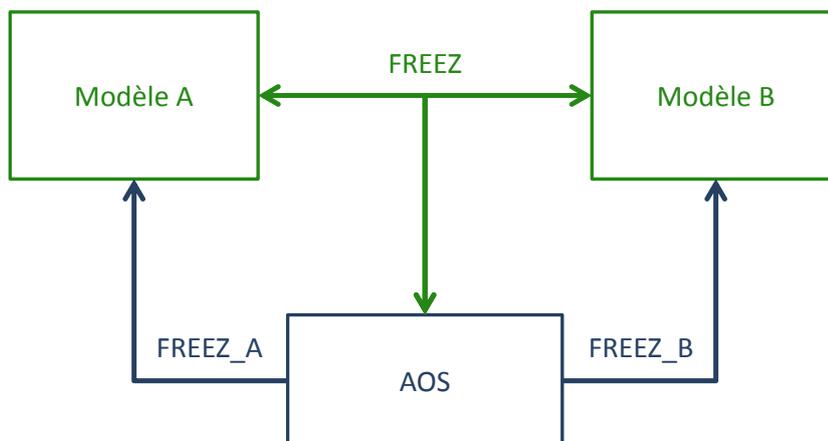


FIGURE 4.25 – Communications entre l'AOS et les modèles testés.

#### 4.3.2.4 Mesure du temps entre les deux indices

Maintenant que nous pouvons laisser un modèle évoluer pour rattraper l'indice de trace de comparaison, il faut mesurer le temps physique utilisé par ce modèle pour arriver à l'indice de trace de comparaison. En effet, pour que les traces de sortie soient équivalentes, il faut que les valeurs de l'horloge globale à l'indice de trace considéré soient identiques.

L'horloge globale est un atout important pour formaliser la trace temporisée mais est peu utilisable telle quelle puisque cette valeur peut devenir très importante (et pénaliser alors grandement le processus de vérification) et que l'on ne peut pas la sauvegarder.

Nous utilisons à la place deux horloges chargées de mesurer le temps écoulé depuis la dernière modification de trace :

- Last\_A est une horloge qui est réinitialisée à chaque fois que l'indice de trace  $i\_A$  du modèle A évolue. Elle mesure donc en pratique le temps écoulé depuis la dernière fois que cette trace a évolué.
- Last\_B est une horloge qui a le même comportement vis-à-vis du modèle B.

Pour que les traces des modèles A et B aient la même valeur d'horloge globale, il faut impérativement qu'au moment du test ( $i\_A == i\_B$ ), les valeurs de Last\_A et Last\_B soient identiques.

Soit  $i\_A = i\_B = \mu$  l'indice de trace de comparaison, on a  $(\tilde{V}_\mu^A, t_\mu^A)$  la valeur de la trace du modèle A à cet indice et  $(\tilde{V}_\mu^B, t_\mu^B)$  celle du modèle B. On ne s'intéresse ici qu'au temps,  $t$  se réfère donc au temps global. Il vient donc, au moment où l'on fait la comparaison :

$$\begin{cases} t_{\text{courant}} = t_\mu^A + \text{Last\_A} \\ t_{\text{courant}} = t_\mu^B + \text{Last\_B} \end{cases}$$

Soit :  $t_\mu^A = t_\mu^B$  implique que Last\_A = Last\_B.

#### 4.3.2.5 Modifications à apporter aux modèles

Les changements des modèles à tester A et B relatifs aux variables FREEZ\_A et FREEZ\_B sont détaillés ci-après :

- Chaque garde de transition du modèle A se voit ajouter `&& !FREEZ_A` afin d'en interdire le franchissement lorsqu'il est gelé (variable FREEZ\_A mise à vrai). Une modification similaire utilisant FREEZ\_B est faite pour le modèle B.
- Une localité spéciale puits est ajoutée pour chaque modèle A et B (et éventuellement pour chaque modèle composant A et B).
- Toutes les localités du modèle A utilisant un invariant se voient ajouter une transition urgente de garde `FREEZ_A` allant vers la localité spéciale. Une modification similaire utilisant FREEZ\_B et la localité spéciale du modèle B est faite pour le modèle B.

La version modifiée du modèle A est présentée dans la figure 4.26. Elle comporte la localité ajoutée `LX` ainsi que deux transitions urgentes permettant de désactiver les localités avec des invariants (`L0` et `L1`) en cas de gel du modèle A.

En ce qui concerne l'AOS, il est modifié pour prendre en compte le comportement des variables FREEZ\_A et FREEZ\_B ainsi que les horloges Last\_A et Last\_B : deux parties

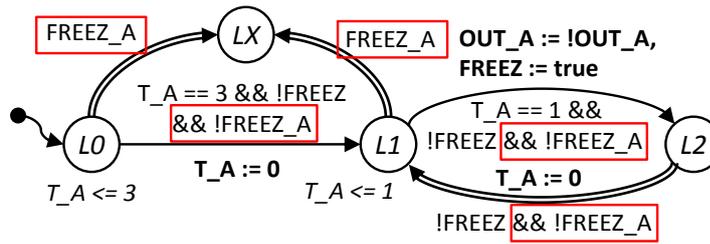


FIGURE 4.26 – Modèle A temporisé modifié.

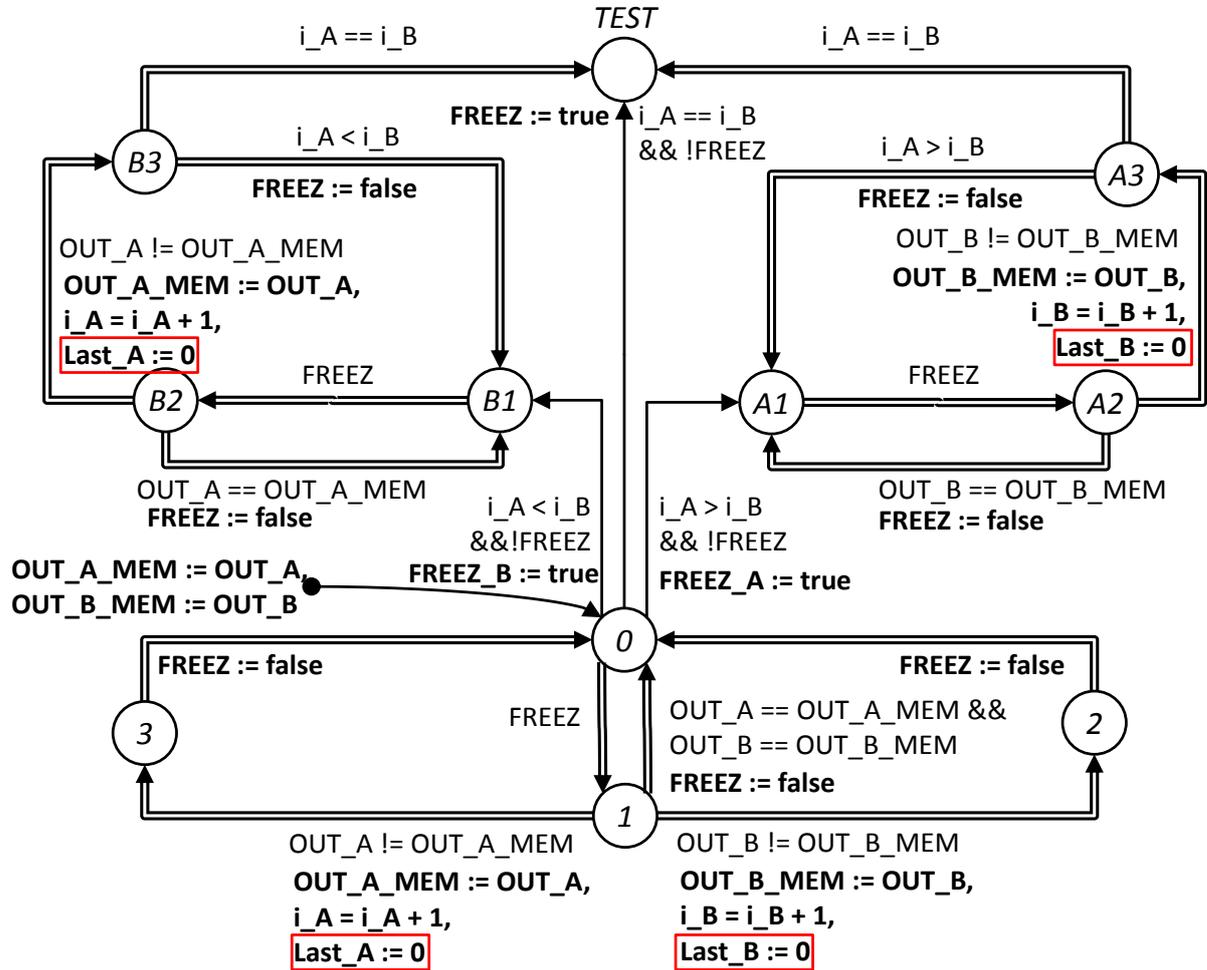
ont été ajoutées selon que l'indice  $i\_A$  est supérieur ou inférieur à l'indice  $i\_B$  ; pour le cas où  $i\_A$  est supérieur à  $i\_B$ , on peut définir le comportement suivant :

- La transition *non urgente* (pour la même raison que pour la transition allant de  $0$  à  $TEST$ ) allant de la localité initiale  $0$  vers la localité  $A1$  est franchie. À cette occasion, la variable  $FREEZ\_A$  est mise à vrai afin de geler le modèle A (puisque'il a l'indice le plus élevé).
- Depuis la localité  $A1$ , on retrouve le cycle de mise à jour de l'indice limité à l'indice  $i\_B$  : lorsque  $FREEZ$  passe à vrai (obligatoirement dû au modèle B puisque A est gelé), on compare les valeurs des sorties de B à celles mémorisées.
- En cas de non-changement, on revient à la localité  $A1$ .
- En cas de changement, on fait évoluer l'indice de trace  $i\_B$  et on met à jour les variables mémorisées (ici,  $OUT\_B\_MEM$ ).
- On teste ensuite si  $i\_B$  a rejoint  $i\_A$ . Si ce n'est pas le cas, on revient en  $A1$  et on remet  $FREEZ$  à faux pour laisser le modèle B rattraper son retard. Sinon (cas  $i\_B$  égal à  $i\_A$ ) on évolue vers la localité  $TEST$  pour comparer les valeurs des sorties. Comme  $FREEZ$  n'a pas été remise à zéro depuis son assignation par le modèle B, les deux modèles ont leurs évolutions gelées.

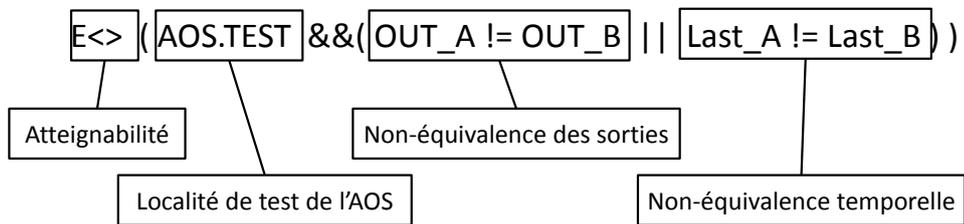
Un raisonnement similaire s'applique à la branche chargée du cas où l'indice de trace  $i\_A$  est inférieur à  $i\_B$ . La remise à zéro des horloges  $Last\_A$  et  $Last\_B$  se fait lorsque l'on incrémente les indices de trace (que l'on ait bloqué un modèle ou non).

De plus, pour que les modèles soient équivalents, il faut dorénavant que les variables soient identiques *et* que  $Last\_A$  soit égale à  $Last\_B$ , ce qui implique que la propriété formelle à vérifier est changée et la figure 4.27 montre la nouvelle version de l'AOS, accompagnée de la nouvelle propriété à vérifier.

La suite d'évolutions de la figure 4.24 est reprise avec les modèles modifiés et le résultat est alors satisfaisant. En effet, lors de l'arrivée en  $TEST$ , l'horloge  $Last\_A$  vaut 2, ce qui permet de déterminer que les traces sont non identiques, et par conséquent que les modèles sont non équivalents. Le détail des évolutions est décrit dans l'annexe E.2 pour des raisons de place.



(a) Version modifiée de l'AOS prenant en compte les variables FREEZ\_A et FREEZ\_B ainsi que les horloges Last\_A et Last\_B.



(b) Version modifiée de la propriété formelle associée.

FIGURE 4.27 – Versions modifiées de l'AOS et de la propriété formelle associée pour prendre en compte des modèles à comparer temporisés.

### 4.3.3 Équivalence de modèles temporisés réagissant à des entrées

Nous nous intéressons maintenant au cas des modèles temporisés réagissant à des variables d'entrée comme présenté sur la figure 4.28.

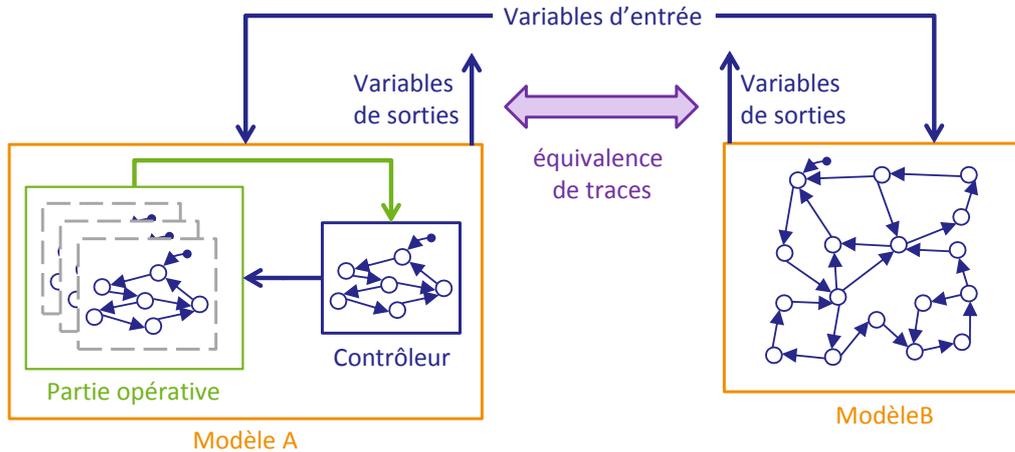


FIGURE 4.28 – Équivalence de traces de modèles temporisés réagissant à des entrées.

#### 4.3.3.1 Modèles des entrées

Les modèles testés pour l'équivalence sont maintenant affectés par les valeurs des variables d'entrée communes, il nous faut donc générer une trace d'entrée à donner aux deux modèles pour vérifier leur équivalence. Afin d'être complet, il faut pouvoir tester *toutes* les traces d'entrée possibles : pour ce faire, nous utilisons dans un premier temps des modèles d'entrées totalement libres.

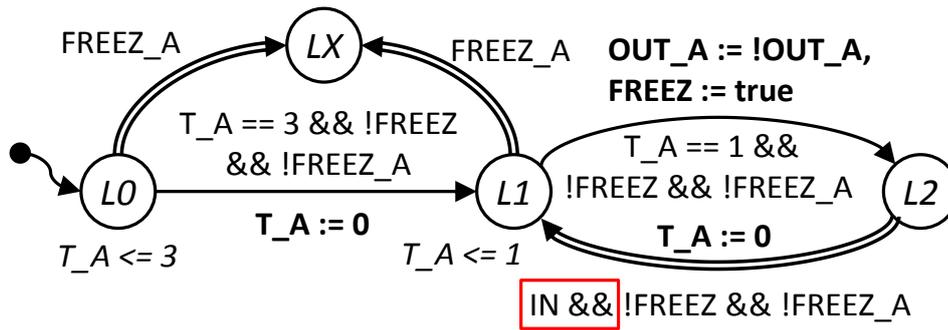
La figure 4.29 montre les modèles A et B utilisés dans cette section : ce sont les mêmes que ceux utilisés dans la section précédente (avec les variables FREEZ et FREEZ\_A / FREEZ\_B et les ajouts relatifs aux localités de gel  $LX$  et  $LW$ ), mais avec les différences suivantes :

- La première transition consomme la même quantité de temps (3 unités) dans les deux modèles.
- La transition urgente du cycle (de  $L2$  à  $L1$  pour le modèle A) nécessite la variable booléenne d'entrée IN à vrai pour être franchie.
- Les deux modèles sont identiques et donc équivalents en trace.

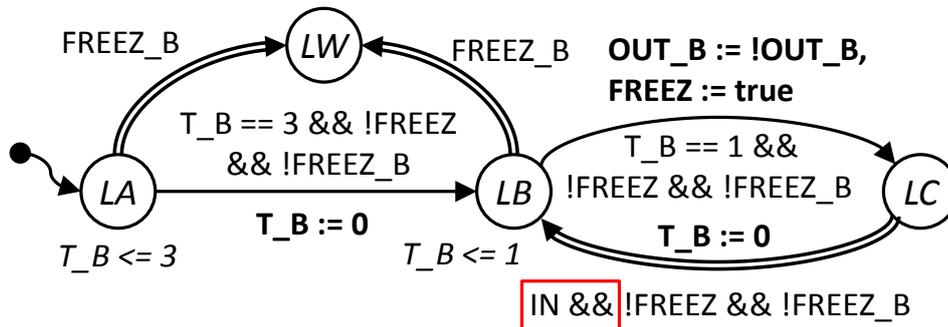
Le modèle de la variable IN est donné dans la figure 4.30, il est libre de toute contrainte mais n'utilise pas de transition urgente pour ne pas interdire de sémantique temporelle. IN est initialisée à faux (non représenté sur l'arc source selon notre convention).

#### 4.3.3.2 Concurrence entre les modèles des entrées et les autres modèles

La figure 4.31 représente une séquence d'évolutions mettant en lumière la concurrence entre les modèles des entrées et ceux testés et son implication sur le résultat de la vérifi-



(a) Modèle A temporisé et réagissant à la variable booléenne IN.



(b) Modèle B temporisé et réagissant à la variable booléenne IN.

FIGURE 4.29 – Modèles A et B utilisés dans cette section.

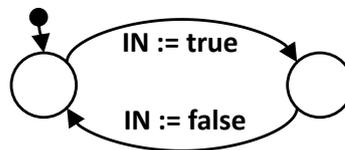


FIGURE 4.30 – Modèle de l'entrée IN.

cation d'équivalence.

La figure 4.31a montre l'état initial des modèles. La figure 4.31b montre la situation des modèles après plusieurs évolutions :

- Après 3 unités de temps, les deux modèles A et B évoluent en  $L1$  et  $LB$  respectivement.
- Ensuite, le modèle de l'entrée IN évolue pour passer IN à vrai.
- Après une unité de temps, le modèle A évolue en  $L2$  et assigne  $OUT\_A$ .
- Le modèle de l'AOS réagit à ce changement et fait évoluer  $i\_A$ .
- Le modèle B évolue en  $LC$  et assigne  $OUT\_B$ . Cette évolution est possible en dépit de la transition urgente franchissable car, comme  $T\_B$  est déjà égale à 1, la sémantique temporelle n'est pas nécessaire ici.
- L'AOS réagit au changement de  $OUT\_B$  et passe  $i\_B$  à 1.

Plusieurs transitions ne consommant pas de temps sont possibles :

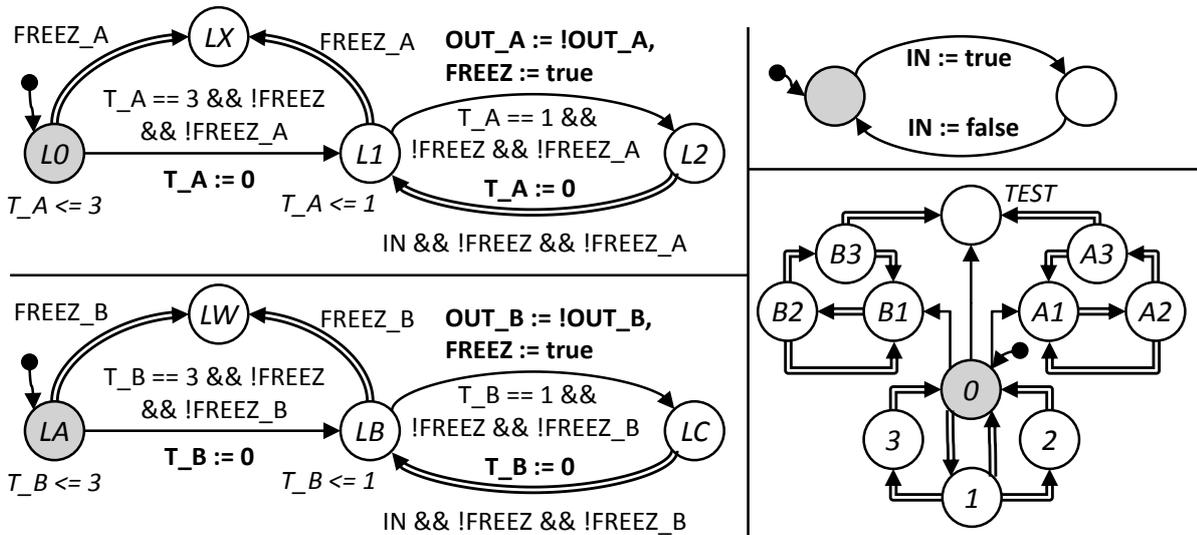
- La transition urgente du modèle A.

- La transition urgente du modèle B.
- La transition sans contrainte temporelle du modèle de l'entrée IN.
- La transition sans contrainte temporelle de l'AOS allant en *TEST*.

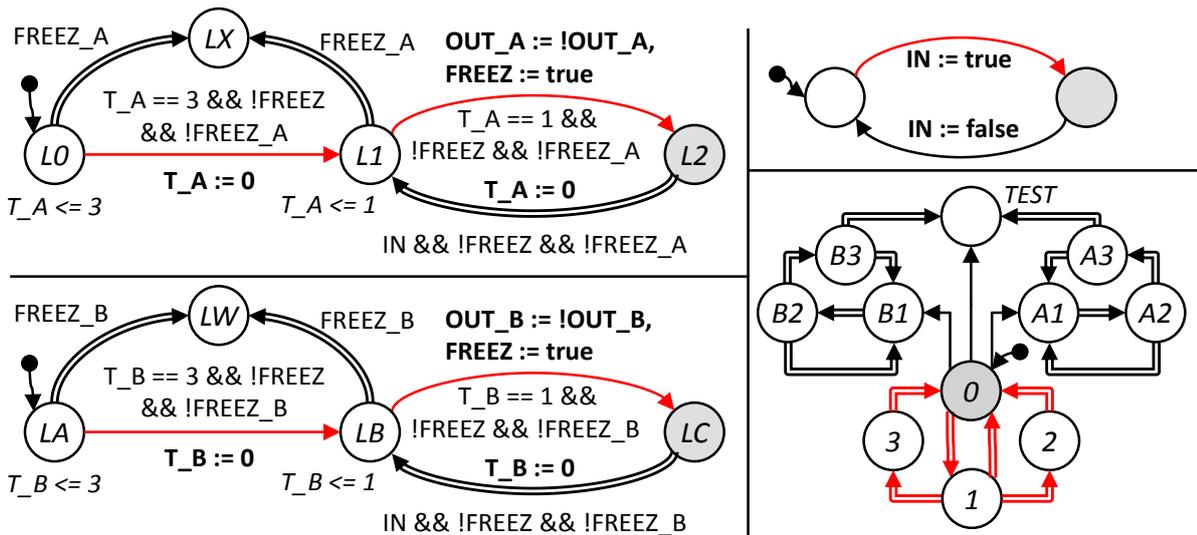
La figure 4.31c montre le franchissement de la transition du modèle A. La localité active devient *L1*.

Ensuite, nous choisissons de faire évoluer le modèle de l'entrée (figure 4.31d) : la variable IN repasse à faux.

Le modèle B est alors bloqué en *LC* et ne peut plus franchir la transition urgente.



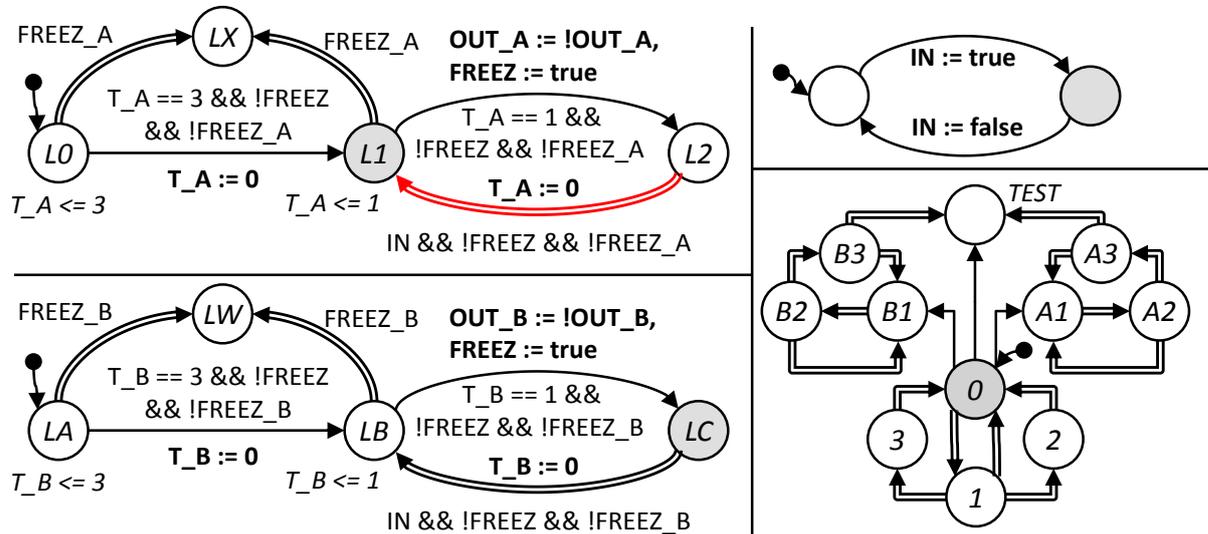
(a) Situation initiale.



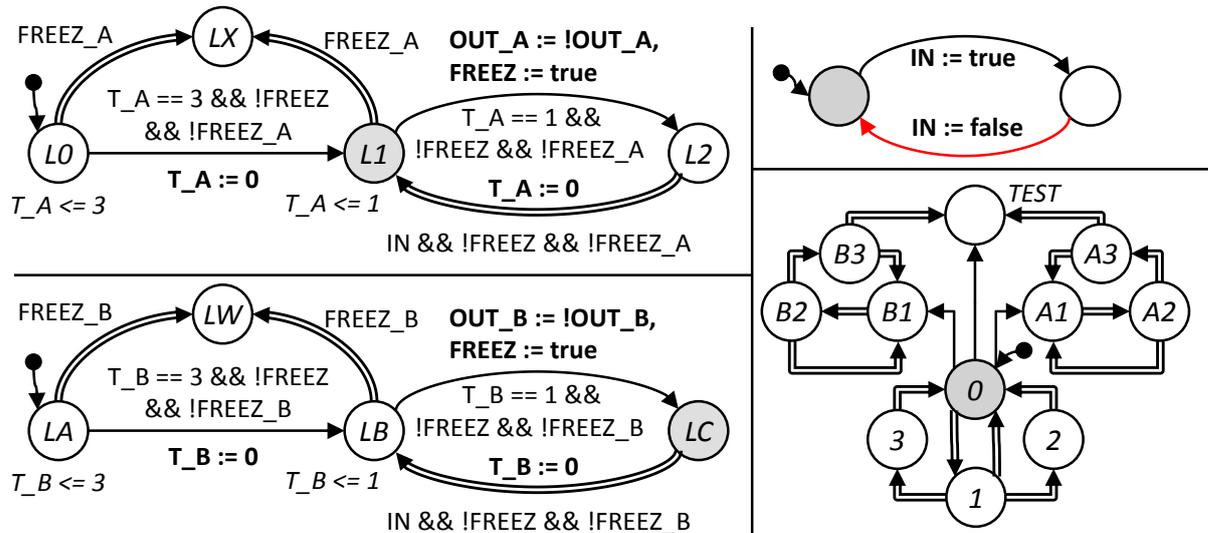
(b) Évolutions des modèles A et B et de l'AOS.

FIGURE 4.31 – Suite d'évolutions mettant en avant la concurrence entre les modèles des entrées et ceux testés (1).

considérant que le modèle de IN n'évolue plus, le modèle A peut évoluer et faire progresser son indice de trace en assignant OUT\_A à faux. On obtient des traces de sortie différentes pour les deux modèles alors qu'ils sont identiques !



(c) Évolution du modèle A (choix).



(d) Évolution du modèle de l'entrée IN (choix) : blocage du modèle B.

FIGURE 4.31 – Suite d'évolutions mettant en avant la concurrence entre les modèles des entrées et ceux testés (2).

### 4.3.3.3 Modifications dues aux modèles des entrées

Les entrées doivent pouvoir varier librement, cependant il faut être certain que les changements ne privilégient pas un modèle par rapport à l'autre. Pour ce faire, nous décidons de temporiser les évolutions des entrées en obligeant au moins une évolution du temps physique entre des variations des variables d'entrée. Ceci revient à considérer que les variables d'entrée changent quand elles le souhaitent mais doivent consommer du temps physique pour cela.

Pour répondre à ces besoins, nous proposons d'utiliser un automate EXT\_M contrôlant les évolutions des modèles des entrées et de modifier les modèles des entrées, les modèles à comparer et le modèle de l'AOS :

**EXT\_M** est l'automate contrôlant les évolutions des modèles des entrées. Il est composé de deux localités représentant les modèles des entrées bloqués (localité initiale  $B$ ) ou autorisés à évoluer (localité  $E$ ). Il assigne la variable booléenne EVOL\_IN qui, mise à vrai, autorise les modèles des entrées à évoluer. Afin de garantir une évolution du temps entre deux modifications de la trace d'entrée, une horloge nommée T\_ext est ajoutée. Elle est utilisée pour temporiser les changements des variables d'entrée : une valeur strictement supérieure à 0 de cette horloge est obligatoire pour permettre un changement de la trace d'entrée (évolution des modèles des entrées). Afin de garantir que les modèles n'évoluent pas, EXT\_M va utiliser la variable FREEZ pour geler les évolutions des modèles comparés. Le changement de la trace d'entrée ne doit pas intervenir alors que l'AOS réagit à un changement de variable de sortie : pour ce faire, la variable FREEZ doit être à faux pour que les modèles des entrées puissent évoluer. Le modèle EXT\_M est détaillé dans la figure 4.32a.

**Les modèles des entrées** (figure 4.32b pour une entrée booléenne et figure 4.32c pour une entrée entière) sont modifiés pour évoluer uniquement lorsque EVOL\_IN est vraie.

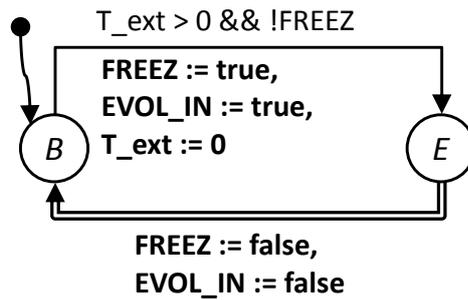
**Les modèles comparés** sont modifiés pour garantir l'équité : toutes les transitions utilisant dans leur garde au moins une variable d'entrée remettent à zéro l'horloge T\_ext afin d'interdire une nouvelle évolution immédiate de la trace d'entrée.

**L'AOS** est modifié pour ne pas réagir lorsque l'automate EXT\_M assigne FREEZ à vrai. Pour ce faire, chaque transition nécessitant FREEZ à vrai pour être franchie se voit ajouter  $\&\&!EVOL\_IN$  pour ne plus être franchissable lorsque cette assignation de FREEZ correspond à un changement de trace d'entrée.

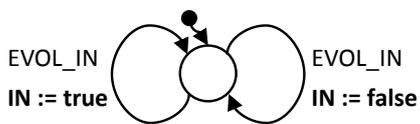
Les modèles comparés sont eux aussi modifiés avec l'ajout de la remise à zéro de l'horloge T\_ext, les figures 4.33a et 4.33b montrent les modèles A et B modifiés.

Enfin, le modèle de l'AOS subit des modifications mineures, sa version finale est représentée dans la figure 4.34.

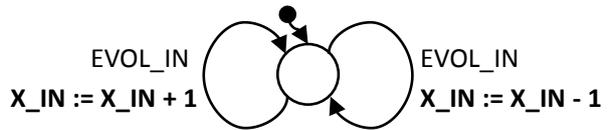
Avec ces modifications, il n'y a plus de concurrences néfastes entre les modèles des entrées et les modèles testés. La séquence d'évolutions présentée dans la figure 4.31 reprise avec les modèles modifiés permet de vérifier l'équivalence des modèles, elle est décrite dans l'annexe E.3 pour des raisons de place.



(a) Automate EXT\_M chargé de limiter les évolutions des entrées.

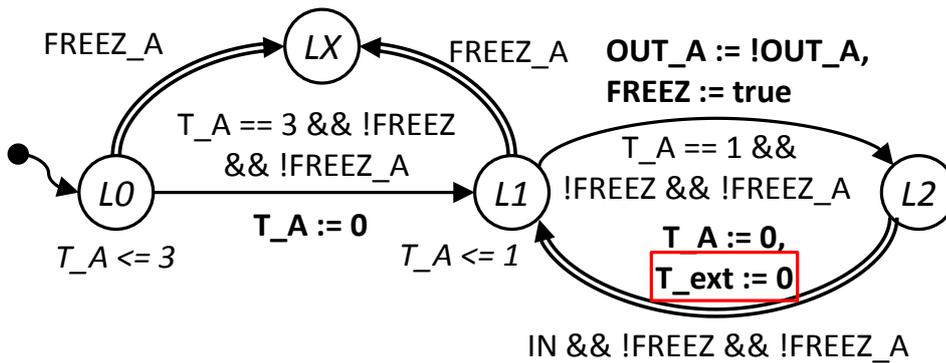


(b) Modèle modifié d'une entrée booléenne IN.

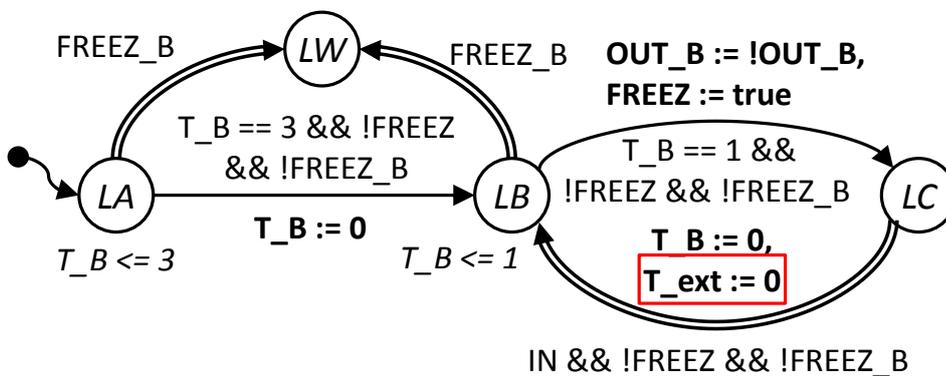


(c) Modèle modifié d'une entrée entière X\_IN.

FIGURE 4.32 – Modèles produisant la trace d'entrée.



(a) Modèle A temporisé et réagissant à la variable booléenne IN modifié.



(b) Modèle B temporisé et réagissant à la variable booléenne IN modifié.

FIGURE 4.33 – Modèles A et B modifiés pour remettre à zéro l'horloge T\_ext utilisée par l'automate EXT\_M.

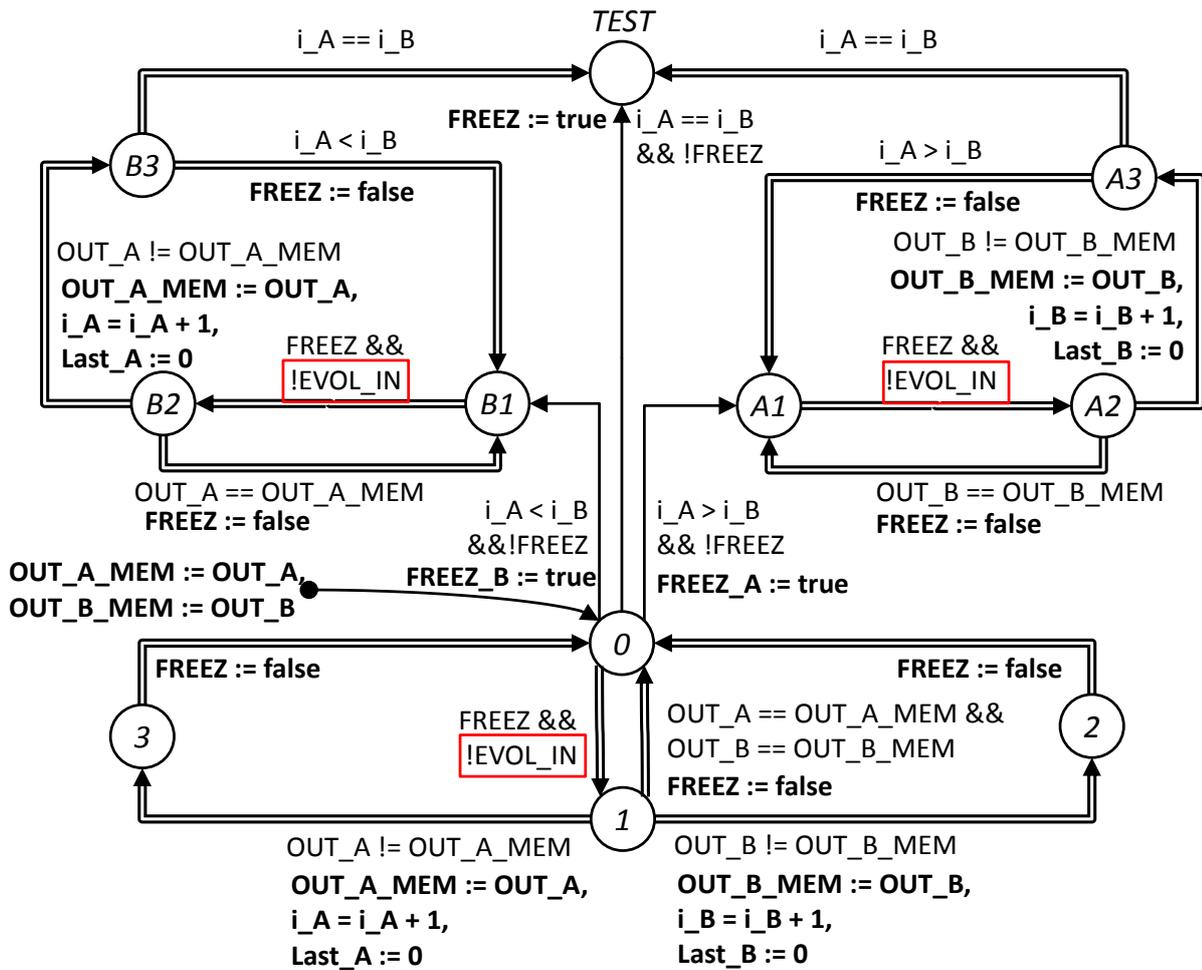


FIGURE 4.34 – Version finale de l'AOS.

## 4.4 Définition et vérification d'une équivalence de traces avec tolérance

### 4.4.1 Motivations

L'hypothèse de la section 4.1.4.3 concernant le déterminisme des modèles à comparer est très avantageuse pour la vérification d'une équivalence de traces. En effet, ainsi, il n'y a qu'une trace de sortie possible pour une trace d'entrée donnée au modèle.

Lors d'une utilisation industrielle, les modèles de partie opérative peuvent être non déterministes afin de modéliser de la concurrence ou même des imprécisions de réalisation. Le modèle du contrôleur, généralement déterministe, ne permet alors pas tout le temps de réaliser un modèle du système bouclé déterministe, ce qui exclut ces modèles de la méthode présentée précédemment.

Afin de pouvoir aider à la conception de modèles multi-échelles, et ce même pour des modèles de systèmes bouclés non déterministes, nous proposons d'utiliser une notion d'équivalence de traces *avec tolérance*.

Dans ces travaux, nous considérons principalement deux catégories de tolérances :

- Une tolérance en valeur, qui représente une différence entre les valeurs des variables produites par les deux modèles au même indice.
- Une tolérance temporelle, qui autorise un modèle à produire une trace décalée dans le temps.

Dans la suite de cette section, nous définissons ce qu'implique d'autoriser ces tolérances dans le cadre d'une équivalence de traces de deux modèles *avec tolérance*. Dans un premier temps, seule une tolérance en valeur est abordée. Dans un second temps, une tolérance temporelle seule est présentée, avec le souci important du respect ou non de l'ordre des indices de trace. Enfin, le cas d'une équivalence avec tolérance en valeur *et* temporelle est présenté. Pour finir, l'impact de ces modifications sur notre procédure de vérification de l'équivalence de traces est présenté.

### 4.4.2 Équivalence de traces avec tolérance en valeur

L'idée ici est de définir une équivalence stricte sur le plan temporel : les valeurs de l'horloge globale doivent être identiques ( $t_i$ ) tout en permettant une certaine imprécision sur les valeurs des variables.

Soit deux ATVD A et B, l'équivalence de traces avec tolérance en valeur ne peut porter que sur des variables communes donc :

- Soit  $V_e = V_e^A \cap V_e^B$  l'ensemble des variables d'entrée communes aux deux ATVD, idéalement  $V_e = V_e^A = V_e^B$ .
- Soit  $V_s = V_s^A \cap V_s^B$  l'ensemble des variables de sortie communes aux deux ATVD, idéalement  $V_s = V_s^A = V_s^B$ .

La notion de tolérance sur les valeurs d'une variable peut difficilement être appliquée à une variable booléenne qui ne peut avoir que deux valeurs : vrai ou faux. Ainsi on définit une fonction de tolérance en valeur  $\Delta V$  sur les variables entières de  $V_s \cap V_z$  telle que :

$$\forall v \in (V_s \cap V_z), \Delta V(v) = [\Delta V_{\text{inf}}^v; \Delta V_{\text{sup}}^v] \text{ avec } \Delta V_{\text{inf}}^v, \Delta V_{\text{sup}}^v \in \mathbb{Z} \text{ et } \Delta V_{\text{inf}}^v \leq \Delta V_{\text{sup}}^v \quad (4.12)$$

On définit alors l'équivalence de traces avec tolérance  $\Delta V$  de l'ATVD B par rapport à l'ATVD A comme :

$$\begin{aligned} B \cong_{\Delta V} A \Rightarrow \\ \forall Tra(V_e), \forall i \in \mathbb{N}, (\tilde{V}_i^A, t_i^A) \in Tra^A(V_s), (\tilde{V}_i^B, t_i^B) \in Tra^B(V_s) : \\ t_i^A = t_i^B \text{ et} \\ \forall v \in (V_s \cap V_z), \tilde{v}_i^A - \tilde{v}_i^B \in \Delta V(v) \end{aligned} \quad (4.13)$$

avec  $\tilde{v}_i^A$  la valeur de la variable  $v$  du modèle  $A$  à l'indice de trace  $i$  et  $\tilde{v}_i^B$  la valeur de la variable  $v$  du modèle  $B$  à l'indice de trace  $i$ .

Cette définition n'est pas symétrique, contrairement à celle de l'équivalence de traces usuelle. Cependant, l'équivalence  $B \cong_{\Delta V} A$  implique nécessairement l'équivalence réciproque  $A \cong_{-\Delta V} B$  où  $-\Delta V$  correspond à la fonction :

$$\forall v \in (V_s \cap V_z), -\Delta V(v) = [-\Delta V_{\text{sup}}^v; -\Delta V_{\text{inf}}^v] \quad (4.14)$$

avec  $\Delta V_{\text{inf}}^v$  et  $\Delta V_{\text{sup}}^v$  issus de la définition de  $\Delta V$ . Par la suite, nous parlerons donc d'équivalence de traces avec tolérance en valeur  $\Delta V$  entre les ATVD A et B sans préciser l'ordre de l'équivalence.

Le choix particulier d'intervalles symétriques par rapport à 0 ( $\Delta V_{\text{sup}}^v + \Delta V_{\text{inf}}^v = 0$ ) permet le retour à une définition symétrique de cette équivalence car, dans ce cas,  $\Delta V = -\Delta V$ .

Le choix d'une fonction  $\Delta V$  particulière où  $\Delta V_{\text{sup}}^v = -\Delta V_{\text{inf}}^v = 0$  nous ramène à la définition usuelle d'une équivalence de traces d'ATVD sans tolérance en valeur.

Un exemple d'ATVD A et B équivalents en trace avec tolérance en valeur est donné dans l'annexe E.4.1 pour des raisons de place. Un exemple de modèle C non équivalent au modèle A est aussi présenté dans la même annexe. Il est accompagné d'une trace d'entrée et des traces de sortie des modèles A et C mettant en évidence cette non-équivalence.

### 4.4.3 Définition d'une équivalence avec tolérance temporelle

La définition d'une équivalence avec tolérance temporelle pose quelques problèmes supplémentaires que nous détaillons dans cette section.

#### 4.4.3.1 Aspect chronologique et intervalle de tolérance

Lorsque l'on imagine une tolérance temporelle sur une trace, on peut distinguer deux cas de figure :

- Soit l'ordre des événements est conservé, et c'est donc du *retard* ou de *l'avance* que l'on tolère sur la trace.
- Soit l'ordre des événements n'est pas conservé, et c'est sur chaque événement que l'on souhaite une tolérance.

Ces deux approches donnent lieu à deux définitions différentes de l'équivalence avec tolérance temporelle *avec* et *sans* conservation de la chronologie.

#### 4.4.3.2 Équivalence avec tolérance temporelle et conservation de la chronologie

Dans ce cas de figure, l'indice  $i$  des deux traces doit faire référence aux mêmes événements, il peut donc toujours être utilisé pour faire la comparaison.

Soit deux ATVD A et B, l'équivalence de traces avec tolérance temporelle et conservation de la chronologie ne peut porter que sur des variables communes, tout comme les équivalences précédentes, on définit alors :

- $V_e = V_e^A \cap V_e^B$  l'ensemble des variables d'entrée communes aux deux ATVD, idéalement  $V_e = V_e^A = V_e^B$ .
- $V_s = V_s^A \cap V_s^B$  l'ensemble des variables de sortie communes aux deux ATVD, idéalement  $V_s = V_s^A = V_s^B$ .

On pose la tolérance temporelle  $\Delta T = [\Delta T_{inf}; \Delta T_{sup}]$  avec  $\Delta T_{inf}, \Delta T_{sup} \in \mathbb{Z}$  et  $\Delta T_{inf} \leq \Delta T_{sup}$ .

On définit alors l'équivalence de traces avec tolérance temporelle  $\Delta T$  de l'ATVD B par rapport à l'ATVD A comme :

- $B \cong_{\Delta T} A \Rightarrow \forall Tra(V_e), \forall i \in \mathbb{N}, (\tilde{V}_i^A, t_i^A) \in Tra^A(V_s)$  et  $(\tilde{V}_i^B, t_i^B) \in Tra^B(V_s)$  vérifient :
- $t_i^A - t_i^B \in \Delta T$
  - $\forall v \in V_s, \tilde{V}_i^A = \tilde{V}_i^B$

À nouveau, les remarques faites pour l'équivalence avec tolérance en valeur s'appliquent ici :

- La relation d'équivalence n'est pas symétrique, mais la relation réciproque  $A \cong_{-\Delta T} B$  se conçoit facilement avec  $-\Delta T = [-\Delta T_{sup}; -\Delta T_{inf}]$ .
- La relation devient symétrique pour les cas où la tolérance temporelle est symétrique par rapport à 0. Si  $\Delta T_{sup} + \Delta T_{inf} = 0$ , alors  $B \cong_{\Delta T} A \equiv A \cong_{-\Delta T} B$ .
- Le cas particulier avec un intervalle temporel nul ( $\Delta T_{sup} = \Delta T_{inf} = 0$ ) permet de revenir à une équivalence de traces classique.

Pour des raisons de place, les exemples de modèles équivalents et non équivalents (accompagnés dans ce cas d'une trace mettant en avant la non-équivalence) sont présentés dans l'annexe E.4.2.

#### 4.4.3.3 Équivalence avec tolérance temporelle sans conservation de la chronologie

Dans ce cas de figure, l'indice  $i$  des deux traces ne permet plus de faire une comparaison directe entre les deux traces car l'ordre des événements des deux traces peut être différent.

Soit deux ATVD A et B, l'équivalence de traces avec tolérance temporelle sans conservation de la chronologie ne peut porter que sur des variables communes, tout comme les équivalences précédentes, on définit alors :

- $V_e = V_e^A \cap V_e^B$  l'ensemble des variables d'entrée communes aux deux ATVD, idéalement  $V_e = V_e^A = V_e^B$ .
- $V_s = V_s^A \cap V_s^B$  l'ensemble des variables de sortie communes aux deux ATVD, idéalement  $V_s = V_s^A = V_s^B$ .

On définit alors la tolérance temporelle  $\Delta T = [\Delta T_{inf}; \Delta T_{sup}]$  avec  $\Delta T_{inf}, \Delta T_{sup} \in \mathbb{Z}$  et  $\Delta T_{inf} < \Delta T_{sup}$ .

$B \cong_{\Delta T} A \Rightarrow \forall Tra(V_e), \forall i \in \mathbb{N}, (\tilde{V}_i^A, t_i^A) \in Tra^A(V_s)$  alors  $\exists j \in \mathbb{N}$  tel que  $(\tilde{V}_j^B, t_j^B) \in Tra^B(V_s)$  vérifie :

- $t_i^A - t_j^B \in \Delta T$
- $\forall v \in V_s, \tilde{V}_i^A = \tilde{V}_j^B$

Cette définition n'impose aucune conservation de l'ordre et, si les tolérances temporelles le permettent, risque même d'utiliser plusieurs fois un même état de la trace B comme équivalent à des états de la trace A.

Le non-respect de l'ordre d'apparition des événements nous semble une tolérance très grande et relativement peu appropriée dans le cadre de la simulation de systèmes industriels manufacturiers (possibilité de réponse à une communication entre contrôleurs avant même l'envoi de celle-ci...).

Cette définition ne sera pas utilisée dans le reste des travaux, du fait de sa difficulté de mise en œuvre et de l'incohérence possible laissée au modèle détaillé.

De plus, le cas particulier d'une tolérance nulle ne renvoie pas à la définition usuelle de l'équivalence de traces.

#### 4.4.4 Proposition d'une équivalence de traces avec tolérance temporelle et en valeur

La proposition faite d'une équivalence avec tolérance temporelle et en valeur reprend les définitions proposées dans les sections précédentes. On s'attachera à l'ordre d'apparition des événements, donc on choisira la définition de l'équivalence de traces avec tolérance temporelle qui conserve la chronologie.

##### 4.4.4.1 Définition d'une équivalence de traces avec tolérance temporelle et en valeur

Soit deux ATVD A et B, l'équivalence de traces avec tolérance temporelle et en valeur ne peut porter que sur des variables communes, tout comme les équivalences précédentes, on définit alors :

- $V_e = V_e^A \cap V_e^B$  l'ensemble des variables d'entrée communes aux deux ATVD, idéalement  $V_e = V_e^A = V_e^B$ .
- $V_s = V_s^A \cap V_s^B$  l'ensemble des variables de sortie communes aux deux ATVD, idéalement  $V_s = V_s^A = V_s^B$ .

On définit alors les fonctions de tolérance :

- La tolérance temporelle  $\Delta T = [\Delta T_{inf}; \Delta T_{sup}]$  avec  $\Delta T_{inf}, \Delta T_{sup} \in \mathbb{Z}$  et  $\Delta T_{inf} \leq \Delta T_{sup}$ .
- La fonction de tolérance en valeur  $\Delta V$  telle que :  $\forall v \in (V_s \cup V_z), \Delta V(v) = [\Delta V_{inf}^v; \Delta V_{sup}^v]$  avec  $\Delta V_{inf}^v, \Delta V_{sup}^v \in \mathbb{Z}$  et  $\Delta V_{inf}^v \leq \Delta V_{sup}^v$ .

L'équivalence de traces avec tolérance temporelle et en valeur de l'ATVD B par rapport à l'ATVD A est définie comme :

$B \cong_{(\Delta T, \Delta V)} A \Rightarrow \forall Tra(V_e), \forall i \in \mathbb{N}, (\tilde{V}_i^A, t_i^A) \in Tra^A(V_s)$  et  $(\tilde{V}_i^B, t_i^B) \in Tra^B(V_s)$  vérifient :

- $t_i^A - t_i^B \in \Delta T$
- $\forall v \in (V_s \cup V_z), \tilde{V}_i^A - \tilde{V}_i^B \in \Delta V(v)$

Cette définition profite des particularités communes aux définitions précédentes, et tout particulièrement :

- La relation d'équivalence n'est pas symétrique, mais la relation réciproque  $A \cong_{(-\Delta T, -\Delta V)} B$  se conçoit facilement avec  $-\Delta T = [-\Delta T_{sup}; -\Delta T_{inf}]$  et  $\forall v \in V_s, -\Delta V(v) = [-\Delta V_{sup}^v; -\Delta V_{inf}^v]$ .
- La relation devient symétrique pour les cas où les tolérances sont symétriques. Si  $\Delta T_{sup} + \Delta T_{inf} = 0$  et  $\forall v \in V_s, \Delta V_{sup}^v + \Delta V_{inf}^v = 0$ , alors  $B \cong_{(\Delta T, \Delta V)} A \equiv A \cong_{(-\Delta T, -\Delta V)} B$ .
- Le cas particulier avec des intervalles de tolérance nuls ( $\Delta T_{sup} = \Delta T_{inf} = 0$  et  $\forall v \in V_s, \Delta V_{sup}^v = \Delta V_{inf}^v = 0$ ) permet de revenir à une équivalence de traces usuelle.

#### 4.4.4.2 Tolérances absolues et cycliques

Toutes les tolérances précédentes ont été définies en absolu sur la trace. Cependant, il peut être utile de définir des tolérances cycliques pour des systèmes à cycles, très présents dans le milieu manufacturier. Ceci revient à autoriser un certain décalage temporel et en valeur à chaque cycle des modèles. Pour implémenter ce changement, il convient de définir une variable particulière  $v_{cycle}$  qui est mise à vrai lors du départ d'un cycle et remise à zéro durant le cycle. À chaque fois que cette variable passe à vrai, on incrémente un compteur de cycles noté  $N_{cycle}$  qui compte le nombre de cycles (début à 1). Les tolérances sont alors définies non plus de manière absolue mais au travers de formules utilisant  $N_{cycle}$ , comme par exemple une tolérance temporelle de la forme  $\Delta T = [-2N_{cycle}, 2N_{cycle}]$  qui correspond à une tolérance symétrique de deux unités de temps par cycle.

### 4.4.5 Vérification d'une équivalence de traces avec tolérance

#### 4.4.5.1 Prise en compte de la tolérance en valeur

Sous couvert d'un test à des indices de trace identiques, la tolérance en valeur n'est pas difficile à prendre en compte.

Il convient de modifier la propriété formelle de manière à prendre en compte la tolérance en valeur associée à la variable testée.

Soit la variable  $v \in V_s$  et la fonction de tolérance  $\Delta V(v) = [\Delta V_{\text{inf}}^v, \Delta V_{\text{sup}}^v]$  associée à l'équivalence  $B \cong_{\Delta V} A$ .

La trace doit vérifier :  $\forall \text{Tra}(V_e), \forall i \in \mathbb{N}, (\tilde{V}_i^A, t_i^A) \in \text{Tra}^A(V_s)$  et  $(\tilde{V}_i^B, t_i^B) \in \text{Tra}^B(V_s)$  vérifient  $\forall v \in V_s, \tilde{V}_i^A - \tilde{V}_i^B \in \Delta V(v)$ , ce qui se traduit au niveau de la propriété associée au test d'équivalence de la variable  $v$  par la modification de la condition booléenne à vérifier pour garantir l'équivalence en valeur, on ne doit donc *jamais* avoir :

$$v^a - v^b > \Delta V_{\text{sup}}^v \text{ ou } v^a - v^b < \Delta V_{\text{inf}}^v,$$

avec  $v^a$  et  $v^b$  la variable  $v$  respective des modèles A et B.

#### 4.4.5.2 Prise en compte de la tolérance temporelle

La tolérance temporelle nécessite quant à elle de mesurer le temps écoulé depuis le départ, ce qui supposerait de mémoriser la valeur des horloges, ce qui n'est pas possible avec le formalisme utilisé et peu souhaitable en terme de mémoire utilisée comme décrit dans la section 4.3.2.4.

Les horloges Last\_A et Last\_B précédemment ajoutées pour vérifier l'équivalence temporelle sans tolérance vont être ici mises à contribution.

Pour rappel, l'horloge Last\_A (respectivement Last\_B) est utilisée pour mesurer le temps écoulé depuis la dernière évolution de la trace du modèle A (respectivement B).

En reprenant le raisonnement de la section 4.3.2.4, on pose à nouveau  $i_A = i_B = \mu$  l'indice de trace de comparaison. On a  $(\tilde{V}_\mu^A, t_\mu^A)$  la valeur de la trace du modèle A à cet indice et  $(\tilde{V}_\mu^B, t_\mu^B)$  celle du modèle B. On ne s'intéresse ici qu'au temps,  $t$  se réfère donc au temps global. On a, au moment où l'on fait la comparaison :

$$\begin{cases} t_{\text{courant}} = t_\mu^A + \text{Last}_A \\ t_{\text{courant}} = t_\mu^B + \text{Last}_B \end{cases}$$

Soit :  $t_\mu^A - t_\mu^B = \text{Last}_B - \text{Last}_A$ .

Pour vérifier l'équivalence avec tolérance temporelle, il faut donc que Last\_B - Last\_A soit compris dans l'intervalle de tolérance. Il faut donc que la situation suivante n'apparaisse *jamais* lors de la comparaison :

$$\text{Last}_B - \text{Last}_A < \Delta T_{\text{inf}} \text{ ou } \text{Last}_B - \text{Last}_A > \Delta T_{\text{sup}}$$

#### 4.4.5.3 Version de l'AOS et des propriétés associées pour la preuve d'équivalence avec tolérances

En prenant en compte les modifications présentées dans les sections précédentes, l'AOS reste inchangé mais la propriété exemple de la figure 4.27b devient celle présentée dans la figure 4.35. On y retrouve l'atteignabilité de la localité de test de l'AOS ainsi que la tolérance temporelle commune à toutes les variables et la tolérance en valeur de la variable considérée.

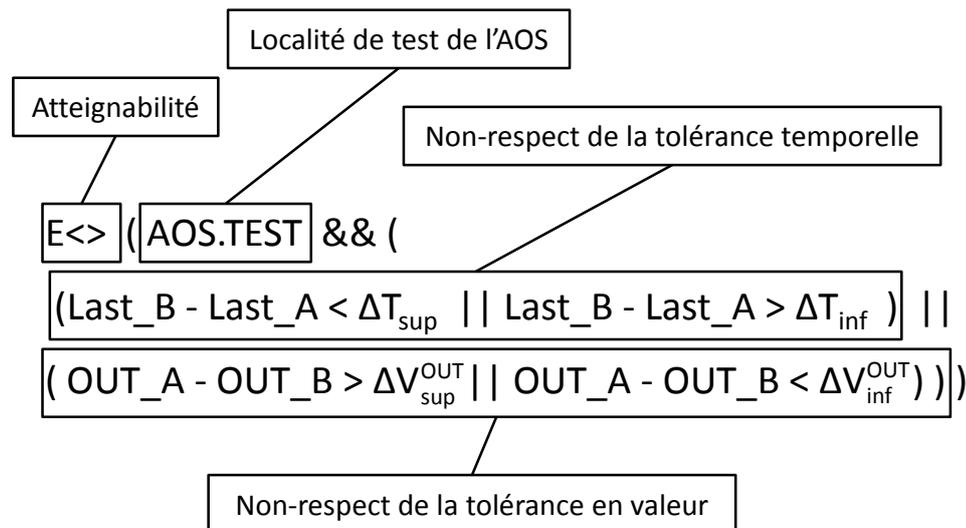


FIGURE 4.35 – Exemple de propriété pour l'équivalence avec tolérance temporelle et en valeur.

## 4.5 Conclusion

Ce chapitre définit une méthode permettant de prouver l'équivalence entre les modèles détaillé  $M_{P_i}^D$  et abstrait  $M_{P_i}^A$  d'un même poste en terme d'entrées/sorties.

En effet, dans un premier temps, la définition d'une équivalence de traces entre deux ATVD a été décrite puis la technique des automates observateurs, couplés à un model-checker, est retenue pour vérifier celle-ci.

Un Automate Observateur-Séquenceur est ajouté pour garantir un résultat correct pour la vérification de modèles temporisés. Un automate EXT\_M est ensuite également ajouté pour garantir l'équité entre modèles testés lors de l'utilisation de modèles des entrées.

Enfin, l'hypothèse de déterminisme des modèles à comparer étant très restrictive vis-à-vis des modèles de systèmes bouclés réels, une définition de l'équivalence de traces avec tolérances en valeur et temporelle est proposée. Les modifications sur la méthode de vérification de l'équivalence sont alors détaillées.



# Conclusions et perspectives

Les contributions proposées dans ce mémoire permettent de modéliser les systèmes bouclés et temporisés de manière plus réaliste. En effet, le chapitre 3 permet de modéliser :

- Une partie opérative construite à partir de composants génériques modélisés dans une bibliothèque. Les concurrences indésirables entre les modèles de la partie opérative sont supprimées en utilisant le mécanisme d’urgence des automates temporisés à variables discrètes.
- Un contrôleur exécutant un code issu d’une spécification Grafcet, traduite sous forme d’équations algébriques et accompagnée de modèles de temporisations.
- Un système bouclé temporisé complet, en utilisant les résultats des deux points précédents mais en :
  - Traitant la phase de lecture/écriture des entrées (utilisation de la variable HOLD),
  - Supprimant les cas de blocage des modèles de la partie opérative (utilisation des variables EVOL\_PL et END\_TI),
  - Garantissant la réactivité et la recherche de stabilité du modèle du contrôleur (ajout de la variable stable).

Ensuite, l’utilisation de modèles de grande dimension étant un frein important lors des analyses, une contribution à la conception de modèles multi-échelles (détaillé et abstrait) de systèmes bouclés temporisés a été faite dans le chapitre 4. Dans ce chapitre nous avons présenté :

- La définition et la vérification d’une équivalence en trace sur des modèles d’automates temporisés à variables discrètes en :
  - Supprimant les concurrences entre les modèles à tester et l’automate observateur (utilisation d’un automate observateur-séquenceur),
  - Mesurant le décalage temporel des traces à comparer (utilisation des horloges Last\_A et Last\_B),
  - Traitant les concurrences avec les modèles des entrées (utilisation d’un automate de contrôle des modèles des entrées).
- La définition d’une équivalence en trace avec tolérance, nécessaire en raison des difficultés de conception d’un modèle abstrait rencontrées pour certains systèmes bouclés, et sa vérification, par une modification de la méthode proposée précédemment.

À l’issue de ces travaux, nos contributions ouvrent plusieurs perspectives :

- À court terme, le développement d’une bibliothèque de composants plus fournie et utilisable dans le milieu industriel apparaît comme une suite logique,
- À plus long terme, l’ajout de composants avec défaillances (blocage de vérins, dé-

faillance de capteurs) dans la bibliothèque de composants et l'étude de leur impact sur les évolutions du modèle final du système bouclé est une piste intéressante.

De même, la recherche d'une méthode permettant d'obtenir un modèle abstrait directement et automatiquement à partir du modèle détaillé d'un système bouclé permettrait de garantir l'équivalence, *a priori*, entre les deux modèles et constituerait une avancée importante. La capacité à produire un modèle abstrait automatiquement pour tout système bouclé doit cependant être modérée. En effet la nécessité d'utiliser une équivalence avec tolérance, présentée dans le chapitre 4, pour prouver l'équivalence entre modèles détaillé et abstrait pour certains systèmes bouclés nous incite à penser que l'abstraction automatique devra peut-être être restreinte à certaines catégories de systèmes bouclés.

Enfin, lorsque les outils le permettront, le passage à des formalismes hybrides pour la conception des modèles détaillés des systèmes bouclés permettra d'amplifier le réalisme des modèles et la finesse des analyses.

# Bibliographie

- R. ALUR : A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
- R. ALUR, C. COURCOUBETIS et D. DILL : Model-checking for real-time systems. *In Proceedings of 5th Annual IEEE Symposium on Logic in Computer Science, LICS'90*, p. 414–425, 1990.
- N. BAUER, S. ENGELL, R. HUUCK, S. LOHMANN, B. LUKOSCHUS, M. REMELHE et O. STURBERG : Verification of PLC programs given as sequential function charts. *In LNCS*, vol. 3147, p. 517–540, 2004.
- G. BEHRMANN, R. DAVID et K. LARSEN : A tutorial on UPPAAL. *LNCS, Formal Methods for the Design of Real-Time Systems*, 3185:200–236, 2004.
- J. BENGTSSON, K. LARSEN, F. LARSSON, P. PETTERSSON et W. YI : UPPAAL – a tool suite for automatic verification of real-time systems. *LNCS*, 1066:232–243, 1996.
- B. BÉRARD, M. BIDOIT, A. FINKEL, F. LAROUSSINIE, A. PETIT, L. PETRUCCI et P. SCHNOEBELEN : *Systems and Software Verification. Model-Checking Techniques and Tools*. Springer, 2001.
- S. BEREZIN : *Model Checking and Theorem Proving : a Unified Framework*. Thèse de doctorat, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, 2002.
- J. C. CAMPOS et J. MACHADO : Pattern-based analysis of automated production systems. *In Proc. of the 13th IFAC Symposium on Information Control Problems in Manufacturing, INCOM'09*, 2009.
- B. DENIS, J.-J. LESAGE et Z. JUAREZ OROZCO : Performance verification of discrete event systems using hybrid model-checking. *In Proceedings of 2nd IFAC Conference on Analysis and Design of Hybrid Systems, ADHS'06*, p. 365–370, Alghero, Italie, juin 2006.
- G. FREY et L. LITZ : Formal methods in PLC programming. *In Proc. of IEEE conference on Systems, Man and Cybernetics*, p. 2431–2436, Nashville, USA, October 2000.
- V. GOURCUFF, O. de SMET et J.-M. FAURE : Improving large-sized PLC programs verification using abstractions. *In IFAC World congress*, 2008.

- H.-M. HANISCH, J. THIEME et O. LUDER, A. ajnd Wienhold : Modeling of PLC behavior by means of timed net condition/event systems. *In Proc. of the 6th International Conference on Emerging Technologies and Factory Automation Proceedings, ETFA'97*, p. 391 – 396, 1997.
- T. A. HENZINGER, Z. MANNA et A. PNUELI : Timed transition systems. *In LNCS, Real Time : Theory in Practice*, vol. 600, p. 226 – 251, 1992a.
- T. A. HENZINGER, X. NICOLLIN, J. SIFAKIS et S. YOVINE : Symbolic model checking for real-time systems. *Information and Computation*, 111:394–406, 1992b.
- IEC 60848 : *International Electrotechnical Committee, Grafcet specification language for sequential function charts*, 2002.
- IEC 61131 : *International Electrotechnical Committee, Programmable controlles - Programming languages*, 1999.
- A. JANOWSKA et P. JANOWSKI : Slicing of timed automata with discrete data. *Fundamenta Informaticae*, 72(1-3):181–195, 2006.
- Z. JUAREZ OROZCO : *Vérification de propriétés quantitatives des systèmes logiques par model-checking hybride*. Thèse de doctorat, École Normale Supérieure de Cachan, 2008.
- S. LAMPERIERE-COUFFIN et J.-J. LESAGE : Formal verification of the sequential part of plc programs. *In Proc. of the 5th Workshop on Discrete Event Systems, WODES'2000*, p. 247–254, 2000.
- K. LARSEN, P. PETTERSSON et W. YI : UPPAAL in a nutshell. *Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, 1997.
- M. LINDAHL, P. PETTERSSON et W. YI : Formal design and analysis of a gear controller : an industrial case study using uppaal. *LNCS, Proceedings of 4th International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, 1384:281–297, 1998.
- J. MACHADO : *Influence de la prise en compte d'un modèle de processus en vérification formelle des Systèmes à Événements Discrets*. Thèse de doctorat, École Normale Supérieure de Cachan & Université du Minho (Portugal), 2006.
- J. MACHADO, B. DENIS et J.-J. LESAGE : A generic approach to build plant models for DES verification purposes,. *In Proc. of 8th International Workshop on Discrete Event Systems (WODES)*, Ann Arbor (Michigan), USA, July 2006a.
- J. MACHADO, B. DENIS, J.-J. LESAGE, J.-M. FAURE. et J. F. D. SILVA : Logic controllers dependability verification using a plant model. *In Proc. of 3rd IFAC Workshop on Discrete-Event System Design (DESDes)*, p. 37–42, Rydzyna, Poland, September 2006b.
- J. MACHADO, B. DENIS et J.-J. LESAGE : Formal verification of industrial controllers : with or without a plant model? *In Proc. of 7th Portuguese Conference on Automatic Control, CONTROLO'06*, Lisboa Portugal, 2006.

- 
- A. MADER et H. WUPPER : Timed automaton models for simple programmable logic controllers. *In Proc. of 11th Euromicro Conference on Real-Time Systems*, p. 114–122, York, England, June 1999.
- M. PERIN et J.-M. FAURE : Building meaningful timed plant models for verification purposes. *In Proceedings of the 13th IFAC Symposium on information control problems in manufacturing, INCOM'09*, p. 970–975, Moscow, Russia, 2009.
- M. PERIN et J.-M. FAURE : Analyse prévisionnelle des fautes des systèmes embarqués discrets par vérification model-based. *In Actes de la Conférence Internationale Francophone d'Automatique*, p. CDRom paper n381, Bucarest, Roumanie, sept. 2008.
- M. PERIN et J.-M. FAURE : Building meaningful timed models of closed-loop DES for verification purposes. *Control Engineering Practice*, 2012.
- P. PETTERSSON : *Modelling and Verification of Real-Time Systems Using Timed Automata : Theory and Practice*. Thèse de doctorat, Department of Computer Systems, Uppsala University, 1999.
- A. PHILIPPOT, M. SAYED MOUCHAWEH et V. CARRÉ-MÉNÉTRIER : Modelling of a discrete manufacturing system by parts of plant. *In 13th IFAC Symposium on Information Control Problem in Manufacturing*, Moscow, jun 2009.
- J. PROVOST, J.-M. ROUSSEL et J.-M. FAURE : Translating grafcet specifications into mealy machines for conformance test purposes. *Control Engineering Practice*, 19(9):947 – 957, 2011.
- A. RABINOVICH : Complexity of equivalence problems for concurrent systems of finite agents. *Information and computation*, 139:111–129, 1997.
- B. ROHÉE, B. RIERA, V. CARRÉ-MÉNÉTRIER et J.-M. ROUSSEL : A methodology to design and check a plant model. *In Proceedings of 3rd IFAC Workshop on Discrete-Event System Design (DESDes'06)*, p. 246–250, Rydzyna, Pologne, juin 2006.
- O. ROSSI et P. SCHNOEBELEN : Formal modeling of timed function blocks for the automatic verification of ladder diagram programs. *In Proc. of the 4th International Conference Automation of Mixed Processes (ADPM'00)*, p. pp. 177–182, 2000.
- J.-M. ROUSSEL et J.-J. LESAGE : Validation and verification of grafcets using state machine. *In Proc. of IMACS-IEEE "CESA '96" IMACS-IEEE "CESA '96" - Lille, France*, p. pp. 758–764, 07 1996.
- S. RUEL : *Évaluation des bornes des performances temporelles des Architectures d'Automatisation en Réseau par preuves itératives de propriétés logiques*. Thèse de doctorat, École Doctorale Sciences Pratiques, ENS Cachan, F 94230 CACHAN, 2009.
- O. STURBERG et S. LOHMANN : Analysis of logic controllers by transformation of sfc into timed automata. *In Proc of the 44th IEEE Conference on Decision and Control, and the European Control Conference*, p. 7720–7725, 2005.

R. van GLABBEEK : The linear time-branching time spectrum i - the semantics of concrete, sequential processes. *In Handbook of Process Algebra, chapter 1*, p. 3–99, 2001.

## Annexe A

# Formalisme d'analyse : les Réseaux d'Automates Temporisés d'UPPAAL (RATU)

Ce formalisme, présenté dans divers travaux (Bengtsson *et al.* (1996), Larsen *et al.* (1997), Lindahl *et al.* (1998), Pettersson (1999), Behrmann *et al.* (2004)), est actuellement très utilisé au travers de la suite logicielle UPPAAL<sup>5</sup>.

Ce formalisme étant de la même famille que les ATVD, les notations déjà utilisées sont reprises au maximum, mais les éléments distinctifs ont logiquement des dénominations différentes.

## A.1 Variables, expressions et contraintes

Soit  $V = V_b \cup V_z$  l'ensemble fini des variables discrètes associées à un automate temporisé du réseau d'automates temporisés d'UPPAAL (RATU)  $A^{UPPAAL}$ , composé de l'ensemble  $V_b$  des variables booléennes à valeurs dans  $\{true, false\}$  et de l'ensemble  $V_z$  des variables entières à valeurs dans  $\mathbb{Z}$ .

Les définitions *des expressions arithmétiques* ( $Expr(V_z)$ ) et *booléennes* ( $B(V)$ ) restent *inchangées* tout comme *l'ensemble des actions* ( $A(V)$ ) ainsi que *l'ensemble des horloges* ( $C$ ).

*La principale différence* vient du fait qu'UPPAAL *intègre les contraintes temporelles* ( $T(C)$ ) *directement dans les gardes* et qu'il autorise *l'utilisation de variables dans les invariants*.

Il faut donc définir un ensemble des relations intégrant les variables et les horloges  $D(V, C)$  tel que :

$$\forall d \in D(V, C), d ::= true \mid b \mid t \mid d \wedge d \mid d \vee d \mid \neg d \quad (\text{A.1})$$

avec  $b \in B(V)$  et  $t \in T(C)$ .

Étant donné que le formalisme sous-jacent d'UPPAAL (réseaux d'automates temporisés - RATU) est particulièrement tourné vers une utilisation en réseau de plusieurs automates, nous ne détaillerons pas la sémantique d'un automate seul.

En effet, ce dernier est proche d'un ATVD, il contient deux possibilités d'évolutions :

- Une évolution des horloges, sans notion d'urgence, à l'image de la sémantique présentée en 2.5.
- Une évolution de la localité active, et éventuellement des valeurs des variables et horloges, au travers du tir d'une transition, à l'image de la sémantique présentée en 2.6.

## A.2 Canaux de communication

UPPAAL utilise deux mécanismes de communication entre les automates d'un réseau : les variables partagées, comme pour les ATVD, ainsi que les canaux de communication. Cette dernière possibilité s'apparente à des étiquettes permettant le tir synchrone de transitions.

---

5. <http://www.uppaal.org>

On définit alors un ensemble  $X$  de canaux de communication, et un ensemble d'actions  $\Sigma(X)$  tel que :

$$\forall \sigma \in \Sigma(X), \sigma ::= \chi! \mid \chi? \mid \tau \quad (\text{A.2})$$

avec :

- $\chi \in X$  un canal de communication.
- $\chi!$  l'action d'émission sur le canal  $\chi$ .
- $\chi?$  l'action d'écoute sur le canal  $\chi$ .
- $\tau$  l'action nulle de communication. Une transition avec comme action de communication  $\tau$  n'est liée à aucun canal de communication.

Dans le formalisme des réseaux d'automates temporisés d'UPPAAL (RATU), chaque transition ne peut avoir qu'une action de communication  $\sigma$ , ce qui signifie que la transition ne peut être liée qu'à un seul canal de communication (voire aucun dans le cas de l'action  $\tau$ ).

La sémantique associée à ces canaux de communication est simple : de manière informelle, pour qu'une transition avec une action de communication puisse être tirée, il faut qu'elle soit franchissable au sens de la garde et des invariants, et qu'une transition de l'autre type (émettrice pour réceptrice, et vice-versa) soit disponible et aussi franchissable. Alors les deux transitions sont tirées simultanément (une transition émettrice et une réceptrice sur un même canal de communication).

Une définition formelle de cette sémantique est donnée dans la section A.5.

Une sémantique de synchronisation multiple est présente dans la thèse de [Pettersson \(1999\)](#). Cette synchronisation de type *broadcast* permet le tir de plusieurs transitions avec action d'écoute lorsqu'une transition avec action d'émission est tirée. Cette sémantique n'est pas complètement formalisée à ce jour et ne sera donc pas utilisée dans ces travaux.

## A.3 Sémantique d'un RATU sans urgence

Un réseau d'automates temporisés d'UPPAAL est un ensemble d'automates  $A_i^{\text{UPPAAL}} = (L_i, l_i^0, V, \tilde{V}_0, C, X, F_i, J_i)$  avec  $i \in \mathbb{N}^*$ . Les ensembles d'*horloges* ( $C$ ), de *variables* ( $V$ ) et de *canaux de communication* ( $X$ ) sont *communs à tous les automates*. Le réseau  $\{A_i^{\text{UPPAAL}}\} = (\bar{L}, \bar{l}_0, V, \tilde{V}_0, C, X, F, J)$  est alors défini par :

- $\bar{L} = (L_1 \times L_2 \times \dots \times L_n)$ , ensemble de toutes les localités.
- $\bar{l} = (l_1, l_2, \dots, l_n)$ , vecteur des localités actives du réseau.
- $\bar{l}_0 = (l_1^0, l_2^0, \dots, l_n^0)$ , vecteur des localités initiales.
- $V$  et  $C$ , ensembles communs de variables et d'horloges, avec  $\tilde{V}_0$  les valeurs des variables à l'initialisation.
- $X$ , ensemble des canaux de communication.
- $F_i \subseteq L \times D(V, C) \times \Sigma(X) \times A(V) \times 2^C \times L$ , ensemble des transitions d'un automate partant d'une localité source  $l \in L$  vers une localité cible  $l' \in L$ , avec une garde  $g' \in D(V, C)$ , une synchronisation éventuelle au travers de l'action de communication  $\sigma \in \Sigma(X)$  et une action  $a \in A(V)$ . Le terme  $2^C$  est à relier à la remise à zéro des

horloges : chaque horloge de  $C$  peut être remise à zéro ou non.  $r \subseteq C$  définit le sous-ensemble (potentiellement vide) d'horloges à remettre à zéro lors du franchissement de la transition.

- $F = \bigcup_i F_i$ , ensemble des transitions du réseau.
- $J_i: L \rightarrow D(V, C)$ , fonction qui assigne un invariant à chaque localité d'un automate, cet invariant pouvant porter sur des variables.
- $J(\bar{l}) = \bigwedge_j J_j(l_j)$ , fonction d'invariant commune.

Deux définitions diffèrent des définitions proposées pour les ATVD :

- $F_i$ , la fonction de transition d'un RATU, remplace  $E_i$ . Les différences proviennent de la définition de la garde ( $g' \in D(V, C)$ ) qui intègre la contrainte temporelle et des actions de synchronisation ( $\sigma \in \Sigma(X)$ ).
- $J$ , la fonction d'invariant des localités, remplace  $I$ . La différence vient ici de la possibilité d'intégrer des variables dans les invariants.

En utilisant les mêmes notations que précédemment (section 2.2.2), la sémantique d'un RATU est un système de transitions  $\langle S, s_0, \rightarrow \rangle$  où un état  $s \in S$  est un vecteur  $(\bar{l}, \tilde{V}, \tilde{C})$ , où  $s_0 = (\bar{l}_0, \tilde{V}_0, \tilde{C}_0)$  est l'état initial et  $\rightarrow \subseteq S \times S$  est la relation de transition telle que :

$$(\bar{l}, \tilde{V}, \tilde{C}) \rightarrow (\bar{l}, \tilde{V}, \tilde{C} + d) \text{ si} \quad (A.3)$$

$$\forall v \in V, \forall c \in C, \forall d \in \mathbb{N}, \forall d' \in [0; d], (\tilde{v}, \tilde{c} + d') \models J(\bar{l})$$

$$(\bar{l}, \tilde{V}, \tilde{C}) \rightarrow (\bar{l}' = \bar{l}[l_i \mapsto l'_i], \tilde{V}', \tilde{C}') \text{ si} \quad (A.4)$$

$$\exists f_i : l_i \xrightarrow{g', \tau, a, r} l'_i, (\tilde{V}, \tilde{C}) \models g', \tilde{C}' = [r \mapsto 0]\tilde{C}, \tilde{V}' = [V_a \xrightarrow{a} \mathbb{Z} \cup \{true, false\}]\tilde{V} \text{ et}$$

$$(\tilde{V}', \tilde{C}') \models J(\bar{l}')$$

$$(\bar{l}, \tilde{V}, \tilde{C}) \rightarrow (\bar{l}' = \bar{l}[l_i \mapsto l'_i, l_j \mapsto l'_j], \tilde{V}', \tilde{C}') \text{ si} \quad (A.5)$$

$$\exists f_i : l_i \xrightarrow{g'_i, \chi^i, a_i, r_i} l'_i \text{ et } f_j : l_j \xrightarrow{g'_j, \chi^j, a_j, r_j} l'_j, (\tilde{V}, \tilde{C}) \models (g'_i \wedge g'_j), \tilde{C}' = [r_i \cup r_j \mapsto 0]\tilde{C},$$

$$\tilde{V}' = [V_{a_i} \cup V_{a_j} \xrightarrow{a_i \cup a_j} \mathbb{Z} \cup \{true, false\}]\tilde{V} \text{ et } (\tilde{V}', \tilde{C}') \models J(\bar{l}')$$

La sémantique A.3 est la sémantique d'évolution temporelle. Comme il n'y a pas de sémantique d'urgence définie, elle n'est limitée que par les invariants des localités actives.

La sémantique A.4 de tir de transition est proche de celle des ATVD, à la principale différence que les valeurs des variables doivent aussi vérifier l'invariant de la localité d'arrivée.

Enfin, la sémantique A.5 de synchronisation permet le tir simultané de deux transitions de deux automates du réseau. Alors que la sémantique de tir d'une transition ne s'applique qu'à *une seule et unique* transition d'un automate du réseau d'automates, cette sémantique permet, au moyen d'un canal de communication et des actions s'y rapportant, de faire évoluer deux automates simultanément. Cette évolution n'est possible que si l'une des transitions est émettrice et l'autre réceptrice vis-à-vis du même canal de communication, et si leurs gardes sont validées et leurs assignations et valeurs d'horloges conformes

aux invariants des deux localités cibles. Comme il y a alors deux actions d'assignation à effectuer simultanément, il peut y avoir contradiction et concurrence en cas de double assignation d'une même variable. Dans ce cas, les assignations de la transition émettrice sont appliquées, puis celles de la transition réceptrice le sont à leur tour.

## A.4 Exemple de RATU sans sémantique d'urgence

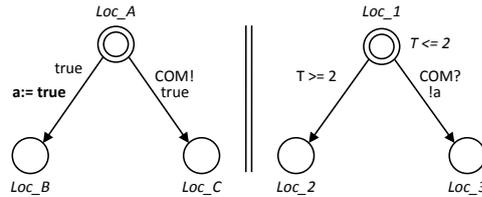


FIGURE A.1 – Réseau de deux automates temporisés d'UPPAAL.

La figure A.1 montre un exemple de réseau d'automates temporisés d'UPPAAL avec :

- Les localités (Loc\_A, Loc\_B, Loc\_C et Loc\_1, Loc\_2, Loc\_3) représentées par des cercles, les noms étant en italique.
- Les localités initiales (Loc\_A et Loc\_1) représentées par des cercles doublés.
- Les invariants attachés aux localités (comme  $T \leq 2$  pour la localité Loc\_1) en italique.
- Les transitions, représentées par des flèches, allant de la localité source à la localité cible.
- Les gardes écrites près des transitions. Les disjonctions ou conjonctions sont notées respectivement  $\parallel$  et  $\&\&$ . L'opérateur complémentaire booléen est noté  $!$ . La garde toujours vraie (true) est notée sur les transitions de cet exemple mais sera par la suite toujours omise.
- Les actions associées à une transition, notées en utilisant l'opérateur d'assignation  $:=$  en gras. Les remises à zéro des horloges utilisent le même opérateur, en limitant la valeur assignée à 0.
- Les actions de communication, notées par le nom du canal suivi de  $?$  pour l'action de réception ou  $!$  pour l'action d'émission (comme ici COM? et COM!). L'action de communication nulle ( $\tau$ , non synchronisation) sera systématiquement omise.

En appliquant les règles de sémantique définies précédemment, on peut obtenir les évolutions possibles depuis la situation initiale (Loc\_A et Loc\_1 actives, variable booléenne  $a$  à faux (*false*) et horloge  $T$  à 0).

**Loc\_A**  $\rightarrow$  **Loc\_B** est possible via la sémantique A.4 car la garde est toujours vérifiée. La variable booléenne  $a$  sera mise à vrai. Le prochain vecteur des localités actives sera alors (Loc\_B, Loc\_1).

**Loc\_1**  $\rightarrow$  **Loc\_2** est aussi possible, il suffit de laisser passer exactement 2 unités de temps (sémantique temporelle A.3 faisant passer  $T$  de 0 à 2), ce qui est autorisé par la garde (car  $T$  est supérieure ou égale à 2) et l'invariant (car  $T$  est inférieure ou égale à 2),

puis de tirer la transition via la sémantique A.4. Le prochain vecteur des localités actives sera alors (Loc\_A,Loc\_2).

**Loc\_A  $\rightarrow$  Loc\_C et Loc\_1  $\rightarrow$  Loc\_3 simultanément** est possible car la garde !a est vérifiée et les deux transitions liées par le canal COM sont franchissables via la sémantique A.5. Le prochain vecteur des localités actives sera alors (Loc\_C,Loc\_3).

En résumé, trois évolutions sont possibles : le tir direct d'une transition, l'évolution du temps permettant le tir d'une autre transition, et enfin le franchissement synchrone de deux transitions au travers du canal de communication COM.

On note qu'aucune de ces évolutions n'est prioritaire par rapport aux autres : les trois évolutions sont possibles et donc en concurrence.

## A.5 Sémantique d'urgence d'un RATU

L'urgence dans UPPAAL peut se traduire de deux manières : au travers d'un attribut d'urgence attaché aux canaux de communication, et au travers d'attributs d'urgence attachés aux localités.

**Les canaux de communication urgents** permettent de donner la priorité aux évolutions synchrones concernées sur l'évolution des horloges. Lorsqu'une évolution synchrone au travers d'un canal urgent est possible (sémantique A.5), la sémantique de passage du temps (sémantique A.3) –et celle-ci *uniquement*– est interdite. En toute logique, les transitions avec une synchronisation au travers d'un canal urgent n'ont pas le droit d'avoir des gardes faisant référence aux horloges. Les canaux de communication urgents sont marqués en italique dans le reste de ces travaux.

**Les attributs urgents et obligatoires des localités** permettent de limiter les sémantiques autorisées en fonction de la nature des localités actives :

- Lorsqu'une localité urgente (*urgent location*) est active, la sémantique d'évolution des horloges est interdite.
- Le cas des localités obligatoires (*committed location*) est plus complexe : lorsqu'une localité obligatoire est active, la sémantique d'évolution du temps est aussi interdite, mais la prochaine évolution (franchissement d'une transition, ou de deux via un canal de communication) devra *obligatoirement* avoir une localité obligatoire comme localité source.

Les localités urgentes sont marquées d'un U et les localités obligatoires d'un C (pour *committed*).

## A.6 Sémantique complète d'un RATU

On conserve les notations utilisées pour la définition de la sémantique sans urgence, mais de nouvelles définitions sont nécessaires :

- $X$  est toujours l'ensemble des canaux de communication, mais on définit  $X_u \subseteq X$  le sous-ensemble des canaux urgents.

- $L$  est toujours l'ensemble des localités, mais on définit  $L_u \subset L$  le sous-ensemble des localités urgentes.
- De même, on définit  $L_c \subset L$ ,  $L_u \cap L_c = \emptyset$  le sous-ensemble des localités obligatoires.

Avec ces nouvelles définitions, on peut définir la sémantique complète d'un RATU  $\{A_i^{\text{UPPAAL}}\}$ , avec  $i \in \{1, \dots, n\}$ , comme un système de transitions  $\langle S, s_0, \rightarrow \rangle$  où un état  $s \in S$ , avec  $S = (L_1, L_2, \dots, L_n) \times V \times C$ , est un triplet  $(\bar{l}, \tilde{V}, \tilde{C})$ ,  $s_0 = (\bar{l}_0, \tilde{V}_0, \tilde{C}_0)$  est l'état initial, et  $\rightarrow \subseteq S \times S$  est la relation de transition telle que :

$$\begin{aligned}
& (\bar{l}, \tilde{V}, \tilde{C}) \rightarrow (\bar{l}, \tilde{V}, \tilde{C} + d) \text{ si} \\
& \quad \forall v \in V, \forall c \in C, \forall d \in \mathbb{N}, \forall d' \in [0; d], (\tilde{v}, \tilde{c} + d') \models J(\bar{l}) \text{ et} \\
& \quad \forall l_k \in \bar{l}, l_k \notin (L_u \cup L_c) \text{ et} \\
& \quad \nexists f_i : l_i \xrightarrow{g'_i, \chi^!, a_i, r_i} l'_i \text{ et } f_j : l_j \xrightarrow{g'_j, \chi^?, a_j, r_j} l'_j \text{ tels que}
\end{aligned} \tag{A.6}$$

$$\begin{aligned}
& \quad \chi \in X_u, (\tilde{V}, \tilde{C}) \models (g'_i \wedge g'_j), \tilde{C}' = [r_i \cup r_j \mapsto 0] \tilde{C}, \\
& \quad \tilde{V}' = [V_{a_i} \cup V_{a_j} \xrightarrow{a_i \cup a_j} \mathbb{Z} \cup \{true, false\}] \tilde{V} \text{ et } (\tilde{V}', \tilde{C}') \models J(\bar{l}')
\end{aligned}$$

$$\begin{aligned}
& (\bar{l}, \tilde{V}, \tilde{C}) \rightarrow (\bar{l}' = \bar{l}[l_i \mapsto l'_i], \tilde{V}', \tilde{C}') \text{ si} \\
& \quad \forall l_k \in \bar{l}, l_k \notin L_c \text{ et } \exists f : l_i \xrightarrow{g', \tau, a, r} l'_i, (\tilde{V}, \tilde{C}) \models g', \tilde{C}' = [r \mapsto 0] \tilde{C}, \\
& \quad \tilde{V}' = [V_a \xrightarrow{a} \mathbb{Z} \cup \{true, false\}] \tilde{V} \text{ et } (\tilde{V}', \tilde{C}') \models J(\bar{l}')
\end{aligned} \tag{A.7}$$

$$\begin{aligned}
& (\bar{l}, \tilde{V}, \tilde{C}) \rightarrow (\bar{l}' = \bar{l}[l_i \mapsto l'_i, l_j \mapsto l'_j], \tilde{V}', \tilde{C}') \text{ si} \\
& \quad \forall l_k \in \bar{l}, l_k \notin L_c \text{ et } \exists f_i : l_i \xrightarrow{g'_i, \chi^!, a_i, r_i} l'_i \text{ et } f_j : l_j \xrightarrow{g'_j, \chi^?, a_j, r_j} l'_j, (\tilde{V}, \tilde{C}) \models (g'_i \wedge g'_j), \\
& \quad \tilde{C}' = [r_i \cup r_j \mapsto 0] \tilde{C}, \tilde{V}' = [V_{a_i} \cup V_{a_j} \xrightarrow{a_i \cup a_j} \mathbb{Z} \cup \{true, false\}] \tilde{V} \text{ et } (\tilde{V}', \tilde{C}') \models J(\bar{l}')
\end{aligned} \tag{A.8}$$

$$\begin{aligned}
& (\bar{l}, \tilde{V}, \tilde{C}) \rightarrow (\bar{l}' = \bar{l}[l_i \mapsto l'_i], \tilde{V}', \tilde{C}') \text{ si} \\
& \quad \exists l_i \in \bar{l}, l_i \in L_c \text{ et } \exists f : l_i \xrightarrow{g', t, \tau, a, r} l'_i, (\tilde{V}, \tilde{C}) \models g', \tilde{C}' = [r \mapsto 0] \tilde{C}, \\
& \quad \tilde{V}' = [V_a \xrightarrow{a} \mathbb{Z} \cup \{true, false\}] \tilde{V} \text{ et } (\tilde{V}', \tilde{C}') \models J(\bar{l}')
\end{aligned} \tag{A.9}$$

$$\begin{aligned}
& (\bar{l}, \tilde{V}, \tilde{C}) \rightarrow (\bar{l}' = \bar{l}[l_i \mapsto l'_i, l_j \mapsto l'_j], \tilde{V}', \tilde{C}') \text{ si} \\
& \quad \exists l_i, l_j \in \bar{l}, l_i \in L_c \text{ ou } l_j \in L_c, \text{ et } \exists f_i : l_i \xrightarrow{g'_i, \chi^!, a_i, r_i} l'_i \text{ et } f_j : l_j \xrightarrow{g'_j, \chi^?, a_j, r_j} l'_j \text{ et} \\
& \quad (\tilde{V}, \tilde{C}) \models (g'_i \wedge g'_j), \tilde{C}' = [r_i \cup r_j \mapsto 0] \tilde{C}, \tilde{V}' = [V_{a_i} \cup V_{a_j} \xrightarrow{a_i \cup a_j} \mathbb{Z} \cup \{true, false\}] \tilde{V} \text{ et} \\
& \quad (\tilde{V}', \tilde{C}') \models J(\bar{l}')
\end{aligned} \tag{A.10}$$

La sémantique temporelle (A.6) est logiquement très impactée par les sémantiques d'urgence. En effet, cette sémantique est interdite quand une localité active est urgente ou obligatoire, mais aussi lorsqu'une évolution synchrone est possible via un canal de communication urgent.

L'impact du comportement des localités urgentes s'arrête ici, mais celui des localités obligatoires est bien plus important, il force à dédoubler les sémantiques de franchissement d'une transition et de synchronisation.

Ainsi, pour les cas où il n'y a pas de localité obligatoire active, on retrouve les sémantiques classiques de tir d'une transition (A.7) et de synchronisation (A.8).

La sémantique A.9 considère le cas où au moins une localité active est obligatoire et où l'on peut franchir une transition sans communication. La localité de départ de la transition franchie doit alors *nécessairement* être une localité obligatoire.

La sémantique A.10 considère quant à elle le cas où une localité active est obligatoire et où il existe deux transitions franchissables liées par un canal de communication (urgent ou non). Dans ce cas, la localité de départ de la transition émettrice et/ou de la transition réceptrice doit être une localité obligatoire.

On peut donc maintenant étudier les différentes priorités des sémantiques. La sémantique temporelle est la plus contrainte car elle nécessite le respect des invariants, la non présence de transitions franchissables liées par un canal de communication urgent et la non présence de localités urgentes ou obligatoires dans les localités actives.

Ensuite, dans le cas où il n'y a pas de localité obligatoire dans les localités actives, les sémantiques de franchissement d'une transition ou de tir synchrone de deux transitions peuvent être en concurrence.

De manière semblable, si au moins une localité active est une localité obligatoire, une concurrence peut apparaître à nouveau entre ces mêmes sémantiques, mais limitée aux transitions avec (au moins) une localité obligatoire comme localité source.

## A.7 Illustration de la sémantique complète sur un RATU

La figure A.2 montre un exemple de réseau de deux automates temporisés d'UPPAAL. Par rapport à l'exemple précédent (figure A.1), les modifications sont les suivantes :

- Le canal de communication COM est maintenant défini comme *urgent*, et donc noté en italique.
- La localité Loc\_C est définie comme obligatoire, elle est marquée d'un C (pour *committed*).
- Une transition a été ajoutée allant de Loc\_C à Loc\_B.
- Une transition a été ajoutée allant de Loc\_3 à Loc\_2.
- Comme précédemment expliqué, les gardes toujours vraies (*true*) ne sont pas marquées.

À partir de l'état initial (Loc\_A et Loc\_1 actives, *a* à faux et T à 0), les évolutions suivantes sont possibles :

**Loc\_A** → **Loc\_B** est possible (avec la sémantique A.7) car la garde (non notée) est toujours vraie.

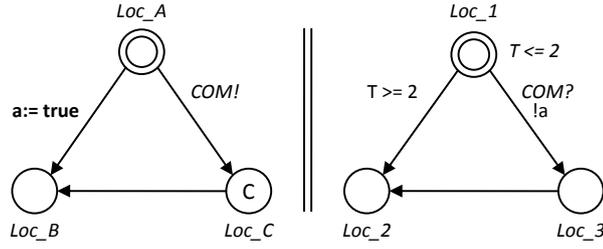


FIGURE A.2 – Exemple de réseau d’automates temporisés d’UPPAAL avec urgence.

**Loc\_A**  $\rightarrow$  **Loc\_C** & **Loc\_1**  $\rightarrow$  **Loc\_3** est possible via le canal de communication COM car les gardes sont vérifiées ( $a$  est faux). En accord avec la sémantique A.8, les localités actives sont alors Loc\_C et Loc\_3, comme montré dans la figure A.3.

L’autre évolution partant d’une localité active (Loc\_1  $\rightarrow$  Loc\_2) n’est pas possible. En effet, les transitions utilisant COM sont franchissables et ce canal est défini comme urgent : la sémantique d’évolution du temps (A.6) est donc interdite. T restant à 0, la garde ne peut pas être validée.

La figure A.3 montre la situation après le tir utilisant le canal de communication COM. Dans cet état, seule la transition **Loc\_C**  $\rightarrow$  **Loc\_B** peut être tirée (sémantique A.9). En effet, la localité Loc\_C étant définie comme obligatoire, la prochaine évolution devra nécessairement partir d’une localité obligatoire (sémantique A.9 ou A.10).

La transition **Loc\_3**  $\rightarrow$  **Loc\_2** ne peut donc pas être franchie en utilisant la sémantique A.7 car Loc\_3 n’est pas une localité obligatoire.

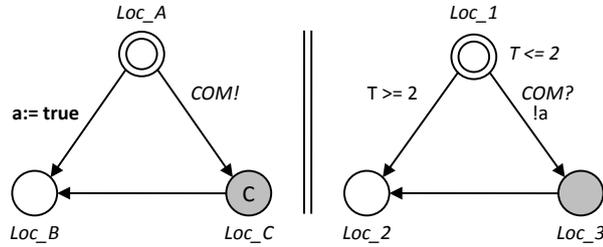


FIGURE A.3 – Exemple de RATU après une évolution synchrone via le canal urgent COM.



## Annexe B

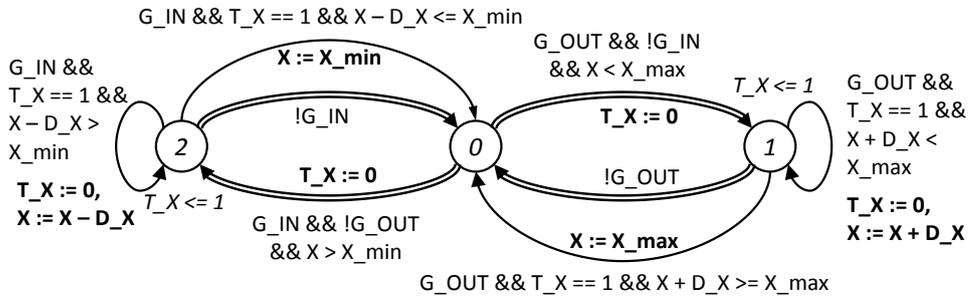
### Modèles de partie opérative relatifs au chapitre 3

## B.1 Exemples de modèles d'une bibliothèque de composants génériques

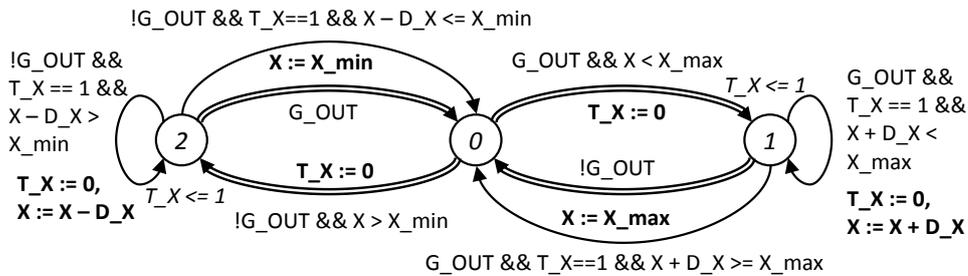
Les modèles présentés dans cette annexe tiennent compte des modifications proposées dans le chapitre 3.

### B.1.1 Actionneurs couplés à des pré-actionneurs

Les modèles des vérins (figures B.1a et B.1b) utilisent les ordres G\_OUT et éventuellement G\_IN. La longueur de tige sortie X est modifiée d'une longueur D\_X toutes les D\_T unités de temps lorsque la tige est en mouvement, dans les limites de X\_min et X\_max qui sont les limites physiques de rentrée/sortie de tige. La localité initiale n'est pas précisée car elle dépend des valeurs initiales des ordres et de la configuration du pré-actionneur.



(a) Modèle générique d'un vérin à double effet associé à un pré-actionneur 5\3 à commande bi-stable.



(b) Modèle générique d'un vérin à double effet associé à un pré-actionneur 5\2 à commande mono-stable.

FIGURE B.1 – Exemples de modèles génériques de vérins avec pré-actionneurs.

Le modèle du moteur est sensible aux ordres G\_POS (mouvement sens trigonométrique) et G\_NEG (sens contraire). La position angulaire X de l'arbre est changée de D\_X degrés toutes les D\_T unités de temps lorsque le moteur est en marche. Ici, le nombre de tour du moteur n'est pas limité mais la grandeur X représentant un angle doit rester dans l'intervalle [0, 359]. Pour ce faire, lors de l'assignation de la prochaine valeur de X, un test est réalisé au moyen d'une division euclidienne. Par exemple, dans le cas d'une position angulaire X augmentant de D\_X, on souhaite le comportement suivant :

$$X \mapsto \begin{cases} X + D\_X & \text{si } X + D\_X < 360 \\ X + D\_X - 360 & \text{si } X + D\_X \geq 360 \end{cases} \quad (\text{B.1})$$

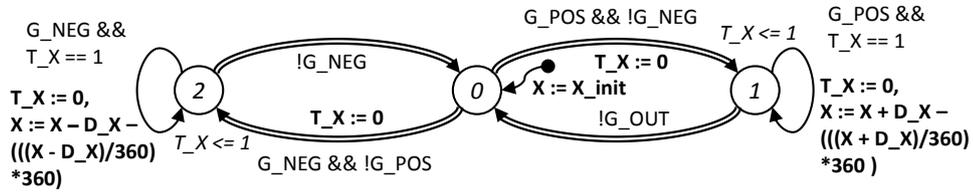


FIGURE B.2 – Modèle générique d'un moteur à deux sens de rotation.

### B.1.2 Capteurs

Le modèle de la figure B.3 réagit à certaines valeurs particulières de la grandeur X et y associe les informations des capteurs IN, MID et OUT. Ces changements de valeur des informations des capteurs induisent une mise à vrai de EVOL\_PL pour permettre au modèle du contrôleur de réagir.

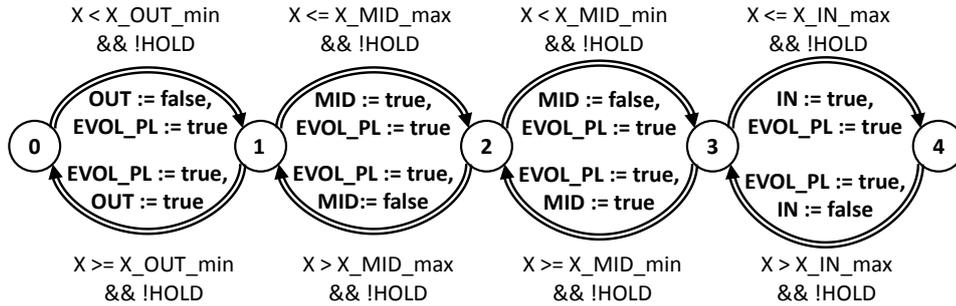
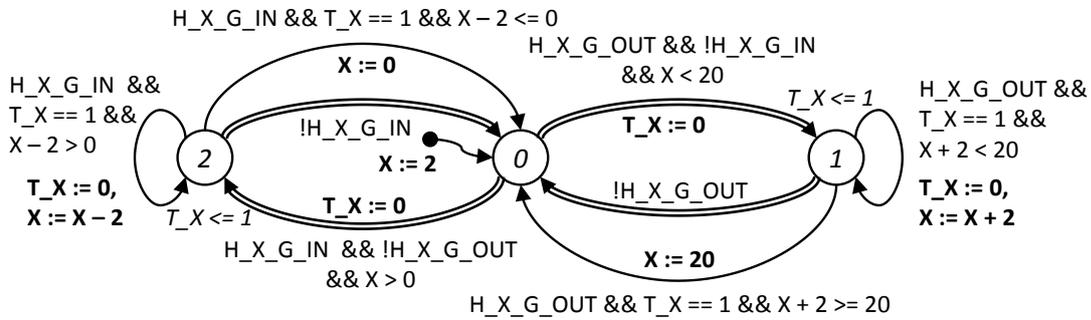
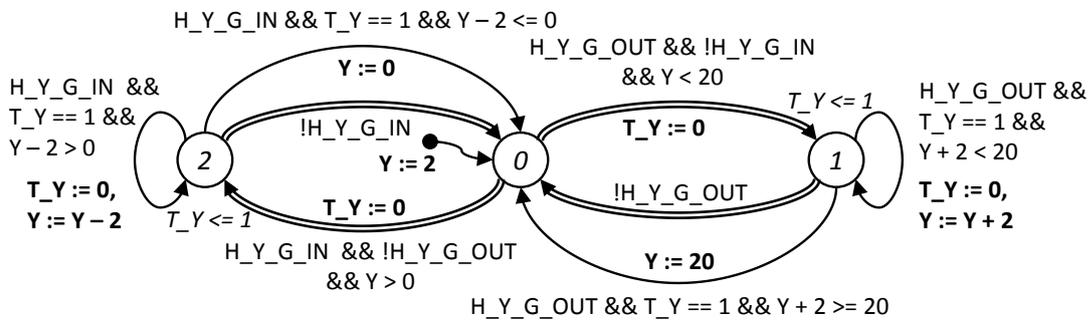


FIGURE B.3 – Modèle générique d'un ensemble de trois capteurs de position.

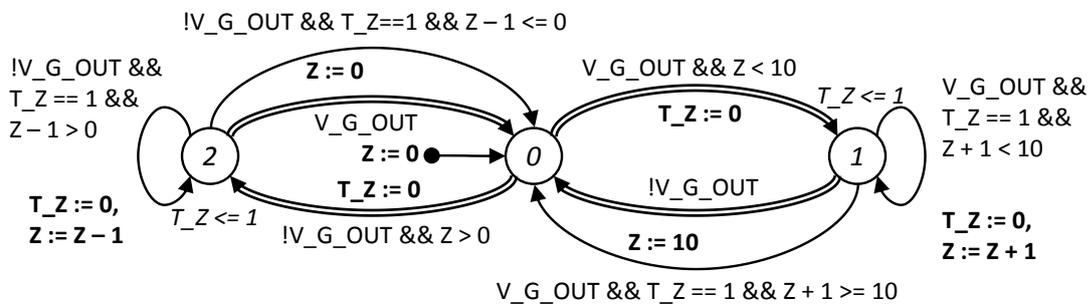
## B.2 Modèles de la partie opérative du manipulateur



(a) Modèle final du vérin horizontal X de l'exemple.

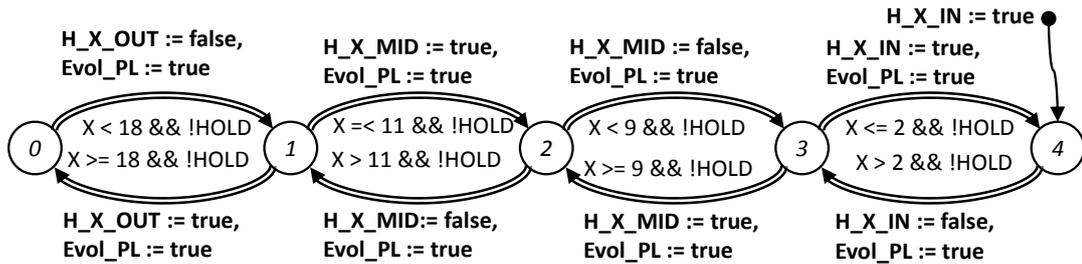


(b) Modèle final du vérin horizontal Y de l'exemple.

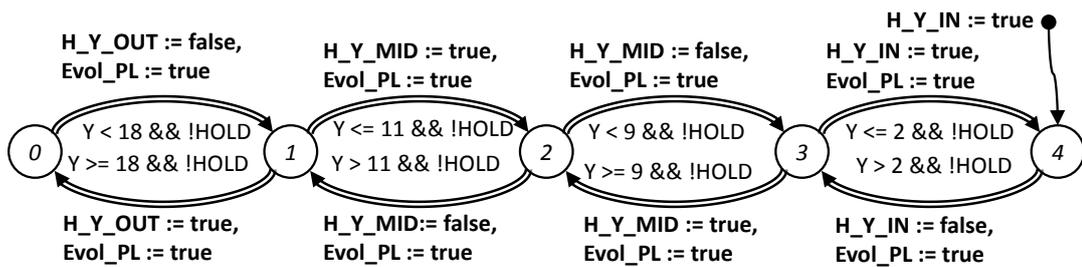


(c) Modèle final du vérin vertical Z de l'exemple.

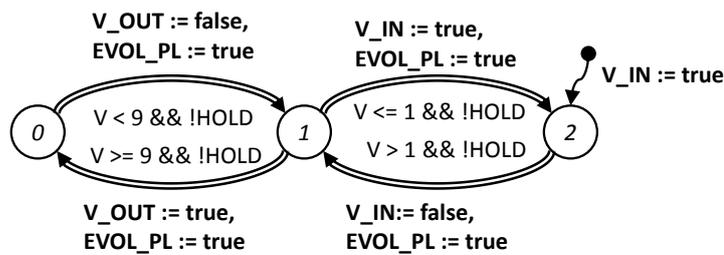
FIGURE B.4 – Modèles finaux des vérins de l'exemple.



(a) Modèle final des capteurs associés au vérin horizontal X.



(b) Modèle final des capteurs associés au vérin horizontal Y.



(c) Modèle final des capteurs associés au vérin vertical Z.

FIGURE B.5 – Modèles finaux des groupes de capteurs de l'exemple.



## Annexe C

### Séquences d'évolutions relatives au chapitre 3

## C.1 Séquence d'évolutions avec les modèles de partie opérative modifiés

L'état de départ (figure C.1a) est identique à celui utilisé dans la figure 3.12 : vérin rentré ( $X$  égale à 2) et à l'arrêt (localité  $0$  active), capteurs détectant une position rentrée ( $H\_X\_IN$  à vrai,  $H\_X\_MID$  et  $H\_X\_OUT$  à faux). La commande de sortie du vérin ( $H\_X\_G\_OUT$ ) est pour l'instant à faux.

La mise à vrai de la variable de contrôle  $H\_X\_G\_OUT$  permet toujours au modèle du vérin d'évoluer vers la localité  $1$ , comme montré dans la figure C.1b.

Ensuite, comme aucune transition urgente n'est franchissable, le passage du temps va à nouveau faire évoluer l'horloge à 1. Ceci permet le tir de la transition modélisant la sortie de la tige comme montré dans la figure C.1c.

Alors que précédemment, deux évolutions étaient possibles (le modèle du vérin pouvait évoluer, comme celui du groupe de capteurs), ce n'est plus possible avec les modèles corrigés. En effet, la transition *urgente* du modèle des capteurs est franchissable, ce qui interdit la sémantique temporelle et donc le franchissement de la transition de mouvement du modèle du vérin. Il ne reste plus que l'évolution du modèle des capteurs possible, comme montré sur la figure C.1d.

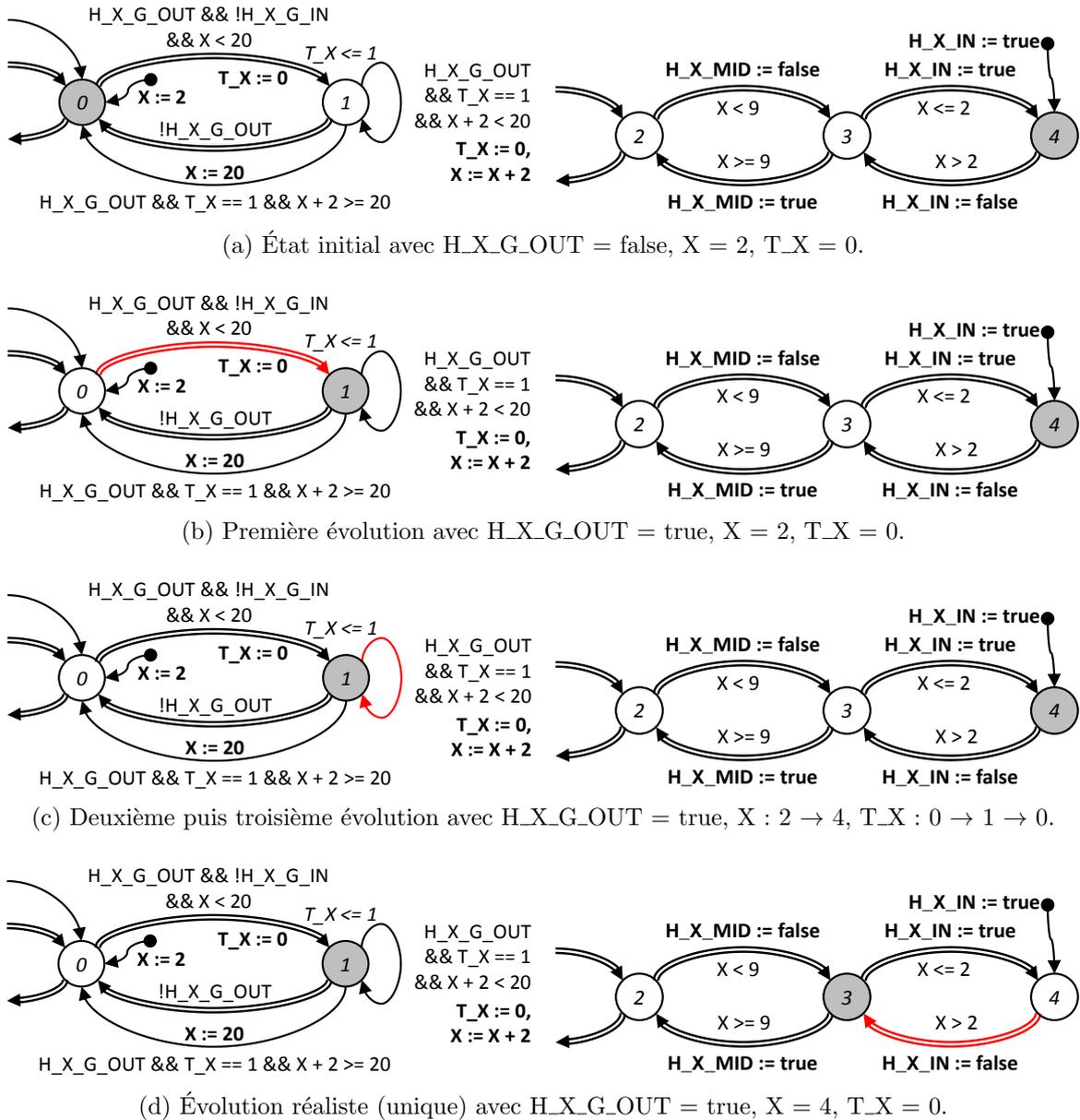


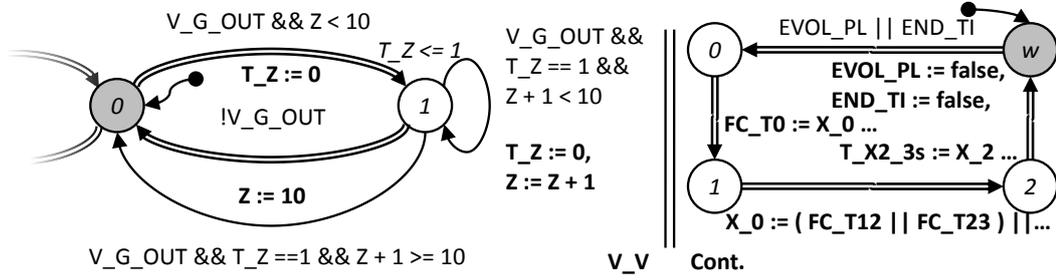
FIGURE C.1 – Séquence d'évolutions avec modèles corrigés.

## C.2 Séquence d'évolutions après introduction des variables EVOL\_PL et END\_TI

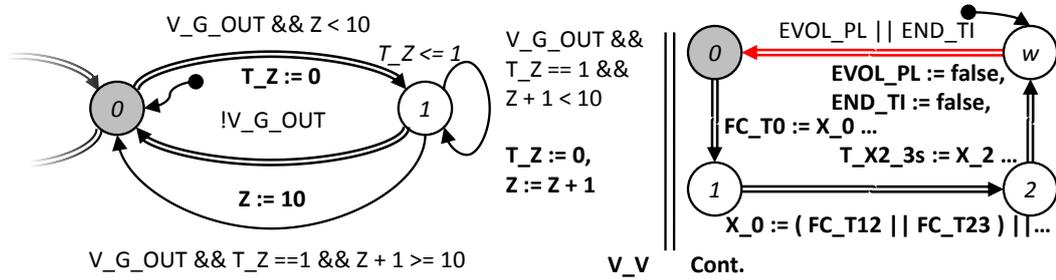
Les modifications apportées par les deux variables EVOL\_PL et END\_TI permettent de supprimer le comportement non souhaité, ainsi la suite d'évolutions de la figure 3.27 n'est plus possible et est remplacée par la suite représentée dans la figure C.2 :

- La situation initiale montrée dans la figure C.2a est identique à la précédente, à la seule différence que la variable EVOL\_PL est à vrai car on suppose que la variable IHM\_B\_START vient de passer à vrai.
- L'évolution suivante, figure C.2b, correspond au franchissement de la nouvelle transition ajoutée : comme EVOL\_PL est vraie, on peut débiter le cycle de l'API, EVOL\_PL est remise à faux et HOLD passe à vrai.
- Les trois évolutions suivantes (calcul des conditions de tir, des étapes actives et des valeurs des variables de sortie) sont inchangées et ne sont pas représentées.
- Une fois le cycle API terminé (localité  $w$  à nouveau active), comme EVOL\_PL n'est plus vraie, un nouveau cycle API n'est pas possible. Le modèle du vérin V\_V évolue donc vers la localité 1 modélisant la sortie de la tige (figure C.2c).
- Enfin, comme il n'y a plus de transition urgente franchissable (car EVOL\_PL et END\_TI sont toutes deux à faux), la sémantique temporelle peut être utilisée et la transition de mouvement du modèle du vérin peut être tirée (figure C.2d).

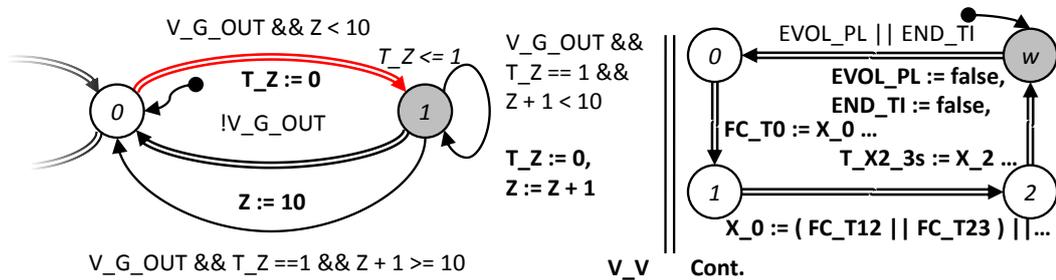
Il n'y a plus de blocage du modèle de la partie opérative. Un autre cycle du modèle du contrôleur est possible dès qu'une entrée change, et ainsi de suite. Dans la séquence précédente, le prochain cycle sera autorisé par le passage de V\_OUT à vrai après plusieurs tirs de l'arc modélisant la sortie de la tige.



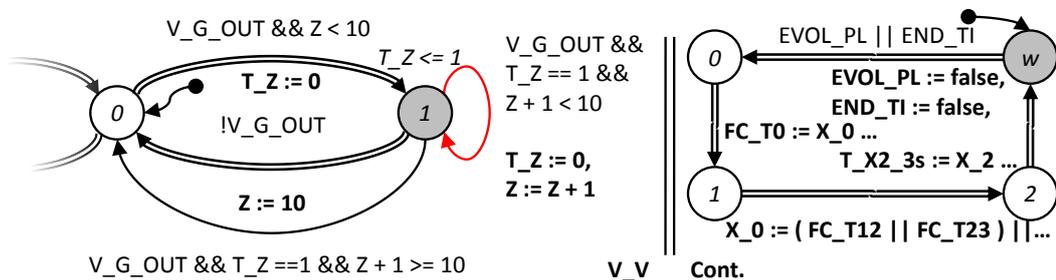
(a) Situation initiale.



(b) Première évolution : début du cycle de l'API.



(c) Cinquième évolution : évolution urgente du modèle du vérin.



(d) Dernière évolution : évolution temporisée du modèle du vérin.

FIGURE C.2 – Séquence d'évolutions avec les modèles modifiés.

## C.3 Séquence d'évolutions après ajout de recherche de stabilité dans le contrôleur

La figure C.3 montre la séquence précédente de la figure 3.31 avec le modèle final du contrôleur :

- La situation initiale est identique avec *stable* à vrai car cette situation est stable (figure C.3a).
- Lorsque le capteur associé à la fin de course du vérin réagit, la variable V\_OUT passe à vrai, en même temps que EVOL\_PL (figure C.3b).
- Ceci autorise le modèle du contrôleur à faire un cycle, comme le montre la figure C.3c. Comme il y a eu au moins une condition de tir validée (FC\_TP1), *stable* est mise à faux. Les sorties ne sont pas émises et on peut directement lancer un nouveau cycle.
- Le second cycle présenté dans la figure C.3d montre que l'étape A-W devient enfin active. Comme il y a eu une condition de tir validée (FC\_TA-W), *stable* reste à faux et les sorties ne sont toujours pas émises.
- Enfin, le troisième cycle (figure C.3e) permet de définir que la situation est stable. Le cycle se termine donc par le retour à la localité initiale, la libération des évolutions des modèles de la partie opérative (HOLD repasse à faux) et l'émission des sorties. La volonté du concepteur est respectée : la pompe à vide est bien lancée en même temps que la temporisation associée à l'activité de l'étape P2.

C.3. Séquence d'évolutions après ajout de recherche de stabilité dans le contrôleur

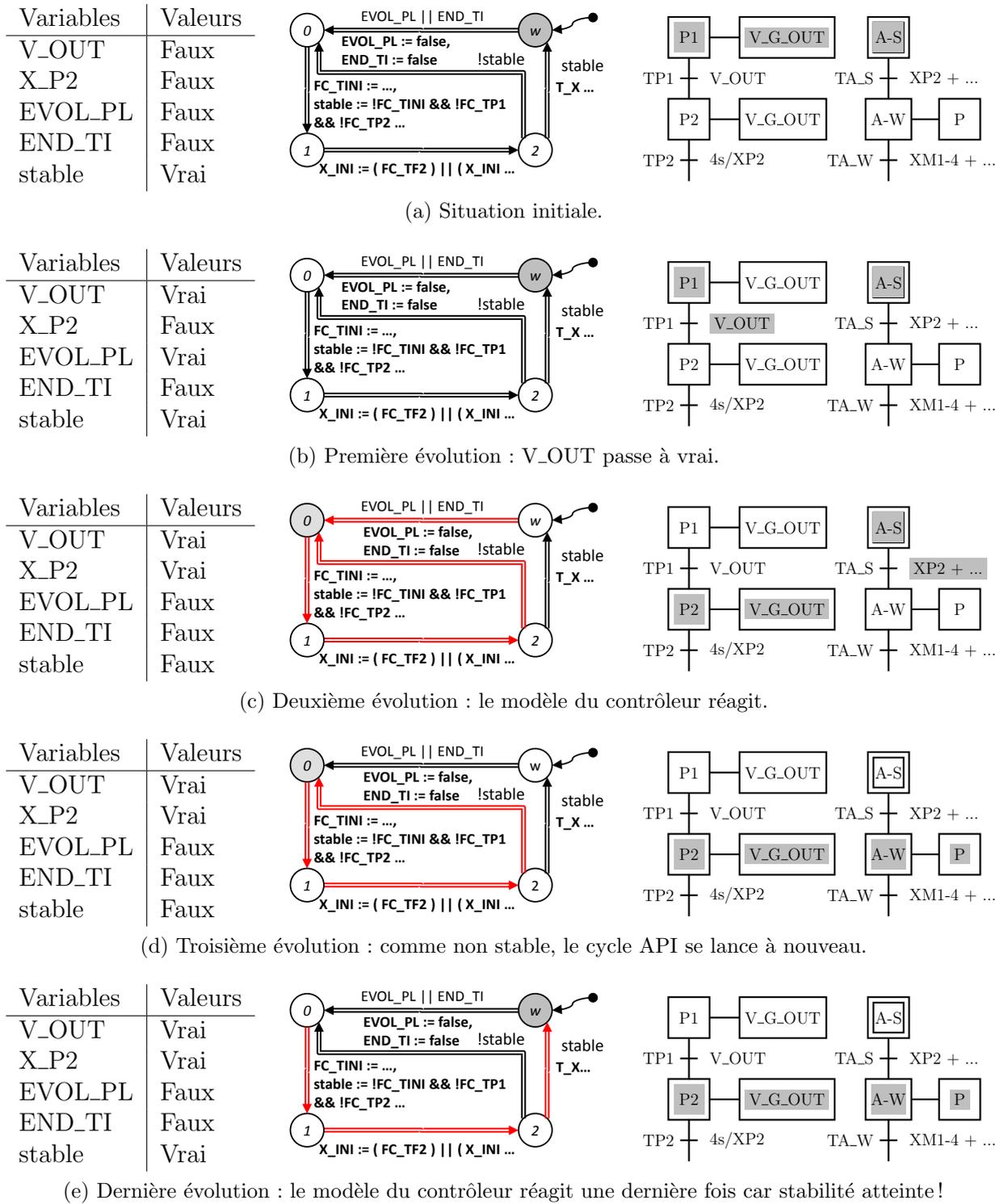


FIGURE C.3 – Séquence d'évolutions du système bouclé avec la version finale du modèle du contrôleur.



## Annexe D

# Équations algébriques du Grafcet et modèles du contrôleur du manipulateur

## **D.1 Équations algébriques associées au Grafset du manipulateur**

$$\begin{aligned}
 FC\_TINI &= X\_INI \wedge IHM\_B\_START \\
 FC\_TP1 &= X\_P1 \wedge V\_OUT \\
 FC\_TP2 &= X\_P2 \wedge T\_XP2\_4s\_Q \\
 FC\_TP3 &= X\_P3 \wedge V\_IN \\
 FC\_TP4 &= X\_P4 \wedge (H\_X\_MID \wedge H\_Y\_MID) \\
 FC\_TP5 &= X\_P5 \wedge V\_OUT \\
 FC\_TT1 &= X\_T1 \wedge TEST\_OK \\
 FC\_TT2 &= X\_T2 \wedge V\_IN \\
 FC\_TT3a &= X\_T3 \wedge X\_PT1 \\
 FC\_TT3b &= X\_T3 \wedge X\_PT2 \\
 FC\_TM1\_1 &= X\_M1\_1 \wedge (H\_X\_OUT \wedge H\_Y\_IN) \\
 FC\_TM1\_2 &= X\_M1\_2 \wedge M1\_RDY \\
 FC\_TM1\_3 &= X\_M1\_3 \wedge V\_OUT \\
 FC\_TM1\_4 &= X\_M1\_4 \wedge T\_X\_M1\_4\_3s\_Q \\
 FC\_TM1\_5 &= X\_M1\_5 \wedge V\_IN \\
 FC\_TM1\_6 &= X\_M1\_6 \wedge H\_Y\_MID \\
 FC\_TM1\_7 &= X\_M1\_7 \wedge M1\_FINISH \\
 FC\_TM1\_8 &= X\_M1\_8 \wedge V\_OUT \\
 FC\_TM1\_9 &= X\_M1\_9 \wedge T\_X\_M1\_9\_4s\_Q \\
 FC\_TM1\_10 &= X\_M1\_10 \wedge V\_IN \\
 FC\_TM1\_11 &= X\_M1\_11 \wedge H\_Y\_OUT \\
 FC\_TM2\_1 &= X\_M2\_1 \wedge (H\_X\_IN \wedge H\_Y\_OUT) \\
 FC\_TM2\_2 &= X\_M2\_2 \wedge M2\_RDY \\
 FC\_TM2\_3 &= X\_M2\_3 \wedge V\_OUT \\
 FC\_TM2\_4 &= X\_M2\_4 \wedge T\_X\_M2\_4\_3s\_Q \\
 FC\_TM2\_5 &= X\_M2\_5 \wedge V\_IN \\
 FC\_TM2\_6 &= X\_M2\_6 \wedge H\_X\_MID \\
 FC\_TM2\_7 &= X\_M2\_7 \wedge M2\_FINISH \\
 FC\_TM2\_8 &= X\_M2\_8 \wedge V\_OUT \\
 FC\_TM2\_9 &= X\_M2\_9 \wedge T\_X\_M2\_9\_4s\_Q \\
 FC\_TM2\_10 &= X\_M2\_10 \wedge V\_IN \\
 FC\_TM2\_11 &= X\_M2\_11 \wedge H\_X\_OUT \\
 FC\_TF1 &= X\_F1 \wedge T\_XF1\_3s\_Q \\
 FC\_TF2 &= X\_F2 \wedge (H\_X\_IN \wedge H\_Y\_IN) \\
 FC\_TAS &= X\_AS \wedge (X\_P2 \vee X\_M1\_9 \vee X\_M2\_9) \\
 FC\_TAW &= X\_AW \wedge (X\_M1\_4 \vee X\_M2\_4 \vee X\_F1) \\
 FC\_TPTa &= X\_NP \wedge (XT1 \wedge TEST\_OK \wedge TEST\_PT1) \\
 FC\_TPTb &= X\_NP \wedge (XT1 \wedge TEST\_OK \wedge \neg(TEST\_PT1)) \\
 FC\_TPT1 &= X\_PT1 \wedge X\_F2 \\
 FC\_TPT2 &= X\_PT2 \wedge X\_F2
 \end{aligned}$$

FIGURE D.1 – Calcul des conditions de tir.

$$\begin{aligned}
 X\_INI &= (FC\_TF2) \vee (X\_INI \wedge \neg(FC\_TINI)) \\
 X\_P1 &= (FC\_TINI) \vee (X\_P1 \wedge \neg(FC\_TP1)) \\
 X\_P2 &= (FC\_TP1) \vee (X\_P2 \wedge \neg(FC\_TP2)) \\
 X\_P3 &= (FC\_TP2) \vee (X\_P3 \wedge \neg(FC\_TP3)) \\
 X\_P4 &= (FC\_TP3) \vee (X\_P4 \wedge \neg(FC\_TP4)) \\
 X\_P5 &= (FC\_TP4) \vee (X\_P5 \wedge \neg(FC\_TP5)) \\
 X\_T1 &= (FC\_TP5) \vee (X\_T1 \wedge \neg(FC\_TT1)) \\
 X\_T2 &= (FC\_TT1) \vee (X\_T2 \wedge \neg(FC\_TT2)) \\
 X\_T3 &= (FC\_TT2) \vee (X\_T3 \wedge \neg(FC\_TT3a \vee FC\_TT3b)) \\
 X\_M1\_1 &= (FC\_TT3a) \vee (X\_M1\_1 \wedge \neg(FC\_TM1\_1)) \\
 X\_M1\_2 &= (FC\_TM1\_1) \vee (X\_M1\_2 \wedge \neg(FC\_TM1\_2)) \\
 X\_M1\_3 &= (FC\_TM1\_2) \vee (X\_M1\_3 \wedge \neg(FC\_TM1\_3)) \\
 X\_M1\_4 &= (FC\_TM1\_3) \vee (X\_M1\_4 \wedge \neg(FC\_TM1\_4)) \\
 X\_M1\_5 &= (FC\_TM1\_4) \vee (X\_M1\_5 \wedge \neg(FC\_TM1\_5)) \\
 X\_M1\_6 &= (FC\_TM1\_5) \vee (X\_M1\_6 \wedge \neg(FC\_TM1\_6)) \\
 X\_M1\_7 &= (FC\_TM1\_6) \vee (X\_M1\_7 \wedge \neg(FC\_TM1\_7)) \\
 X\_M1\_8 &= (FC\_TM1\_7) \vee (X\_M1\_8 \wedge \neg(FC\_TM1\_8)) \\
 X\_M1\_9 &= (FC\_TM1\_8) \vee (X\_M1\_9 \wedge \neg(FC\_TM1\_9)) \\
 X\_M1\_10 &= (FC\_TM1\_9) \vee (X\_M1\_10 \wedge \neg(FC\_TM1\_10)) \\
 X\_M1\_11 &= (FC\_TM1\_10) \vee (X\_M1\_11 \wedge \neg(FC\_TM1\_11)) \\
 X\_M2\_1 &= (FC\_TT3b) \vee (X\_M2\_1 \wedge \neg(FC\_TM2\_1)) \\
 X\_M2\_2 &= (FC\_TM2\_1) \vee (X\_M2\_2 \wedge \neg(FC\_TM2\_2)) \\
 X\_M2\_3 &= (FC\_TM2\_2) \vee (X\_M2\_3 \wedge \neg(FC\_TM2\_3)) \\
 X\_M2\_4 &= (FC\_TM2\_3) \vee (X\_M2\_4 \wedge \neg(FC\_TM2\_4)) \\
 X\_M2\_5 &= (FC\_TM2\_4) \vee (X\_M2\_5 \wedge \neg(FC\_TM2\_5)) \\
 X\_M2\_6 &= (FC\_TM2\_5) \vee (X\_M2\_6 \wedge \neg(FC\_TM2\_6)) \\
 X\_M2\_7 &= (FC\_TM2\_6) \vee (X\_M2\_7 \wedge \neg(FC\_TM2\_7)) \\
 X\_M2\_8 &= (FC\_TM2\_7) \vee (X\_M2\_8 \wedge \neg(FC\_TM2\_8)) \\
 X\_M2\_9 &= (FC\_TM2\_8) \vee (X\_M2\_9 \wedge \neg(FC\_TM2\_9)) \\
 X\_M2\_10 &= (FC\_TM2\_9) \vee (X\_M2\_10 \wedge \neg(FC\_TM2\_10)) \\
 X\_M2\_11 &= (FC\_TM2\_10) \vee (X\_M2\_11 \wedge \neg(FC\_TM2\_11)) \\
 X\_F1 &= (FC\_TM1\_11 \vee FC\_TM2\_11) \vee (X\_F1 \wedge \neg(FC\_TF1)) \\
 X\_F2 &= (FC\_TF1) \vee (X\_F2 \wedge \neg(FC\_TF2)) \\
 X\_A\_S &= (FC\_T\_A\_W) \vee (X\_A\_S \wedge \neg(FC\_T\_A\_S)) \\
 X\_A\_W &= (FC\_T\_A\_S) \vee (X\_A\_W \wedge \neg(FC\_T\_A\_W)) \\
 X\_NP &= (FC\_TPT1 \vee FC\_TPT2) \vee (X\_NP \wedge \neg(FC\_TNP\_a \vee FC\_TNP\_b)) \\
 X\_PT1 &= (FC\_TNP\_a) \vee (X\_PT1 \wedge \neg(FC\_TPT1)) \\
 X\_PT2 &= (FC\_TNP\_b) \vee (X\_PT2 \wedge \neg(FC\_TPT2))
 \end{aligned}$$

FIGURE D.2 – Calcul des variables d'activité.

$$\begin{aligned}
 T\_X\_P2\_4s &= X\_P2 \\
 T\_X\_F1\_3s &= X\_F1 \\
 T\_X\_M1\_4\_3s &= X\_M1\_4 \\
 T\_X\_M1\_9\_4s &= X\_M1\_9 \\
 T\_X\_M2\_4\_3s &= X\_M2\_4 \\
 T\_X\_M2\_9\_4s &= X\_M2\_9 \\
 M1\_STILL &= X\_M1\_3 \wedge X\_M1\_4 \wedge X\_M1\_5 \wedge X\_M1\_8 \wedge X\_M1\_9 \wedge X\_M1\_10 \\
 M2\_STILL &= X\_M2\_3 \wedge X\_M2\_4 \wedge X\_M2\_5 \wedge X\_M2\_8 \wedge X\_M2\_9 \wedge X\_M2\_10 \\
 P &= X\_A\_W \\
 V\_G\_OUT &= X\_P1 \wedge X\_P2 \wedge X\_P5 \wedge X\_T1 \wedge X\_M1\_3 \wedge X\_M1\_4 \wedge X\_M1\_8 \wedge X\_M1\_9 \wedge X\_M2\_3 \\
 &\quad \wedge X\_M2\_4 \wedge X\_M2\_8 \wedge X\_M2\_9 \\
 H\_X\_G\_OUT &= (X\_P4 \vee \neg(H\_X\_MID)) \wedge (X\_M1\_1 \vee \neg(H\_X\_OUT)) \wedge X\_M2\_6 \wedge X\_M2\_11 \\
 H\_X\_G\_IN &= (X\_M2\_1 \vee \neg(H\_X\_IN)) \wedge (X\_F2 \vee \neg(H\_X\_IN)) \\
 H\_Y\_G\_OUT &= (X\_P4 \vee \neg(H\_Y\_MID)) \wedge X\_M1\_6 \wedge X\_M1\_11 \wedge (X\_M2\_1 \vee \neg(H\_Y\_OUT)) \\
 H\_Y\_G\_IN &= (X\_M1\_1 \vee \neg(H\_Y\_IN)) \wedge (X\_F2 \vee \neg(H\_Y\_IN)) \\
 IHM\_L\_P &= X\_A\_W \\
 IHM\_L\_PT1 &= X\_PT1 \\
 IHM\_L\_PT2 &= X\_PT2 \\
 TEST\_GO &= X\_T1
 \end{aligned}$$

FIGURE D.3 – Calcul des valeurs des sorties.

## **D.2 Première version du modèle du contrôleur du manipulateur**

Ce modèle se compose du modèle du contrôleur contenant les équations algébriques représentant la spécification Grafcet. Les trois jeux d'équations sont détaillés dans la section précédente, mais on peut noter que :

- Le calcul des conditions de tir (figure D.1) prend en compte les variables issues des temporisations.
- Le calcul des étapes actives (figure D.2) ne comporte pas de relation aux temporisations.
- Le calcul des valeurs des sorties (figure D.3) comporte les variables de contrôle des temporisations.

Ce modèle du contrôleur est complété par les modèles des temporisations associées aux étapes P2, M1-4, M2-4, M1-9, M2-9 et F1.

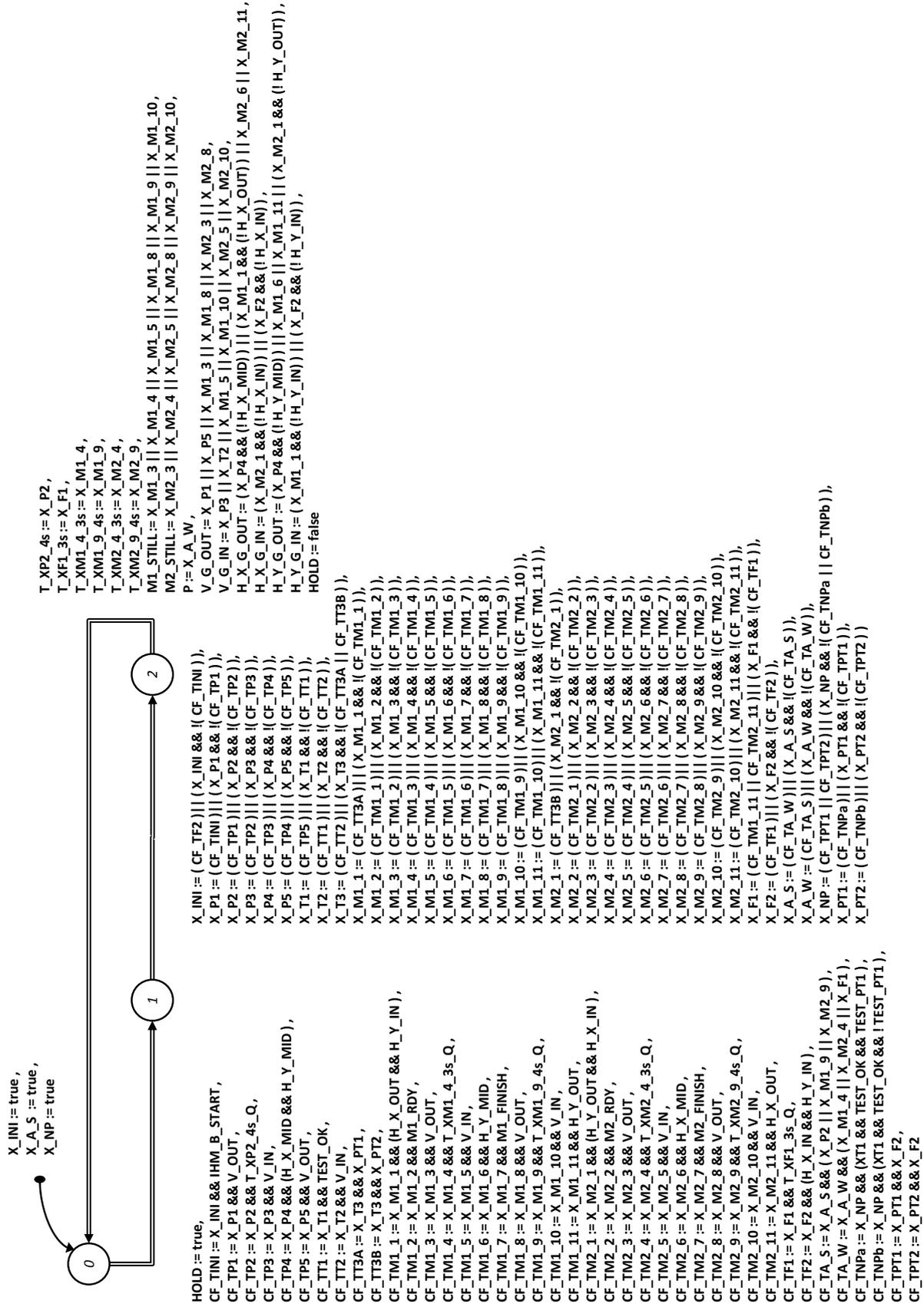
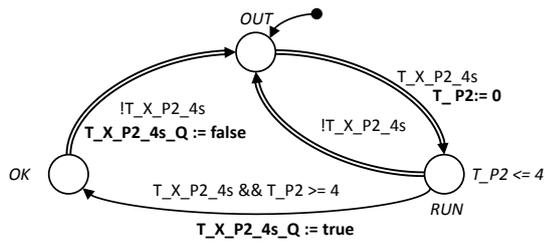
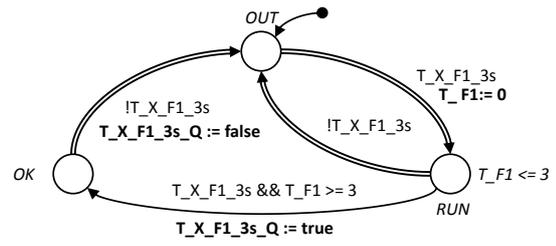


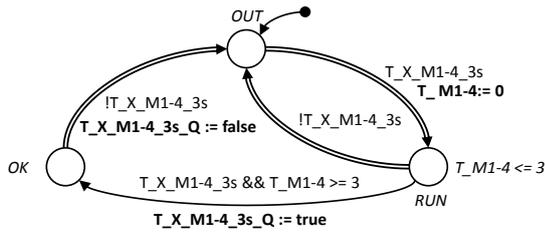
FIGURE D.4 – Premier modèle du contrôleur de l'exemple.



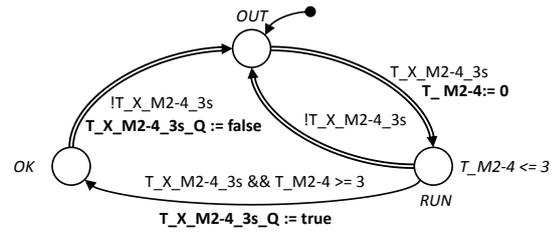
(a) Modèle de la temporisation associée à l'étape P2.



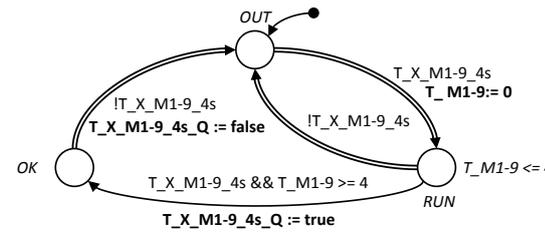
(b) Modèle de la temporisation associée à l'étape F1.



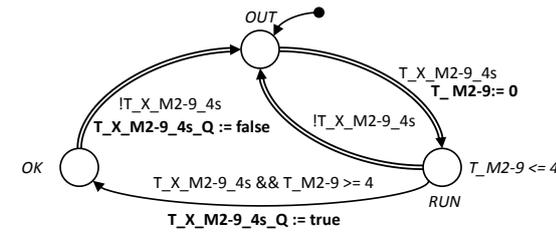
(c) Modèle de la temporisation associée à l'étape M1-4.



(d) Modèle de la temporisation associée à l'étape M2-4.



(e) Modèle de la temporisation associée à l'étape M1-9.

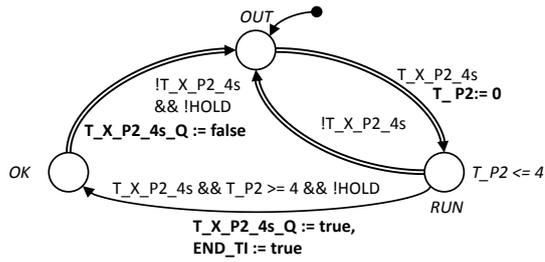


(f) Modèle de la temporisation associée à l'étape M2-9.

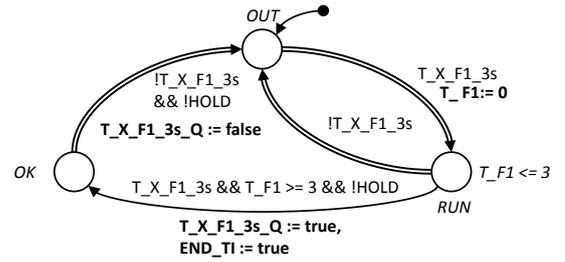
FIGURE D.5 – Modèles des temporisations de l'exemple.

## D.3 Modèle final avec recherche de stabilité du contrôleur

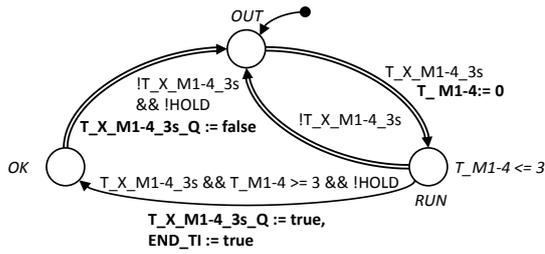
Les modèles des temporisations (figure D.6) ont été modifiés pour prendre en compte la variable HOLD et la variable END\_TI. Le modèle du contrôleur (figure D.7) comprend les modifications relatives aux variables EVOL\_PL et END\_TI, ainsi que la recherche de stabilité au travers de la variable *stable*.



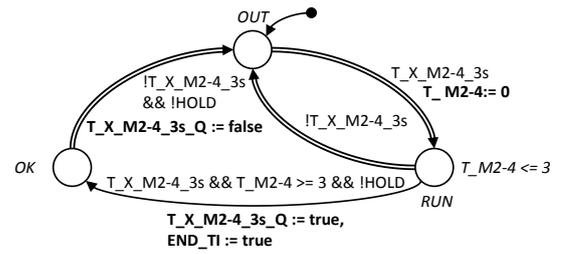
(a) Modèle final de la temporisation associée à l'étape P2.



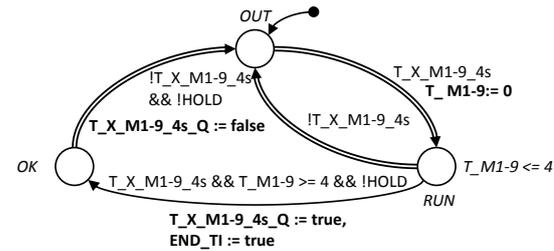
(b) Modèle final de la temporisation associée à l'étape F1.



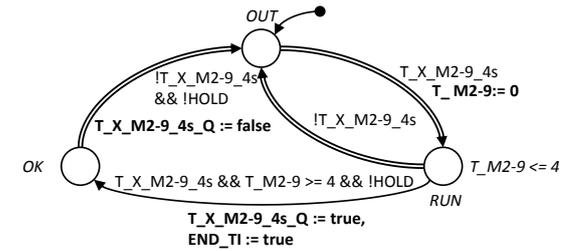
(c) Modèle final de la temporisation associée à l'étape M1-4.



(d) Modèle final de la temporisation associée à l'étape M2-4.



(e) Modèle final de la temporisation associée à l'étape M1-9.



(f) Modèle final de la temporisation associée à l'étape M2-9.

FIGURE D.6 – Modèles finaux des temporisations de l'exemple.

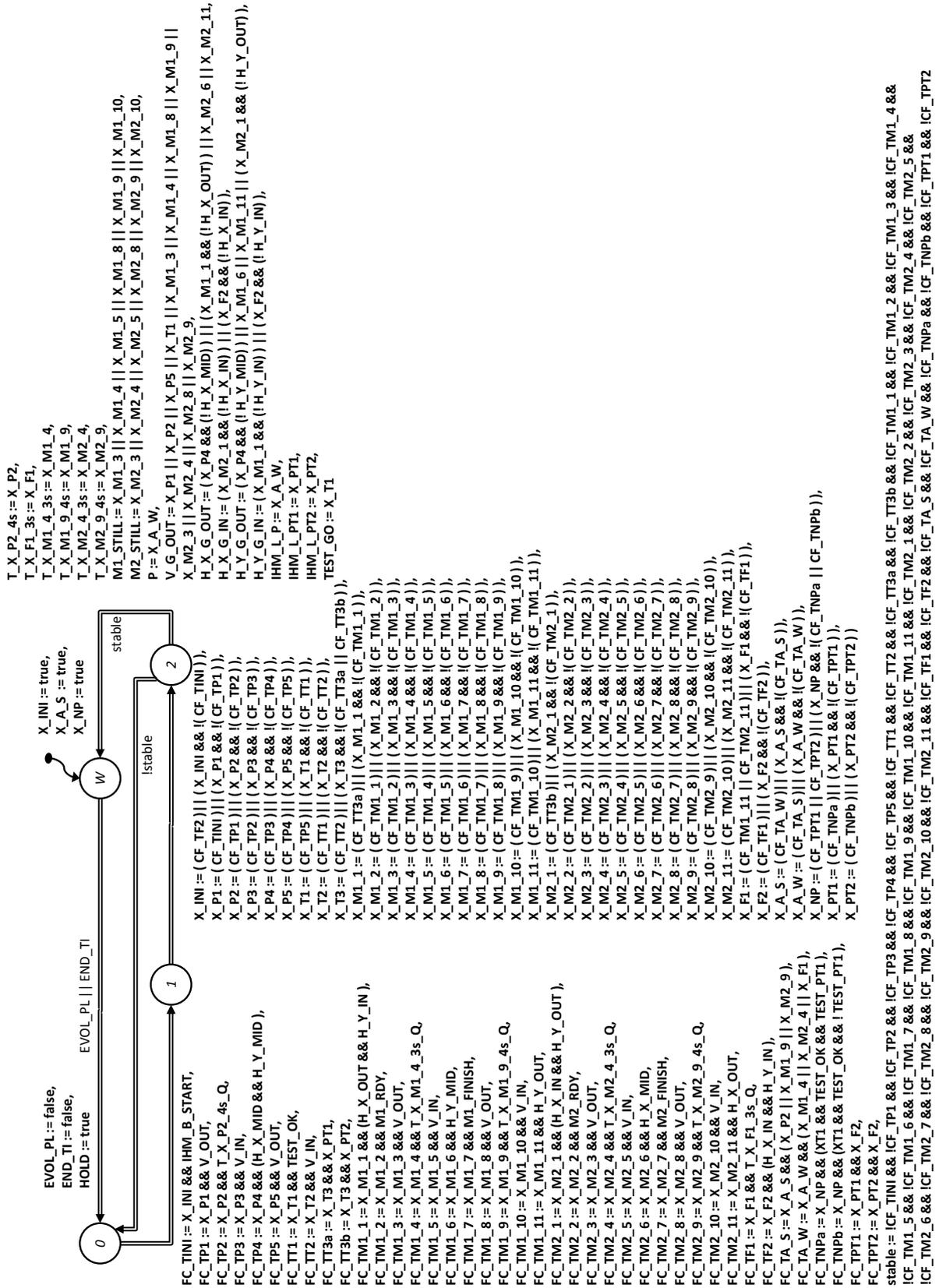


FIGURE D.7 – Modèle final du contrôleur de l'exemple.

## Annexe E

# Séquences d'évolutions et traces relatifs à la vérification de l'équivalence de traces

## E.1 Séquence d'évolutions avec ajout de l'Automate Observateur Séquenceur

La figure E.1a montre l'état initial des trois modèles. Lorsque le modèle A évolue (figure E.1b), il assigne à vrai la variable OUT\_A. Avec les modifications apportées, il assigne également la variable FREEZ à vrai, ce qui empêche toute évolution des modèles A et B. Après la réaction de l'AOS (figures E.1c et E.1d), l'indice de trace du modèle A ( $i_A$ ) vaut 1 alors que celui du modèle B est resté à 0 (pas d'évolution de B) : la comparaison des variables OUT\_A et OUT\_B ne peut pas se faire. Une fois que B aura évolué (et l'AOS fait passer  $i_B$  à 1), on aura le même indice de trace pour les deux modèles, et on pourra comparer les valeurs de leurs variables. Le model-checker confirme que les deux modèles sont équivalents en trace car l'état *KO* de l'AOS n'est pas atteignable.

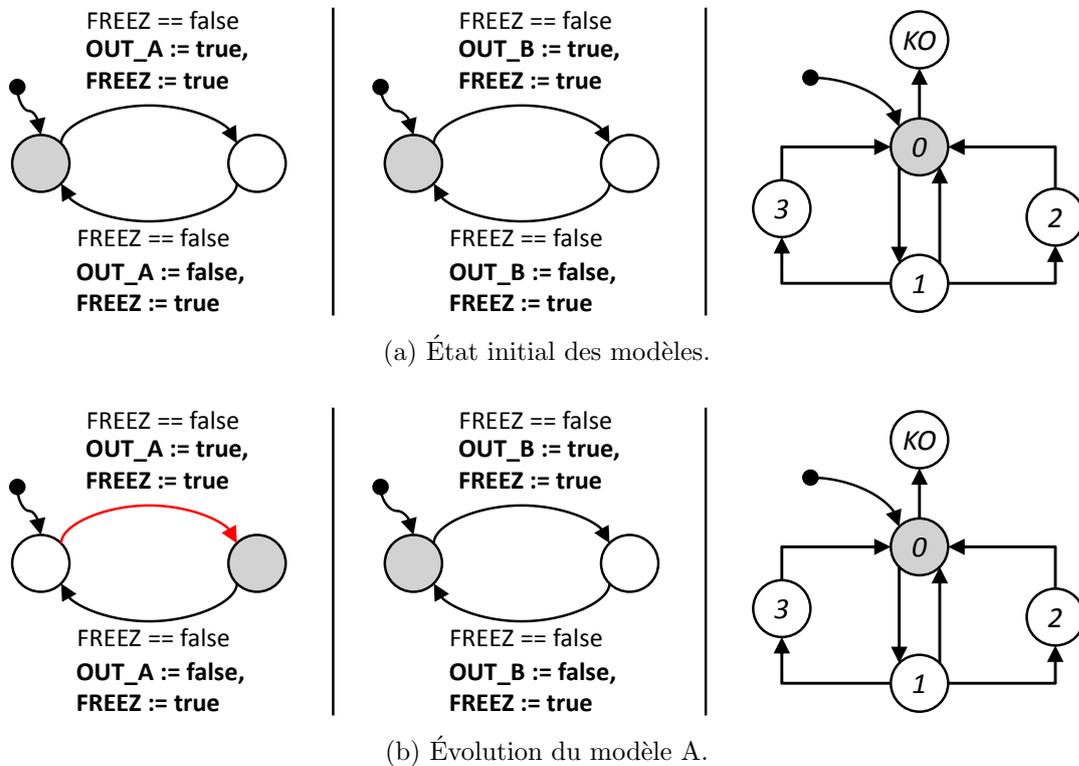
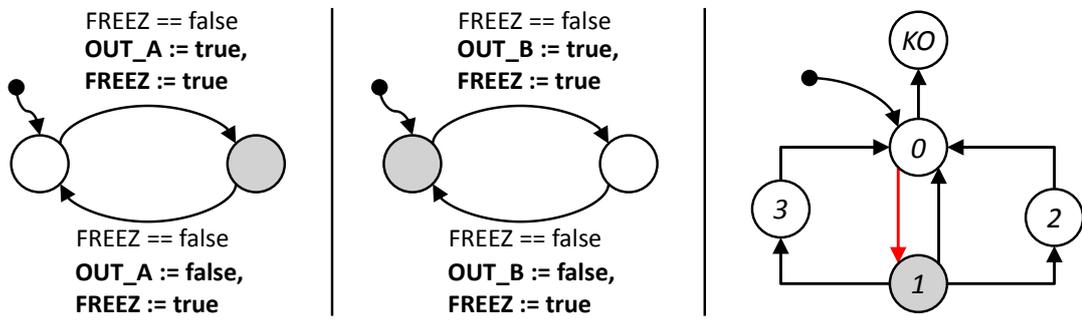
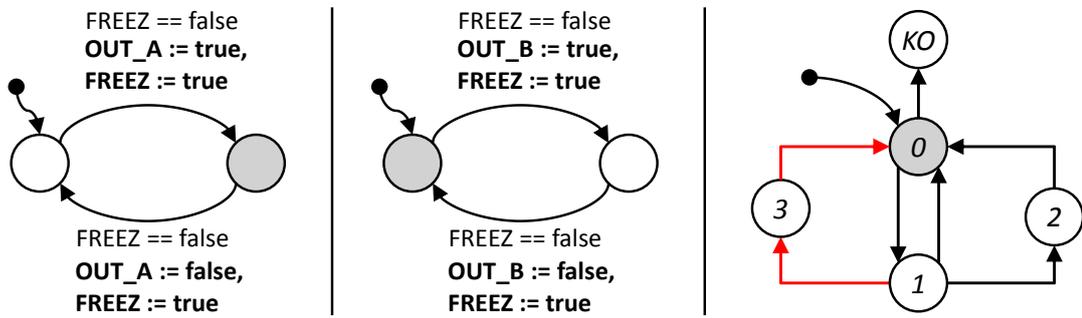


FIGURE E.1 – Séquence d'exemple reprise avec le nouvel AOS (1).



(c) Réaction de l'AOS.



(d) L'AOS fait évoluer l'indice de trace du modèle A et remet FREEZ à faux.

FIGURE E.1 – Séquence d'exemple reprise avec le nouvel AOS (2).

## E.2 Séquence d'évolutions avec un AOS mesurant les décalages temporels de traces

La figure E.2 représente la séquence de la figure 4.24 avec les modèles modifiés. La situation initiale est rappelée dans la figure E.2a. Ensuite, pour des raisons de place, plusieurs évolutions ont été représentées sur la figure E.2b :

- Après 3 unités de temps, le modèle A évolue en  $L1$ .
- Après une autre unité de temps, ce modèle évolue à nouveau. La localité active devient  $L2$  et  $OUT\_A$  est assignée à vrai (initialisée à faux), tout comme  $FREEZ$ .
- Le modèle de l'AOS réagit et passe  $i\_A$  à 1. L'horloge  $Last\_A$  est remise à zéro.
- La transition urgente du modèle A est franchie, la localité active redevient  $L1$ .

À cet instant, plusieurs évolutions concurrentes sont possibles. Nous choisissons de définir l'indice actuel le plus grand ( $i\_A = 1$ ) comme indice de comparaison des traces. La figure E.2c montre ce choix : l'AOS évolue vers la localité  $A1$  (car  $i\_A > i\_B$ ) et la variable  $FREEZ\_A$  passe à vrai. En réaction, le modèle A évolue vers la localité  $LX$  (seule évolution possible).

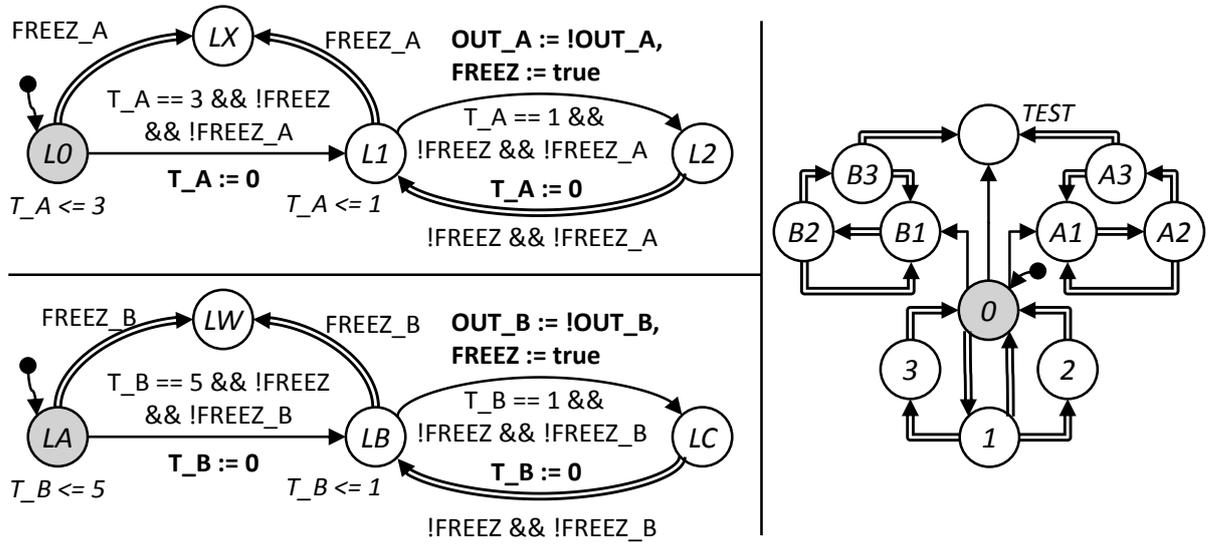
La figure E.2d montre la suite de la séquence, qui ne concerne que le modèle B et l'AOS :

- Après une unité de temps, le modèle B évolue en  $LB$ .  $Last\_A$  évolue de 0 à 1 (car toutes les horloges évoluent simultanément).
- Après encore une unité de temps, le modèle B évolue en  $LC$ . Ce faisant, il assigne  $OUT\_B$  à vrai (initialisée à faux).  $Last\_A$  passe à 2.
- L'AOS réagit : il passe  $i\_B$  à 1 et remet  $Last\_B$  à 0. Comme  $i\_A$  est égal à  $i\_B$ , il ne revient pas en  $A1$  mais la localité  $TEST$  devient active.

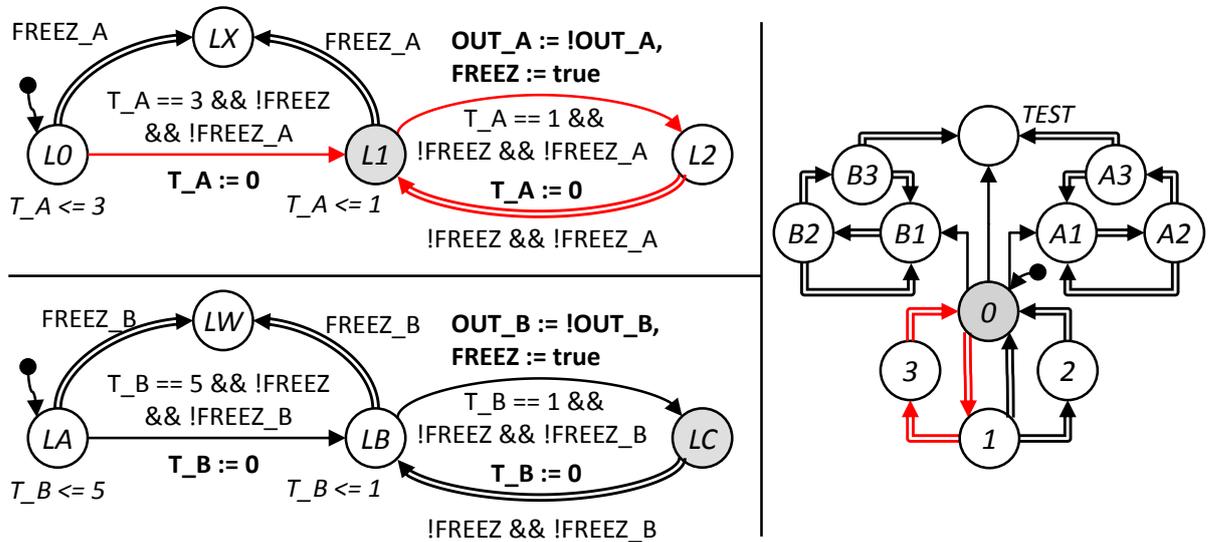
La comparaison des traces à cet indice peut alors avoir lieu :

- $OUT\_A$  est égale à  $OUT\_B$  (toutes deux à vrai).
- $Last\_B$  vaut 0 (elle vient d'être remise à 0 par l'AOS) tandis que  $Last\_A$  vaut 2 (pas remise à zéro pendant les évolutions de B). On retrouve ici les deux unités de temps d'écart des traces : elles ne sont pas équivalentes !

Lors d'un test en utilisant le model-checker, la propriété formelle est vérifiée : on peut atteindre la localité  $TEST$  de l'AOS avec des traces différentes. Les modèles A et B sont *détectés comme non équivalents*.

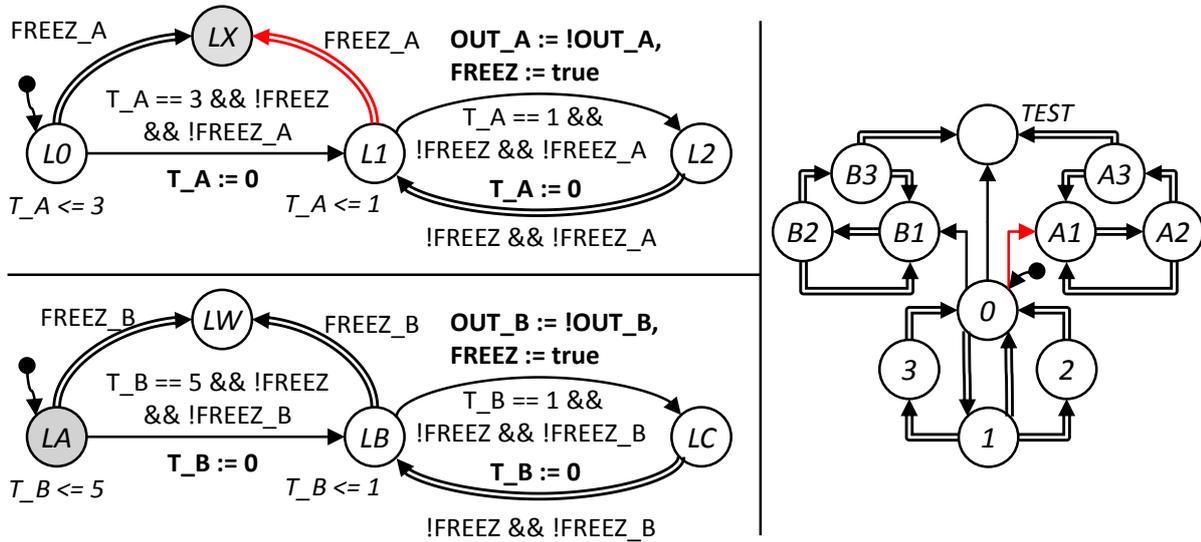


(a) Situation initiale.

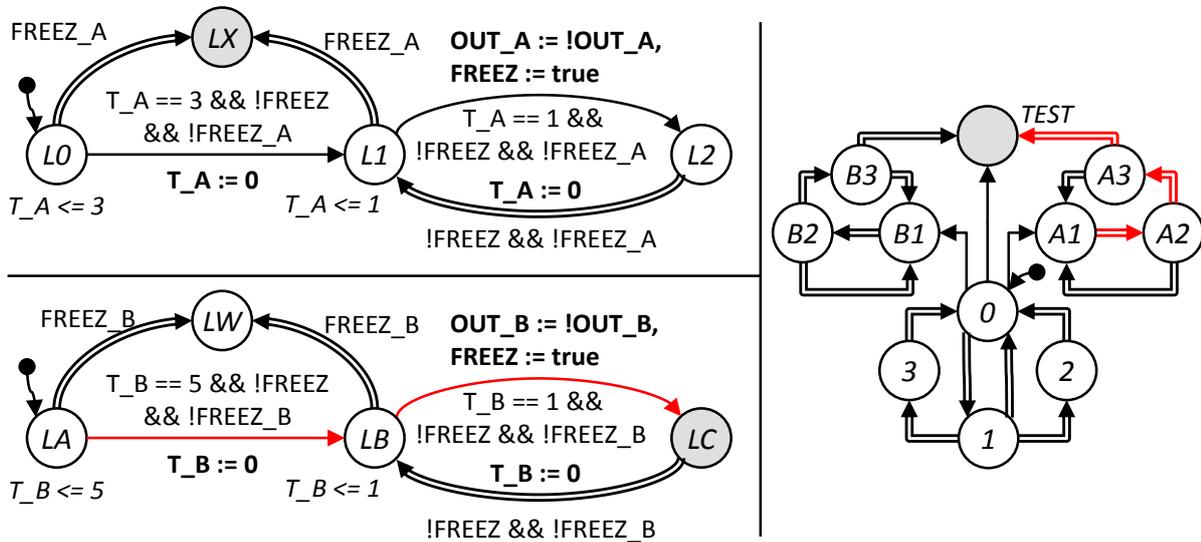


(b) Évolutions du modèle A et réaction de l'AOS.

FIGURE E.2 – Séquence de la figure 4.24 avec les modèles modifiés (1).



(c) Choix de l'indice de comparaison et conséquences.



(d) Évolutions du modèle B et réaction de l'AOS.

FIGURE E.2 – Séquence de la figure 4.24 avec les modèles modifiés (2).

## E.3 Séquence d'évolutions avec modèles modifiés des entrées

La figure E.3 représente la même suite d'évolutions que la figure 4.31 mais avec les modèles modifiés.

La figure E.3a montre l'état initial des modèles, identique à celui de la figure 4.31a. La figure E.3b montre la situation des modèles après plusieurs évolutions identiques à celles présentées dans la figure 4.31b :

- Après 3 unités de temps, les deux modèles A et B évoluent en  $L1$  et  $LB$  respectivement.
- Après une unité de temps, le modèle A évolue en  $L2$  et assigne  $OUT\_A$ .
- Le modèle de l'AOS réagit à ce changement et fait évoluer  $i\_A$ .
- Le modèle B évolue en  $LC$  et assigne  $OUT\_B$ . Cette évolution est possible en dépit de la transition urgente franchissable car, comme  $T\_B$  est déjà égale à 1, la sémantique temporelle n'est pas nécessaire ici.
- L'AOS réagit au changement de  $OUT\_B$  et passe  $i\_B$  à 1.

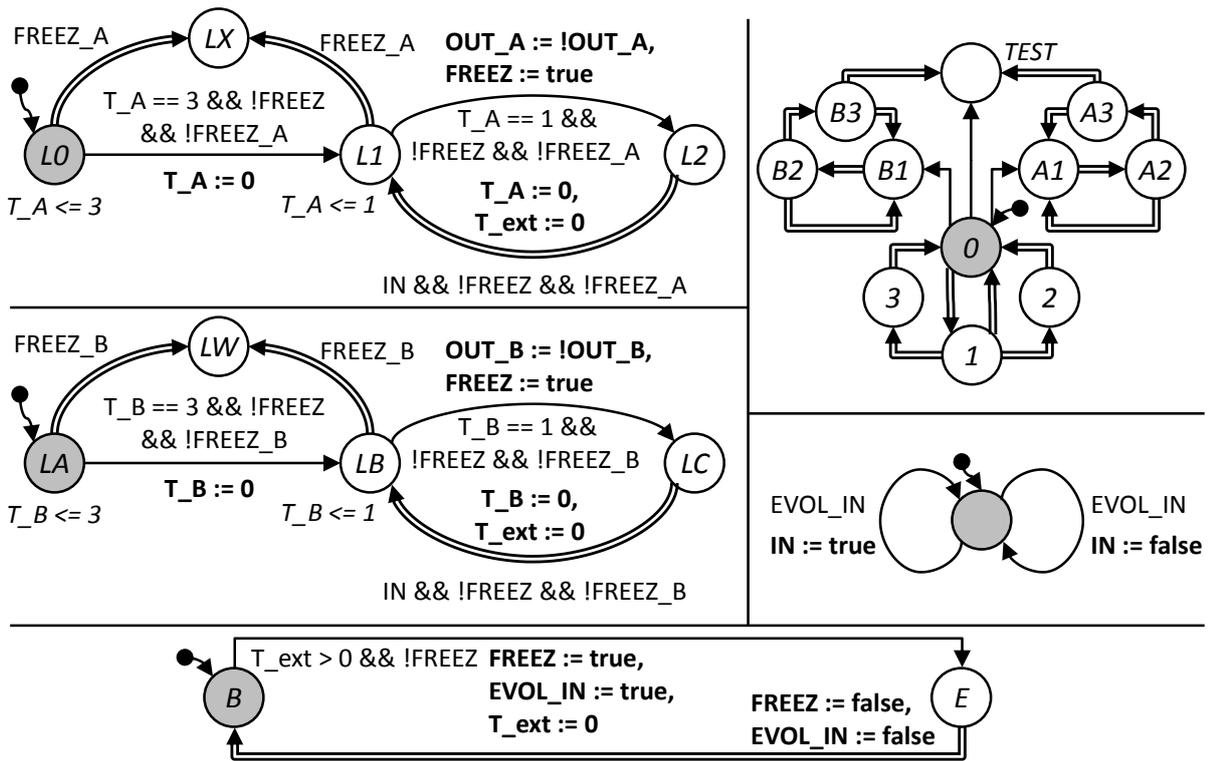
Plusieurs transitions ne consommant pas de temps sont possibles :

- La transition urgente du modèle A.
- La transition urgente du modèle B.
- La transition sans contrainte temporelle du modèle  $EXT\_M$ .
- La transition sans contrainte temporelle de l'AOS allant en  $TEST$ .

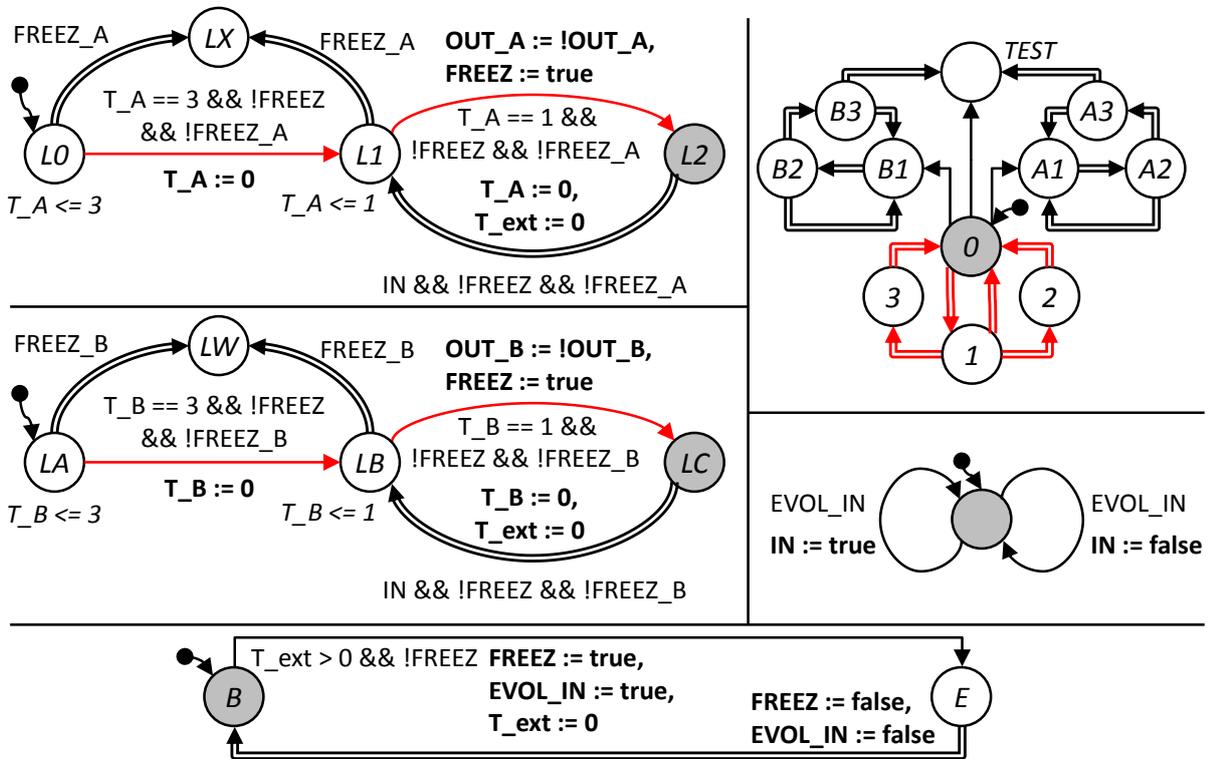
Ensuite, nous choisissons de faire évoluer la trace d'entrée (figure E.3c) comme pour la suite d'évolutions de la figure 4.31 : l'automate  $EXT\_M$  évolue en  $E$ , ce faisant il bloque les évolutions des modèles au travers de  $FREEZ$  et l'horloge  $T\_ext$  est remise à zéro. L'entrée  $IN$  change alors de valeur : elle passe à vrai.

La figure E.3d montre le retour de l'automate  $EXT\_M$  en  $B$  (évolution de la trace d'entrée interdite et relaxe des modèles car  $FREEZ$  repasse à faux). Le franchissement de la transition du modèle A rend la localité  $L1$  active.  $T\_ext$  est remise à zéro.

Contrairement à la suite d'évolutions précédente, la trace d'entrée ne peut plus évoluer à nouveau car  $T\_ext$  est nulle, ainsi le modèle B, par sa transition urgente, garantit l'équité : il peut lui aussi franchir la transition utilisant  $IN$  et n'est plus en retard par rapport au modèle A.

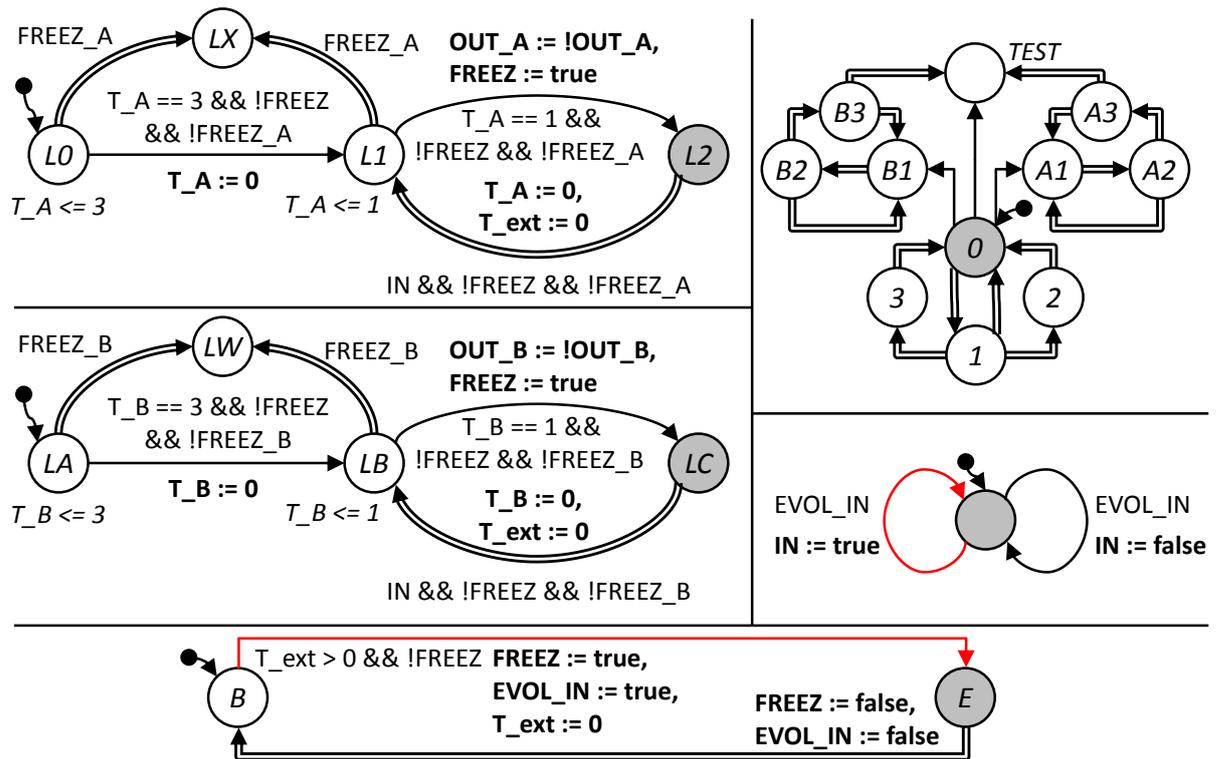


(a) État initial.

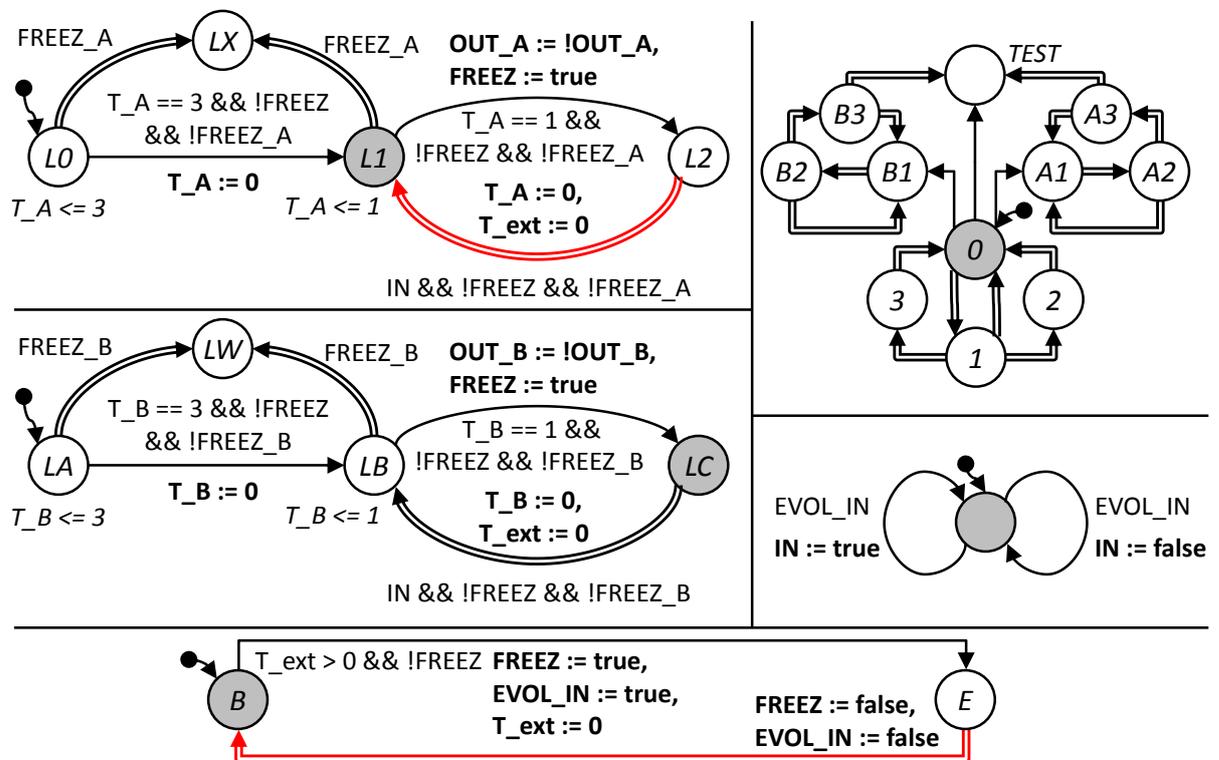


(b) Évolutions des modèles A et B et de l'AOS.

FIGURE E.3 – Suite d'évolutions avec les modèles modifiés pour la concurrence des modèles des entrées (1).



(c) Évolution de la trace d'entrée (choix).



(d) Évolution du modèle A, les entrées ne peuvent plus changer immédiatement, le modèle B n'est plus bloqué.

FIGURE E.3 – Suites d'évolutions avec les modèles modifiés pour la concurrence des modèles des entrées (2).

## E.4 Modèles et traces servant d'exemple aux équivalences avec tolérances

### E.4.1 Modèles et traces d'exemple de l'équivalence avec tolérance en valeur

La figure E.4 montre l'exemple de deux ATVD A et B avec comme entrée commune  $e$  et comme sorties les variables  $a, b$  et  $x$ . Ils sont équivalents de traces avec une tolérance  $\Delta V$  définie pour  $B \cong_{\Delta V} A$  à valeurs dans  $V_e = \{a, b, x\}$ , telle que :

- $\Delta V(a) = 0$
- $\Delta V(b) = [-1; 1]$
- $\Delta V(x) = [-2; 0]$

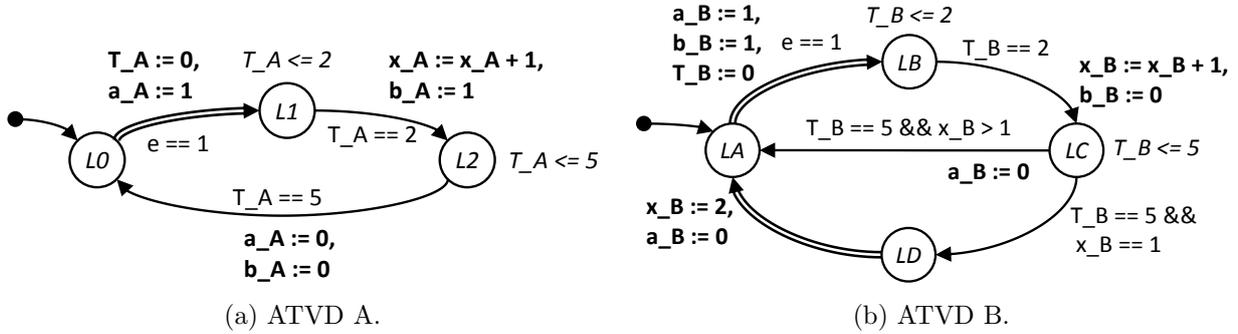


FIGURE E.4 – Exemple d'ATVD équivalents.

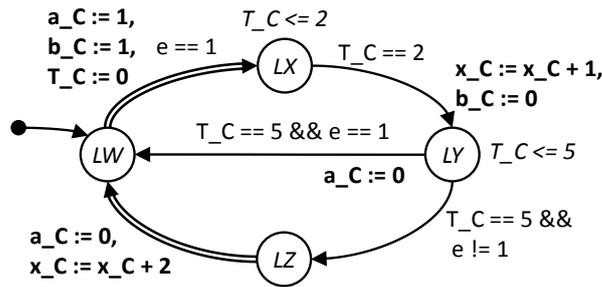


FIGURE E.5 – Exemple d'un ATVD C non équivalent au modèle A.

La figure E.5 montre un modèle C non équivalent au modèle A en utilisant l'équivalence de traces  $C \cong_{\Delta V} A$  avec la même fonction  $\Delta V$ . La table E.1 montre la trace d'entrée (E.1a) qui donne les traces de sortie de A (E.1b) et C (E.1c). On remarque alors dans la trace de C que  $x_C$  arrive à 6, ce qui est hors de l'intervalle autorisé (car  $x_A - x_C = 2 - 6 = -4 \notin [-2; 0] = \Delta V(x)$ ).

$e$	0	1	1	0	0	1	1	0	0
$t$	0	0	4	4	6	6	8	8	12

(a) Trace d'entrée donnée aux modèles A et C.

$a_A$	0	1	1	1	1	0	0	1	1	1	1	0
$b_A$	0	0	0	1	1	0	0	0	0	1	1	0
$x_A$	0	0	0	1	1	1	1	1	1	2	2	2
$t$	0	0	2	2	5	5	6	6	8	8	11	11

(b) Trace de sortie de A.

$a_C$	0	1	1	1	1	0	0	1	1	1	1	0
$b_C$	0	1	1	0	0	0	0	1	1	0	0	0
$x_C$	0	0	0	1	1	3	3	3	3	4	4	6
$t$	0	0	2	2	5	5	6	6	8	8	11	11

(c) Trace de sortie de C.

TABLE E.1 – Traces d'entrée et de sortie montrant la non équivalence des modèles A et C.

## E.4.2 Modèles et traces d'exemple de l'équivalence avec tolérance temporelle et conservation de la chronologie

La figure E.6 montre l'exemple de deux ATVD A et B équivalents de traces avec une tolérance  $\Delta T = [-1; 0]$ .

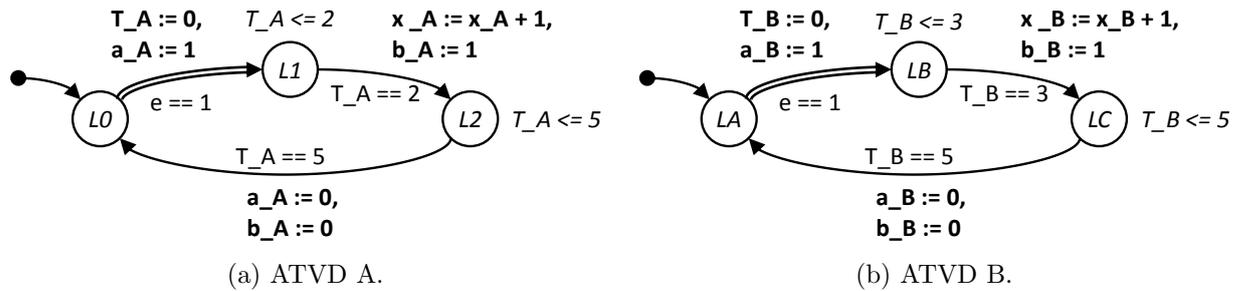


FIGURE E.6 – Exemple d'ATVD équivalents.

La figure E.7 montre un modèle C non équivalent au modèle A en utilisant l'équivalence de traces  $C \cong_{\Delta T} A$  avec la même tolérance  $\Delta T$ . La table E.2 montre la trace d'entrée (E.2a) qui donne les traces de sortie de A (E.2b) et C (E.2c). On remarque alors dans la trace de C que la date du dernier indice de la trace ne respecte pas la tolérance (car  $t^A - t^B = 4 - 6 = -2 \notin [-1; 0] = \Delta T$ ).

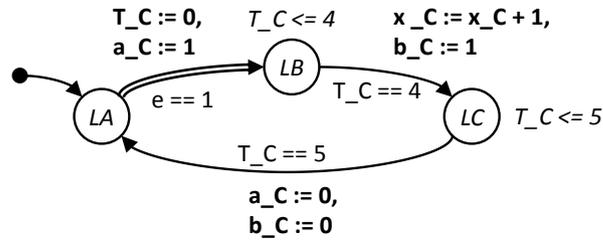


FIGURE E.7 – Exemple d'un ATVD non équivalent à l'ATVD A.

$e$	0	0	1	1
$t$	0	2	2	10

(a) Trace d'entrée donnée à A et C.

$a_A$	0	0	1	1	<b>1</b>
$b_A$	0	0	0	0	<b>1</b>
$x_A$	0	0	0	0	<b>1</b>
$t$	0	2	2	4	<b>4</b>

(b) Trace de sortie de A.

$a_C$	0	0	1	1	<b>1</b>
$b_C$	0	0	0	0	<b>1</b>
$x_C$	0	0	0	0	<b>1</b>
$t$	0	2	2	6	<b>6</b>

(c) Trace de sortie de C.

TABLE E.2 – Traces d'entrée et de sortie montrant la non équivalence des modèles A et C.



## Résumé

L'analyse d'une chaîne de production manufacturière complète, composée de plusieurs systèmes bouclés temporisés, en utilisant des modèles détaillés est presque impossible à cause des problèmes liés aux tailles des modèles. Une solution pour contourner ces difficultés consiste à utiliser des techniques d'analyse multi-échelles utilisant à la fois des modèles détaillés des système bouclés, lorsque nécessaire, mais aussi des modèles abstraits de certains systèmes bouclés dès que possible.

Afin de garantir les résultats lors d'analyses multi-échelles, il convient de garantir que les modèles détaillés des systèmes bouclés ne permettent pas d'évolutions indésirables ne représentant pas un comportement réaliste des système bouclés, et que les modèles abstraits des système bouclés utilisés pendant l'analyse ont un comportement identique aux modèles détaillés de ces mêmes systèmes bouclés.

La première contribution de ces travaux est une méthodologie de conception de modèles détaillés, utilisant des automates temporisés, de systèmes bouclés temporisés permettant de supprimer les évolutions irréalistes. Les solutions proposées pour y parvenir se basent sur une conception modulaire, des mécanismes d'urgence et des variables partagées.

La seconde contribution de ces travaux est axée sur la vérification de l'équivalence entre les modèles détaillé et abstrait d'un même système bouclé. Pour ce faire, une équivalence conforme aux exigences d'une analyse multi-échelles est retenue, puis une méthode basée sur un automate observateur-séquenceur couplé à un model-checker est décrite. Enfin, la prise en compte d'une équivalence avec tolérances (en valeur et/ou temporelle) est détaillée.

## Abstract

The analysis of a complete industrial production line, composed of several closed-loop systems, using detailed models is almost impossible due to size-related issues. A solution consists in performing a multi-scale analysis which uses some abstract models in place of detailed ones. In order to guarantee the analysis result, the detailed models need to have a correct behavior, and the abstract model of a closed-loop system has to be equivalent to the detailed model of the same closed-loop system.

The first contribution details the construction process and the solutions used to build a correct –exempt from unrealistic evolutions– model of a timed closed-loop system. This is achieved by using an urgency semantics upon timed automata with synchronization variables.

The second contribution consists in proposing an equivalence which is suitable for the multi-scale analysis and then in proposing a technique –using formal methods– to prove the equivalence between the abstract and detailed models.