UNIVERSITÉ DE RENNES 1

THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Européenne de Bretagne

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Informatique

École doctorale Matisse

présentée par

Sébastien CHÉDOR

préparée à l'unité de recherche INRIA – UMR6074
INRIA Université de Rennes 1

Diagnostic, opacité et test de conformité pour des systèmes récursifs.

Thèse soutenue à Rennes
le 7 Janvier 2014

devant le jury composé de :

David PICHARDIE
Professeur ENS Cachan / *Président*

Didier CAUCAL
DR de recherche CNRS / *Rapporteur*

Stefan HAAR
DR de recherche INRIA / *Rapporteur*

Nikolai KOSMATOV
Chargé de recherche CEA / *Examinateur*

Thierry JÉRON
DR de Recherche INRIA / *Directeur de thèse*

Christophe MORVAN
Maître de conférence, Université Marne la Vallée / *Co-directeur de thèse*

Remerciements

Je remercie Didier CAUCAL et Stephan HAAR d'avoir accepté d'être les rapporteurs de ce document et d'avoir pris le temps de le relire. Je les remercie également pour les retours qu'ils m'ont fait, qui m'ont permis d'améliorer la qualité de ce document.

Je remercie également les autres membre du jury, en particulier Nokolai KOSMATOV, pour avoir fait le déplacement et pour les retours qu'ils m'a fait. Je remercie aussi David PICHARDIE d'avoir présidé ce jury.

Je remercie Christophe MORVAN pour sa patience à m'encadrer tout au long de ces années. Pour m'avoir toujours soutenu, y compris dans les moment les plus difficiles. Je le remercie aussi pour sa capacité à décoder mes idées et pour m'avoir aidé à les organiser ainsi que pour avoir su aiguiller mes recherches. Je le remercie enfin pour le temps qu'il m'a accordé malgré une situation inconfortable entre Rennes et Marne-la-Vallée.

Je remercie Thierry JÉRON, à la fois en temps que directeur de thèse et chef d'équipe. Il a su se montrer patient et disponible, tout en rendant l'ambiance de travail agréable.

Je remercie l'ensemble de l'équipe VerTeCS, Hervé, Nathalie, Yliès, Amélie, Srinivas, Paulin, Nicolas, et de façon plus large l'équipe SUMO. Je pense qu'il est rare de trouver une équipe de recherche aussi accueillante J'ai toujours pu trouver des conseil pour progresser dans cette thèse. Je remercie plus particulièrement Hervé ainsi que Sophie avec lesquels j'ai travaillé.

Je remercie aussi l'ensemble de mon groupe d'amis, Loïg, Amélie et Philippe, Paulin, Nicolas, Nathanaël, Benjamin, Koubi, Ali, Tomo, Bastien, Aurélien, Yoann, Jeff, Delphine, Pierre Emmanuel, Mickaël, Philippe et j'espère ne pas en oublier, pour toutes les discussions que j'ai pu avoir avec eux et pour m'être souvent dit, et grâce à eux, que cette thèse m'a aussi beaucoup apporté sur le plan personnel.

Enfin je remercie ma famille, mes parents et surtout ma soeur Carine ainsi que Jean-Jacques, pour avoir toujours été là et pour m'avoir énormément soutenu. Je remercie plus généralement tous les membre de ma famille qui m'ont toujours dit que je pouvais compter sur eux en cas de besoin et qui m'ont même proposé de relire mon manuscrit.

Table des matières

Table des matières **3**

Table des figures **5**

Liste des notations **6**

Introduction **7**

1 Présentation des modèles **13**
1.1 Notations mathématiques . . . 13
1.1.1 Ensembles . . . 13
1.1.2 Monoïdes . . . 14
1.1.3 Relations . . . 14
1.2 Les graphes et leurs extensions . . . 15
1.2.1 Graphes . . . 15
1.2.2 Graphes étiquetés . . . 17
1.2.3 Extensions des graphes étiquetés . . . 18
1.3 Propriétés des automates . . . 20
1.3.1 Déterminisation . . . 20
1.3.2 Opérations rationnelles sur les automates . . . 22
1.4 Observation partielle . . . 26
1.4.1 Clôture d'un DES . . . 27
1.4.2 Estimateurs d'état. . . . 28

2 Diagnostic et Opacité **33**
2.1 Problèmatique de l'observation partielle . . . 33
2.1.1 Modélisation des propriétés . . . 33
2.1.2 Supervision . . . 35
2.2 Opacité . . . 36
2.3 Diagnosticabilité . . . 38
2.4 Diverses extensions . . . 40
2.4.1 Opacité bornée synchrone . . . 40
2.4.2 Opacité bornée asynchrone . . . 43

2.4.3 Opacité infinie synchrone . . . 45
2.4.4 Opacité infinie asynchrone . . . 45
2.4.5 Détection de fuites d'informations . . . 46

3 Test de conformité **49**
3.1 Généralités sur le test . . . 49
3.2 La relation de conformité IOCO . . . 51
3.2.1 Blocages . . . 51
3.2.2 Définition de la relation de conformité **ioco** . . . 52
3.3 Cas de tests . . . 53
3.3.1 Equations préliminaires . . . 54
3.3.2 Correction . . . 54
3.3.3 Exhaustivité . . . 55
3.3.4 Sévérité . . . 55
3.3.5 Complétude . . . 56
3.4 Testeur canonique . . . 56
3.4.1 Opérations sur les automates . . . 56
3.4.2 Complétion en sortie . . . 57
3.4.3 Déterminisation . . . 57
3.5 Sélection par objectif . . . 58
3.5.1 Produit . . . 59
3.5.2 Analyse de co-accessibilité . . . 59
3.5.3 Verdicts . . . 59
3.5.4 Propriétés des cas de test . . . 60

4 Modèles de la récursivité **63**
4.1 Hiérarchie de Chomsky . . . 63
4.1.1 Grammaires . . . 63
4.1.2 Langages algébriques . . . 65
4.1.3 Automates à pile . . . 65
4.1.4 Résultats utiles sur les automates à pile . . . 66
4.1.5 Propriétés des langages algébriques . . . 66
4.2 Autres modèles de la récursivité . . . 67
4.2.1 Machines récursives à états (RSM) . . . 67
4.2.2 Grammaires déterministes de graphes . . . 68
4.2.3 Automates à pile d'ordre supérieur . . . 69
4.3 Modèles restreints de la récursivité . . . 70
4.3.1 Automates à pile visible . . . 70
4.3.2 Grammaires synchronisées et pondérées . . . 71

5 Systèmes récursifs de tuiles **73**
5.1 Définition des RTS . . . 73
5.2 Tuiles de chemins . . . 77
5.3 Propriétés élémentaires héritées des grammaires de graphe . . . 80
5.3.1 Accessibilité . . . 80
5.3.2 Produit avec un graphe fini . . . 82
5.4 Clôture de la sémantique d'un RTS . . . 83
5.5 RTS pondérées . . . 88
5.5.1 Définition . . . 88
5.5.2 Déterminisation . . . 88
5.5.3 Produit de RTS . . . 89

6 Diagnosticabilité et opacité pour les systèmes récursifs de tuiles **91**
6.1 Résultats de diagnosticabilité et d'opacité pour les systèmes récursifs de tuiles . 91
6.1.1 Diagnosticabilité . . . 91
6.1.2 Opacité . . . 93
6.2 Extensions des problèmes d'observation partielle. . . . 94
6.2.1 Opacité bornée synchrone . . . 94
6.2.2 Opacité bornée asynchrone . . . 95
6.2.3 Opacité infinie synchrone . . . 95
6.2.4 Opacité infinie asynchrone . . . 96
6.2.5 Diagnostic d'une fuite d'information . . . 96

7 Test de systèmes modélisés par les RTS **99**
7.1 Exemple . . . 99
7.2 Construction du testeur canonique . . . 101
7.2.1 Identification des événements admissibles depuis un sommet . . . 101
7.2.2 Calcul de la suspension d'un système engendré par un RTS . . . 102
7.2.3 Clôture de la sémantique d'un RTS . . . 102
7.2.4 Complétion en sortie de la sémantique d'un RTS . . . 102
7.2.5 Déterminisation . . . 103
7.3 Sélection par objectifs . . . 104
7.4 Propriétés des cas de test générés . . . 106
7.5 Génération à la volée . . . 107
7.5.1 Génération des cas de test . . . 108
7.5.2 Propriétés des cas de test générés à la volée . . . 109
7.5.3 Application de la génération de cas de test à la volée . . . 110

Conclusion et perspectives **111**

Bibliographie **113**

Table des figures

1.1 Représentation sagittale de l'automate de l'exemple 1. 19
1.2 Restriction à la partie accessible à partir de {1} de l'automate des parties obtenu à partir de l'automate de la figure 1.1 21
1.3 Deux automates. 23
1.4 Union des deux automates de la figure 1.3 23
1.5 Concaténation des deux automates de la figure 1.3 24
1.6 Itération de l'automate représenté en figure 1.5 24
1.7 Produit entre les deux automates de la figure 1.3 25
1.8 Automate reconnaissant le complémentaire du langage reconnu par l'automate de la figure 1.2 26
1.9 Une τ-transition 29
1.10 $\tau * a$-simulation 29
1.11 $a\tau*$-simulation 29
1.12 Clôture mixte 29
1.13 Un LTS 31
1.14 Estimateur d'état courant 31
1.15 Estimateur d'état initial 31
1.16 Détail des états de l'estimateur d'état initial 31

2.1 Un DES 34
2.2 Un patron de faute 34
2.3 Produit entre le patron de faute de la figure 2.2 et du DES de la figure 2.1 . . . 34
2.4 Un système non-opaque 37
2.5 Moniteur du système représenté en figure 2.4 38
2.6 Un DES avec deux couleurs. 41
2.7 Exécutions possibles du DES de la figure 2.6 correspondant à l'observation *aabb*. 41
2.8 DES $\mathcal{M}_{\|k}$ obtenu avec $k = 2$ et le DES de la figure 2.6 43
2.9 DES $\mathcal{M}_{\tilde{k}}$ obtenu avec $k = 2$ et le DES de la figure 2.6 44

3.1 Illustration de la relation de conformité **ioco** 53
3.2 Un IOLTS 58
3.3 Suspension de l'IOLTS de la figure 3.2 58
3.4 Complétion de l'IOLTS de la figure 3.3 58

3.5 Déterminisation de l'IOLTS de la figure 3.4 . . . 58
3.6 Cas de test. . . . 60

5.1 Axiome du RTS de l'exemple 17. . . . 75
5.2 Tuile A du RTS de l'exemple 17 . . . 75
5.3 Tuile B du RTS de l'exemple 17 . . . 75
5.4 La tuile $\mathcal{T}_{\mathcal{M}}(t^0_{\mathcal{M}})$ . . . 77
5.5 La tuile ${\mathcal{T}_{\mathcal{M}}}^2(t^0_{\mathcal{M}})$ . . . 77
5.6 La tuile A_{11} . . . 79
5.7 La tuile A_{12} . . . 79
5.8 La tuile A_{13} . . . 79
5.9 La tuile A_{21} . . . 79
5.10 La tuile A_{22} . . . 79
5.11 La tuile A_{23} . . . 79
5.12 La tuile A_{31} . . . 79
5.13 La tuile A_{32} . . . 79
5.14 La tuile A_{33} . . . 79
5.15 L'axiome . . . 81
5.16 La tuile $B_{\{1,2\}}$ . . . 81
5.17 La tuile $A_{\{2,3\}}$ . . . 81
5.18 La tuile $A_{\{1,2,3\}}$ . . . 81
5.19 Représentation schématique de la troisième étape de la clôture . . . 86
5.20 Représentation des quatre ensembles d'états décrits pour la cinquième étape de la clôture en § 5.4. . . . 87

7.1 Le RTS modélisant le programme de l'exemple 21 . . . 101
7.2 Exemple d'un testeur canonique. . . . 105
7.3 Exemple d'un cas de test . . . 106

Introduction

L'informatique est la science qui concerne le traitement automatique de l'information par des machines. Elle possède un spectre extrêmement large. On peut citer l'écriture de programmes pour ordinateur, la transformation d'images, l'édition de séquences vidéo, le calcul de trajectoires astronomiques, le stockage de données, les systèmes de communication, et bien d'autres aspects encore.

Les fondements de l'informatique

Les fondements de cette science sont les mathématiques en générale, et en particulier une branche spécifique qu'on désigne souvent par *mathématiques discrètes*. Cette branche des mathématiques étudie les objets discrets (par opposition aux objets continus qui se fondent sur les nombres réels ou complexes), et qui ne sont pas de nature numérique, le plus souvent. Ainsi la théorie des langages s'intéresse aux ensembles composés de mots, eux-même formés de lettres prises dans un alphabet (ensemble fini de symboles). Cette branche de l'informatique a permis de développer un grand nombre de techniques pour programmer les ordinateurs, ou encore pour formaliser le fonctionnement de systèmes complexes. En outre, la théorie des langages interagit avec d'autres disciplines fondamentales telles que la théorie des graphes. Cette dernière connaît également de nombreuses applications, par exemple la gestion de flux.

Aujourd'hui, ces disciplines sont appliquées pour modéliser ou spécifier des systèmes. En effet, les structures critiques et complexes ont besoin d'être représentées de façon précise. Ensuite, lorsqu'on dispose d'une modélisation, de nombreuses possibilités existent. La plus naturelle consiste à produire ces systèmes directement à partir de leur modèle. Une autre approche consiste à faire la supposition que ces dispositifs sont effectivement conformes à leur spécification et à vérifier certaines propriétés sur cette représentation symbolique. Cette dernière approche est, en générale, appelée vérification formelle [58], et est utilisée y compris par les constructeurs de processeurs [37]. Lorsqu'on définit une représentation formelle pour un système, on peut y incorporer un ensemble d'informations relatives à d'éventuels dysfonctionnements, à des comportements confidentiels, ou d'autres propriétés qui pourront être vérifiées.

Quelques modèles formels

Avant d'aborder les problèmes qui peuvent être étudiés en vérification formelle, il convient de donner quelques éléments précis sur la modélisation des systèmes.

De façon générale, dans ce travail, nous étudierons des systèmes à événements discrets (DES). Ces systèmes sont constitués d'un ensemble d'états (contenant quelques sous-ensembles distingués, tels que les états initiaux) ; ces états sont connectés entre eux par des transitions étiquetées par des événements. L'ensemble des événements est fini, alors que les ensembles d'états ou des transitions peuvent être infinis. Lorsque tous les ensembles définissant un DES sont finis, on dit qu'il est lui-même *fini*. Les DES infinis peuvent être de natures très diverses. En vérification formelle, on utilise des DES qui possèdent une description finie. Le modèle le plus simple de tels DES est certainement celui des systèmes à pile. Ces systèmes sont décrits par un ensemble d'états et de transitions, mais, une configurations d'un tel système est définie par un état et un *mot de pile*. Les transitions (dont la description est finie) connectent des configurations selon leurs états et *le sommet* de leur pile. Ainsi, il est éventuellement possible, à partir d'une configuration donnée, d'atteindre, par une suite de transitions, un ensemble infini de configurations (par exemple en faisant croitre la taille de la pile à chaque transition). Les systèmes à pile sont anciens et ont été largement étudiés [33, 56, 8, 14, 68]. Ces automates ont été étendus par les grammaires déterministes de graphes, [29, 22, 23]. Brièvement, ces grammaires sont définies par un ensemble de règles qui sont formées par un arc qui se récrit en un graphe. Sous réserve de restreindre de façon appropriée ces ensembles de règles, on obtient une caractérisation élégante des DES engendrés par les automates à pile.

Il existe bien d'autres approches pour donner une caractérisation finie d'un DES infinis. On peut citer, par exemple les automates temporisés [3], définis par un automate fini auquel est associé une ou plusieurs horloges dont les valeurs contraignent le franchissement des transitions. Comme les valeurs des horloges ne sont pas bornées, a priori, le comportement d'un tel DES est infini. Les réseaux de Petri [59] sont un autre exemple majeur de systèmes infinis décrits par un ensemble fini de règles. Ces réseaux sont définis par un nombre fini de *places*, chacune pouvant contenir une quantité non-bornée de *jetons*. Ils sont utilisés comme modèles de la concurrence, puisque chaque transition est connectée à quelques places, et que deux transitions mettant en jeu des ensembles disjoints de places peuvent être franchies dans un ordre quelconque. Réciproquement, seule une transition, parmi deux en compétition pour une ressource, pourra être franchie.

Dans ce document, on se concentrera sur les modèles de la récursivité que sont les grammaires de graphes. Plus précisément, nous en donnerons une définition alternative : *les systèmes récursifs de tuiles*. Ce choix apporte, à nos yeux, une plus grande simplicité d'exposition.

Domaines d'application : diagnostic, opacité et génération de tests de conformité

Diagnostic. Une branche de la vérification formelle, la *supervision*, s'intéresse à la construction de superviseurs (appelés également moniteurs) qui sont exécutés conjointement au système. Ces outils, fournissent des indications sur l'état réel du système au cours de son exécution. En général, dans ce contexte, le modèle établit une classification des événements qui se produisent dans le système en observables d'une part et inobservables d'autre part. Ainsi le diagnostic, qui sera largement considéré dans ce document, vise à la construction automatique d'un superviseur qui détecte la survenue d'une faute. A l'utilisation, un tel superviseur peut émettre trois signaux : **oui** (la défaillance s'est produite avec certitude), **non** (il est impossible que la défaillance se soit produite), **equi** (il est possible, mais pas certain que la défaillance se soit produite). La formalisation de ce type de problème a été faite en [61]. Une observation importante est que, pour certains systèmes, il est possible que la réponse **equi** soit émise arbitrairement longtemps, alors même qu'une défaillance s'est effectivement produite. Dans un tel cas de figure le superviseur du diagnostic (appelé diagnostiqueur) a un intérêt très limité.

Le *problème de la diagnosticabilité* consiste à déterminer si le meilleur diagnostiqueur risque de répondre perpétuellement **equi** lorsqu'une défaillance s'est effectivement produite. Nous considérerons de façon précise ce problème pour des systèmes ayant un comportement infini. Cette question de la diagnosticabilité a été considérée pour les systèmes finis, avec un algorithme polynomial pour le résoudre [20, 69, 44]. Elle a également été étudiée pour les systèmes temporisés. Ainsi, [66] démontre que la diagnosticabilité est équivalente à l'absence de comportement Zénon dans un automate dérivé de l'original, ce qui induit une borne dans PSPACE. Ces travaux sont étendus dans [15]. Alors que Tripakis considère le problème du diagnostic dans toute sa généralité, il ne s'intéresse pas au diagnostiqueur, qui est construit *à la volée*. L'article de Bouyer *et al.* s'intéresse à un problème légèrement différent : celui de l'existence d'un diagnostiqueur dans une classe particulière d'automates. Ainsi, lorsqu'on souhaite obtenir un automate temporisé déterministe, le problème est complet pour la classe 2EXPTIME. Et il est complet pour PSPACE pour la sous-classe des automates temporisés déterministes *à enregistrement d'événements* ([5]).

Le problème de la diagnosticabilité pour les réseaux de Petri est indécidable, mais il est possible de construire un diagnostiqueur [67]. Ce contexte est étendu dans [10] à l'aide de technique de transformations de graphes, ce qui permet à ce modèle de représenter des systèmes avec de la mobilité, et une évolution de la topologie.

Les travaux conduits dans le présent document sont dans la continuité de [55] qui a considéré le problème de la diagnosticabilité pour les automates à pile visible, et leurs extensions d'ordre supérieur. Les auteurs y établissent l'indécidabilité de ce problème dans le cas général, ce qui les à conduit à définir une restriction simple pour laquelle le problème devient décidable. Dans cette ligne, [48] établit des résultats similaires pour une classe légèrement différente.

Opacité. Concernant la confidentialité, un problème naturel consiste à déterminer si un observateur de l'exécution d'un système sera en mesure de déterminer avec certitude que le système a atteint l'un des états d'un ensemble distingué, considérés comme *secrets*. Cette question a été formulée, et étudiée dans [17, 9, 30]. La complexité de ce problème y est établie : il est complet pour PSPACE. Pour les systèmes infinis, [21] en démontre l'indécidabilité dans le cas général. Ces problèmes ne semblent pas avoir été abordés pour les réseaux de Petri. En revanche, une autre approche consiste à ne plus apporter une réponse binaire à cette question, mais plutôt une réponse probabiliste [11]. Ainsi, le problème devient : un observateur peut-il établir avec certitude que la probabilité que le système ait atteint un état secret est supérieur à une certaine valeur ?

Génération formelle de tests de conformité. La génération formelle de tests de conformité consiste à dériver des séries de tests (qu'on appelle *suite de tests*) à partir d'une spécification du système. Chaque test (appelé *cas de test*) est ensuite exécuté sur une *implémentation* du système (par exemple un programme, ou un ensemble de programmes). Chaque cas de test fournit un *verdict* qui détermine s'il est réussi ou s'il a échoué. Ce mode de fonctionnement a été introduit dans [16]. Pour pouvoir engendrer une suite de tests pertinents à partir d'une modélisation, il est indispensable de pouvoir exprimer avec précision la notion de conformité.

En 1996, Tretmans a défini la relation **ioco** [64]. Cette relation formalise la conformité, et permet ensuite de certifier des propriétés de la suite de tests (tels que la correction ou la sévérité). La relation **ioco** est aujourd'hui largement utilisée dans le contexte du test de conformité des systèmes réactifs. Ces systèmes sont des DES où l'alphabet des événements est une partition entre les entrées qui sont fournies au système par un élément extérieur et les sorties qui sont produites par le système lui-même. Il est également possible qu'un système possède des événements internes qui ne sont pas observables de l'extérieur. Lorsque ces systèmes sont finis, de nombreux travaux, parmi lesquelles [42, 65], ont traité de la génération de suite de tests, ainsi que de leurs propriétés. Des approches générales ont été étudiées pour des systèmes à états infinis [43, 35]. Très récemment, plusieurs travaux ont été conduits pour étendre ces techniques à des systèmes modélisés par des automates temporisés [51, 49, 12].

Pour ce qui est des systèmes modélisés par des automates à pile, un travail les a considérés de façon très restreinte [28].

Plan détaillé

L'essentiel des contributions de ce travail a été publié dans [24, 26, 25, 27]. Elles sont présentés en détails dans les chapitres 5, 6 et 7.

Chapitre 1 : Dans ce chapitre, nous allons introduire les différentes notions formelles nécessaires à ce document.

Dans un premier temps nous présenterons les préliminaires mathématiques indispensables en particulier des notations sur les ensembles et sur les monoïdes. Par la suite, nous présenterons les graphes et leur diverses extensions, en ajoutant l'étiquetage, le coloriage, ou la reconnaissance de mots. En particulier nous définissons les deux modèles finis que nous utilisons, à savoir les systèmes de transitions étiquetées avec entrées et sorties (IOLTS) et les systèmes à événements discrets (DES). Pour avoir une base formelle sur les transformations que nous utiliserons par la suite nous présentons les opérations élémentaires qui sont utilisées sur les automates. Enfin, nous nous intéressons à l'observation partielle de ces systèmes et nous montrons différentes façons de traiter cette spécificité.

Chapitre 2 **:** Lorsque des systèmes modélisés par des DES (ou des IOLTS) sont exécutés, le plus souvent, seule une partie de leur comportement est visible pour un observateur extérieur. Ceci induit le principe d'observation partiel du système à son exécution. Cette remarque permet de définir plusieurs familles de problèmes. Ainsi, le *diagnostic*, formalisé par exemple dans [61, 69], consiste à observer le comportement du système, et déterminer si une défaillance s'est produite. Le problème d'*opacité*, [17, 9, 30], quant-à lui, vise à identifier si un observateur du comportement du système sera à même de déterminer avec certitude si ce dernier a atteint un ensemble états identifiés comme secrets. Dans le chapitre 2, on s'intéresse à ces deux problèmes ainsi qu'à quelques extensions.

Chapitre 3 **:** Ce chapitre présente le *test de conformité* utilisant la relation de conformité **ioco**. Tout d'abord nous définissons **ioco**, puis nous présentons les étapes de la génération de cas de tests à partir de la spécification. Nous montrons comment construire le testeur canonique, puis comment raffiner celui-ci au moyen d'un objectif de test.

Chapitre 4 **:** Ce chapitre est consacré aux modèles de la récursivité. La façon la plus simple pour étendre le modèle des automates finis, tout en conservant des propriétés décidables, consiste à lui adjoindre une pile.

Les modèles reposant sur une pile permettent, de façon naturelle, de représenter les comportements des programmes récursifs. Plus généralement ces modèles ont été considérablement mis à profit en informatique, que ce soit pour leur bon comportement pour définir la syntaxe de langages, ou encore pour définir des langages d'arbres, tel que XML.

Dans ce chapitre, nous examinerons plusieurs modèles de récursivité. Dans un premier temps, nous examinerons ceux-ci sous l'angle des langages pour quelques systèmes de récriture de mots. Puis, nous considérerons des objets plus riches qui engendrent les mêmes langages, mais dont les structures offrent des possibilités plus importantes.

Chapitre 5 **:** Dans ce chapitre, nous présentons notre propre modèle, à savoir les systèmes récursifs de tuiles. Ils servent à modéliser les systèmes récursifs en restant proches des langages de programmation. Tout comme les grammaires déterministes de graphes, il s'agit de générateurs finis des graphes réguliers. Les systèmes récursifs de tuiles, sont un ensemble de tuiles dont

chacune est un graphe fini, et qui peuvent s'assembler entre elles afin de former des graphes de plus grande taille.

Les RTS sont fortement inspirés des grammaires déterministes de graphes à récriture d'hyperarcs. On ne considère que des terminaux d'arité 2, et les graphes engendrés peuvent contenir des états de degré infini. Il s'agit essentiellement d'une reformulation syntaxique. La motivation de cette reformulation provient du fait que les grammaires de graphes sont utilisées dans de très nombreux contextes, le plus souvent pour définir des ensembles de graphes finis. Ainsi, pour lever toute ambiguïté, il a été choisi, dans ce manuscrit, d'utiliser une formulation légèrement différente qui porte l'accent sur les tuiles, qui sont de simples graphes finis, et les supports et frontières qui permettent de les composer. Cependant, les résultats connus sur les graphes réguliers peuvent être simplement adaptés sur les systèmes récursifs de tuiles.

Chapitre 6 : Dans ce chapitre, nous aborderons différents problèmes liés à l'observation partielle, pour les RTS. Plus précisément, il s'agit des problèmes de diagnosticabilité et d'opacité déjà abordés au chapitre 2 et considérés ici sur des DES infinis modélisés par des RTS. Nous verrons que ces problèmes sont indécidables dans le cas général, mais qu'ils deviennent décidables pour une sous-classe décidable de RTS : les $\mathcal{C}$wRTS (introduite au chapitre 5). Nous considérerons aussi quelques unes des extensions au problème de diagnosticabilité proposées au chapitre 2 dans le contexte des RTS.

Chapitre 7 : Dans ce chapitre, nous considérons la génération de cas de tests pour les RTS. Nous nous concentrons en particulier sur les RTS pondérés, qui sont déterminisables, et nous proposons un algorithme de génération hors-ligne qui fonctionne avec par la sélection de comportements spécifiés par un objectif de test décrit par un DES fini. Comme pour le chapitre 3, nous examinons les opérations nécessaires à la génération de test hors-ligne conduite par des objectifs : tout d'abord la suspension, puis, la clôture, la complétion en sortie et enfin la déterminisation permettent d'obtenir un testeur canonique.

Par la suite, dans le contexte de la sélection par objectif, deux autres opérations sont utilisées : le produit avec l'objectif de test et l'analyse de co-accessibilité.

En outre, comme l'opération de déterminisation n'est pas toujours effective, nous proposons aussi une approche en ligne en décrivant un algorithme à la volée.

Chapitre 1

Présentation des modèles

Dans ce chapitre, nous allons introduire les différentes notions formelles nécessaires à ce document.

Dans un premier temps nous présenterons les préliminaires mathématiques indispensables en particulier des notations sur les ensembles et sur les monoïdes. Par la suite nous présenterons les graphes et leur diverses extensions, en ajoutant l'étiquetage, le coloriage, ou la reconnaissance de mots. En particulier nous définissons les deux modèles finis que nous utilisons, à savoir les systèmes de transitions étiquetées avec entrées et sorties (IOLTS) et les systèmes à événements discrets (DES). Pour avoir une base formelle sur les transformations que nous utiliserons par la suite, nous présentons les opérations élémentaires qui sont utilisées sur les automates. Enfin, nous nous intéressons à l'observation partielle de ces systèmes et nous montrons différentes façons de traiter cette spécificité.

1.1 Notations mathématiques

L'ensemble de ce travail repose sur un certain nombre de notions formelles clé.

1.1.1 Ensembles

L'ensemble des entiers naturels est noté $\mathbb{N}$. Si n est un entier, on note $[n] \stackrel{\text{def}}{=} \{0, 1, 2, \cdots, n\}$ l'ensemble des entiers positifs inférieurs ou égaux à n. L'ensemble vide est noté $\emptyset$.

Si E est un ensemble, on note $|E|$ son *cardinal* ; c'est un entier lorsque cet ensemble est fini, c'est le symbole $+\infty$ sinon. Un *couple* est un ensemble ordonné de deux éléments, une *paire* un ensemble non-ordonné de deux éléments.

On utilise les symboles $\in$ et $\subseteq$ pour désigner respectivement l'appartenance et l'inclusion.

La notation 2^E désigne l'ensemble des parties d'un ensemble E. Si E et F sont deux ensembles, on note $E - F$ la *différence* entre ces deux ensembles : $E - F \stackrel{\text{def}}{=} \{x \in E \mid x \notin F\}$. Si F est une partie d'un ensemble E, on note $\overline{F}$ son *complémentaire* dans E, c'est à dire $E - F$. Si E et F sont des ensembles arbitraires, on note $E \times F$ le *produit cartésien* de E et F : l'ensemble des couples (x, y) pour tout x élément de E et pour tout y élément de F. On peut

étendre ce produit à une collection finie d'ensembles, les éléments de l'ensemble construit sont alors des n-uplets.

1.1.2 Monoïdes

Soit M un ensemble, et $\cdot$ une opération binaire sur cet ensemble ; on note $(M, \cdot)$ l'ensemble M muni de $\cdot$.

On dit que $(M, \cdot)$ est un *monoïde* si l'opération $\cdot$ est associative et possède un élément neutre (noté 1_M) : pour tout triplet d'éléments u, v, w dans M, on a $(u \cdot v) \cdot w = u \cdot (v \cdot w)$ et, pour tout élément u dans M, $u \cdot 1_M = 1_M \cdot u = u$.

Une partie $A \subseteq M$ d'un monoïde M est appelée *sous-monoïde* lorsque, $1_M \in A$ et, pour tous u et v éléments de A, $u \cdot v \in A$.

A présent, on définit quelques opérations sur les monoïdes. L'opération d'un monoïde peut s'étendre à l'ensemble de ses parties : on note $A \cdot B$ le produit de deux parties A et B défini par $A \cdot B \stackrel{\mathbf{def}}{=} \{u \cdot v \mid u \in A \wedge v \in B\}$; la *puissance* d'une partie A est définie par récurrence ; on pose $A^0 \stackrel{\mathbf{def}}{=} \{1_M\}$, ensuite, pour n strictement positif, $A^n \stackrel{\mathbf{def}}{=} A \cdot A^{n-1}$, le *plus du produit* est défini à partir de la puissance $A^+ \stackrel{\mathbf{def}}{=} \bigcup_{i=1}^{\infty} A^i$.

On définit également *l'étoile du produit* : $A^* \stackrel{\mathbf{def}}{=} \bigcup_{i=0}^{\infty} A^i$.

On peut remarquer que A^* est le plus petit sous-monoïde de M contenant A. Il est appelé monoïde *engendré* par A.

Un monoïde M est dit *libre* s'il existe une partie $P \subseteq M$, telle que $P^+ = M$ (*resp.* $P^* = M$) et, pour tout élément $u \in M$ il existe une unique suite finie $(u_i)_{i \in [n]}$ d'éléments de P telle que $u = u_1 \cdot u_2 \cdots u_n$ (*resp.* l'élément neutre d'un monoïde est le produit de zéro élément du monoïde).

Langages On s'intéresse à l'étude des monoïdes libres finiment engendrés. Un *alphabet* Σ est un ensemble fini, non-vide, de symboles appelés *lettres*.

Un *mot* w sur Σ, de *longueur* $|w|$ est une application de $[|w|]$ dans Σ, ou de façon équivalente est un $|w|$-uplet $(w(1), \ldots, w(|w|))$ de lettres de Σ noté $w(1) \ldots w(|w|)$. Pour $k \leq |w|$, on note w_k le k-ième préfixe de w, c'est à dire le mot $(w(1), \ldots, w(k))$

On note ε le mot de longueur nulle, appelé *mot vide*. On note $\tilde{w}$ le *miroir* de w, défini par récurrence : $\tilde{\varepsilon} = \varepsilon$, et $\widetilde{aw} = \tilde{w}a$ (où a est une lettre, et w un mot).

L'ensemble Σ^* des mots sur Σ est un monoïde pour le produit de *concaténation* défini par $u \cdot w = u(1) \ldots u(|u|)w(1) \ldots w(|w|)$ pour tous mots u et w sur Σ.

Ainsi Σ^* est le monoïde libre engendré par Σ. Une partie d'un monoïde libre est appelée un *langage* sur Σ.

1.1.3 Relations

Soient E et E' deux ensembles. On appelle *relation* toute partie de $E \times E'$. Si R est une relation, on note R^{-1} la relation $\{(v, u) \mid (u, v) \in R\}$ dans $E' \times E$, dite *relation inverse*. Si

u est un élément de E, l'ensemble $R(u) \stackrel{\text{def}}{=} \{v \mid (u,v) \in R\}$ est l'image de u par R. On peut étendre cette notation aux ensembles : pour $F \subseteq E$, on note $R(F) = \{v \mid \exists u \in F, v \in R(u)\}$. On note souvent $u\,R\,v$ pour $(u,v) \in R$. Le *domaine* (*resp. l'image*) d'une relation est l'ensemble $Dom(R)$ (*resp.* $Im(R)$), défini par : $Dom(R) \stackrel{\text{def}}{=} \{u \mid \exists v \in E', u\,R\,v\}$ (*resp.* $Im(R) \stackrel{\text{def}}{=} \{v \mid \exists u \in E, u\,R\,v\}$) ; ainsi $Im(R) = Dom(R^{-1})$.

Une *relation d'équivalence*, $R \subseteq E \times E$, est une relation *réflexive* (pour tout élément $a \in E$, on a $a\,R\,a$), *symétrique* (pour toute paire d'éléments a, b de E, on a : $a\,R\,b$ entraîne $b\,R\,a$) et enfin *transitive* (pour tout triplet d'éléments a, b, c de E, on a $a\,R\,b$ et $b\,R\,c$ entraîne $a\,R\,c$). Lorsque R est une relation d'équivalence sur E, la classe $[a]_R$ d'un élément a de E est l'ensemble des éléments qui sont en relation avec a : $[a]_R \stackrel{\text{def}}{=} \{b \mid a\,R\,b\}$. On peut alors définir le *quotient* d'un ensemble par une relation : $E/R \stackrel{\text{def}}{=} \{[a]_R \mid a \in E\}$.

Fonctions Une *fonction* est une relation f de $E \times E'$ telle que pour tout élément e de E, on ait $|f(e)| \leqslant 1$. Une *application* est une fonction f telle que $Dom(f) = E$. L'ensemble des applications de E dans E' est noté ${E'}^E$. Une fonction est dite *surjective* si $Im(f) = E'$, *injective* si pour tout u et v dans $Dom(f)$, $f(u) = f(v) \implies u = v$, et enfin *bijective* si elle est à la fois injective et surjective. Les ensembles que nous utiliserons seront, sauf précision contraire, *dénombrables* autrement dit, en bijection avec $\mathbb{N}$.

Un *morphisme* est une application entre deux ensembles E_1 et E_2 munis d'opérations $\cdot_1$ et $\cdot_2$, et telle que, pour tous éléments u et v de E_1, $f(u \cdot_1 v) = f(u) \cdot_2 f(v)$. Un *isomorphisme* est un morphisme bijectif. Un *endomorphisme* est un morphisme d'un ensemble dans lui-même.

1.2 Les graphes et leurs extensions

Nous présentons ici les graphes ainsi que leurs diverses extensions. Si ces modèles sont souvent définis pour un nombre fini d'*états* et de transitions, leur définition reste valable dans le cas infini. Dans ce paragraphe, les résultats sont présentés aussi bien pour des objets finis qu'infinis.

Suivant le contexte, il arrive que des objets identiques soient désignés par des termes différents. Ainsi, on parle en général d'arc et de sommets pour un graphe alors qu'on emploie plutôt transition et états pour les automates. Afin de limiter la profusion de vocabulaire et de notations, certains termes seront donc utilisés à la place des termes habituellement utilisés (nous parlerons, par exemple, d'états et de transitions dans un graphe).

1.2.1 Graphes

Définition 1.1. Un *graphe* G est un couple (Q_G, Δ_G) où Q_G est l'ensemble des *états* et $\Delta_G \subseteq Q_G \times Q_G$ est la relation de transition. Quand les éléments de Δ_G sont des couples, le graphe est dit *orienté*, s'il s'agit de paires, le graphe est dit *non-orienté*.

Dans la suite de ce document, nous considérerons uniquement des graphes orientés.

Accessibilité Étant donné un graphe G, un élément (p, q) de Δ_G est appelé transition et est notée $p \underset{G}{\rightarrow} q$ ou, plus simplement, $p \rightarrow q$, s'il n'y a pas d'ambiguïté sur G. L'état p est la *source* de la transition (p, q) et q est sa *cible* ; q est un *successeur* de p et p est un *prédécesseur* de q. Il est possible d'étendre cette notation aux ensembles, on note $Q_1 \underset{G}{\rightarrow} Q_2$ avec $Q_1, Q_2 \subseteq Q_G$ quand $Q_2 = \left\{ q_2 \in Q_G \mid \exists q_1 \in Q_1, q_1 \underset{G}{\rightarrow} q_2 \right\}$. Comme pour les relations, on note respectivement $Dom(\Delta_G)$ et $Im(\Delta_G)$ l'ensemble des sources et des cibles.

On note G^+ le graphe obtenu par clôture transitive du graphe G défini inductivement : $G_1 \overset{\text{def}}{=} G$, et pour tout $1 \leq n$,

- $Q_{G_n} = Q_G$,
- $\Delta_{G_{n+1}} \overset{\text{def}}{=} \left\{ (p, r) \mid \exists q \in Q_G, p \underset{G}{\rightarrow} q \wedge q \underset{G_n}{\longrightarrow} r \right\}$.

On note $p \underset{G}{\Longrightarrow} q$ si $p \underset{G^+}{\longrightarrow} q$ ou plus simplement $p \Longrightarrow q$ lorsqu'il n'y a pas de doute sur G. Soit $p \in Q_G$, on dit qu'un état $q \in Q_G$ est accessible depuis p lorsque $p \Longrightarrow q$. Comme précédemment, on peut étendre cette notation aux ensemble d'états, on note $Q_1 \underset{G}{\Longrightarrow} Q_2$ si $Q_1 \underset{G^+}{\longrightarrow} Q_2$.

Degré Le *degré sortant* d'un sommet q est ${d^+}_G(q) \overset{\text{def}}{=} |\{p \mid (q, p) \in \Delta_G\}|$, le nombre de transitions issues de cet état. De la même manière, le *degré entrant* d'un sommet q est ${d^-}_G(q) \overset{\text{def}}{=} |\{p \mid (p, q) \in \Delta_G\}|$, le nombre de transitions aboutissant à cet état. Le *degré* $deg_G(q) \overset{\text{def}}{=} {d^+}_G(q) + {d^-}_G(q)$ d'un état q est la somme de ses degrés entrant et sortant. On dit qu'un graphe est de degré fini si tout ses sommets sont de degré fini. On dit qu'il est de degré borné si il existe un entier majorant le degré de chacun de ses sommets. On dit qu'un graphe est de degré infini s'il a un sommet de degré infini. Enfin, on note $d(G) = |\{deg\ (q)|\ q \in Q_G\}|$ le *nombre de degrés* d'un graphe G.

Définition 1.2. Un *graphe coloré* est un quadruplet $G = (Q_G, \Lambda_G, \Delta_G, col_G)$ ou (Q_G, Δ_G) est un graphe, Λ_G est l'ensemble des *couleurs* et $col_G \subseteq Q_G \times \Lambda_G$ est la relation de coloriage.

Les couleurs servent à modéliser des propriétés qui seront attachée aux états.

On remarquera qu'un état peut porter plusieurs couleurs et qu'une même couleur peut être partagée par plusieurs états. On note $col_G(q) \overset{\text{def}}{=} \{\lambda \in \Lambda_G \mid (q, \lambda) \in col_G\}$ et $col_G^{-1}(\lambda) \overset{\text{def}}{=} \{q \in Q_G \mid (q, \lambda) \in col_G\}$.

Définition 1.3. Une *structure de Kripke* est un quintuplet $K = (Q_K, \Lambda_K, Q_K^0, \Delta_K, col_K)$ où $(Q_K, \Lambda_K, \Delta_K, col_K)$ est un graphe coloré, et $Q_K^0 \subseteq Q_K$ est un ensemble d'états initiaux. De plus, tout état de Q_K a au moins un successeur.

Les structures de Kripke sont souvent utilisées en *model checking* : le graphe représente le comportement d'un système qui peut changer d'état, et les couleurs représentent un ensemble de propositions élémentaires qui peuvent être vérifiées ou non dans ces états.

1.2.2 Graphes étiquetés

Les graphes peuvent servir à modéliser des systèmes qui changent d'état. En général, ces systèmes changent d'état à cause d'événements qui se produisent. Pour modéliser ces événements, il est possible d'ajouter des *étiquettes* aux transitions, de la même manière que les couleurs pour les états.

Définition 1.4. Un *graphe étiqueté* G est un triplet $(Q_G, \Sigma_G, \Delta_G)$ où Q_G est l'ensemble des *états*, Σ_G est un alphabet d'événements et $\Delta_G \subseteq Q_G \times \Sigma_G \times Q_G$ est la relation de transition.

En comparaison avec les graphes simples, la relation de transition dépend maintenant aussi de Σ_G. Pour $(p, a, q) \in \Delta_G$, p est toujours la source, q est toujours la cible et a est l'*étiquette*. On définit $\Delta_G(p, a) \stackrel{\text{def}}{=} \{q \in Q_G \mid (p, a, q) \in \Delta_G\}$.

La clôture transitive G^+ est obtenue de la même manière, inductivement, à l'exception des transitions qui sont maintenant étiquetées par des mots : $G_1 \stackrel{\text{def}}{=} G$, et pour tout $1 \leq n$, $\Delta_{G_{n+1}} \stackrel{\text{def}}{=} \left\{(p, aw, r) \mid \exists q \in Q_G, p \xrightarrow[G]{a} q \wedge q \xrightarrow[G_n]{w} r\right\}$. On note $p \xRightarrow[G]{w} q$ si $p \xrightarrow[G^+]{w} q$ ou plus simplement $p \stackrel{w}{\Longrightarrow} q$ lorsqu'il n'y a pas de doute sur G. Soit $p \in Q_G$, on dit qu'un état $q \in Q_G$ est accessible depuis p par le mot w lorsque $p \stackrel{w}{\Longrightarrow} q$. On étend la relation de transition aux mots par $\Delta_G(p, w) \stackrel{\text{def}}{=} \left\{q \in Q_G \mid p \xRightarrow[G]{w} q\right\}$.

Comme précédemment, on peut étendre les notations $\rightarrow$, $\Longrightarrow$ et la relation de transition aux ensembles.

Soit $\Sigma' \subseteq \Sigma_G$, on note $G_{|\Sigma'}$ le graphe restreint aux transitions étiquetées par des éléments de Σ' : $\Delta_{G_{|\Sigma'}} = \Delta_G \cap (Q_G \times \Sigma' \times Q_G)$.

La *trace* (ou ensemble des étiquettes des chemins), $L_G(Q_1, Q_2)$, de G allant de l'ensemble des états Q_1 à l'ensemble des états Q_2 est la partie de Σ_G^+ suivante :

$$L_G(Q_1, Q_2) \stackrel{\text{def}}{=} \left\{w \in \Sigma_G^+ \mid \exists\, p \in Q_1, \exists q \in Q_2, p \xRightarrow[G]{w} q\right\}$$

Un *chemin* γ est une suite de transitions $(q_i, a_1, q_{i+1})_{i \in I}$ indexée par un intervalle I de $\mathbb{N}$ contenant 0. Le sommet q_0 est l'extrémité initiale du chemin, notée $init(\gamma)$. Lorsque I est borné on dit que $q_{|I|+1}$ est l'extrémité finale du chemin, notée $fin(\gamma)$.

Propriétés locales Un graphe étiqueté G est *localement déterministe* si deux transitions distinctes de même source ont des étiquettes distinctes : $p \xrightarrow[G]{a} q \ \wedge \ p \xrightarrow[G]{a} q'$ implique $q = q'$.

Un graphe G est dit *complet* si, pour chaque étiquette a, tout état est source d'au moins une transition étiquetée par a : $\forall a \in \Sigma_G, \forall p \in Q_G, \exists p \xrightarrow[G]{a} q \in \Delta_G$.

Relations entre graphes L'*isomorphisme* de graphes correspond à l'idée d'abstraire le nom des états d'un graphe. Précisément, deux graphes étiquetés G et G' sont dits *isomorphes*, s'il existe une bijection f de Q_G sur $Q_{G'}$ telle que en posant $\Sigma_G \cup \Sigma_{G'} = \Sigma$: $\forall a \in \Sigma, \forall p, q \in Q_G, p \xrightarrow[G]{a} q \Leftrightarrow h(p) \xrightarrow[G']{a} h(q)$.

La *bisimulation* correspond à l'isomorphisme « à redondance de structure près ». Elle a été introduite initialement par Park [57] et reprise par Milner [54]. Tout comme l'isomorphisme, il s'agit d'une relation définie entre les états de deux graphes.

Tout d'abord, une *simulation* est une relation entre les états de deux graphes : une relation $R \subseteq Q_G \times Q_{G'}$ est une *simulation*, si, pour tout $p, q \in Q_G, p' \in Q_{G'}$, avec $p\,R\,p'$ et $p \xrightarrow[G]{a} q$, alors il existe $q' \in Q_{G'}$ tel que $p' \xrightarrow[G']{a} q'$ et $q\,R\,q'$. Lorsqu'il existe une simulation entre G et G', on dit que G' *simule* G.

A présent, une relation $R \subseteq Q_G \times Q_{G'}$ est une *bisimulation* (forte), si, pour tout $p\,R\,p'$, on a :

1. dès que $p \xrightarrow[G]{a} q$, alors il existe q' dans G' tel que $p' \xrightarrow[G']{a} q'$, et $q\,R\,q'$
2. dès que $p' \xrightarrow[G']{a} q'$, alors il existe q dans G tel que $p \xrightarrow[G]{a} q$, et $q\,R\,q'$

Lorsqu'il existe une bisimulation entre G et G' on dit qu'ils sont *bisimilaires* ; ceci signifie aussi que G simule G' et que G' simule G.

1.2.3 Extensions des graphes étiquetés

Nous allons maintenant présenter diverses extensions des graphes étiquetés.

Définition 1.5. Un *système de transition étiqueté* (LTS) $\mathcal{M}$ est un quadruplet $(Q_\mathcal{M}, \Sigma_\mathcal{M}, Q^0_\mathcal{M}, \Delta_\mathcal{M})$ où $(Q_\mathcal{M}, \Sigma_\mathcal{M}, \Delta_\mathcal{M})$ est un graphe étiqueté et $Q^0_\mathcal{M}$ un ensemble d'états initiaux.

Les systèmes de transitions étiquetés sont connus dans la littérature sous le nom de LTS (Labelled Transition Systems) mais aussi sous le nom de DES (Discrete Event Systems). L'utilisation de l'un ou l'autre des termes dépend essentiellement de la communauté, leur signification formelle est la même. Nous utiliserons les DES dans un sens plus spécifique, et nous conservons l'acronyme LTS pour les systèmes de transition étiquetés.

La présence d'un ensemble d'états initiaux nous permet d'introduire la notion d'*exécution* : ce sont des chemins dont l'extrémité initiale appartient à $Q^0_\mathcal{M}$. Étant donnée une exécution γ, on appelle *trace de l'exécution* la suite d'étiquettes des transitions de γ qui est notée $tr(\gamma)$. Remarquons que l'ensemble des traces d'exécutions est *clos par préfixe*, c'est-à-dire que si $uw \in {\Sigma_\mathcal{M}}^*$ est une trace d'exécutions, alors u est une trace d'exécution.

Un système de transition étiqueté est dit *globalement déterministe* si il est localement déterministe et que $Q^0_\mathcal{M}$ est un singleton.

Définition 1.6. Un *automate* est un quintuplet $\mathcal{A} = (Q_\mathcal{A}, \Sigma_\mathcal{A}, Q^0_\mathcal{A}, \Delta_\mathcal{A}, F_\mathcal{A})$ où $(Q_\mathcal{A}, \Sigma_\mathcal{A}, Q^0_\mathcal{A}, \Delta_\mathcal{A})$ est un système de transition étiqueté et $F_\mathcal{A}$ est un ensemble d'*états terminaux*.

Les états terminaux permettent la notion de *reconnaissance*. Un mot w est *reconnu* par l'automate $\mathcal{A}$ si il existe un état q de $F_\mathcal{A}$ tel que q est accessible par w depuis un état de $Q^0_\mathcal{A}$.

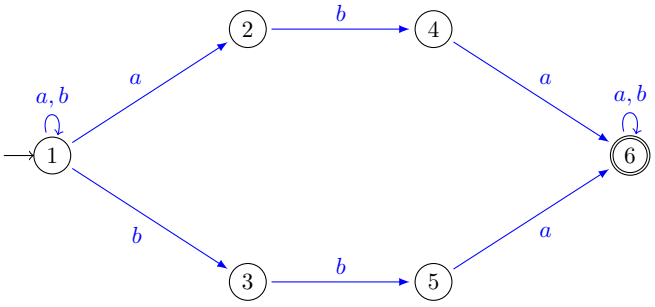


FIGURE 1.1 – Représentation sagittale de l'automate de l'exemple 1.

Le *langage reconnu* par un automate est l'ensemble des mots reconnus par celui-ci, c'est-à-dire $L_{\mathcal{A}}(Q^0_{\mathcal{A}}, F_{\mathcal{A}})$, on le note $L_{\mathcal{A}}$.

Nous présenterons un certain nombre de transformations sur les automates dans le paragraphe suivant.

Exemple 1. *La figure 1.1 est la représentation sagittale de l'automate* $\mathcal{A} = (Q_{\mathcal{A}}, \Sigma_{\mathcal{A}}, Q^0_{\mathcal{A}}, \Delta_{\mathcal{A}}, F_{\mathcal{A}})$ *avec :*

- $Q_{\mathcal{A}} = \{1, 2, 3, 4, 5, 6\}$
- $\Sigma_{\mathcal{A}} = \{a, b\}$
- $Q^0_{\mathcal{A}} = \{1\}$
- $\Delta_{\mathcal{A}} = \{(1, a, 1), (1, b, 1), (1, a, 2), (2, b, 4), (4, a, 6), (1, b, 3), (3, b, 5), (5, a, 6), (6, a, 6), (6, b, 6)\}$
- $F_{\mathcal{A}} = \{6\}$

Définition 1.7. Un *système à événements discrets* (DES) est donné par un sextuplet $\mathcal{M} = (Q_{\mathcal{M}}, \Sigma_{\mathcal{M}}, \Lambda_{\mathcal{M}}, Q^0_{\mathcal{M}}, \Delta_{\mathcal{M}}, col_{\mathcal{M}})$ tel que $(Q_{\mathcal{M}}, \Sigma_{\mathcal{M}}, Q^0_{\mathcal{M}}, \Delta_{\mathcal{M}})$ est un système de transition étiqueté et $(Q_{\mathcal{M}}, \Lambda_{\mathcal{M}}, Q^0_{\mathcal{M}}, \Delta_{\mathcal{M}}, col_{\mathcal{M}})$ est une structure de Kripke.

Les DES sont des graphes étiquetés, colorés, munis d'états initiaux. Ce sera notre modèle principal par la suite, en particulier dans le chapitre sur les problèmes d'observation partielle.

On appelle *langage reconnu* par un DES, l'ensemble des traces d'exécution de ce DES. Comme un DES n'est pas nécessairement complet, il est possible que cet ensemble soit strictement inclus dans $\Sigma^*_{\mathcal{M}}$. Si on considère deux couleurs λ et λ', le *langage depuis* λ vers λ' est l'ensemble des traces des chemins depuis un sommet coloré par λ vers un sommet coloré par λ'. On note $L_{\mathcal{M}}(\lambda, \lambda') \stackrel{\text{def}}{=} L_{\mathcal{M}}(col^{-1}_{\mathcal{M}}(\lambda), col^{-1}_{\mathcal{M}}(\lambda'))$ où simplement $L_{\mathcal{M}}(\lambda')$ si $col^{-1}_{\mathcal{M}}(\lambda) = Q^0_{\mathcal{M}}$.

Dans certains contextes, il est utile de distinguer formellement les entrées d'un système, ses sorties, et les actions internes, inobservables. La notion d'IOLTS introduit ce type de nuance en explicitant une partition de l'alphabet.

Définition 1.8. Un *système de transition étiqueté avec entrées et sorties* (IOLTS) est un DES $\mathcal{M} = (Q_{\mathcal{M}}, \Sigma_{\mathcal{M}}, \Lambda_{\mathcal{M}}, Q^0_{\mathcal{M}}, \Delta_{\mathcal{M}}, col_{\mathcal{M}})$ tel que l'ensemble des événements forme une partition :

$\Sigma_{\mathcal{M}} = \Sigma^?_{\mathcal{M}} \cup \Sigma^!_{\mathcal{M}} \cup \Sigma^i_{\mathcal{M}}$ où $\Sigma^?_{\mathcal{M}}$ est l'ensemble des *entrées*, $\Sigma^!_{\mathcal{M}}$ celui des sorties et $\Sigma^i_{\mathcal{M}}$ celui des événements internes.

En général, les IOLTS ne sont pas colorés. Nous utilisons donc ici des IOLTS colorés que nous désignerons simplement IOLTS. Les IOLTS sont des modèles habituellement utilisés pour représenter des système réactifs, c'est à dire des systèmes qui interagissent avec leur environnement. Dans ce contexte, les entrées représentent les actions de l'utilisateur (entrée clavier par exemple), les sorties représentent les réponses du système (affichage) et les événements internes représentent tous les autres événements que l'utilisateur ne peut ni contrôler, ni observer (des calculs internes par exemple). Les IOLTS seront utilisés dans les deux chapitres sur le test, et les DES seront utilisés dans le reste du document.

1.3 Propriétés des automates

Nous allons maintenant présenter diverses opérations sur les automates. L'intérêt de présenter ces opérations sur les automates est qu'elles s'étendent naturellement aux DES. En faire une présentation spécifique pour les DES ne ferait qu'alourdir ce document. La construction de la majorité de ces opérations ne dépend pas de l'ensemble des états terminaux. Elles seront donc réutilisables avec le système de couleurs.

1.3.1 Déterminisation

Automate des parties Si un automate n'est pas déterministe, il est possible de construire un automate déterministe reconnaissant le même langage en utilisant l'automate des parties. Pour un automate $\mathcal{A} = (Q_{\mathcal{A}}, \Sigma_{\mathcal{A}}, Q^0_{\mathcal{A}}, \Delta_{\mathcal{A}}, F_{\mathcal{A}})$, l'automate des parties est l'automate $det(\mathcal{A}) = (2^{Q_{\mathcal{A}}}, \Sigma_{\mathcal{A}}, \{Q^0_{\mathcal{A}}\}, \Delta', F')$ où

- $2^{Q_{\mathcal{A}}}$ est l'ensemble des parties de $Q_{\mathcal{A}}$
- $\Delta' \subseteq (2^{Q_{\mathcal{A}}} \times \Sigma_{\mathcal{A}} \times 2^{Q_{\mathcal{A}}})$ et $(Q_1, a, Q_2) \in \Delta'$ si $Q_1 \xrightarrow[\mathcal{A}]{a} Q_2$
- $F' = \left\{P \in 2^{Q_{\mathcal{A}}} \mid P \cap F_{\mathcal{A}} \neq \emptyset\right\}$.

Cette construction est possible pour n'importe quel automate mais n'est pas toujours effective si l'automate est infini. Le résultat de cette construction est un automate déterministe qui reconnaît le même langage $L_A(Q_0, F) = L_{det(A)}(\{Q^0_{\mathcal{A}}\}, F')$.

Exemple 2. *La figure 1.2 représente l'automate des parties (déterministe) obtenu à partir de l'automate non-déterministe de l'example 1.*

On observe que l'ensemble $2^{Q_{\mathcal{A}}}$ est de taille exponentielle par rapport à celle de $Q_{\mathcal{A}}$.

Déterminisation à la volée La construction en utilisant l'automate des parties ne permet pas toujours de produire une représentation finie d'un automate infini. De plus, elle est coûteuse en mémoire. Afin de réduire ce coût en mémoire, il est possible de ne garder en mémoire que les états atteints par les exécutions possibles. Cette façon de procéder, appelée déterminisation à la volée, est beaucoup plus proche de la vision machine d'un automate. Étant donné un

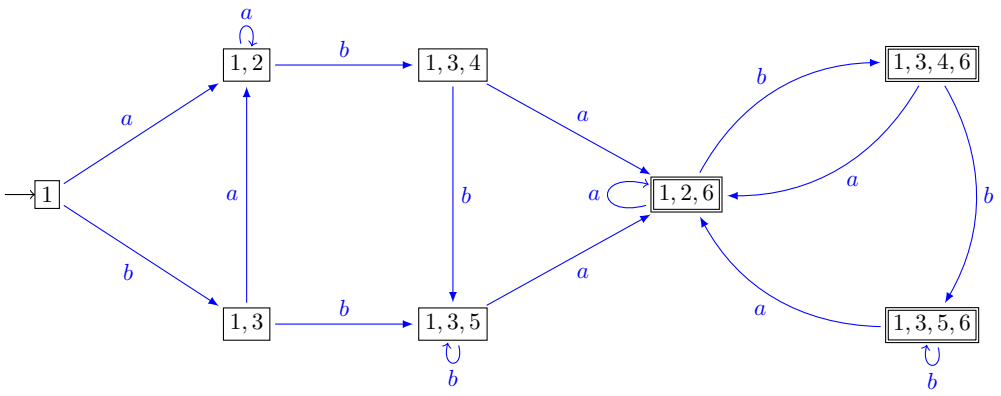


FIGURE 1.2 – Restriction à la partie accessible à partir de $\{1\}$ de l'automate des parties obtenu à partir de l'automate de la figure 1.1

mot w, dont $w(n)$ est la n-ième lettre, il s'agit de construire une suite d'ensembles d'états courants possibles : $(Q_n^{cour})_n$, où Q_n^{cour} représente l'ensemble des états accessibles par le préfixe $w(1)w(2)...w(n)$.

Algorithme de déterminisation à la volée

Données : un automate $\mathcal{A}$ et un mot w.

Initialement : $Q_0^{cour} = Q_{\mathcal{A}}^0$

Pour $1 \leq k \leq |w|$ **:** $Q_k^{cour} := \left\{q' \in Q \mid \exists q \in Q_{k-1}^{cour}, (q, w(k), q') \in \Delta_{\mathcal{A}}\right\}$

Acceptation : Le mot w est accepté si l'ensemble $Q_{|w|}^{cour}$ contient un état terminal : $Q_{|w|}^{cour} \cap F \neq \emptyset$.

Comparaison des deux méthodes

Complexité spatiale : L'automate des parties a un nombre d'état exponentiel en fonction du nombre d'états de l'automate initial. La méthode à la volée occupe un espace qui peut aller jusqu'à l'ensemble des états de l'automate initial ; si celui-ci est fini, l'espace occupé est donc au maximum celui-ci de l'automate initial. En revanche, si l'automate est infini, la croissance de l'ensemble des états courants suit au pire, une exponentielle dont la base est le degré sortant maximal de l'automate.

Complexité temporelle : L'automate des parties ayant une taille exponentielle, il faut un temps exponentiel en fonction du nombre d'état pour le construire. Cependant, une fois celui-ci construit, le calcul des états accessibles par un mot donnée est linéaire dans la

taille du mot, ce qui le rend intéressant si on s'en sert comme une machine. Le coût initial de la méthode à la volée est nul, cependant à chaque nouvel événement, il faut un temps linéaire dans la taille de l'ensemble des états courants pour calculer l'ensemble des états suivant. Comme cet ensemble croît de manière exponentielle, le calcul des états accessibles par un mot est exponentiel en fonction de la taille de celui-ci. Dans le cas fini, l'ensemble des états accessibles est borné par le nombre total d'états de l'automate initial, le coût est donc linéaire.

Calculabilité : Comme souligné précédemment, si l'automate est infini, la construction de l'automate des parties peut requérir un nombre infini d'étapes. En revanche, l'utilisation de la méthode à la volée est toujours possible lorsque le calcul de l'ensemble des successeurs d'un ensemble de sommets est effectif.

Propriétés : L'utilisation de la méthode à la volée n'autorise l'analyse que d'un unique mot. Elle ne permet donc pas de décider de propriétés globables sur l'automate telles que l'universalité. Par la suite, nous utiliserons donc l'une ou l'autre des constructions. Lorsqu'il s'agira de décider de propriétés, nous préférerons l'automate des parties. Quand, au contraire, nous chercherons à produire un moniteur, ou un testeur, il sera pertinent d'utiliser la méthode à la volée qui a l'avantage de fonctionner dans tous les cas.

Adaptation aux couleurs En général, on considérera qu'un ensemble d'états P possède une couleur λ dès que $P \cap col_{\mathcal{A}}^{-1}(\lambda) \neq \emptyset$. Il sera occasionnellement utile de considérer une définition plus contraignante : $P \subseteq col_{\mathcal{A}}^{-1}(\lambda)$.

1.3.2 Opérations rationnelles sur les automates

Le théorème de Kleene (ci-dessous) énonce l'équivalence entre les automates finis et les *expressions régulieres*. L'ensemble des langages rationnels sur un alphabet Σ est l'ensemble :

- qui contient le langage vide
- qui contient les langages $\{a\}$ où $a \in \Sigma$ est une lettre de l'alphabet
- qui est clos par union
- qui est clos par concaténation
- qui est clos par itération (fermeture de Kleene).

Théorème de Kleene. *Sur un alphabet fini Σ, il y a égalité entre les langages rationnels et les langages reconnaissables par un automate fini.*

Nous allons examiner plus précisément certains des opérateurs booléens sur les automates et nous les introduirons dans l'ordre énoncé précédemment.

Union

Soient $\mathcal{A} = (Q_{\mathcal{A}}, \Sigma_{\mathcal{A}}, Q^0_{\mathcal{A}}, \Delta_{\mathcal{A}}, F_{\mathcal{A}})$ et $\mathcal{B} = (Q_{\mathcal{B}}, \Sigma_{\mathcal{B}}, Q^0_{\mathcal{B}}, \Delta_{\mathcal{B}}, F_{\mathcal{B}})$ avec $Q_{\mathcal{A}} \cap Q_{\mathcal{B}} = \emptyset$ et $\Sigma_{\mathcal{A}} \cup \Sigma_{\mathcal{B}} = \Sigma$. L'automate $\mathcal{A}_{\cup}\mathcal{B} = (Q, \Sigma, Q^0, \Delta, F)$ est défini de la façon suivante :

- $Q = Q_{\mathcal{A}} \cup Q_{\mathcal{B}} \cup \{q\}$ avec $q \notin Q_{\mathcal{A}} \cup Q_{\mathcal{B}}$

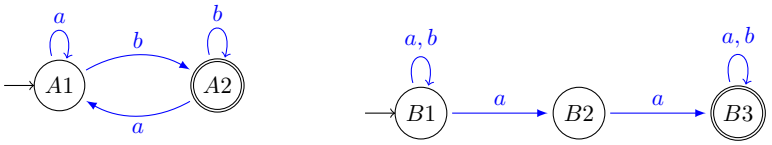


FIGURE 1.3 – Deux automates.

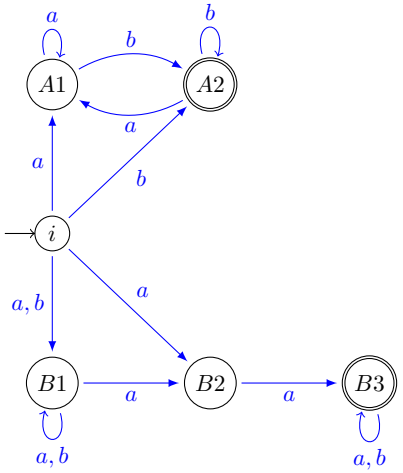


FIGURE 1.4 – Union des deux automates de la figure 1.3

- $Q^0 = \{q\}$
- $\Delta = \Delta_{\mathcal{A}} \cup \Delta_{\mathcal{B}} \cup \{(q, a, q_{\mathcal{A}}) \mid \exists q^0_{\mathcal{A}} \in Q^0_{\mathcal{A}}, (q^0_{\mathcal{A}}, a, q_{\mathcal{A}}) \in \Delta_{\mathcal{A}}\}$
  $\cup \{(q, a, q_{\mathcal{B}}) \mid \exists q^0_{\mathcal{B}} \in Q^0_{\mathcal{B}}, (q^0_{\mathcal{B}}, a, q_{\mathcal{B}}) \in \Delta_{\mathcal{B}}\}$
- $F = F_{\mathcal{A}} \cup F_{\mathcal{B}}$

$\mathcal{A}_\cup\mathcal{B}$ reconnaît tous les mots qui sont dans $L_{\mathcal{A}}$ ou $L_{\mathcal{B}}$, $L_{\mathcal{A}_\cup\mathcal{B}} = L_{\mathcal{A}} \cup L_{\mathcal{B}}$.

Exemple 3. *La figure 1.4 représente l'automate obtenu par union entre les deux DES de la figure 1.3.*

Adaptation aux couleurs Lorsque les automates possèdent des couleurs, ils les conservent lorsqu'on réalise l'union.

Concaténation

Soient $\mathcal{A} = (Q_{\mathcal{A}}, \Sigma_{\mathcal{A}}, Q^0_{\mathcal{A}}, \Delta_{\mathcal{A}}, F_{\mathcal{A}})$ et $\mathcal{B} = (Q_{\mathcal{B}}, \Sigma_{\mathcal{B}}, Q^0_{\mathcal{B}}, \Delta_{\mathcal{B}}, F_{\mathcal{B}})$ avec $Q_{\mathcal{A}} \cap Q_{\mathcal{B}} = \emptyset$ et $\Sigma_{\mathcal{A}} \cup \Sigma_{\mathcal{B}} = \Sigma$. L'automate $\mathcal{A} \cdot \mathcal{B} = (Q, \Sigma, Q^0, \Delta, F)$ est défini de la façon suivante :

- $Q = Q_{\mathcal{A}} \cup Q_{\mathcal{B}}$
- $Q^0 = Q^0_{\mathcal{A}}$
- $\Delta = \Delta_{\mathcal{A}} \cup \Delta_{\mathcal{B}} \cup \{(q_{\mathcal{A}}, a, q_{\mathcal{B}}) \in F_{\mathcal{A}} \times \Sigma \times Q_{\mathcal{B}} \mid \exists q'_{\mathcal{B}} \in Q^0_{\mathcal{B}}, (q'_{\mathcal{B}}, a, q_{\mathcal{B}}) \in \Delta_{\mathcal{B}}\}$
- $F = F_{\mathcal{B}} \cup F_{\mathcal{A}}$ si $F_{\mathcal{B}} \cap Q^0_{\mathcal{B}} \neq \emptyset$
  $F_{\mathcal{B}}$ sinon

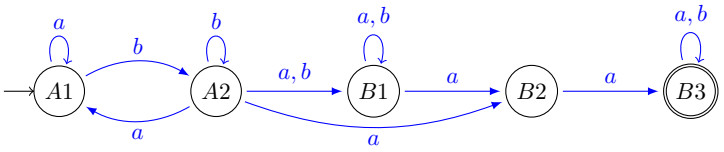


FIGURE 1.5 – Concaténation des deux automates de la figure 1.3

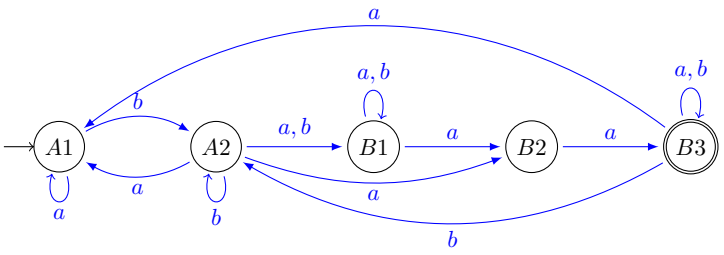


FIGURE 1.6 – Itération de l'automate représenté en figure 1.5

$\mathcal{A} \cdot \mathcal{B}$ reconnaît tous les mots qui sont la concaténation d'un mot de $L_\mathcal{A}$ et de $L_\mathcal{B}$. On notera que les états terminaux s'expriment différemment suivant que l'automate $\mathcal{B}$ reconnaît le mot vide ou pas.

Adaptation aux couleurs La concaténation, et par la suite l'itération, sont des transformations qui dépendent des états terminaux. Elles ne s'adaptent donc pas avec les couleurs. Nous n'en aurons pas vraiment besoin dans la suite du document, mais elles sont présentées par souci d'exhaustivité.

Itération

L'itération L^* d'un langage L est le plus petit langage qui vérifie l'équation $L^* = \{\varepsilon\} \cup L^*.L$. Soit $\mathcal{A} = (Q_\mathcal{A}, \Sigma_\mathcal{A}, Q^0_\mathcal{A}, \Delta_\mathcal{A}, F_\mathcal{A})$ un automate, on définit l'automate $\mathcal{A}^* = (Q_\mathcal{A}, \Sigma_\mathcal{A}, Q^0_\mathcal{A}, \Delta', F_\mathcal{A})$ où $\Delta' = \Delta_\mathcal{A} \cup \big\{(q_1, a, q_2) \in F_\mathcal{A} \times \Sigma_\mathcal{A} \times Q_\mathcal{A} \mid \exists q'_2 \in Q^0_\mathcal{A},\ \text{tel que } (q'_2, a, q_2) \in \Delta_\mathcal{A}\big\}$ qui reconnaît l'itération $L^*_\mathcal{A}$ du langage reconnu par $\mathcal{A}$.

Exemple 4. *La figure 1.5 représente l'automate obtenu par concaténation des deux automates de la figure 1.3. La figure 1.6 représente l'itération de l'automate obtenu en figure 1.5*

Intersection (Produit synchrone d'automates)

Soient $\mathcal{A} = (Q_\mathcal{A}, \Sigma_\mathcal{A}, Q^0_\mathcal{A}, \Delta_\mathcal{A}, F_\mathcal{A})$ et $\mathcal{B} = (Q_\mathcal{B}, \Sigma_\mathcal{B}, Q^0_\mathcal{B}, \Delta_\mathcal{B}, F_\mathcal{B})$. L'automate $\mathcal{A}_\times\mathcal{B} = (Q, \Sigma, Q^0, \Delta, F)$ est défini de la façon suivante :

- $Q = Q_\mathcal{A} \times Q_\mathcal{B}$
- $\Sigma = \Sigma_\mathcal{A} \cap \Sigma_\mathcal{B}$
- $Q^0 = Q^0_\mathcal{A} \times Q^0_\mathcal{B}$

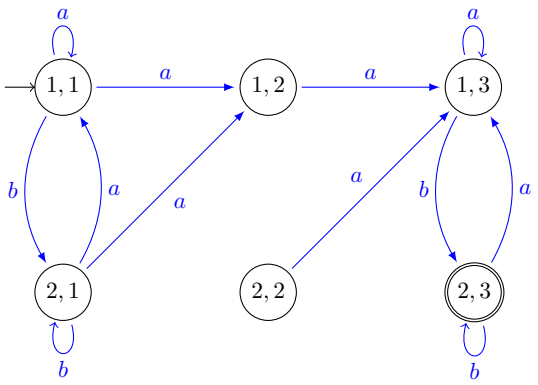


FIGURE 1.7 – Produit entre les deux automates de la figure 1.3

- $\Delta = \{((q_\mathcal{A}, q_\mathcal{B}), a, (q'_\mathcal{A}, q'_\mathcal{B})) \mid (q_\mathcal{A}, a, q'_\mathcal{A}) \in \Delta_\mathcal{A} \wedge (q_\mathcal{B}, a, q'_\mathcal{B}) \in \Delta_\mathcal{B}\}$
- $F = F_\mathcal{A} \times F_B$

$\mathcal{A}_\times\mathcal{B}$ est le *produit* entre $\mathcal{A}$ et $\mathcal{B}$ et reconnaît tous les mots qui sont dans $L_\mathcal{A}$ et $L_\mathcal{B}$, $L_{\mathcal{A}_\times\mathcal{B}} = L_\mathcal{A} \cap L_\mathcal{B}$. Cette construction est aussi appelée *produit synchrone*.

Exemple 5. *La figure 1.7 représente l'automate obtenu par produit entre les deux automates de la figure 1.3.*

Adaptation aux couleurs Le produit s'adapte très facilement aux couleurs. L'automate produit est coloré par des couples de couleurs. On a $\Lambda_{\mathcal{A}_\times\mathcal{B}} = \Lambda_\mathcal{A} \times \Lambda_\mathcal{B}$ et $((q, q')(\lambda, \lambda')) \in col_{\mathcal{A}_\times\mathcal{B}}$ si $(q, \lambda) \in col_\mathcal{A}$ et si $(q', \lambda') \in col_\mathcal{B}$.

Complémentaire

Soit $\mathcal{A} = (Q_\mathcal{A}, \Sigma_\mathcal{A}, Q^0_\mathcal{A}, \Delta_\mathcal{A}, F_\mathcal{A})$ un automate déterministe et complet. L'automate $\neg\mathcal{A} = (Q_\mathcal{A}, \Sigma_\mathcal{A}, Q^0_\mathcal{A}, \Delta_\mathcal{A}, F')$ où $F' = Q_\mathcal{A} \setminus F_\mathcal{A}$ est le *complémentaire* de $\mathcal{A}$ et reconnaît tous les mots qui sont dans $\Sigma^*_\mathcal{A} \setminus L_\mathcal{A}$. Pour un automate quelconque, obtenir le complémentaire est donc possible après une première étape de déterminisation. Cette étape de déterminisation peut être faite en utilisant l'automate des parties ou bien la construction à la volée. Dans le cas de l'automate des parties, on obtient un automate déterministe et on peut donc utiliser la méthode décrite ci-dessus. Dans le cas de la construction à la volée, il faut alors complémenter la condition d'acception, un mot w appartient donc au langage complémentaire, si et seulement, aucun des états de $Q^{cour}_{|w|}$ n'est terminal, $Q^{cour}_{|w|} \cap F_\mathcal{A} = \emptyset$.

Exemple 6. *La figure 1.8 représente l'automate obtenu par complémentation de l'automate de la figure 1.2.*

Adaptation aux couleurs Il ne s'agit pas ici d'une transformation d'automate à proprement parler. Seule la condition d'acceptation change. Comme l'automate $\mathcal{A}$ est donné sous forme déterministe, il suffit de faire le complémentaire $col_{\neg\mathcal{A}} = (Q_\mathcal{A} \times \Lambda_\mathcal{A}) \setminus col_\mathcal{A}$.

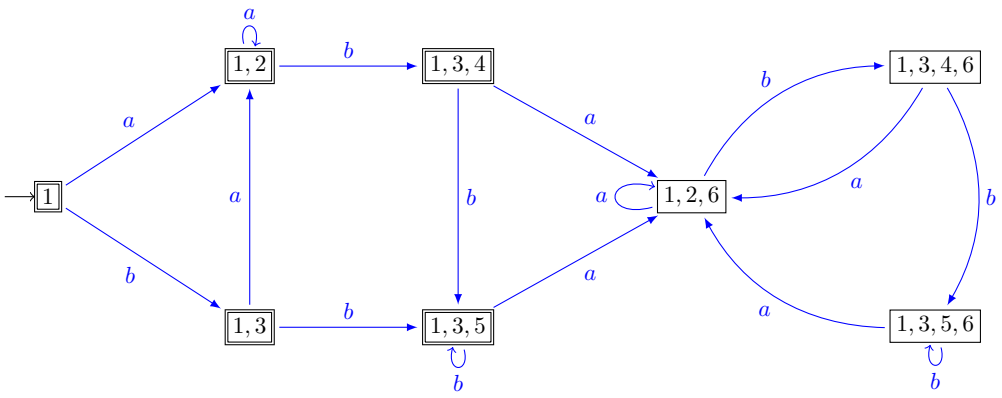


FIGURE 1.8 – Automate reconnaissant le complémentaire du langage reconnu par l'automate de la figure 1.2

1.4 Observation partielle

Quand on cherche à modéliser un système réel, il est possible d'utiliser l'un des objets précédemment décrit. Les états représentent les différentes configurations possibles du système, qui évoluent au fil des événements. Nous avons vu dans la définition des IOLTS qu'il existait différents types d'événements. Dans ce paragraphe nous considérons uniquement qu'il y a ceux que l'on peut observer (dans le cas des IOLTS les entrées et les sorties) et ceux qui ne sont pas observables. Un aspect prépondérant de ce travail concerne la capacité à établir, ou déduire à l'exécution, des informations, sur l'état interne du système, uniquement à partir de l'observation de son comportement.

Modélisation formelle Une façon d'exprimer l'observation partielle pour un DES consiste à établir une partition de l'alphabet des événements : pour un DES $\mathcal{M}$ donné, $\Sigma_{\mathcal{M}} = \Sigma^o_{\mathcal{M}} \cup \Sigma^i_{\mathcal{M}}$. L'ensemble $\Sigma^o_{\mathcal{M}}$ est celui des événements observables, dans le cas des IOLTS $\Sigma^o_{\mathcal{M}} = \Sigma^?_{\mathcal{M}} \cup \Sigma^!_{\mathcal{M}}$. L'ensemble $\Sigma^i_{\mathcal{M}}$ est celui des événements non-observables. Concrètement, comme il n'est pas possible de distinguer les événements non-observables les uns des autres, on peut considérer que $\left|\Sigma^i_{\mathcal{M}}\right| = 1$; on pose en général $\Sigma^i_{\mathcal{M}} = \{\tau\}$.

Dans ce contexte, les seules choses qu'on perçoit du système sont les événements observables. On introduit donc la notion d'*observation*. Une *observation* est la *projection* d'une trace d'exécution sur l'alphabet des événements observables. Plus formellement, étant donnée une trace d'exécution w, on note $\pi_{\Sigma^o}(w)$ la projection sur Σ^o construite de la manière suivante : $\pi_{\Sigma^o}(\varepsilon) = \varepsilon$ et pour tout $w \in \Sigma^*$, si $a \in \Sigma^o$ alors $\pi_{\Sigma^o}(aw) = a\pi_{\Sigma^o}(w)$, sinon $\pi_{\Sigma^o}(aw) = \pi_{\Sigma^o}(w)$.

Soit $\sigma = \pi_{\Sigma^o}(w)$ avec $w \in \Sigma^*_{\mathcal{M}}$ on note $p \overset{\sigma}{\underset{\mathcal{M}}{\rightsquigarrow}} q$ si $p \overset{w}{\underset{\mathcal{M}}{\Longrightarrow}} q$. De la même manière que les traces d'exécutions, on peut noter que l'ensemble des observations est clos par préfixe.

On note $\mathsf{Obs}_{\mathcal{M}}(\mathsf{P})$ l'ensemble des observations de $\mathcal{M}$ qui aboutissent en P, c'est à dire l'ensemble $\left\{\sigma \in {\Sigma^o_{\mathcal{M}}}^* \mid \exists p \in Q^0_{\mathcal{M}}, q \in P, p \overset{\sigma}{\underset{\mathcal{M}}{\rightsquigarrow}} q\right\}$. Par extension, pour une couleur λ, on note $\mathsf{Obs}_{\mathcal{M}}(\lambda) = \mathsf{Obs}_{\mathcal{M}}(\mathsf{col}^{-1}_{\mathcal{M}}(\lambda))$ et $\mathsf{Obs}_{\mathcal{M}} = \mathsf{Obs}_{\mathcal{M}}(\mathsf{Q}_{\mathcal{M}})$ enfin, on note pour tout $\sigma \in {\Sigma^o_{\mathcal{M}}}^*$, $\mathsf{Obs}^{-1}_{\mathcal{M}}(\sigma) = \left\{q \in Q_{\mathcal{M}} \mid \exists p \in Q^0_{\mathcal{M}}, p \overset{\sigma}{\underset{\mathcal{M}}{\rightsquigarrow}} q\right\}$.

On dit qu'un DES $\mathcal{M}$ est *complet à l'observation* si pour tout $q \in Q_{\mathcal{M}}$, et pour tout $a \in \Sigma^o_{\mathcal{M}}$ il existe $q' \in Q_{\mathcal{M}}$ tel que $q \overset{a}{\underset{\mathcal{M}}{\rightsquigarrow}} q'$.

On peut noter qu'un DES peut être déterministe quand on considère l'ensemble des événements et ne plus l'être si on considère uniquement les événements observables.

On appelle *langage observé* la projection sur Σ_o du langage reconnu par le DES.

Pour un DES donné, il peut être intéressant de construire un DES privé des événements non observables, reconnaissant le même langage observable. Pour cela nous présentons différentes méthodes. Comme les DES sont aussi des automates, les techniques qui s'appliquent pour ceux-ci s'appliquent aussi pour les DES.

1.4.1 Clôture d'un DES

Traditionnellement, un automate qui contient des transitions étiquetées par τ est non-déterministe. Les procédures qui permettent de supprimer ces transitions sont donc souvent associées à la déterminisation, ce que nous présentons dans un premier temps. Cependant, la déterminisation est une construction exponentielle, tandis que la suppression des transitions non-observable peut-être effectuée avec une complexité polynomiale si on rapproche ce problème des algorithmes d'accessibilité. Nous présenterons donc deux autres constructions dans ce but. Les limites de ces deux constructions (non préservation des langages de couleur à couleur) nous amènent à en introduire une dernière, au prix d'un ajout d'états, quadratique en fonction du nombre d'états du DES initial.

Clôture par déterminisation Une approche simple est d'utiliser l'automate des parties afin de faire simultanément une clôture et une déterminisation. Il suffit simplement d'exprimer la relation de transition. Pour un DES $\mathcal{M} = (Q_{\mathcal{M}}, \Sigma_{\mathcal{M}}, \Lambda_{\mathcal{M}}, Q^0_{\mathcal{M}}, \Delta_{\mathcal{M}}, col_{\mathcal{M}})$, le DES des parties est le DES $\mathcal{M}' = (2^{Q_{\mathcal{M}}}, \Sigma_{\mathcal{M}}, \Lambda_{\mathcal{M}}, Q^{0\prime}, \Delta', col')$ dans lequel :

- $Q^{0\prime} = Q^0_{\mathcal{M}} \cup \left\{q \in Q_{\mathcal{M}} \mid \exists q_0 \in Q^0_{\mathcal{M}}, q_0 \overset{\varepsilon}{\underset{\mathcal{M}}{\rightsquigarrow}} q\right\}$
- $\Delta' = \left\{(P_1, a, P_2) \in 2^{Q_{\mathcal{M}}} \times \Sigma_{\mathcal{M}} \times 2^{Q_{\mathcal{M}}} \mid P_1 \overset{a}{\underset{\mathcal{M}}{\rightsquigarrow}} P_2\right\}$
- pour $P \in 2^{Q_{\mathcal{M}}}$, $col'(P) = \bigcup_{q \in P}(col(q))$

Clôture par ajout de transitions Comme la construction du DES des parties n'est effective que si le DES est fini, il existe une autre solution pour faire une clôture qui consiste à supprimer les transitions τ et à les remplacer directement par la transition observable qui

suit. Plus formellement, pour un DES $\mathcal{M} = (Q_\mathcal{M}, \Sigma_\mathcal{M}, \Lambda_\mathcal{M}, Q^0_\mathcal{M}, \Delta_\mathcal{M}, col_\mathcal{M})$, on construit un DES $clo(\mathcal{M}) = (Q_\mathcal{M}, \Sigma_M, \Lambda_\mathcal{M}, Q^{0\prime}, \Delta', col_\mathcal{M})$ où $(q_1, a, q_2) \in \Delta'$ si, et seulement si, $a \in \Sigma_o$ et $q_1 \overset{a}{\underset{\mathcal{M}}{\Longrightarrow}} q_2$.

Il existe deux cas particuliers à cette clôture. On parle de τ^*a-simulation, si lorsqu'on considère le chemin $q_1 \overset{a}{\underset{\mathcal{M}}{\rightsquigarrow}} q_2$, la trace du chemin est de la forme τ^*a. On parle de $a\tau^*$-simulation, si elle est de la forme $a\tau^*$.

Ces deux méthodes de clôture ont chacune leurs inconvénients. Ainsi, la clôture par déterminisation n'est, en général, pas effective si le DES n'est pas fini. D'autre part, si on considère l'ensemble des langages observés depuis une couleur vers une autre (et non plus uniquement depuis un état initial), il est possible de perdre de l'information. Dans l'exemple 8, on perd le passage par 2 (1.10) ou par 1 (1.11).

Clôture mixte Afin d'éviter les inconvénients précédents, il est possible de considérer un autre type de clôture. Pour chaque chemin de τ-transition supprimé, un état est ajouté. Le passage par cet état représente le fait que le chemin est emprunté. Plus précisément, étant donné un DES $\mathcal{M} = (Q_\mathcal{M}, \Sigma_\mathcal{M}, \Lambda_\mathcal{M}, Q^0_\mathcal{M}, \Delta_\mathcal{M}, col_\mathcal{M})$, le DES $clo(A) = (Q', \Sigma_\mathcal{M}, \Lambda_\mathcal{M}, Q^{0\prime}, \Delta', col')$ est obtenu avec :

- $Q' = \left\{(q_1, q_2) \in Q_\mathcal{M} \times Q_\mathcal{M} \mid q_1 \overset{\tau^*}{\underset{\mathcal{M}}{\Longrightarrow}} q_2\right\}$
- $Q^{0\prime} = (Q^0_\mathcal{M} \times Q_\mathcal{M}) \cap Q'$
- $\Delta' = \{((q_1, q_2), a, (q'_1, q'_2)) \mid a \in \Sigma^o, (q_1, q_2) \in Q', (q'_1, q'_2) \in Q', (q_2, a, q'_1) \in \Delta_\mathcal{M}\}$
- $col' = \{((q_1, q_2), \lambda) \in Q' \times \Lambda_\mathcal{M} \mid (q_1, \lambda) \in col_\mathcal{M} \vee (q_2, \lambda) \in col_\mathcal{M}\}$.

Exemple 7. *Les figures 1.10, 1.11, 1.12 représentent respectivement la τ^*a-simulation, la $a\tau^*$-simulation et et la clôture mixte d'une partie d'un DES représenté à la figure 1.9. On ne représente qu'une seule transition τ entre les états* 1 *et* 2*. Les états $p1$ et $p2$ représentent les prédécesseurs respectifs de* 1 *et* 2*, tandis que les états $s1$ et $s2$ représentent leurs successeurs. Les étiquettes des transitions sont arbitraires et représentent les entrées et sorties des sommets* 1 *et* 2*. La version utilisant l'automate des parties n'est pas représentée ici car elle contiendrait* 64 *états.*

1.4.2 Estimateurs d'état.

Lorsque seules les observations du système peuvent être obtenues, il peut être intéressant de connaître l'état courant du système. C'est pour l'essentiel ce que nous chercherons à faire dans le chapitre suivant quand nous construirons des moniteurs. Les deux procédés suivants ont pour objectif d'extraire de l'information des observations : l'estimateur d'état courant, sans la donnée de l'état initial, détermine quel peut être l'état courant tandis que l'estimateur d'état initial, sans la donnée de l'état courant vise à déterminer l'état initial du système. Pour ces deux constructions, on ne se préoccupe pas des couleurs, elle seront donc présentées pour des LTS.

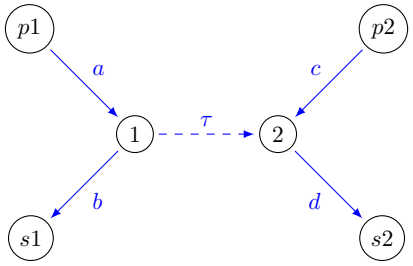


FIGURE 1.9 – Une τ-transition

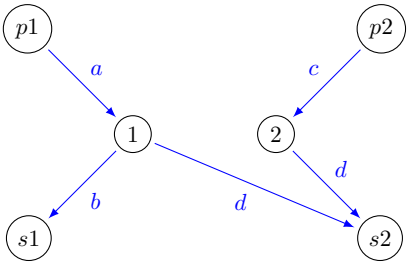


FIGURE 1.10 – $\tau * a$-simulation

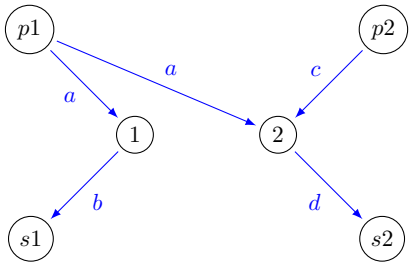


FIGURE 1.11 – $a\tau*$-simulation

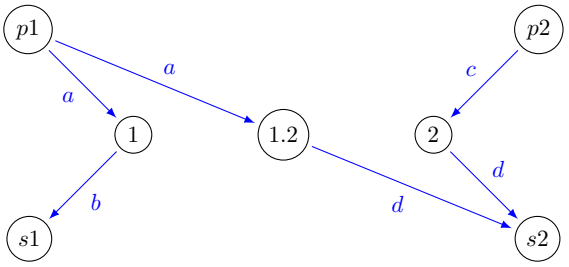


FIGURE 1.12 – Clôture mixte

Estimateur d'état courant L'estimateur d'état courant (CSE) est simplement un LTS des parties dans lequel l'état initial est l'ensemble total des états possibles. Plus formellement, étant donné un LTS $\mathcal{M} = (Q_\mathcal{M}, \Sigma_\mathcal{M}, Q^0_\mathcal{M}, \Delta_\mathcal{M})$, l'estimateur d'état courant est le LTS $EC_\mathcal{M} = (2^{Q_\mathcal{M}}, \Sigma_\mathcal{M}, Q^{0\prime}, \Delta')$ dans lequel :

- $Q^{0\prime} = \{Q_\mathcal{M}\}$
- $\Delta' = \left\{ (P_1, a, P_2) \in 2^{Q_\mathcal{M}} \times \Sigma_\mathcal{M} \times 2^{Q_\mathcal{M}} \mid P_1 \overset{a}{\underset{\mathcal{M}}{\rightsquigarrow}} P_2 \right\}$

Le LTS des parties permet de disposer d'un estimateur d'état courant lorsqu'on connaît avec précision :

1. l'ensemble des états initiaux et
2. l'observation dans sa totalité.

Il suffit alors, en partant de l'état initial du LTS des parties, de suivre l'observation et de lire les états possibles comme faisant partie de l'ensemble d'état atteint.

Parfois, on ne connaît pas l'état initial, ou on prend l'exécution en cours, il nous manque le début de celle-ci. Dans ces cas, on utilise alors l'estimateur d'état courant décrit ci-dessus en prenant pour ensemble d'état initiaux celui qui les contient tous. On peut adapter ce procédé si on possède des informations supplémentaires (par exemple un sous ensemble d'états possibles).

Estimateur d'état initial L'estimateur d'état initial (ISE) est un peu plus complexe. Il s'agit de garder simultanément les états courants possibles ainsi que les états initiaux qui correspondent. Plus formellement, étant donné un LTS $\mathcal{M} = (Q_\mathcal{M}, \Sigma_\mathcal{M}, Q^0_\mathcal{M}, \Delta_\mathcal{M})$, l'estimateur d'état initial est le LTS $EI_\mathcal{M} = (2^{Q_\mathcal{M} \times Q_\mathcal{M}}, \Sigma_\mathcal{M}, Q^{0\prime}, \Delta')$ dans lequel :

- $Q^{0\prime} = \{(q, q) \mid q \in Q\mathcal{M}\}$
- Δ' est l'ensemble des triplets (Q_1, a, Q_2) tels que $(q_1, q_3) \in Q_2$ si il existe $q_2 \in Q_{\mathcal{M}}$ tel que $(q_1, q_2) \in Q_1$ et $(q_2, a, q_3) \in \Delta_{\mathcal{M}}$.

On procède en deux étapes pour lire l'ensemble des états initiaux possibles dans l'estimateur d'état initial :

- En partant de l'état initial de l'ISE, et en suivant une observation, on atteint un état P de l'ISE.
- Un état q appartient à l'ensemble des états initiaux possibles, si il existe un couple (q, q') dans P.

Exemple 8. *Les figures 1.14 et 1.15 représentent respectivement les estimateurs d'état courants et initiaux du LTS représenté à la figure 1.13. L'estimateur d'état courant se lit comme un LTS classique. Pour l'estimateur d'état initial, en revanche, une fois l'état déterminé, il faut se reporter au tableau disposé à droite à la figure 1.16 qui représente le détail des états du DES de la figure 1.15. On peut remarquer que les estimateurs contiennent un unique état initial.*

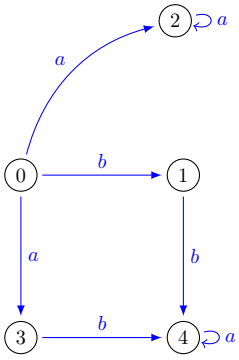


FIGURE 1.13 – Un LTS

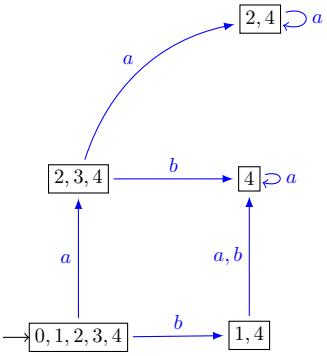


FIGURE 1.14 – Estimateur d'état courant

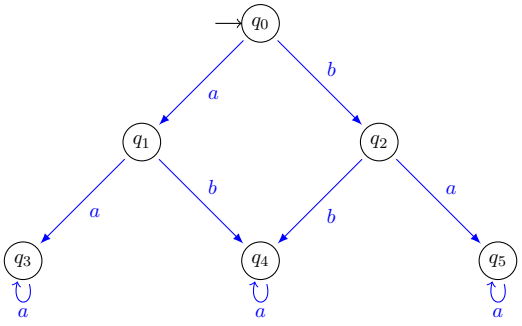


FIGURE 1.15 – Estimateur d'état initial

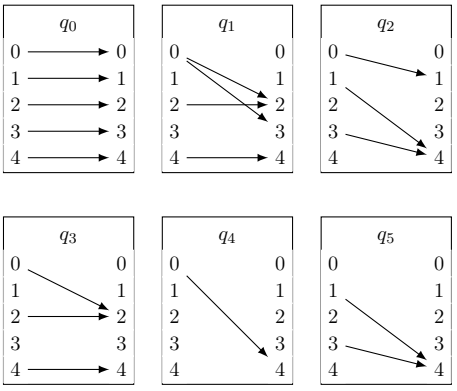


FIGURE 1.16 – Détail des états de l'estimateur d'état initial

Chapitre 2

Diagnostic et Opacité

Lorsque des systèmes modélisés par des DES ou des LTS) sont exécutés, le plus souvent, seule une partie de leur comportement est visible pour un observateur extérieur. Ceci induit le principe d'observation partiel du système à son exécution. Cette remarque permet de définir plusieurs familles de problèmes. Ainsi, le diagnostic, formalisé par exemple dans [61, 69], consiste à observer le comportement du système, et à déterminer si une défaillance s'est produite. Le problème d'opacité, [17, 9, 30], quant-à lui, vise à identifier si un observateur du comportement du système sera à même de déterminer avec certitude si ce dernier a atteint un ensemble d'états identifiés comme secrets. Dans ce chapitre, on s'intéresse à ces deux problèmes ainsi qu'à quelques unes de leurs extensions.

2.1 Problèmatique de l'observation partielle

On l'a vu, les questions de diagnostic et d'opacité sont toutes deux relatives à l'observation partielle d'un système. On suppose donc de n'observer qu'un sous ensemble des événements du système. Ces problèmes ont plusieurs points communs : la connaissance d'une spécification du système et l'accès à des observations.

2.1.1 Modélisation des propriétés

Notre but est de vérifier que les systèmes que nous étudions possèdent un certain nombre de propriétés. Il existe plusieurs méthodes pour décrire ces propriétés : propriétés d'état, langage d'exécution, ou langage d'observation. Nous montrerons que sous certaines conditions, ces descriptions peuvent toutes se représenter via des propriétés d'état.

Propriété d'état. Dans ce travail, nous utiliserons les propriétés d'état pour des systèmes de transitions, ce qui consiste à attribuer une couleur qui représente la propriété à tous les états qui la vérifient.

Langage d'observations. Il est possible de modéliser la propriété via un langage qui représente l'ensemble des observations. Ce type de description externe est souvent considéré

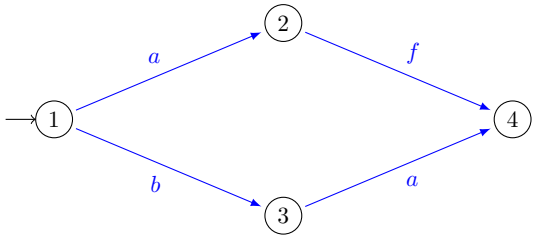


FIGURE 2.1 – Un DES

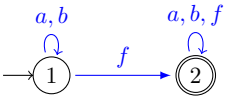


FIGURE 2.2 – Un patron de faute

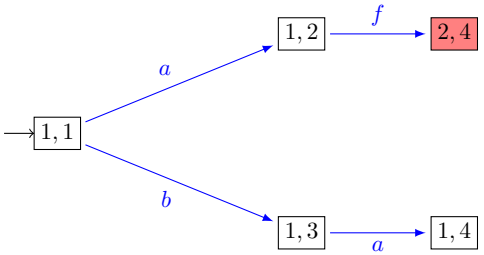


FIGURE 2.3 – Produit entre le patron de faute de la figure 2.2 et du DES de la figure 2.1

quand on spécifie la propriété à l'aide d'une logique. Un reconnaisseur, c'est à dire un système qui reconnaît la propriété, sera exécuté en parallèle et indiquera si la propriété est vérifiée ou non. Il est possible de modéliser la synchronisation des deux systèmes en faisant le produit des modèles (un DES, pour le système étudié).

Ce type de construction nécessite cependant que la construction du produit soit effective, ce qui est le cas si les deux système sont donnés par des DES finis.

Langage d'événements. Pour décrire une propriété, il est possible de l'associer à des séquences de transitions. Dans le cas du diagnostic par exemple, il est usuel de considérer qu'un événement *faute* peut se produire et que c'est l'occurrence de cet événement que l'on veut mettre en évidence.

Souvent ce type de langage est assez simple et reflète la survenue d'un événement particulier à un moment. Une fois que l'événement s'est produit, la propriété reste perpétuellement vraie (comme dans le patron représenté à la figure 2.2) .

Plus généralement, lorsqu'on définit une propriété par langage d'événement, ce langage est reconnu par un DES, dont certains états sont terminaux. Ainsi, il est possible de se ramener à une propriété d'état en faisant le produit entre ce DES, et celui qui modélise le système [46].

Exemple 9. *La figure 2.1 est un DES modélisant un système $\mathcal{M}$. Le DES représenté à la figure 2.2 définit un langage d'événements L qui correspond à un ensemble d'exécutions* fautives *(qu'on cherche à identifier). La figure 2.3 illustre le produit entre les deux DES précédents. Cre produit constitue une modélisation par une propriété d'état du langage L sur le système $\mathcal{M}$.*

2.1.2 Supervision

Quand un système est partiellement observé, il peut être intéressant, étant donné une couleur, de savoir si le système est dans un état possédant cette couleur. Ce problème est appelé *supervision* et il est résolu par la construction d'un *moniteur*, qui sera exécuté conjointement au système.

Moniteur

Étant donné un DES $\mathcal{M}$ et une couleur $\lambda \in \Lambda_{\mathcal{M}}$, on appelle *moniteur*, une fonction $Mon_{(\mathcal{M},\lambda)} : \mathsf{Obs}_{\mathcal{M}} \to \{\mathbf{oui}, \mathbf{non}, \mathbf{equi}\}$. De façon informelle, ces trois verdicts ont la signification suivante : le verdict **oui** signifie que σ atteint λ avec certitude, le verdict **non** signifie qu'il ne l'atteint pas, et le verdict **equi** signifie qu'il est possible d'atteindre λ comme de ne pas l'atteindre. Quand le verdict renvoyé est **oui** ou **non**, on dit que le verdict est *clair*, dans le cas contraire il est *équivoque*.

Un moniteur est *correct*, si quand un verdict clair est renvoyé, alors la propriété est garantie d'être vérifiée ou violée. Plus formellement, un moniteur est correct si

$$\begin{cases} Mon_{(\mathcal{M},\lambda)}(\sigma) = \mathbf{oui} & \Longrightarrow \mathsf{Obs}_{\mathcal{M}}^{-1}(\sigma) \subseteq col_{\mathcal{M}}^{-1}(\lambda) \\ Mon_{(\mathcal{M},\lambda)}(\sigma) = \mathbf{non} & \Longrightarrow \mathsf{Obs}_{\mathcal{M}}^{-1}(\sigma) \cap col_{\mathcal{M}}^{-1}(\lambda) = \emptyset \end{cases}$$

Un moniteur est *précis*, si quand la propriété est garantie d'être vérifiée ou violée, alors un verdict clair est renvoyé. Plus formellement, un moniteur est correct si

$$\begin{cases} \mathsf{Obs}_{\mathcal{M}}^{-1}(\sigma) \subseteq col_{\mathcal{M}}^{-1}(\lambda) & \Longrightarrow Mon_{(\mathcal{M},\lambda)}(\sigma) = \mathbf{oui} \\ \mathsf{Obs}_{\mathcal{M}}^{-1}(\sigma) \cap col_{\mathcal{M}}^{-1}(\lambda) = \emptyset & \Longrightarrow Mon_{(\mathcal{M},\lambda)}(\sigma) = \mathbf{non} \end{cases}$$

Lorsqu'on s'intéresse à construire ce moniteur de façon effective, il sera alors représenté par un DES déterministe (pour l'unicité du verdict) sur l'alphabet des événements observables de $\mathcal{M}$ et le verdict associé à une exécution et une couleur, sera la couleur de l'état atteint (**oui**, **non**, **equi**).

Construction d'un moniteur pour les DES finis

Lorsqu'on dispose d'un DES fini $\mathcal{M}$ qui modélise un système, il existe plusieurs façons de construire un moniteur correct et précis pour ce système et une couleur λ. En pratique, deux techniques sont les plus fréquentes. La première consiste à construire un DES déterministe pour les observables, et attribuer des couleurs pertinentes aux états. La seconde technique consiste à réaliser une construction *à la volée*, et donc à effectuer un parcours de $\mathcal{M}$, et à produire un verdict selon l'ensemble d'états atteints.

Construction par parties Pour un DES fini, il est possible de construire un moniteur en construisant le DES des parties de la clôture. Il est aussi possible d'utiliser directement la technique de clôture par parties. Pour un DES $\mathcal{M}$ et une couleur λ, on peut construire un moniteur à partir de $det(\mathcal{M}) = (Q_{det(\mathcal{M})}, \Sigma_{det(\mathcal{M})}, \Lambda_{det(\mathcal{M})}, Q^0_{det(\mathcal{M})}, \Delta_{det(\mathcal{M})}, col_{det(\mathcal{M})})$, le DES des parties. Le DES $\mathcal{M}on = (Q_{det(\mathcal{M})}, \Sigma_{det(\mathcal{M})}, \Lambda_{\mathcal{M}on}, Q^0_{det(\mathcal{M})}, \Delta_{det(\mathcal{M})}, col_{\mathcal{M}on})$ est défini de la façon suivante :

- $\Lambda_{\mathcal{M}on} = \{\mathbf{oui}, \mathbf{non}, \mathbf{equi}\}$,
- $\begin{cases} (P, \mathbf{oui}) \in col_{\mathcal{M}on} & \text{si } \forall p \in P, (p, \lambda) \in col_{\mathcal{M}} \\ (P, \mathbf{non}) \in col_{\mathcal{M}on} & \text{si } \forall p \in P, (p, \lambda) \notin col_{\mathcal{M}} \\ (P, \mathbf{equi}) \in col_{\mathcal{M}on} & \text{sinon} \end{cases}$

On peut vérifier aisément que ce moniteur est correct.

Construction à la volée La construction par parties n'est pas toujours effective dans le cas où le DES est infini. Même lorsque cette construction est possible, sa taille peut être un problème. Il est donc parfois judicieux d'utiliser la construction à la volée présentée au chapitre 1. En voici un rappel :

Données : un DES $\mathcal{M}$, une couleur λ et une observation σ.

Initialement : $Q_0^{cour} = Q^0_{\mathcal{M}}$

Pour $1 \leq k \leq |\sigma|$ **:** $Q_k^{cour} := \left\{ q' \in Q \mid \exists q \in Q_{k-1}^{cour}, q \overset{\sigma(k)}{\underset{\mathcal{M}}{\rightsquigarrow}} q' \right\}$

Verdict : $\begin{cases} \mathbf{oui} & \text{si l'ensemble } Q_{|\sigma|}^{cour} \subseteq col_{\mathcal{M}}^{-1}(\lambda) \\ \mathbf{non} & \text{si l'ensemble } Q_{|\sigma|}^{cour} \cap col_{\mathcal{M}}^{-1}(\lambda) = \emptyset \\ \mathbf{equi} & \text{sinon} \end{cases}$

2.2 Opacité

La notion d'opacité a été introduite dans [53] et a été définie pour les systèmes de transitions dans [17]. Il s'agit de déterminer si un attaquant, qui connaît la spécification du système, est en mesure, à partir des observations du système, de détecter un état secret ou d'identifier un comportement secret qui se produirait lors de l'exécution. Il est possible de spécifier ces états du système de bien des façons (comme pour toute propriété, voir paragraphe § 2.1.1). Dans ce document, une couleur spécifique désignera les états secrets d'un DES modélisant le système. Comme nous l'avons déjà vu, toute autre caractérisation reposant sur un DES ou un automate fini serait équivalente.

Définition 2.1. Une couleur λ est *opaque* dans un DES $\mathcal{M}$ si, pour toute exécution γ de $\mathcal{M}$ telle que $fin(\gamma) \in col_{\mathcal{M}}^{-1}(\lambda)$, alors il existe une exécution γ' de $\mathcal{M}$ avec $\pi_{\Sigma^o_{\mathcal{M}}}(\gamma) = \pi_{\Sigma^o_{\mathcal{M}}}(\gamma')$ telle que $fin(\gamma') \notin col_{\mathcal{M}}^{-1}(\lambda)$.

Intuitivement, λ est opaque si, toute exécution dont l'extrémité finale est colorée par λ est masquée par une exécution de même observation dont l'extrémité finale n'est pas colorée par λ. Cela peut être traduit de manière équivalente en utilisant les moniteurs : λ est opaque dans $\mathcal{M}$ si pour tout moniteur correct et précis :

$$\forall \sigma \in {\Sigma^o_{\mathcal{M}}}^*, Mon_{(\mathcal{M},\lambda)}(\sigma) \neq \textbf{oui} \tag{2.1}$$

Exemple 10. *Étant donné le DES $\mathcal{M}$ de la figure 2.4, avec $\Sigma^o_{\mathcal{M}} = \{a, b\}$. Le secret est donné par l'ensemble d'états $\{2, 6\}$ représenté en rouge.*

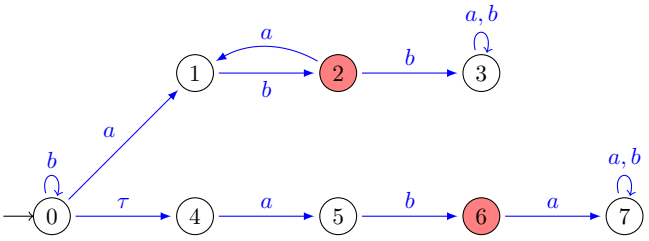


FIGURE 2.4 – Un système non-opaque

La couleur rouge n'est pas opaque car après l'observation d'une trace de la forme b^*ab, l'attaquant sait de façon sûre que le système est dans un état coloré. Il ne sait en revanche pas si il est en 2 ou en 6 mais il sait que l'état courant est rouge.

Problème. Le problème de l'opacité est le suivant :

Données : un DES $\mathcal{M}$ et une couleur $\lambda \in \Lambda_{\mathcal{M}}$

Question : la couleur λ est-elle opaque dans $\mathcal{M}$?

Dans le contexte d'un DES $\mathcal{M}$ à états finis, résoudre ce problème consiste à vérifier l'accessibilité du verdict **oui** dans un moniteur correct et précis. Pour rappel, si $\exists \sigma \in \Sigma^o_{\mathcal{M}}, Mon_{(\mathcal{M},\lambda)}(\sigma) =$ **oui**, cela contredit l'équation 2.1 et donc λ n'est pas opaque dans $\mathcal{M}$.

Pour les DES finis, il suffit donc :

1. de construire le moniteur de la fonction $Mon_{(\mathcal{M},\lambda)}$
2. de vérifier l'accessibilité d'un état coloré par **oui**.

Comme on l'a vu, il est possible de construire un DES fini qui est le moniteur correct et précis. L'accessibilité d'un état renvoyant le verdict **oui** est ainsi décidable en temps polynomial en fonction de la taille du moniteur. En revanche, le moniteur étant de taille exponentielle, on a le résultat suivant.

Théorème 1. *[30] Le problème de l'opacité sur les DES finis est* PSPACE*-complet.*

Démonstration. La dureté est démontrée par réduction du problème de l'universalité sur les automates finis, c'est à dire savoir si un automate $\mathcal{A}$ reconnaît le langage $\Sigma^*_{\mathcal{A}}$. Ce problème est PSPACE-complet.

Soit $\mathcal{A} = (Q_\mathcal{A}, \Sigma_\mathcal{A}, Q^0_\mathcal{A}, \Delta_\mathcal{A}, F_\mathcal{A})$ un automate dont on veut savoir si il est universel. On construit alors le DES $\mathcal{M} = (Q_\mathcal{A} \cup \{q\}, \Sigma_\mathcal{A}, \{\lambda\}, Q^0_\mathcal{A} \cup \{q\}, \Delta_\mathcal{A} \cup \{(q, \Sigma_\mathcal{A}, q)\}, (q, \lambda))$ avec $q \notin Q_\mathcal{A}$ un nouvel état. Par construction, $\mathcal{M}$ est opaque, si, et seulement si, $\mathcal{A}$ reconnaît le langage universel. Donc l'universalité de $\mathcal{A}$ se réduit à l'opacité de $\mathcal{M}$.

Réciproquement, la construction du moniteur à la volée donne un algorithme dans *pspace*. □

Exemple 11. *Considérons le DES défini dans l'exemple 10. Le moniteur associé est représenté par la figure 2.5. Les états colorés par* **oui** *sont représentés en rouge, les états colorés par* **equi** *en bleu et les états colorés par* **non** *sont blancs. L'état* $\{2, 6\}$ *est accessible depuis l'état initial, le secret n'est donc pas opaque.*

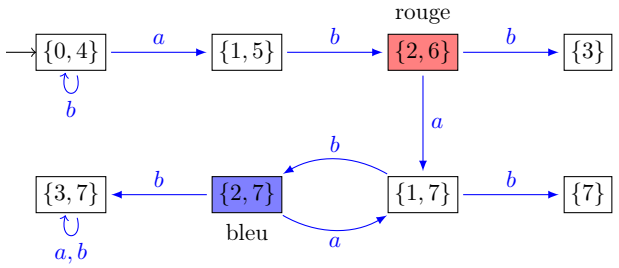


FIGURE 2.5 – Moniteur du système représenté en figure 2.4

2.3 Diagnosticabilité

Le problème de la diagnosticabilité a été introduit dans [61]. Il consiste à déterminer, au cours de l'exécution d'un système si une certaine propriété peut-être détectée avec certitude, en un temps fini après qu'elle soit satisfaite. En général, cela sert à détecter une défaillance interne du système, et on cherche donc à la détecter. On appellera donc *exécution fautive*, une exécution dont l'un des états porte la couleur λ. L'une des différences avec l'opacité est que le langage des observations qui vérifient la propriété est clos par extension. Pour un DES $\mathcal{M}$ et une couleur λ, on dit que λ est un *piège* lorsque pour tout $q \in Q_\mathcal{M}$ tel que $(q, \lambda) \in col_\mathcal{M}$, quel que soit $q' \in Q_\mathcal{M}$, si $q \overset{a}{\underset{\mathcal{M}}{\rightsquigarrow}} q'$ alors $(q', \lambda) \in col_\mathcal{M}$.

Une première approche pour détecter un passage par la couleur λ est de construire un moniteur correct et précis de λ. Étant donné que le langage est clos par extension, le fait de renvoyer un verdict sur l'état courant nous suffit à détecter un passage par la couleur λ par le passé. Cependant, le moniteur, même correct et précis, ne garantit pas toujours que tout passage par λ sera détecté. Il est en effet possible de renvoyer perpétuellement le verdict **equi**. C'est par exemple le cas quand la couleur λ est opaque.

La notion de diagnosticabilité à été définie (par [61]) pour caractériser l'absence de ce risque.

Définition 2.2. Une couleur λ est k-diagnosticable ($k \in \mathbb{N}$) dans un DES $\mathcal{M}$, si pour toute exécution fautive de trace σ, pour tout prolongement σ' de σ avec $|\sigma'| \geq k$, $Mon_{(\mathcal{M},\lambda)}(\sigma\sigma') =$ **oui**.

Cette notion de k-diagnosticabilité peut-être étendue naturellement pour définir la diagnosticabilité, en exprimant qu'un système est diagnosticable lorsqu'il existe un k tel qu'il soit k-diagnosticable. Cette définition historique peut poser problème pour les systèmes infinis. Dans ce document, nous utilisons une définition alternative (présentée ci-dessous) qui est équivalente pour les systèmes finis, et qui autorise plus de systèmes diagnosticables dans le cas infini.

Définition 2.3. Une couleur λ est diagnosticable dans un DES $\mathcal{M}$, si pour toute exécution fautive de trace σ, il existe $k \in \mathbb{N}$ tel que pour tout prolongement σ' de σ avec $|\sigma'| \geq k$, $Mon_{(\mathcal{M},\lambda)}(\sigma\sigma') =$ **oui**.

Remarque 2.1. Pour les DES finis $\mathcal{M}$, la diagnosticabilité implique la k-diagnosticabilité pour k le nombre d'état de $\mathcal{M}$, et réciproquement, il est trivial que la k-diagnosticabilité implique la diagnosticabilité (voir aussi [61]).

La définition précédente peut être reformulée en terme d'exécutions infinies.

Lemme 1. *([55]) Une couleur λ n'est pas diagnosticable dans un DES $\mathcal{M}$, si il existe deux exécutions infinies ayant la même observation telle que l'une est fautive et l'autre non.*

Problème. On définit le problème de la diagnosticabilité ainsi

Données : un DES $\mathcal{M}$ et une couleur $\lambda \in \Lambda_{\mathcal{M}}$

Question : la couleur λ est-elle diagnosticable dans $\mathcal{M}$?

Le problème de la diagnosticabilité est décidable pour les systèmes finis.

Théorème 2. *([62]) Le problème de la diagnosticabilité pour les systèmes finis est décidable en temps polynomial.*

Soit $\mathcal{M}$ un DES fini, et λ une couleur qui est un piège dans $\mathcal{M}$, on peut déterminer si cette dernière est diagnosticable en procédant aux trois étapes suivantes.

Clôture La première étape consiste à construire le DES $clo(\mathcal{M})$ afin de ne conserver que le comportement observable de $\mathcal{M}$. Il est possible de faire cette étape sans perdre d'information car λ est un piège. Cette opération est quadratique en fonction du nombre d'états pour les DES finis.

Autoproduit On calcule le produit du DES $clo(\mathcal{M})$ avec lui même afin d'obtenir le DES $\mathcal{M}_{\times}\mathcal{M}$. L'ensemble des états, l'alphabet, l'état initial, et les transitions sont définies comme

au paragraphe 1.3.2. On attribue les couleurs, $\Lambda_{\mathcal{M}_\times\mathcal{M}} = \{\mathbf{oui}, \mathbf{non}, \mathbf{equi}\}$ de la façon suivante :

$$\begin{cases} ((p,q), \mathbf{oui}) \in col_{\mathcal{M}_\times\mathcal{M}} & \text{si } (p,\lambda) \in col_{\mathcal{M}} \wedge (q,\lambda) \in col_{\mathcal{M}} \\ ((p,q), \mathbf{non}) \in col_{\mathcal{M}_\times\mathcal{M}} & \text{si } (p,\lambda) \notin col_{\mathcal{M}} \wedge (q,\lambda) \notin col_{\mathcal{M}} \\ ((p,q), \mathbf{equi}) \in col_{\mathcal{M}_\times\mathcal{M}} & \text{sinon} \end{cases}$$

Cette construction est quadratique en fonction du nombre d'états de $clo(\mathcal{M})$ pour les DES finis.

Accessibilité Dans un DES à états finis, il existe une exécution infinie, si, et seulement si, il existe un circuit. La dernière étape est donc de vérifier qu'il existe un circuit d'états colorés par **equi** dans $\mathcal{M}_\times\mathcal{M}$ et que ce dernier est accessible. La complexité de cette vérification est linéaire en fonction du nombre d'états pour les DES finis.

La construction présentée ici utilise l'auto-produit. Quand l'auto-produit n'est pas effectif, la construction peut s'avérer complexe de résoudre le problème de la diagnosticabilité. De fait, nous verrons dans le chapitre 6 que le problème est indécidable dans le cas général.

2.4 Diverses extensions

La propriété d'opacité que nous avons considérée jusqu'à maintenant porte sur l'état actuel du système vis-à-vis d'une observation. Lorsqu'une observation ne permet pas de déterminer avec certitude cet état, il est possible que quelques observations supplémentaires, autorisent à restreindre rétrospectivement les états possibles. Avec ces nouvelles informations il est alors possible de déduire que, par le passé, le secret a été vérifié.

Pour définir des notions d'opacité qui prennent en compte le passé, il y a deux axes sur lesquels il est possible d'agir. Le premier axe est le temps pendant lequel on garde l'information en mémoire : infini ou borné. Le second axe nuance la caractérisation de la violation d'opacité : synchrone ou asynchrone. Dans le cas de l'opacité synchrone, on veut qu'à un moment dans le passé, tous les états possibles aient été colorés simultanément. Dans le cas de l'opacité asynchrone, on veut seulement s'assurer que quelque soit le chemin emprunté, un état coloré ait été traversé. Dans la figure 2.7, à la fin de l'exécution (et pas avant), on est sûr d'être passé par un état bleu, et on est sûr que par le passé, tous les états ont été rouges simultanément.

Ces notions ont été étudiées par [36], [60] dans leur version synchrone.

Dans les quatre paragraphes qui suivent, nous fixerons un DES $\mathcal{M}$ et une couleur λ.

2.4.1 Opacité bornée synchrone

Le but de cette variante est de détecter si dans un passé borné en nombre d'observations, l'ensemble des états possibles ont tous été colorés par λ.

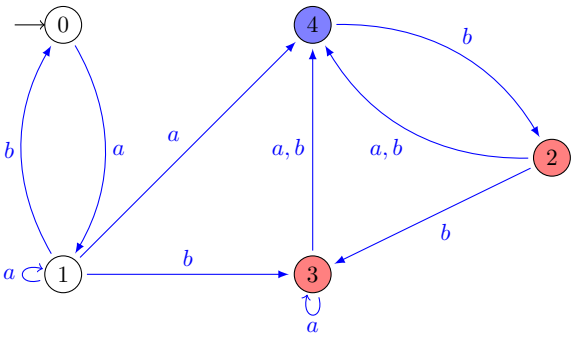


FIGURE 2.6 – Un DES avec deux couleurs.

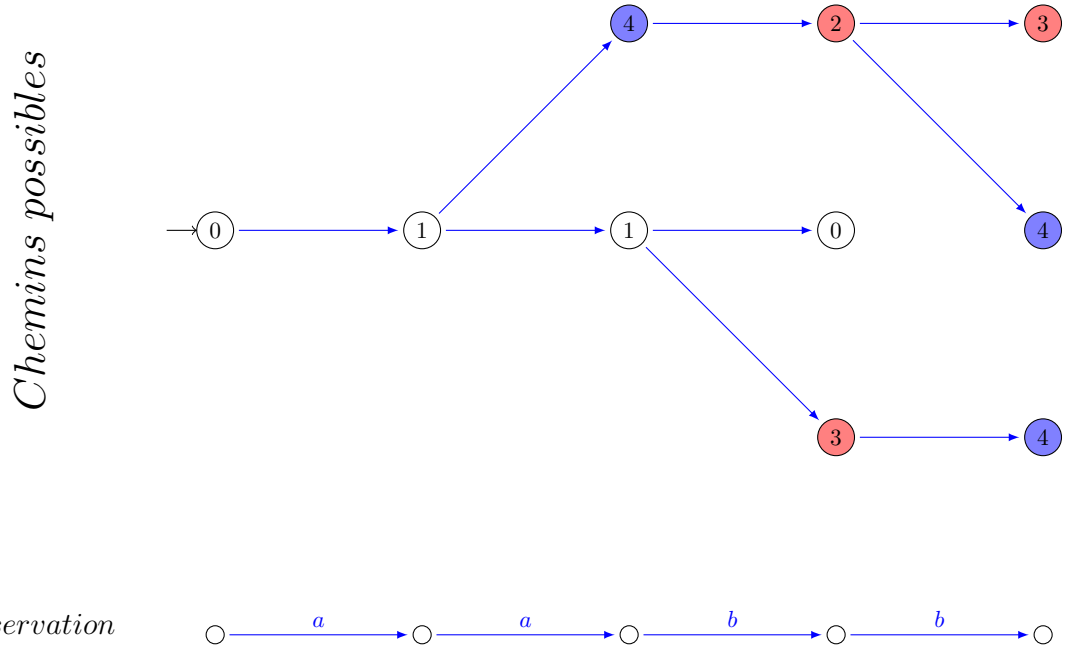


FIGURE 2.7 – Exécutions possibles du DES de la figure 2.6 correspondant à l'observation *aabb*.

Définition 2.4. Une couleur λ est *k-s-opaque* pour un DES $\mathcal{M}$, si pour toute observation $\sigma \in {\Sigma^o_\mathcal{M}}^*$, et pour tout suffixe $\sigma_2 \in {\Sigma^o_\mathcal{M}}^*$ de σ, avec $|\sigma_2| \leq k$ et $\sigma = \sigma_1\sigma_2$: si il existe $q_0 \in Q^0_\mathcal{M}$ et $q_1, q_2 \in Q_\mathcal{M}$ tels que $(q_0 \overset{\sigma_1}{\underset{\mathcal{M}}{\rightsquigarrow}} q_1 \overset{\sigma_2}{\underset{\mathcal{M}}{\rightsquigarrow}} q_2 \wedge (q_1, \lambda) \in col_\mathcal{M})$ alors il existe $q'_0 \in Q^0_\mathcal{M}$ et $q'_1, q'_2 \in Q_\mathcal{M}$ tels que $(q'_0 \overset{\sigma_1}{\underset{\mathcal{M}}{\rightsquigarrow}} q'_1 \overset{\sigma_2}{\underset{\mathcal{M}}{\rightsquigarrow}} q'_2 \wedge (q'_1, \lambda) \notin col_\mathcal{M})$.

La k-s-opacité définie ici correspond à la *k-step opacity* définie par [36].

Problème. Le problème de la k-s-opacité est le suivant :

Données : un DES $\mathcal{M}$ et une couleur λ.

Question : La couleur λ est-elle k-s-opaque pour $\mathcal{M}$?

Pour résoudre ce problème, on procède en trois étapes. Dans un premier temps, on encode les k dernières couleurs d'état pour chaque exécution. Dans un second temps, on construit une version modifiée d'un moniteur. Enfin, on vérifie l'accessibilité d'un état colorié par **oui**.

Pour encoder les occurrences de λ sur les k derniers états, on fait le produit de chaque état avec un sous ensemble de $[k]$. Plus formellement, on construit le DES $\mathcal{M}_{\|k}$, définit de la manière suivante :

- $Q_{\mathcal{M}_{\|k}} = Q_\mathcal{M} \times 2^{[k]}$,
- $\Sigma_{\mathcal{M}_{\|k}} = \Sigma_\mathcal{M}$,
- $\Lambda_{\mathcal{M}_{\|k}} = 2^{[k]}$,
- $Q^0_{\mathcal{M}_{\|k}} = \big\{(q, \emptyset) \mid q \in Q^0_\mathcal{M}, (q, \lambda) \notin col_\mathcal{M}\big\} \cup \big\{(q, \{0\}) \mid q \in Q^0_\mathcal{M}, (q, \lambda) \in col_\mathcal{M}\big\}$,
- $(q, K) \xrightarrow[\mathcal{M}_{\|k}]{a} (q', K')$ si $q \xrightarrow[\mathcal{M}]{a} q'$ et $\begin{cases} & (q', \lambda) \in col_\mathcal{M} \wedge K' = \{i \leq k \mid i - 1 \in K\} \cup \{0\} \\ \text{ou} & (q, \lambda) \notin col_\mathcal{M} \wedge K' = \{i \leq k \mid i - 1 \in K\} \end{cases}$
- $col_{\mathcal{M}_{\|k}} = \big\{((q, K), K) \mid q \in Q_\mathcal{M}, K \in 2^{[k]}\big\}$.

Exemple 12. *Le DES représenté à la figure 2.8, avec λ représenté en bleu, est la partie accessible depuis l'état initial du DES $\mathcal{M}_{\|k}$ obtenu pour $k = 2$ et avec le DES de la figure 2.6. On représente les sous-ensembles de $[k]$ à l'aide de $k+1$-uplets sur $\{0, 1\}$ (des triplets dans le cas présent). Cette représentation a pour but de faciliter l'interprétation de ces ensembles comme une fenêtre glissante sur l'historique, 1 représentant la présence de λ et 0 son absence.*

L'opération suivante consiste à construire un moniteur $\mathcal{M}on$ à partir de $\mathcal{M}_{\|k}$. Cependant, il n'y a plus une unique couleur, mais 2^{k+1} couleurs. Il faut donc définir précisément à quels états sont associés les couleurs **oui**, **non** et **equi**. Le DES $\mathcal{M}on$ est construit à partir de $\mathcal{M}_{\|k}$ en utilisant la construction par parties. Tous les éléments sont définis exactement de la même manière qu'en 2.1.2 à l'exception du coloriage qui est définie de la manière suivante :

$$\begin{array}{ll} (P, \textbf{oui}) \in col_{\mathcal{M}on} & \text{si } \exists i \in \mathbb{N}, \forall (p, K) \in P, i \in K \\ (P, \textbf{non}) \in col_{\mathcal{M}on} & \text{si } \forall (p, K) \in P, K = \emptyset \\ (P, \textbf{equi}) \in col_{\mathcal{M}on} & \text{sinon} \end{array}$$

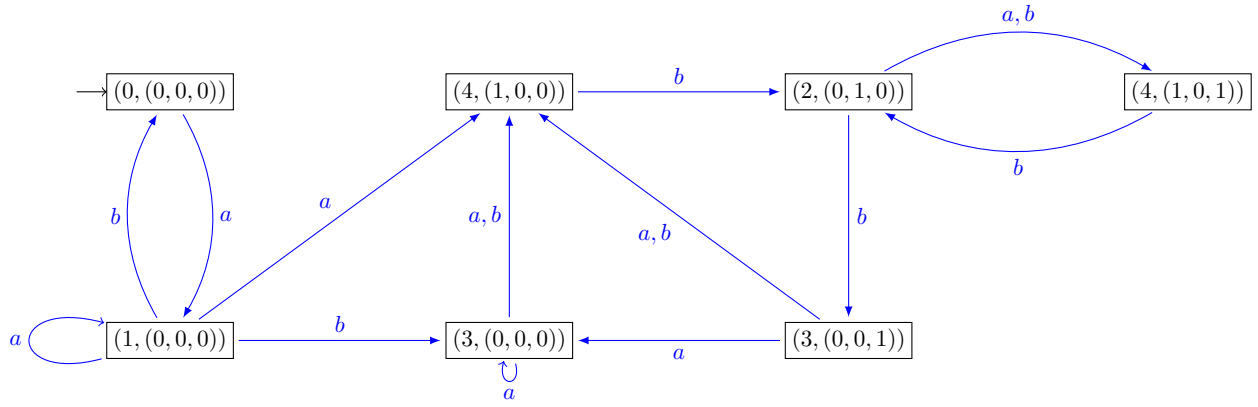


FIGURE 2.8 – DES $\mathcal{M}_{\|k}$ obtenu avec $k = 2$ et le DES de la figure 2.6

Ensuite, il suffit simplement de vérifier si un état coloré par **oui** est accessible dans $\mathcal{M}on$.

2.4.2 Opacité bornée asynchrone

Le but de cette variante est de détecter si, dans un passé borné, tous les chemins ont traversé un état coloré par λ.

Définition 2.5. Une couleur λ est *k-a-opaque* pour un DES $\mathcal{M}$, si pour toute observation $\sigma \in {\Sigma^o_\mathcal{M}}^*$ telle que $|\sigma_2| \leq k$ et $\sigma = \sigma_1\sigma_2$: si il existe $q_0 \in Q^0_\mathcal{M}$ et $q_1, q_2 \in Q_\mathcal{M}$ tels que $(q_0 \overset{\sigma_1}{\underset{\mathcal{M}}{\rightsquigarrow}} q_1 \overset{\sigma_2}{\underset{\mathcal{M}}{\rightsquigarrow}} q_2 \wedge (q_1, \lambda) \in col_\mathcal{M})$ alors il existe un chemin γ tel que $tr(\gamma) = \sigma_1\sigma_2$ et tel que pour tout $i \leq k$, $(fin(\gamma(|\gamma| - i)), \lambda) \notin col_\mathcal{M}$.

Dit de façon moins formelle, la k-a-opacité reflète qu'on peut toujours trouver, pour le suffixe de longueur au plus k d'une observation, un tronçon d'exécution compatible, qui évite totalement la couleur λ.

Problème. Le problème de la k-a-opacité est le suivant :

Données : un DES $\mathcal{M}$ et une couleur λ.

Question : La couleur λ est-elle k-a-opaque pour $\mathcal{M}$?

Pour résoudre ce problème, on procède en deux temps. Dans un premier temps, on construit un système $\mathcal{M}_{\tilde{k}}$ qui encode l'occurrence la plus proche d'un état coloré par λ pour chaque exécution. Ensuite, on vérifie l'opacité ordinaire sur $\mathcal{M}_{\tilde{k}}$.

Cette construction est plus simple que la précédente. Cependant comme on est maintenant asynchrone, il faut aussi mémoriser les passages par des états colorés lors de chemins non-observables. Formellement, on construit le DES $\mathcal{M}_{\tilde{k}} = (Q_{\mathcal{M}_{\tilde{k}}}, \Sigma_{\mathcal{M}_{\tilde{k}}}, \Lambda_{\mathcal{M}_{\tilde{k}}}, Q^0_{\mathcal{M}_{\tilde{k}}}, \Delta_{\mathcal{M}_{\tilde{k}}}, col_{\mathcal{M}_{\tilde{k}}})$ de la manière suivante :

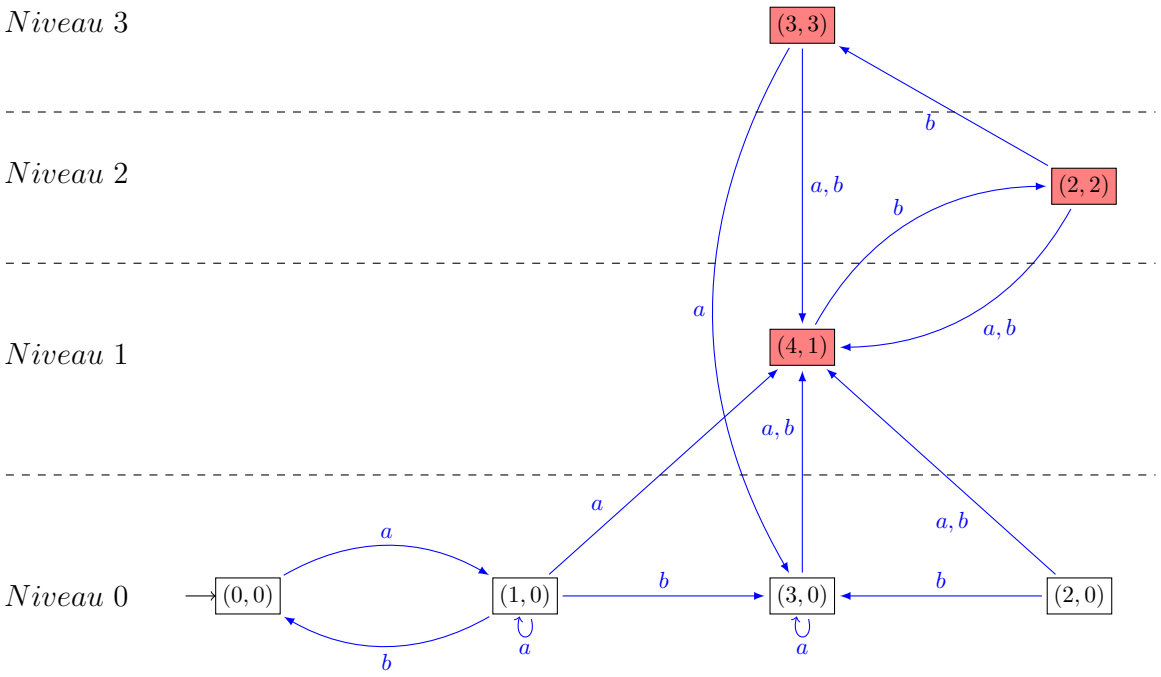


FIGURE 2.9 – DES $\mathcal{M}_{\tilde{k}}$ obtenu avec $k = 2$ et le DES de la figure 2.6

- $Q_{\mathcal{M}_{\tilde{k}}} = Q_{\mathcal{M}} \times \{i \in \mathbb{N} \mid i \leq k+1\}$,
- $\Sigma_{\mathcal{M}_{\tilde{k}}} = \Sigma_{\mathcal{M}}$,
- $\Lambda_{\mathcal{M}_{\tilde{k}}} = \{\lambda\}$,
- $Q^0_{\mathcal{M}_{\tilde{k}}} = \big\{(q,0) \mid q \in Q^0_{\mathcal{M}}, (q,\lambda) \notin col_{\mathcal{M}}\big\} \cup \big\{(q,1) \mid q \in Q^0_{\mathcal{M}}, (q,\lambda) \in col_{\mathcal{M}}\big\}$,
- $(q,i) \xrightarrow[\mathcal{M}_{\tilde{k}}]{a} (q',i')$ si $a \in \Sigma^o_{\mathcal{M}}$, $q \overset{a}{\underset{\mathcal{M}}{\rightsquigarrow}} q'$ et $\begin{cases} (q',\lambda) \in col_{\mathcal{M}} \wedge i' = 1 \\ \text{ou} \quad (q',\lambda) \notin col_{\mathcal{M}} \wedge \begin{cases} i = i' = 0 \\ \text{ou} \quad 1 \leq i \wedge i+1 = i' \\ \text{ou} \quad i = k+1 \wedge i' = 0 \end{cases} \end{cases}$
- $col_{\mathcal{M}_{\tilde{k}}} = \{((q,i),\lambda) \mid i \geq 1\}$.

Exemple 13. *Encore une fois, λ est représenté par la couleur bleue. Le DES illustré à la figure 2.9 est la partie accessible depuis l'état initial du DES $\mathcal{M}_{\tilde{k}}$ obtenu pour $k = 2$ à partir du DES de la figure 2.6. A chaque fois qu'un état coloré est traversé, on atteint le niveau 1. Ensuite, tant qu'on ne traverse pas d'état coloré, on monte dans les niveaux. Si on ne croise pas d'état coloré pendant $k+1$ étapes, on redescend au niveau 0.*

Le DES $\mathcal{M}_{\tilde{k}}$ obtenu est un DES coloré. Il suffit donc d'appliquer la construction du moniteur en 2.1.2 puis de vérifier l'accessibilité d'un état coloré par **oui**.

2.4.3 Opacité infinie synchrone

Le but de cette variante est de détecter si dans un passé non borné, tous les chemins ont traversé un état coloré par λ, au même moment (vis-à-vis des observations). Cette propriété a été étudiée par Saboori et Hadjicostis [60] qui ont proposé une solution que nous allons rappeler ici.

Définition 2.6. Une couleur λ est ∞-*s-opaque* pour un DES $\mathcal{M}$, si pour toute observation $\sigma \in {\Sigma^o_{\mathcal{M}}}^*$, et pour tout suffixe $\sigma_2 \in {\Sigma^o_{\mathcal{M}}}^*$ de σ, avec $\sigma = \sigma_1\sigma_2$: si il existe $q_0 \in Q^0_{\mathcal{M}}$ et $q_1, q_2 \in Q_{\mathcal{M}}$ tels que $(q_0 \overset{\sigma_1}{\underset{\mathcal{M}}{\rightsquigarrow}} q_1 \overset{\sigma_2}{\underset{\mathcal{M}}{\rightsquigarrow}} q_2 \wedge (q_1, \lambda) \in col_{\mathcal{M}})$ alors il existe $q'_0 \in Q^0_{\mathcal{M}}$ et $q'_1, q'_2 \in Q_{\mathcal{M}}$ tels que $(q'_0 \overset{\sigma_1}{\underset{\mathcal{M}}{\rightsquigarrow}} q'_1 \overset{\sigma_2}{\underset{\mathcal{M}}{\rightsquigarrow}} q'_2 \wedge (q'_1, \lambda) \notin col_{\mathcal{M}})$.

La ∞-s-opacité définie ici correspond à l'opacité infinie définie par [60].

Problème. Le problème de la ∞-s-opacité est le suivant :

Données : un DES $\mathcal{M}$ et une couleur λ.

Question : La couleur λ est-elle ∞-s-opaque pour $\mathcal{M}$?

Pour résoudre ce problème, on va là aussi construire un système pour lequel la propriété sera mise en valeur. Cependant, on se contentera de vérifier l'accessibilité dans le système obtenu, d'un ensemble d'état colorés.

La construction présentée dans [60] est assez simple car elle utilise les deux estimateurs introduits en 1.4.2. Il s'agit de construire un estimateur d'état courant. Puis à chaque état P de l'estimateur d'état courant, on concatène un estimateur d'état initial dont l'état initial est $\{(q, q) \mid q \in P\}$. On peut remarquer que le système obtenu est non-déterministe.

On décide de l'opacité en vérifiant s'il existe dans cette construction, un état accessible composé d'un ensemble d'états colorés. Si c'est le cas, le système n'est pas opaque.

La justification de la construction est de choisir de façon non déterministe les deux observations σ et σ'. Il suffit alors en suivant σ de trouver l'ensemble des états accessibles par σ depuis un état initial. Par la suite, on devine aussi σ' qui va nous donner une trajectoire à suivre, afin de raffiner le premier ensemble d'états obtenu en un ensemble qui ne contient que des états colorés par λ.

2.4.4 Opacité infinie asynchrone

La dernière extension d'opacité à laquelle nous nous intéressons est l'opacité infinie asynchrone. Il s'agit en fait de la variante d'opacité la plus simple à vérifier.

Définition 2.7. Une couleur λ est ∞-*a-opaque* pour un DES $\mathcal{M}$, si pour toute observation $\sigma \in {\Sigma^o_{\mathcal{M}}}^*$ telle que $\sigma = \sigma_1\sigma_2$: si il existe $q_0 \in Q^0_{\mathcal{M}}$ et $q_1, q_2 \in Q_{\mathcal{M}}$ tels que $(q_0 \overset{\sigma_1}{\underset{\mathcal{M}}{\rightsquigarrow}} q_1 \overset{\sigma_2}{\underset{\mathcal{M}}{\rightsquigarrow}}$

$q_2 \wedge (q_1, \lambda) \in col_{\mathcal{M}}$) alors il existe un chemin γ tel que $tr(\gamma) = \sigma_1\sigma_2$ et tel que pour tout $i \leq |\gamma|$, $(fin(\gamma(|\gamma| - i)), \lambda) \notin col_{\mathcal{M}}$.

Ainsi, la ∞-a-opacité reflète que pour toute observation qui atteint λ, on peut trouver une exécution compatible qui évite totalement λ.

Problème. Le problème de la ∞-a-opacité est le suivant :

Données : un DES $\mathcal{M}$ et une couleur λ.

Question : La couleur λ est-elle ∞-a-opaque pour $\mathcal{M}$?

Le problème peut être résolu en deux étapes. La première consiste à colorer tous les états accessibles depuis un état coloré. Puis on vérifie l'opacité ordinaire du système ainsi obtenu.

Pour colorer les successeurs des états colorés, nous utilisons la construction suivante. Étant donné un DES $\mathcal{M}$, le DES $\mathcal{M}_{\tilde{\omega}} = (Q_{\tilde{\omega}}, \Sigma_{\tilde{\omega}}, \Lambda_{\tilde{\omega}}, Q^0_{\tilde{\omega}}, \Delta_{\tilde{\omega}}, col_{\tilde{\omega}})$ est obtenu de la façon suivante :

- $Q_{\tilde{\omega}} = Q_{\mathcal{M}} \times \{0, 1\}$
- $\Sigma_{\tilde{\omega}} = \Sigma_{\mathcal{M}}$
- $\Lambda_{\tilde{\omega}} = \{\lambda\}$
- $Q^0_{\tilde{\omega}} = \{(q_0, 0) \mid q_0 \notin col_{\mathcal{M}}(\lambda) \wedge q_0 \in Q^0_{\mathcal{M}}\} \cup \{(q_0, 1) \mid q_0 \in col_{\mathcal{M}}(\lambda) \wedge q_0 \in Q^0_{\mathcal{M}}\}$
- $(q, i) \xrightarrow[\mathcal{M}_{\tilde{\omega}}]{a} (q', i')$ si $a \in \Sigma^o_{\mathcal{M}}$ et $q \overset{a}{\underset{\mathcal{M}}{\rightsquigarrow}} q'$ et $\begin{cases} \text{si} & (q', \lambda) \in col_{\mathcal{M}} \quad \text{alors} \quad i' = 1 \\ \text{sinon, si} & (q', \lambda) \notin col_{\mathcal{M}} \quad \text{alors} \quad i' = i \end{cases}$
- $col_{\tilde{\omega}} = \{((q, 1), \lambda) \mid q \in Q_{\mathcal{M}}\}$

Là encore, le DES obtenu est déjà coloré par λ, il suffit donc simplement de vérifier l'opacité de λ dans $\mathcal{M}_{\tilde{\omega}}$.

2.4.5 Détection de fuites d'informations

La dernière extension à laquelle nous nous intéressons consiste à conjuguer les problèmes d'opacité et de diagnosticabilité ; cette extension a été introduite pour les DES finis dans [30]. Sur le DES $\mathcal{M}$, on considère qu'il existe deux sous-ensembles (qui peuvent avoir une intersection non vide) dans l'alphabet $\Sigma_{\mathcal{M}}$. L'ensemble $\Sigma^a_{\mathcal{M}} \subseteq \Sigma_{\mathcal{M}}$ est l'alphabet de l'*attaquant* et l'ensemble $\Sigma^d_{\mathcal{M}} \subseteq \Sigma_{\mathcal{M}}$ est l'alphabet du *défenseur*. L'attaquant est confronté à un problème de supervision classique (§ 2.1.2) : il doit déterminer si le système est dans un état coloré par une couleur $\lambda \in \Lambda_{\mathcal{M}}$. Il observe le système à travers l'alphabet $\Sigma^a_{\mathcal{M}}$. Le but du défenseur est de déterminer si l'attaquant a pu déduire que le système était dans un état coloré par λ. Il observe le système à travers l'alphabet $\Sigma^d_{\mathcal{M}}$ et doit résoudre un problème de diagnosticabilité dont la propriété est *« l'attaquant sait que le système est passé par un état coloré »*.

Nous rappelons à présent la méthode de [30]. Comme l'attaquant doit résoudre un problème de supervision, le moniteur de λ a entièrement la connaissance de celui-ci. On note $\mathcal{M}on$ le moniteur obtenu grâce à la construction par parties présentée en 2.1.2. Cependant, pour la

diagnosticabilité, on s'intéresse à des langages qui sont clos par extension. En effet, une fois que l'attaquant sait que le système est passé par un état coloré par λ, il n'oublie pas cette information. Il faut donc transformer le moniteur afin de refléter cette propriété.

A partir de $\mathcal{M}on = (Q_{\mathcal{M}on}, \Sigma_{\mathcal{M}on}, \Lambda_{\mathcal{M}on}, Q^0_{\mathcal{M}on}, \Delta_{\mathcal{M}on}, col_{\mathcal{M}on})$, on construit le DES $Att = (Q_{Att}, \Sigma_{\mathcal{M}}, \Lambda_{Att}, Q^0_{\mathcal{M}on}, \Delta_{Att}, col_{Att})$ défini de la façon suivante :

- $Q_{Att} = Q_{\mathcal{M}on} \times \{1, 2\}$,
- $\Lambda_{Att} = \{\lambda'\}$ avec $\lambda \notin (\Lambda_{\mathcal{M}} \cup \{\mathbf{oui}, \mathbf{non}, \mathbf{equi}\})$ une nouvelle couleur
- $(q, x) \xrightarrow[a]{Att} (q', x')$ $\begin{cases} \text{si } (x = x' = 1) & \wedge \ (q \xrightarrow[\mathcal{M}on]{a} q') & \wedge \ ((q, \mathbf{oui}) \notin col_{\mathcal{M}}) \\ \text{ou } (x = 1) \wedge (x' = 2) & \wedge \ (q \xrightarrow[\mathcal{M}on]{a} q') & \wedge \ ((q, \mathbf{oui}) \in col_{\mathcal{M}}) \\ \text{ou } (x = x' = 2) & \wedge \ (q \xrightarrow[\mathcal{M}on]{a} q') & \end{cases}$
- $((q, x), \lambda') \in col_{Att}$, si $(q, \mathbf{oui}) \in col_{\mathcal{M}on}$ ou $x = 2$

Le DES Att est un descripteur externe de la propriété que l'on cherche à diagnostiquer. Comme exprimé en 2.1.1, il est donc possible de se ramener au cas des couleurs en faisant le produit entre Att et le DES initial $\mathcal{M}$, on le note alors $\mathcal{M}_{\times A}$.

Ces constructions permettent d'établir le résultat suivant :

Proposition 2.1. *[30] Une fuite d'information de $\mathcal{M}$ vers un attaquant observant $\Sigma^a_{\mathcal{M}}$ est détectée de manière certaine si, et seulement si, λ' est diagnosticable dans $\mathcal{M}_{\times A}$ par rapport à $\Sigma^d_{\mathcal{M}}$.*

Ainsi, pour résoudre ce problème, on peut utiliser les techniques de vérification de la diagnosticabilité déjà vues.

Dans ce chapitre nous avons examiné un ensemble de problèmes d'observations partielles. Nous avons rappelé les résultats connus pour les systèmes à états finis. Dans la suite de ce document, nous examinerons ces problèmes pour des familles particulières de systèmes infinis.

Chapitre 3

Test de conformité

3.1 Généralités sur le test

Des défauts de conception pour un logiciel critique peuvent entraîner d'importantes pertes, humaines ou monétaires. Afin d'éviter ces pertes, il est souhaitable de passer par une phase de validation, mais cela peut être coûteux en temps et n'apporter qu'une confiance relative dans le logiciel validé. Pour minimiser ce temps et augmenter cette confiance dans la phase de validation, il peut être intéressant de la formaliser et de l'automatiser. Deux grandes méthodes de validation sont la vérification et le test. Pour la vérification, on s'assure que le système vérifie un certain nombre de propriétés, souvent en analysant son code, ou un modèle. Le *test* au contraire, va consister à utiliser le système dans des conditions aussi proches que possible de son environnement réel d'utilisation, tout en cherchant à couvrir un ensemble large de cas d'utilisation.

Il est possible de tester différents aspects du système : les performances (en quantité ou en qualité), la conformité ou la robustesse. Les performances peuvent être le temps de réponse du système, la quantité de données traitées ou le taux d'erreur. La conformité consiste à vérifier que le système a un comportement correct (pour un sens précis que nous définirons) tandis que la robustesse cherche à vérifier que son comportement est acceptable quand les entrées sont incorrectes.

Par la suite, nous nous concentrons la conformité, plus précisément le test *« boite noire »*. Dans ce contexte, on suppose disposer d'une *spécification* du comportement attendu du système, ainsi que d'une *implémentation*. Aucune supposition n'est faite sur la réalisation exacte de cette implémentation. De plus, nous n'avons pas accès au code source de cet objet. Ainsi l'interaction avec cette implémentation se fait suivant les événements observables de la spécification. On considère que ces événements sont de deux natures : les entrées qui devront être fournies par l'utilisateur (et donc le testeur) et les sorties qui ne sont pas contrôlées par l'utilisateur.

Tester un système se fait en exécutant, sur l'implémentation, une *suite de tests*, qui est un ensemble de *cas de test*, sur l'implémentation. A l'issue d'un cas de test, celui-ci peut échouer si une erreur est détectée, ou réussir dans le cas contraire. Quand il s'agit de tester une fonction, le cas de test est simplement une valeur d'entrée et une valeur de sortie attendue. Le test réussi si la valeur de sortie est la valeur attendue, et échoue sinon. Pour les systèmes réactifs, qui peuvent interagir avec leur environnement, il faut non seulement donner une séquence d'entrée, mais aussi tenir compte des sorties observées. Afin de modéliser ces systèmes réactifs, nous utiliserons le modèle des IOLTS introduit dans le chapitre 1. Pour un IOLTS $\mathcal{M} = (Q_{\mathcal{M}}, \Sigma_{\mathcal{M}}, \Lambda_{\mathcal{M}}, Q^0_{\mathcal{M}}, \Delta_{\mathcal{M}}, col_{\mathcal{M}})$, on notera $\Sigma^?_{\mathcal{M}}$ les entrées et $\Sigma^!_{\mathcal{M}}$ les sorties. De plus on supposera que $\Sigma_{\mathcal{M}}$ est une partition avec $\Sigma_{\mathcal{M}} = \Sigma^?_{\mathcal{M}} \cup \Sigma^!_{\mathcal{M}} \cup \{\tau\} = \Sigma^o_{\mathcal{M}} \cup \{\tau\}$ où τ représente les actions internes (qu'il est inutile de distinguer).

Cas de test Comme nous nous intéressons dans ce document au test de conformité de type boite noire, nous n'avons accès qu'aux traces de l'implémentation. Ces traces sont composées d'entrées et de sorties, le testeur va donc devoir choisir les entrées. Le cas de test a alors deux intérêts, étant donnée une séquence d'observations : d'une part proposer des entrées, et d'autre part renvoyer des verdicts.

Plus formellement, on considérera que le cas de test est représenté par un IOLTS $\mathcal{TC} = (Q_{\mathcal{TC}}, \Sigma_{\mathcal{TC}}, \Lambda_{\mathcal{TC}}, Q^0_{\mathcal{TC}}, \Delta_{\mathcal{TC}}, col_{\mathcal{TC}})$ tel que la couleur d'un état représente le verdict retourné par une exécution qui aboutit à cet état. Le comportement de l'implémentation est modélisé par un IOLTS $\mathcal{I} = (Q_{\mathcal{I}}, \Sigma_{\mathcal{I}}, \Lambda_{\mathcal{I}}, Q^0_{\mathcal{I}}, \Delta_{\mathcal{I}}, col_{\mathcal{I}})$. L'exécution de $\mathcal{TC}$ sur $\mathcal{I}$ forme un nouveau système qui peut être modélisée par le produit $\mathcal{TC} \times \mathcal{I}$ où les actions communes sont synchronisées. Le produit reconnaît les traces de $\mathcal{I}$ sous la contrainte de $\mathcal{TC}$.

Pour définir les verdicts, on utilise des couleurs, le verdict renvoyé après une exécution est la couleur de l'état atteint suite à cette exécution. Si on veut que chaque test ne renvoie qu'un seul verdict, il faudra alors que chaque état n'ait qu'une seule couleur. Nous nous intéressons dans un premier temps, au verdict *Échec* qui est renvoyé quand une erreur est détectée. Il faut noter que, du fait du non-déterminisme de l'implémentation, plusieurs verdicts peuvent être renvoyés par un même cas de test ; on parlera donc de *possibilité* d'échec d'un test. Si un test $\mathcal{TC}$ à la possibilité d'échouer sur l'implémentation $\mathcal{I}$, on note $\mathcal{I}$ *fails* $\mathcal{TC}$. Formellement,

$$\mathcal{I} \textit{ fails } \mathcal{TC} \stackrel{\textbf{def}}{=} \mathsf{Obs}_{\mathcal{I}} \cap \mathsf{Obs}_{\mathcal{TC}}(\textit{Échec}) \neq \emptyset. \tag{3.1}$$

Dans le cadre du test boite noire, l'implémentation, est inconnue ; il n'est donc pas possible de faire directement le produit. L'objectif est donc d'être capable de synthétiser un ensemble de cas de test (une *suite de test*) qui vérifie un ensemble de propriétés quelle que soit l'implémentation sur laquelle il est exécuté. Pour définir ces propriétés, il nous faut d'abord formaliser ce qu'est le comportement conforme d'une implémentation.

3.2 La relation de conformité IOCO

La première étape pour estimer si un système a un comportement conforme est d'en donner une spécification. La spécification doit être présentée dans un langage clair et suffisamment expressif ; nous utiliserons pour cela les IOLTS. Nous disposons donc de deux objets : une spécification que nous noterons $\mathcal{S} = (Q_{\mathcal{S}}, \Sigma_{\mathcal{S}}, \Lambda_{\mathcal{S}}, Q^0_{\mathcal{S}}, \Delta_{\mathcal{S}}, col_{\mathcal{S}})$ et une implémentation qui sera supposée être modélisée par un IOLTS $\mathcal{I} = (Q_{\mathcal{I}}, \Sigma_{\mathcal{I}}, \Lambda_{\mathcal{I}}, Q^0_{\mathcal{I}}, \Delta_{\mathcal{I}}, col_{\mathcal{I}})$. On suppose que ces deux IOLTS reposent sur le même alphabet d'actions observables $\Sigma^o_{\mathcal{S}} = \Sigma^o_{\mathcal{I}}$. La seconde étape, détaillée plus loin, consiste à définir une relation de conformité entre une implémentation et une spécification. Ici nous considérons la relation **ioco**(pour Input Output Conformance).

On peut supposer, par ailleurs, que l'implémentation est complète pour les entrées. Si cette supposition n'est pas vérifiée, il est possible de considérer l'un des cas suivant : l'entrée est acceptée mais n'a aucun effet (boucle sur un état) ou l'entrée est acceptée et entraîne un dysfonctionnement du système (envoie sur un état puits).

Enfin on note $Out_{\mathcal{M}}(q)$, l'ensemble des événements visibles qui peuvent se produire immédiatement après l'état q, dans $\mathcal{M}$: $Out_{\mathcal{M}}(q) \stackrel{\textbf{def}}{=} \left\{ a \in \Sigma^o_{\mathcal{M}} \mid \exists q' \in Q_{\mathcal{M}}, q \overset{a}{\underset{\mathcal{M}}{\rightsquigarrow}} q' \right\}$. Par extension, pour une observation σ, $Out_{\mathcal{M}}(\sigma) \stackrel{\textbf{def}}{=} \left\{ a \in \Sigma^o_{\mathcal{M}} \mid \exists q_0 \in Q^0_{\mathcal{M}}, \exists q \in Q_{\mathcal{M}}, q_0 \overset{\sigma}{\underset{\mathcal{M}}{\rightsquigarrow}} q, a \in Out_{\mathcal{M}}(q) \right\}$. Enfin, on note $Out^?_{\mathcal{M}}(\sigma)$ et $Out^!_{\mathcal{M}}(\sigma)$ les restrictions respectives de $Out_{\mathcal{M}}(\sigma)$ sur les entrées et les sorties.

3.2.1 Blocages

Nous avons vu que l'ensemble des actions observables est partagé en entrées et sorties. Cependant, si il est possible d'observer les sorties, il est aussi possible d'observer l'absence de celles-ci en utilisant des *temporisateurs*. Ces temporisateurs permettent d'observer un « blocage ». Il est alors nécessaire de savoir si un tel blocage est autorisé par la spécification. Ces blocages sont considérés comme des sorties, et sont modélisés par sortie particulière qu'on note δ (avec $\delta \notin \Sigma_{\mathcal{S}}$).

Dans un IOLTS $\mathcal{M}$, il existe deux types de blocages :

- Un état q est en *blocage de sortie* (outputlock) si il n'existe aucune transition de sortie ou de transition interne franchissable, $Out_{\mathcal{M}}(q) \subseteq \Sigma^?_{\mathcal{M}}$. En particulier il est en *blocage total* (deadlock) si il n'existe aucune transition franchissable du tout, $Out_{\mathcal{M}}(q) = \emptyset$.
- Un état q est en *blocage vivant* (livelock) si il appartient à un chemin infini d'actions internes, $\forall n \in \mathbb{N}, \exists w \in (\Sigma_{\mathcal{M}})^n, \exists q_n \in Q_{\mathcal{M}}, \pi(w) = \varepsilon \wedge q \overset{w}{\underset{\mathcal{M}}{\Longrightarrow}} q'$. Si il appartient à un circuit c'est un *blocage vivant cyclique*. Si on s'intéresse à des systèmes infinis, il peut exister un nombre infini d'état différents dans ce chemin, on parle alors de *blocage vivant divergent*.

On note $quiescent(\mathcal{M})$ l'ensemble des états en situation de blocage dans l'IOLTS $\mathcal{M}$.

Suspension Pour un IOLTS $\mathcal{M}$, on peut ainsi définir un nouvel IOLTS $Susp(\mathcal{M})$, qui est la *Suspension* de $\mathcal{M}$, dans lequel les blocages sont rendus explicites par l'ajout de boucles étiquetées par δ.

Définition 3.1 (Suspension)**.** Soit $\mathcal{M} = (Q_{\mathcal{M}}, \Sigma_{\mathcal{M}}, \Lambda_{\mathcal{M}}, Q^0_{\mathcal{M}}, \Delta_{\mathcal{M}}, col_{\mathcal{M}})$ un IOLTS : sa *suspension*, notée $Susp(\mathcal{M})$ est l'IOLTS $(Q_{\mathcal{M}}, \Sigma_{Susp(\mathcal{M})}, \Lambda_{\mathcal{M}}, Q^0_{\mathcal{M}}, \Delta_{Susp(\mathcal{M})}, col_{\mathcal{M}})$ où :
- $\Sigma_{Susp(\mathcal{M})} = \Sigma_{\mathcal{M}} \cup \{\delta\}$ avec $\delta \in \Sigma^!_{Susp(\mathcal{M})}$ et $\delta \notin \Sigma_{\mathcal{M}}$,
- $\Delta_{Susp(\mathcal{M})} = \Delta_{\mathcal{M}} \cup \{(q, \delta, q) \mid q \in quiescent(M)\}$.

On peut remarquer que $Susp(\mathcal{M})$ peut ne pas être calculable pour un IOLTS avec un nombre infini d'états. Par la suite, on notera $\Sigma^{!\delta}_{\mathcal{M}}$ pour $\Sigma^!_{\mathcal{M}} \cup \{\delta\}$ et $\Sigma^{o\delta}_{\mathcal{M}}$ pour $\Sigma^o_{\mathcal{M}} \cup \{\delta\}$.

Les observations de $Susp(\mathcal{M})$ notées $\mathsf{SObs}_{\mathcal{M}}$ sont appelées les *observations suspendues* de $\mathcal{M}$. Par extension on note $\mathsf{SObs}_{\mathcal{M}}(\lambda)$ les observations suspendues de $\mathcal{M}$ qui atteignent λ. Elles représentent le comportement observable de $\mathcal{M}$, en prenant en compte les blocages, et sont à la base de la définition de la relation de conformité **ioco**.

3.2.2 Définition de la relation de conformité ioco

La relation de conformité **ioco** (pour Input Output COnformance) a été introduite par Tretmans [64]. Étant donnée une spécification donnée par un IOLTS $\mathcal{S}$, une implémentation donnée par un IOLTS $\mathcal{I}$ est dite conforme à sa spécification $\mathcal{S}$ si, après n'importe quelle trace suspendue de $\mathcal{S}$ exécutée sur $\mathcal{I}$, l'implémentation $\mathcal{I}$ ne retourne que des sorties spécifiées dans $\mathcal{S}$. Plus formellement :

Définition 3.2 (Relation de conformité)**.** Soient $\mathcal{S}$ et $\mathcal{I}$ deux IOLTS avec le même alphabet d'événements observables ($\Sigma^o_{\mathcal{S}} = \Sigma^o_{\mathcal{I}}$), et $\mathcal{I}$ complet en entrée ; la relation **ioco** est définie comme suit :

$$\mathcal{I} \textbf{ ioco } \mathcal{S} \quad \stackrel{\textbf{def}}{=} \quad \forall \sigma \in \mathsf{SObs}_{\mathcal{S}}, Out^!_{Susp(\mathcal{I})}(\sigma) \subseteq Out^!_{Susp(\mathcal{S})}(\sigma).$$

N'oublions pas que l'objectif du test de conformité est de vérifier que le système répond d'une façon acceptable quand il reçoit une entrée correcte. Ainsi, **ioco** n'impose pas que les traces de la spécification soient exactement les mêmes que celles d'une implémentation pour que cette dernière soit conforme. Précisément, la relation de conformité **ioco** autorise deux cas où les traces de l'implémentation peuvent différer de celles de la spécification.

En cas d'entrée non-spécifiée, toutes les sorties qui suivent sont considérées comme conformes. En effet, ce qui nous intéresse est de tester le comportement d'un système dans des conditions normales, ce qui n'est pas le cas avec une entrée non spécifiée.

La non observation d'une sortie ne rend pas une implémentation non-conforme. La conformité consiste à s'assurer qu'il ne se produit pas de comportement indésirable et non que tous les comportements possibles se produisent effectivement. Ceci ne signifie pas, pour autant, que ne

rien observer est conforme. En effet, ne rien observer signifie observer un blocage, qui est une sortie et doit donc être spécifié. Cette possibilité de non observation se produit donc quand plusieurs sorties conformes sont possibles et que l'une d'elles n'est jamais observée.

Exemple 14. *La figure 3.1 présente deux exemples de sorties non conformes (remplacement d'une sortie et création d'une nouvelle sortie). On peut aussi voir les deux modifications par rapport à la spécification qui ne compromettent pas la conformité. Les implémentations sont complètes en entrée, mais par soucis de simplification, les entrées non-spécifiées ne sont pas représentées.*

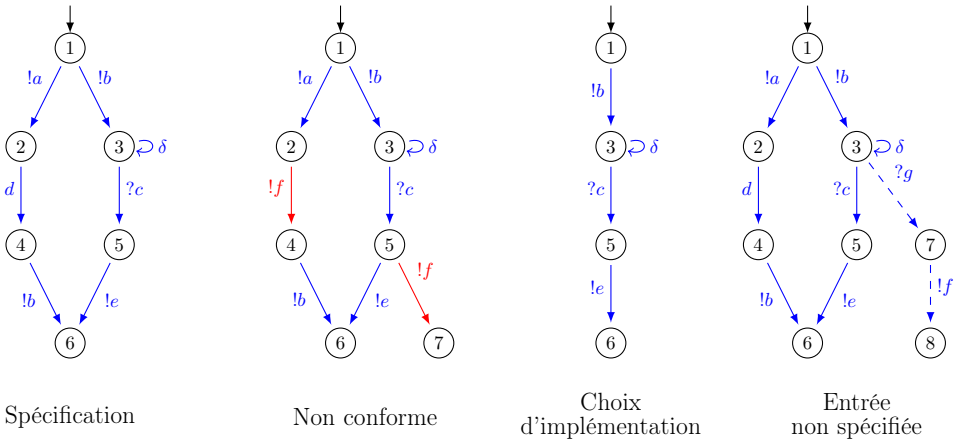


FIGURE 3.1 – Illustration de la relation de conformité **ioco**

Il est possible d'exprimer la relation de conformité avec les traces de la spécification et de l'implémentation. On appelle *observations fautives minimale* d'un modèle $\mathcal{M}$, notées $\mathsf{MinFObs}(\mathcal{M})$, les observations σ de la spécification suivies d'une sortie non spécifiée après σ. Formellement, $\mathsf{MinFObs}(\mathcal{M}) \stackrel{\textbf{def}}{=} (\mathsf{SObs}_{\mathcal{M}} \cdot \Sigma^{!\delta}_{\mathcal{M}}) \setminus \mathsf{SObs}_{\mathcal{M}}$. Il a été prouvé [43] que

$$\mathcal{I} \textbf{ ioco } \mathcal{M} \Leftrightarrow \mathsf{SObs}_{\mathcal{I}} \cap \mathsf{MinFObs}(\mathcal{M}) = \emptyset. \tag{3.2}$$

On peut noter que l'ensemble des traces non-conformes est alors $\mathsf{MinFObs}(\mathcal{M}) \cdot \Sigma_{\mathcal{M}}{}^*$. C'est cette formulation de **ioco** que nous utiliserons par la suite.

3.3 Cas de tests

Revenons à la génération de cas de test. Maintenant que nous avons défini quel est le bon comportement d'une implémentation, nous pouvons définir ce qu'est une bonne suite de tests. Étant donné que l'implémentation est quelconque, elle ne peut pas être utilisée pour définir ce qu'est une bonne suite de test. Il est donc nécessaire d'identifier un ensemble de propriétés qui pourront être déterminées indépendamment de l'implémentation [45].

3.3.1 Equations préliminaires

Pour une suite de tests $\mathcal{TS}$, on s'intéresse à trois propriétés : correction, exhaustivité et sévérité. Il est possible de donner une définition générale pour ces propriétés, mais elles peuvent aussi être exprimées en termes d'observations en utilisant la formulation de [43] afin de remplacer la caractérisation de **ioco** par celle sur les observations.

L'expression en termes de traces utilise le fait que dans l'implémentation, les blocages sont rendus explicites via des temporisateurs. Il suppose donc que :

$$\mathsf{Obs}_{\mathcal{I}} = \mathsf{SObs}_{\mathcal{I}} \tag{3.3}$$

Nous utilisons aussi les équations 3.1 et 3.2, ainsi que le résultat suivant sur les langages.

Ce résultat va nous permettre d'établir des conditions simples à vérifier pour les propriétés évoquées ci-dessus. On peut remarquer que par définition, les mots engendrés par une implémentation forment un langage clos par préfixe (si ua est produit pour $u \in \Sigma^*$, alors u a été produit). Ainsi, on note $Pc(\Sigma^*)$ la famille des langages préfixe clos sur Σ^*, et $Exc(\Sigma^*)$ la famille de leurs complémentaires (les langages clos par extension, donc les langages L tels que $L \cdot \Sigma^* \subseteq L$).

Lemme 2. *Pour tout langages L_1, L_2 sur Σ^* la formule suivante est vraie :*

$$(\forall L \in Pc(\Sigma^*), (L \cap L_1 = \emptyset) \implies (L \cap L_2 = \emptyset)) \implies L_2 \subseteq L_1 \cdot \Sigma^*$$

Le lemme 2 exprime que si un langage L_2 n'a pas d'intersection avec un ensemble clos par préfixe quelconque L, dès qu'un langage L_1 n'en a pas, alors L_2 est inclus dans la clôture par extension de L_1.

Démonstration. Posons $L' = \Sigma^* \setminus (L_1 \cdot \Sigma^*)$. Comme $L_1 \cdot \Sigma^*$ est clos par extension, alors L' est préfixe clos. Si la prémisse de ce lemme est vraie, alors en particulier pour L', on a : $((\Sigma^* \setminus L_1 \cdot \Sigma^*) \cap L_1 = \emptyset) \implies ((\Sigma^* \setminus L_1 \cdot \Sigma^*) \cap L_2 = \emptyset)$ ce qui implique $(\Sigma^* \setminus L_1 \cdot \Sigma^*) \cap L_2 = \emptyset$ et donc $L_2 \subseteq L_1 \cdot \Sigma^*$. □

Dans la suite, on notera $\mathcal{IMP}$ l'ensemble des langages possibles pour les implémentations. Cette famille correspond, pour un alphabet d'événements observables fixé Σ^o, à l'ensemble $Pc(\Sigma^{o*})$.

3.3.2 Correction

Définition 3.3. $\mathcal{TS}$ est *correcte* (on dit parfois non biaisée) si lorsqu'une implémentation est correcte, aucun test ne la rejette.

Plus formellement :

$$\forall \mathcal{I} \in \mathcal{IMP}, (\mathcal{I} \textbf{ ioco } \mathcal{S} \implies \forall \mathcal{TC} \in \mathcal{TS}, \neg(\mathcal{I} \textit{ fails } \mathcal{TC}))$$

En utilisant les équations 3.1, 3.2 et 3.3 :

$$(\forall \mathcal{I} \in \mathcal{IMP}, (\mathsf{SObs}_{\mathcal{I}} \cap \mathsf{MinFObs}(\mathcal{S})) = \emptyset) \implies \forall \mathcal{TC} \in \mathcal{TS}, \mathsf{SObs}_{\mathcal{I}} \cap \mathsf{Obs}_{\mathcal{TC}}(\acute{E}chec) = \emptyset$$

On remplace la quantification sur $\mathcal{TC}$ par une union :

$$(\forall \mathcal{I} \in \mathcal{IMP}, (\mathsf{SObs}_{\mathcal{I}} \cap \mathsf{MinFObs}(\mathcal{S})) = \emptyset) \implies (\mathsf{SObs}_{\mathcal{I}} \cap \bigcup_{\mathcal{TC} \in \mathcal{TS}} \mathsf{Obs}_{\mathcal{TC}}(\acute{E}chec)) = \emptyset$$

Enfin on utilise le lemme 2 afin de supprimer le quantificateur sur $\mathcal{I}$:

$$\bigcup_{\mathcal{TC} \in \mathcal{TS}} \mathsf{Obs}_{\mathcal{TC}}(\acute{E}chec) \subseteq \mathsf{MinFObs}(\mathcal{S}) \cdot \Sigma^{o*}$$

3.3.3 Exhaustivité

Définition 3.4. $\mathcal{TS}$ est *exhaustive* si pour toute implémentation non conforme, il existe un test qui peut échouer.

Formellement :

$$\forall \mathcal{I} \in \mathcal{IMP}, (\neg(\mathcal{I} \ \mathbf{ioco} \ \mathcal{S}) \implies \exists \mathcal{TC} \in \mathcal{TS}, \mathcal{I} \ \textit{fails} \ \mathcal{TC})$$

On utilise à nouveau les équations 3.1, 3.2 et 3.3 et on remplace la quantification sur $\mathcal{TC}$ par une union :

$$\forall \mathcal{I} \in \mathcal{IMP}, (\mathsf{SObs}_{\mathcal{I}} \cap \mathsf{MinFObs}(\mathcal{S})) \neq \emptyset \implies (\mathsf{SObs}_{\mathcal{I}} \cap \bigcup_{\mathcal{TC} \in \mathcal{TS}} \mathsf{Obs}_{\mathcal{TC}}(\acute{E}chec)) \neq \emptyset$$

La contraposée de cette assertion avec le lemme 2 induit :

$$\mathsf{MinFObs}(\mathcal{S}) \subseteq \bigcup_{\mathcal{TC} \in \mathcal{TS}} \mathsf{Obs}_{\mathcal{TC}}(\acute{E}chec) \cdot \Sigma^{o*}$$

3.3.4 Sévérité

Définition 3.5. $\mathcal{TS}$ est *stricte* si, toute exécution révélant une non conformité fait échouer tout cas de test compatible.

$$\forall \mathcal{I} \in \mathcal{IMP}, \forall \mathcal{TC} \in \mathcal{TS}, \neg((\mathcal{TC} \times \mathcal{I}) \ \mathbf{ioco} \ \mathcal{S}) \implies \mathcal{I} \ \textit{fails} \ \mathcal{TC}$$

On utilise les équations 3.1, 3.2 et 3.3 :

$$\forall \mathcal{I} \in \mathcal{IMP}, \forall \mathcal{TC} \in \mathcal{TS}, (\mathsf{SObs}_{\mathcal{TC}} \cap \mathsf{SObs}_{\mathcal{I}} \cap \mathsf{MinFObs}(\mathcal{S})) \neq \emptyset \implies (\mathsf{SObs}_{\mathcal{I}} \cap \mathsf{Obs}_{\mathcal{TC}}(\acute{E}chec)) \neq \emptyset$$

La contraposée de cette assertion avec le lemme 2 induit la propriété suivante :

$$\forall \mathcal{TC} \in \mathcal{TS}, \mathsf{Obs}_{\mathcal{TC}} \cap \mathsf{MinFObs}(\mathcal{S}) \subseteq \mathsf{Obs}_{\mathcal{TC}}(\acute{E}chec) \cdot \Sigma^{*}$$

3.3.5 Complétude

Définition 3.6. Une suite de tests est dite *complète* si elle est, à la fois, correcte, exhaustive et stricte.

D'une manière générale, dans les propriétés précédentes, on pourrait être tenté de remplacer le fait qu'un test *peut échouer* par le fait qu'il échoue effectivement. Cependant, les sorties de l'implémentation ne sont pas contrôlées par le cas de test, ceci conjugué à la propriété de complétude entraîne une contradiction. Une suite doit rejeter toute implémentation non-conforme. Cependant, il est possible qu'une telle implémentation soit non-conforme pour une unique exécution ($\sigma!a$), mais que celle-ci soit masquée par une exécution conforme de même longueur ($\sigma!b$). Tout cas de test pouvant rejeter la première doit également accepter la seconde, puisqu'elle est conforme. Ainsi il n'est pas possible de satisfaire la complétude si on exige l'échec impératif d'un test révélant la non-conformité.

Ainsi, nous parlons de *possibilité d'erreur*. L'exhaustivité signifie donc qu'il existe dans la suite un test qui *peut échouer* sans que celui-ci n'ait l'assurance d'échouer. Même avec une suite de test *exhaustive*, il faut un temps infini et une propriété d'équité pour être certain de n'avoir aucune non-conformité.

Dans ce travail, nous allons donc chercher à engendrer des suites qui vérifient les trois propriétés ci-dessus, c'est-à-dire complètes et strictes. On peut observer que les trois propriétés sont vérifiées si $\mathsf{MinFObs}(\mathcal{S}) \cdot \Sigma_{\mathcal{M}}{}^* = \mathsf{Obs}_{\mathcal{TC}}(\acute{E}chec) \cdot \Sigma_{\mathcal{M}}{}^*$. Pour assurer ces propriétés, il suffit donc de construire un reconnaisseur de $\mathsf{MinFObs}(\mathcal{S}) \cdot \Sigma_{\mathcal{M}}{}^*$, appelé *testeur canonique*.

3.4 Testeur canonique

Par définition des observations fautives minimales, $\mathsf{MinFObs}(\mathcal{S}) \stackrel{\mathbf{def}}{=} (\mathsf{SObs}_{\mathcal{S}} \cdot \Sigma_{\mathcal{S}}^{!\delta}) \setminus \mathsf{SObs}_{\mathcal{S}}$. Une première approche peut donc être d'utiliser les constructions déjà présentées sur les automates.

3.4.1 Opérations sur les automates

En utilisant les opérations ensemblistes élémentaires, on peut écrire que

$$\begin{aligned} \mathsf{MinFObs}(\mathcal{S}) &\stackrel{\mathbf{def}}{=} \mathsf{SObs}_{\mathcal{S}} \cdot \Sigma_{\mathcal{S}}^{!\delta} \setminus \mathsf{SObs}_{\mathcal{S}} \\ &= \mathsf{SObs}_{\mathcal{S}} \cdot \Sigma_{\mathcal{S}}^{!\delta} \cap \overline{\mathsf{SObs}_{\mathcal{S}}} \end{aligned}$$

Il suffit donc, pour construire un reconnaisseur de ces traces, de réaliser les opérations suivantes à partir de la spécification suspendue $Susp(\mathcal{S})$:

- Construire la concaténation $Susp(\mathcal{S})_{\Sigma}$ de $Susp(\mathcal{S})$ avec l'automate qui reconnait $\Sigma_{\mathcal{S}}^{!\delta}$
- Déterminiser $Susp(\mathcal{S})$ pour obtenir $det(Susp(\mathcal{S}))$
- Complémenter $det(Susp(\mathcal{S}))$ pour obtenir $\overline{det(Susp(\mathcal{S}))}$
- Faire le produit entre $\overline{det(Susp(\mathcal{S}))}$ et $Susp(\mathcal{S})_{\Sigma}$

Si ces opérations sont simples à mettre en oeuvre dans le cas d'une spécification donnée par un IOLTS à états finis, la plupart d'entre elles deviennent plus complexes à mettre en oeuvre si l'IOLTS à un nombre infini d'états. De plus, dans le cas d'une spécification infinie, le résultat obtenu n'est pas nécessairement déterministe, il est donc difficile d'utiliser ce reconnaisseur en tant que machine.

3.4.2 Complétion en sortie

La première étape est de construire un reconnaisseur de $\mathsf{SObs}_{\mathcal{S}} \cdot \Sigma_{\mathcal{S}}^{!\delta}$. La spécification suspendue $Susp(\mathcal{S}) = (Q_{\mathcal{S}}, \Sigma_{\mathcal{S}} \cup \{\delta\}, \Lambda_{\mathcal{S}}, Q_{\mathcal{S}}^0, \Delta_{Susp(\mathcal{S})}, col_{\mathcal{S}})$ reconnaît déjà $\mathsf{SObs}_{\mathcal{S}}$. Il suffit de s'en servir pour construire une spécification complétée et suspendue :
$CS(\mathcal{S}) = (Q_{CS(\mathcal{S})}, \Sigma_{CS(\mathcal{S})}, \Lambda_{CS(\mathcal{S})}, Q_{CS(\mathcal{S})}^0, \Delta_{CS(\mathcal{S})}, col_{CS(\mathcal{S})})$ avec :
- $Q_{CS(\mathcal{S})} = Q_{\mathcal{S}} \cup \{q_F\}$ où $q_F \notin Q_{\mathcal{S}}$ est un nouvel état,
- $\Sigma_{CS(\mathcal{S})} = \Sigma_{\mathcal{S}} \cup \{\delta\}$,
- $\Lambda_{CS(\mathcal{S})} = \Lambda_{\mathcal{S}} \cup NS$ où $NS \notin \Lambda_{\mathcal{S}}$ est une nouvelle couleur,
- $Q_{CS(\mathcal{S})}^0 = Q_{\mathcal{S}}^0$,
- $\Delta_{CS(\mathcal{S})} = \Delta_{Susp(\mathcal{M})} \cup \{(p, a, q_F) \mid p \in Q_{\mathcal{S}}, a \in \Sigma_{\mathcal{S}}, a \notin Out_{\mathcal{S}}(p)\}$,
- $col_{CS(\mathcal{S})} = col_{\mathcal{S}} \cup (q_F, NS)$.

La couleur NS représente les comportements non spécifiés. Cependant, seules les observations sont importantes : si pour une observation σ il existe deux exécutions, l'une menant dans NS et l'autre avec un comportement spécifié, alors le test n'échoue pas. Pour éviter ce genre de situation, nous procédons à une déterminisation.

3.4.3 Déterminisation

Si l'IOLTS $CS(\mathcal{S})$ est fini, on peut utiliser la construction par parties. Le testeur canonique est un IOLTS $Can = (Q_{Can}, \Sigma_{Can}, \Lambda_{Can}, Q_{Can}^0, \Delta_{Can}, col_{Can})$ tel que :
- $Q_{Can} = 2^{Q_{CS(\mathcal{S})}}$,
- $\Sigma_{Can} = \Sigma_{\mathcal{S}}^o \cup \{\delta\}$,
- $\Lambda_{Can} = \{\acute{E}chec\}$
- $Q_{Can}^0 = \{Q_{\mathcal{S}}^0\}$
- $(Q_1, a, Q_2) \in \Delta_{Can} \subseteq (2^{Q_{CS(\mathcal{S})}} \times \Sigma_{\mathcal{S}}^{o\delta} \times 2^{Q_{CS(\mathcal{S})}})$ si $Q_1 \overset{a}{\underset{\mathcal{A}}{\rightsquigarrow}} Q_2$,
- $(P, \acute{E}chec) \in col_{Can}$ si $\forall q \in P, (q, NS) \in col_{CS(\mathcal{S})}$.

Par construction, le testeur canonique reconnaît par la couleur $\acute{E}chec$ l'ensemble des observations σ dont toutes les exécutions aboutissent dans l'état q_F de $CS(\mathcal{S})$.

Or, par définition de $CS(\mathcal{S})$, $\sigma \in \mathsf{SObs}_{\mathcal{S}} \cdot \Sigma_{\mathcal{S}}^{!\delta}$. Et, si $\sigma \notin \mathsf{MinFObs}(\mathcal{S})$ et $\sigma \in \mathsf{SObs}_{\mathcal{S}}$, alors $\exists q \in CS(\mathcal{S})$ avec $q \neq q_F$ tel que $\exists q_0 \in Q_{CS(\mathcal{S})}^0, q_0 \overset{\sigma}{\underset{CS(\mathcal{S})}{\rightsquigarrow}} q$. Donc $P \in Q_{Can}$ tel que $\{Q_{\mathcal{S}}^0\} \overset{\sigma}{\underset{Can}{\Longrightarrow}} P$ n'est pas coloré par $\acute{E}chec$, c'est-à-dire que $\mathsf{SObs}_{\mathcal{S}} = \mathsf{Obs}_{Can}(\neg \acute{E}chec)$.

On obtient donc que $\mathsf{Obs}_{Can}(\acute{E}chec) = (\mathsf{SObs}_{\mathcal{S}} \cdot \Sigma_{\mathcal{S}}^{!\delta}) \setminus \mathsf{SObs}_{\mathcal{S}} = \mathsf{MinFObs}(\mathcal{S})$.

Exemple 15.

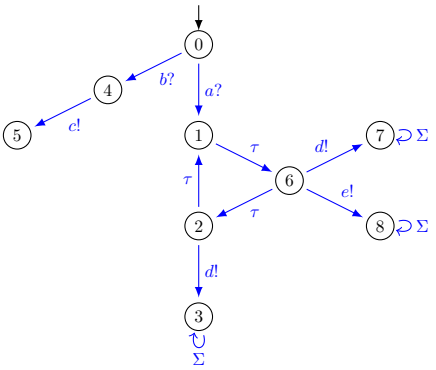


FIGURE 3.2 – Un IOLTS

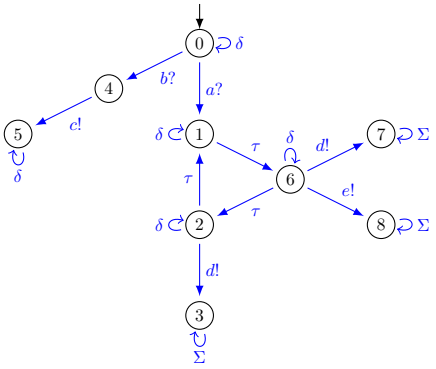


FIGURE 3.3 – Suspension de l'IOLTS de la figure 3.2

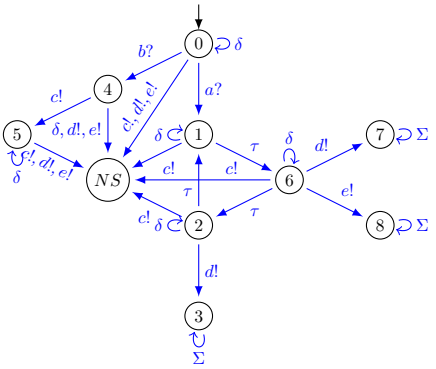


FIGURE 3.4 – Complétion de l'IOLTS de la figure 3.3

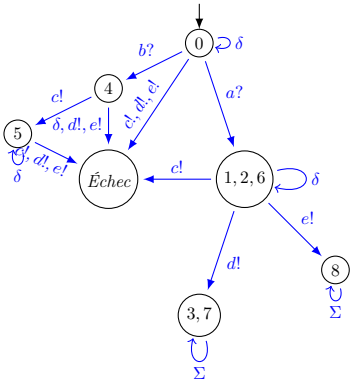


FIGURE 3.5 – Déterminisation de l'IOLTS de la figure 3.4

Le testeur canonique est à lui seul un cas de test qui vérifie toutes les propriétés énoncées précédemment. Cependant, même si le testeur canonique est déterministe, il reste à faire des choix sur les entrées. De plus, il peut être intéressant de ne pas suivre toujours le même chemin afin de tester un maximum de choses. Pour cela on va utiliser des *objectifs de test* qui vont guider le testeur afin de le mener vers un but fixé.

3.5 Sélection par objectif

Dans ce paragraphe, on va utiliser des *objectifs de tests* qui vont restreindre les choix du testeur afin d'explorer une partie ciblée du système et de rendre un verdict dans tous les cas. Formellement, un objectif de test est un IOLTS complet en sortie, sur le même alphabet que la spécification, avec un but, représenté par une couleur particulière *Accept*. Le test prend fin quand une exécution atteint la couleur *Accept*. Les paragraphes qui suivent décrivent les opérations qui vont permettre de composer efficacement l'objectif avec le testeur canonique.

3.5.1 Produit

Si le testeur est destiné à détecter les non conformités, l'objectif de test $\mathcal{TP}$, est lui destiné à choisir les entrées afin d'explorer correctement le système. Formellement, on restreint le comportement du testeur canonique en calculant le produit entre l'objectif de test et le testeur canonique.

Quand la spécification est donnée par un IOLTS fini, le testeur canonique est lui aussi fini. Si l'objectif de test est lui aussi fini, on peut alors utiliser le produit défini dans le chapitre 1 afin de construire un nouvel objet $S_{\times}TP$ qui reconnaîtra certaines observations minimales fautives tout en étant guidé par l'objectif de test.

En ce qui concerne l'attribution des couleurs *Échec* et *Accept*, elles sont portées par les couples dont au moins l'un des états porte la couleur. Nous reviendrons par la suite sur les verdicts.

3.5.2 Analyse de co-accessibilité

Le produit entre le testeur canonique et l'objectif de test peut avoir supprimé certaines exécutions, aussi bien dans l'objectif que dans le testeur. De plus, comme tous les événements ne sont pas contrôlés, il est possible que le système aille dans un état dans lequel la couleur *Accept* n'est plus accessible.

Afin d'éviter qu'un test se prolonge en de telles circonstances, on colore tous les états co-accessibles depuis ceux colorés par *Accept* par la couleur *Aucun*, ce qui permet par la suite d'éliminer les autres.

3.5.3 Verdicts

On donne à tous les états une couleur finale en respectant l'ordre de priorité décroissant suivant :

- *Échec*
- *Succès*, si l'état est coloré par *Accept* mais pas par *Échec*
- *Aucun*
- *Inconc*

Un état ne garde que sa couleur la plus prioritaire. Si il ne porte aucune couleur, on lui attribue la couleur *Inconc*. Les verdicts retournés sont les couleurs portées par l'état atteint (qui est unique puisque le système est déterministe). Le verdict *Échec* signifie que le test à échoué. Le verdict *Succès* signifie que le test à réussi. Le verdict *Aucun* signifie que le test n'est pas encore fini. Le verdict *Inconc* signife que *Accept* n'est plus accessible ; en règle générale, on interrompt l'exécution d'un tel test.

L'IOLTS obtenu est noté $\mathcal{TC}$ est est appellé *cas de test*.

Elagage Dans le but d'éviter que le système explore des états colorés par *Inconc*, il est possible d'élaguer les chemins qui s'y poursuivent. Pour cela, il suffit de supprimer les transitions $p \xrightarrow[\mathcal{TP}]{a} q$ pour lesquelles $(p, Inconc) \in col_{\mathcal{TP}}$.

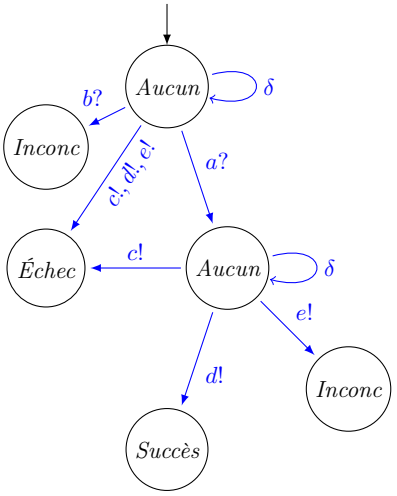


FIGURE 3.6 – Cas de test.

Exemple 16. *La figure 3.6 représente le cas de test obtenu après sélection, à partir de l'IOLTS de la figure 3.5. L'objectif de test est un IOLTS à trois état qui accepte le mot a?d!. Les chemins qui faisaient suite à un état coloré par Inconc ou Succès ont été supprimés.*

3.5.4 Propriétés des cas de test

La construction d'un cas de test fait que les propriétés de correction et de sévérité sont héritées du testeur canonique. Pour ce qui est de l'exhaustivité, chaque cas de test engendré par cette méthode est une restriction du testeur canonique. Il serait donc possible de perdre l'exhaustivité. Cependant, pour toute observation fautive minimale σ, il est possible de construire un objectif de test $\mathcal{TP}_\sigma$ qui reconnaît exactement σ. La construction de $\mathcal{TP}_\sigma$ assure que cette observation fautive sera testée.

Même si il n'est pas possible d'obtenir l'exhaustivité avec un nombre fini de cas de test, cette construction étant possible pour toutes les observations fautives, on peut dire que l'exhaustivité est atteinte à la limite.

Précision La sévérité assure que le cas de test renverra le verdict *Échec* dès que possible, il existe une propriété similaire pour le verdict *Succès*.

Un cas de test $\mathcal{TC}$ est *précis*, si lorsque qu'un état coloré par *Accept* mais pas par *Échec* est atteint, alors le verdict *Succès* est renvoyé immédiatement. Plus formellement $\mathcal{TC}$ est précis par rapport à $\mathcal{TP}$ si :

$$\mathsf{Obs}_{\mathcal{TC}}(\mathit{Succès}) = \mathsf{Obs}_{\mathcal{TP}}(\mathit{Accept}) \cap \mathsf{SObs}_{\mathcal{S}} \cap \mathsf{SObs}_{\mathcal{TC}}$$

Par construction, les états colorés par *Succès* sont ceux colorés par *Accept* dans $\mathcal{TP}$ et non colorés par *Échec* dans *Can*. En conséquence, $\mathsf{Obs}_{\mathcal{TC}}(\mathit{Succès}) = \mathsf{Obs}_{\mathcal{TP}}(\mathit{Accept}) \cap \mathsf{SObs}_{\mathcal{S}}$

et comme $\mathsf{Obs}_{\mathcal{TC}}(\mathit{Succès}) \subseteq \mathsf{Obs}_{\mathcal{TC}}$, on obtient la condition sur la précision.

Chapitre 4

Modèles de la récursivité

Ce chapitre est consacré aux modèles de la récursivité. La façon la plus simple pour étendre le modèle des automates finis, en conservant des propriétés décidables consiste à lui adjoindre une pile. Il est d'ailleurs instructif d'observer que si on adjoint une *file* à un automate fini, ceci définit un modèle ayant l'expressivité des machines de Turing.

Les modèles reposant sur une pile permettent, de façon naturelle, de représenter les comportements des programmes récursifs. Plus généralement, ces modèles ont été considérablement mis à profit en informatique, que ce soit pour définir la syntaxe de langages de programmation, ou encore pour définir des langages d'arbres, tel que XML.

Dans ce chapitre, nous examinerons plusieurs modèles de la récursivité. Dans un premier temps, nous examinerons ceux-ci sous l'angle des langages pour quelques systèmes de récriture de mots. Puis, nous considérerons des objets plus riches qui engendrent les mêmes langages, mais dont les structures offrent des possibilités plus importantes.

4.1 Hiérarchie de Chomsky

Les langages algébriques constituent le modèle fondamental de la récursivité.

4.1.1 Grammaires

Les grammaires de mots ont été originellement introduites dans le but de définir la structure des phrases en langue naturelle. Il existe plusieurs groupes de langages, chacun défini par un reconnaisseur particulier. Leur intérêt en informatique est que ces reconnaisseurs sont des systèmes de récriture de mot avec des propriétés spécifiques. En général, les transformations de modèles garantissent, à minima, la préservation du langage reconnu.

Définition 4.1. Une *grammaire de mot* est un quadruplet $Gr = (\Sigma, \Gamma, Reg, Z)$ où :
- Σ est un alphabet de *symboles terminaux*,
- Γ est un alphabet de *symboles non terminaux*, avec $\Sigma \cap \Gamma = \emptyset$,

– Reg est l'ensemble des *règles*,
– $Z \in \Gamma$ est l'*axiome*

On note $\Sigma \cup \Gamma = \mathcal{X}$. Une règle est une expression de la forme $\alpha \to \beta$ où α est un mot de $\mathcal{X}^+$ et β est un mot de $\mathcal{X}^*$.

Pour une grammaire, on note $\underset{Gr}{\Rightarrow}$ la relation de *récriture* définie par :

$$\forall w_1, w_2 \in \mathcal{X}^*, w_1 \alpha w_2 \underset{Gr}{\Rightarrow} w_1 \beta w_2 \text{ si } \alpha \to \beta \in Reg$$

On note $\underset{Gr}{\Rightarrow}^*$ la clôture transitive de cette relation. Le *langage engendré* par une grammaire, noté $L(Gr)$, est l'ensemble des mots w de Σ^* tels que $Z \underset{Gr}{\Rightarrow}^* w$.

Hiérarchie de Chomsky On distingue quatre types généraux de grammaires, qui sont définis par restrictions sur les règles. Il est possible d'associer à chaque restriction un reconnaisseur, c'est à dire un automate (potentiellement infini) qui reconnaît ce langage. Pour plus de précision sur cette hiérarchie, il est possible de se référer à [41] ou [7].

Les restrictions

Grammaires de type 0 **:** aucune contrainte n'est donnée sur les règles. Les langages engendrés sont appelés *langages récursivement énumérables.*

Grammaires de type 1 **:** On appelle généralement *grammaires contextuelles* les grammaires de type 1. Les règles sont de la forme $\alpha_1 A \alpha_2 \to \alpha_1 \beta \alpha_2$, avec $\alpha_1, \alpha_2 \in \mathcal{X}^*$, $\beta \in \mathcal{X}^+$ et $A \in \Gamma$. Les langages engendrés sont appelés *langages contextuels.*

Grammaires de type 2 **:** aussi appelées *grammaires algébriques* ou parfois *grammaires hors-contexte.* Les règles sont de la forme $A \to \beta$, avec $A \in \Gamma$ et $\beta \in \mathcal{X}^*$. Les langages sont appelés *langages algébriques.*

Grammaires de type 3 **:** ou *grammaires régulières.* Les règles sont de la forme $A \to aB$ (grammaire linéaire droite) ou $A \to Ba$ (grammaire linéaire gauche). Les langages engendrés sont appelés *langages réguliers.* D'après le théorème de Kleene, les langages réguliers sont les langages rationnels.

Les reconnaisseurs

Les langages engendrés par les différents types de grammaires sont les suivants :
– type 0 : les machines de Turing.
– type 1 : les machines de Turing à mémoire linéairement bornée ([50]).
– type 2 : les automates à pile, que nous présenterons par la suite.
– type 3 : les automates finis, par le théorème de Kleene.

Dans les trois premiers chapitres de ce document, nous avons présenté des constructions et des résultats sur des modèles finis, qui appartiennent donc à la classe de type 3. Dans la suite de ce document, nous nous intéressons à des modèles qui reconnaissent des langages algébriques. L'intérêt principal de ces modèles est qu'ils sont particulièrement adaptés aux systèmes récursifs. On peut aussi noter qu'ils sont utilisés en compilation, en particulier pour leur aptitude à reconnaître les expressions bien parenthèsées, ou d'arbres, structures fondamentales des langages de programmation.

4.1.2 Langages algébriques

4.1.3 Automates à pile

Un automate à pile est un automate muni d'une structure de pile qui sert à stocker des informations au cours de son exécution.

Définition 4.2. Un *automate à pile* $\mathcal{A}$ est un sextuplet $(Q_\mathcal{A}, \Sigma_\mathcal{A}, \Gamma_\mathcal{A}, Q^0_\mathcal{A}, \bot_\mathcal{A}, \Delta_\mathcal{A}, F_\mathcal{A})$ où $Q_\mathcal{A}, \Sigma_\mathcal{A}, Q^0_\mathcal{A}$ et $F_\mathcal{A}$ représentent respectivement l'ensemble des états, l'alphabet d'événements, l'ensemble des états initiaux et l'ensemble des états terminaux, et

- $\Gamma_\mathcal{A}$ est l'*alphabet de pile*, avec $\Gamma_\mathcal{A} \cap \Sigma_\mathcal{A} = \emptyset$,
- $\bot_\mathcal{A} \in \Gamma_\mathcal{A}$ est le *symbole de fond de pile*,
- $\Delta_\mathcal{A} \subseteq Q_\mathcal{A} \times \Gamma_\mathcal{A} \times \Sigma_\mathcal{A} \times Q_\mathcal{A} \times \Gamma^*_\mathcal{A}$ est la *relation de transition*.

La sémantique ou *graphe des transitions* d'un automate à pile est l'automate $[\![\mathcal{A}]\!] = (Q_{[\![\mathcal{A}]\!]}, \Sigma_{[\![\mathcal{A}]\!]}, Q^0_{[\![\mathcal{A}]\!]}, \Delta_{[\![\mathcal{A}]\!]}, F_{[\![\mathcal{A}]\!]})$ où :

- $Q_{[\![\mathcal{A}]\!]} = Q_\mathcal{A} \times \Gamma^*_\mathcal{A}$,
- $\Sigma_{[\![\mathcal{A}]\!]} = \Sigma_\mathcal{A}$,
- $Q^0_{[\![\mathcal{A}]\!]} = Q^0_\mathcal{A} \times \{\bot_\mathcal{A}\}$,
- $\Delta_{[\![\mathcal{A}]\!]} = \{((p, AS), a, (q, \beta S)) \mid (p, a, A, \beta, q) \in \Delta_\mathcal{A} \wedge S \in \Gamma^*_\mathcal{A})\}$,
- $F_{[\![\mathcal{A}]\!]} = F_\mathcal{A} \times \Gamma^*_\mathcal{A}$.

Remarque 4.1. De façon classique, les transitions d'un automate à pile sont de trois sortes : internes, d'empilement (push) et de dépilement (pop).

Il convient de faire la distinction entre les éléments de $Q_\mathcal{A}$ que l'on appellera *états de contrôle* et les états dans la sémantique qui sont les couple formés de l'état de contrôle et d'un mot sur $\Gamma_\mathcal{A}$ appelé mot de *pile*. Souvent, on appelle *configuration* (de l'automate $\mathcal{A}$) les états du graphe de transition. Le premier symbole de la pile est appelé le *symbole de sommet de pile*.

L'automate à pile sert de description finie pour l'automate infini qu'est sa sémantique. Un mot est *reconnu* par un automate à pile, si il est reconnu par sa sémantique. En conséquence, un automate à pile est dit *déterministe* (resp. *complet*) si sa sémantique est déterministe (resp. complète).

4.1.4 Résultats utiles sur les automates à pile

Nous présentons ici une série de résultats sur les automates à pile. Il s'agit essentiellement de reprendre les résultats du chapitre 1 sur les automates et de signaler ceux qui peuvent s'adapter facilement et ceux qui ne sont plus valides. Nous ne donnerons pas de construction, en particulier parce que le modèle qui nous intéressera par la suite n'est pas celui des automates à pile ; nous présenterons les opérations sur ce modèle directement.

Accessibilité

Proposition 4.1. *[14] Pour un automate à pile $\mathcal{A}$, l'ensemble des configurations accessibles depuis les états initiaux $Q^0_{\mathcal{A}}$ peut être reconnu par un automate fini sur $\Gamma_{\mathcal{A}} \cup Q_{\mathcal{A}}$. Cet automate peut être construit de façon effective en temps polynomial.*

Ce résultat peut être utilisé pour vérifier qu'une configuration est accessible, que toutes les configurations d'un ensemble sont accessibles ou qu'au moins l'une d'elles est accessible.

Déterminisation A la différence des automates finis, il existe des langages algébriques qui ne sont reconnus par aucun automate à pile déterministe. C'est le cas par exemple du langage $w\tilde{w}$ avec $w \in \Sigma^*$ qui est l'ensemble des palindromes de longueur paire sur Σ.

4.1.5 Propriétés des langages algébriques

Propriétés positives L'ensemble des langages algébriques est clos par union, par concaténation et par itération de Kleene.

L'intersection entre un langage algébrique avec un langage régulier est algébrique.

Propriétés négatives L'ensemble des langages algébriques n'est pas clos par intersection : par exemple, le langage $\{a^n b^n c^n \mid n \geq 0\} = \{a^n b^n c^* \mid n \geq 0\} \cap \{a^* b^n c^n \mid n \geq 0\}$ n'est pas algébrique.

L'ensemble des langages algébriques n'est pas clos par complément. Par exemple, l'ensemble des mots de la forme ww avec $w \in \Sigma^*$ n'est pas algébrique, mais son complémentaire l'est.

Problème indécidables

Problème. La vacuité de l'intersection pour une classe de langages C est le suivant :

Données : Deux langages L_1 et L_2 dans C

Question : Est-ce que $L_1 \cap L_2$ est vide ?

Proposition 4.2. *Le problème du vide de l'intersection de deux langages algébriques est indécidable.*

Problème. Le problème de l'inclusion pour deux classes de langages C_1, C_2 est le suivant :

Données : Deux langages $L_1 \in C_1$ et $L_2 \in C_2$

Question : Est-ce que L_1 est inclus dans L_2 ?

Proposition 4.3. *Le problème de l'inclusion pour la classe des langages réguliers et celle des langages algébriques est indécidable.*

4.2 Autres modèles de la récursivité

Les grammaires de type 2 et les automates à pile ne sont pas les seuls modèles de la récursivité. Nous allons voir d'autres modèles plus généraux sur le plan structurel.

4.2.1 Machines récursives à états (RSM)

Les machines récursives à états ont été introduites par Alur *et al.* [4] pour modéliser les systèmes récursifs.

Définition 4.3. Une *machine récursive à états* (RSM) $\mathcal{M}$, sur un alphabet Σ, est donnée par un ensemble de *composants* $\{\mathcal{M}_1, \mathcal{M}_2, ..., \mathcal{M}_k\}$ où chaque composant est un quintuplet $\mathcal{M}_i = (N_i \cup B_i, Y_i, En_i, Ex_i, \Delta_i)$ tel que :

- N_i est un ensemble de *nœuds*, et B_i disjoint de N_i est un ensemble de *boites*,
- Y_i est un étiquetage qui associe à chaque boite B_i l'indice d'un des composants,
- $En_i \subseteq N_i$ est l'ensemble des *nœuds d'entrée*, et $Ex_i \subseteq N_i$ est l'ensemble des *nœuds de sortie*,
- Δ_i est la *relation de transition* et contient les triplets (p, a, q) où :
  1. $a \in \Sigma$,
  2. soit p est un nœud qui n'est pas une sortie ($p \in N_i \setminus Ex_i$), soit p est un couple (b, x) avec $b \in B_i$ et $x \in Ex_j$ pour $j = Y_i(b)$,
  3. soit q est un nœud qui n'est pas une entrée ($p \in N_i \setminus En_i$), soit q est un couple (b, x) avec $b \in B_i$ et $x \in En_j$ pour $j = Y_i(b)$.

On utilise le terme de *port* pour faire référence aux couples qui correspondent aux nœuds d'entrée et nœuds de sortie d'un composant.

La sémantique d'une machine récursive $\mathcal{M}$ à états est donnée par le graphe étiqueté $(Q_\mathcal{M}, \Sigma_\mathcal{M}, \Delta_\mathcal{M})$. On définit $Q \subseteq B^*N$ avec $B = \bigcup_i B_i$ et $N = \bigcup_i N_i$ par l'ensemble des $(b_1, b_2, ..., b_t, q)$ où pour tout $1 \leq i < t$, $Y_{j_i}(b_i) = j_i + 1$ et $Y_{j_t}(b_t) = j$ avec $q \in N_j$. En ce qui concerne la définition des transitions, on considère que pour $p = (b_1, b_2, ..., b_t, q)$ avec $q \in N_j$ et $b_t \in B_i$, une transition (p, a, p') appartient à $\Delta_\mathcal{M}$ si une des conditions suivantes est vérifiée

- $(q, a, q') \in \Delta_j$ pour q' un nœud de $\mathcal{M}_j$ et $p' = (b_1, b_2..., b_t, q')$,
- $(q, a, (b', q')) \in \Delta_j$ pour une boite b' de $\mathcal{M}_j$ et $p' = (b_1, b_2..., b_t, b', q')$,
- $q \in Ex_j$ et $((b_t, q), a, q') \in \Delta_i$ pour un nœud q' de A_i et $p' = (b_1, b_2..., b_{t-1}, q')$,
- $q \in Ex_j$ et $((b_t, q), a, (b', q')) \in \Delta_i$ pour une boite b' de A_i et $p' = (b_1, b_2..., b_{t-1}, b', q')$.

Proposition 4.4. *[4] Pour toute RSM $\mathcal{M}$, il existe un automate à pile $\mathcal{A}$ tel que la sémantique de $\mathcal{M}$ est isomorphe à celle de $\mathcal{A}$.*

Ce résultat implique que si on fixe dans la sémantique un nombre fini d'état initiaux et d'états terminaux, les langages reconnus sont des langages algébriques.

Nous présentons ici les RSM par soucis d'exhaustivité, et parce que ce modèle est très proche de celui que nous introduisons. Cependant, notre inspiration vient en grande partie des grammaires déterministes de graphes, que nous allons maintenant présenter.

4.2.2 Grammaires déterministes de graphes

Les grammaires de graphes sont des outils conçus suivant l'inspiration des grammaires de mots pour construire des familles de graphes. Dans [29], Courcelle utilise des grammaires *déterministes* de graphes pour définir un unique graphe (potentiellement infini) à partir d'un ensemble fini de productions. Caucal a également fortement contribué à l'étude de ce modèle, par exemple en démontrant que lorsqu'ils sont de degré fini, les graphes engendrés par ces grammaires sont exactement les graphes des transitions des automates à pile. Il existe plusieurs travaux présentant celles-ci, mais l'un des plus exhaustif est l'étude [22]. Avant de présenter les grammaires déterministes de graphes, il nous faut étendre la définition de graphe en permettant des hyperarcs.

Étant donné un ensemble de *sommets* V, et un ensemble fini, gradué F d'étiquettes, avec $\rho : F \to \mathbb{N}$, la fonction d'*arité* de F, on pose $F_n \stackrel{\text{def}}{=} \{f \in F \mid \rho(f) = n\}$ pour tout $n \in \mathbb{N}$. Un *hyperarc* d'arité k est un élément de $F_k V^k$. Un *hypergraphe* H est un ensemble d'hyperarcs tel que $V_H = \{v \in V \mid FV^*vV^* \cap H \neq \emptyset\}$ est fini ou dénombrable.

Définition

Une *grammaire déterministe de graphes* à récriture d'hyperarcs (HR-grammaires) R est un ensemble fini de règles de la forme $fv_1v_2...v_{\rho(f)} \to H$ dans lesquelles $fv_1v_2...v_{\rho(f)}$ est un hyperarc joignant des sommets deux à deux distincts et H est un hypergraphe fini tel que $\{v_1, ..., v_{\rho(f)}\} \subseteq V_H$. On note :

- $\Gamma_R = \{f \in F \mid \exists f x_1...x_{\rho(f)} \in Dom(R)\}$ les symboles *non-terminaux*,
- $T_R = \{f \in F \setminus \Gamma_R \mid \exists P \in Im(R), \exists f x_1...x_{\rho(f)} \in P\}$ les symboles *terminaux*.

De plus on impose que T_R ne contienne que des symboles d'arité 1 (les couleurs, qu'on note Λ_R) ou 2 (les étiquettes, qu'on note Σ_R). On impose aussi qu'il existe un unique non-terminal $Z \in \Gamma_R$ d'arité 0, l'*axiome*. Chaque symbole non-terminal n'apparaît qu'une seul fois dans le membre gauche d'une règle, d'où le qualificatif de grammaire déterministe.

Étant donnée une grammaire R, la récriture $\underset{R}{\to}$ est la relation définie sur les hypergraphes de la façon suivante : $H_1 \underset{R}{\to} H_2$ si il existe un hyperarc non-terminal $X = Ax_1...x_{\rho(A)}$ dans H_1 et une règle $Av_1...v_{\rho(A)} \to H$ dans R telle que X peut être remplacé par H dans H_1 :

$$H_2 = (H_1 \setminus X) \cup h(H)$$

avec h une fonction qui associe v_i à chaque x_i, et où les autres sommets de H sont envoyés de façon injective sur des sommets que ne sont pas dans V_{H_1}.

On note $\underset{R,X}{\rightarrow}$ la récriture d'un hyperarc X, et on étend cette notation aux ensembles d'hyperarcs E par $\underset{R,E}{\rightarrow}$. On appelle *récriture parallèle complète*, notée $H_1 \underset{R}{\Rightarrow} H_2$ la récriture de l'ensemble de tous les non-terminaux de H_1.

On note $[H] = H \cap T_R V_H^*$ la restriction de H aux arcs terminaux.

Un *graphe engendré* par une grammaire R est un graphe défini par

$$\bigcup_{n\in\mathbb{N}} [H_n] \text{ avec } Z \underset{R}{\rightarrow} H_0 \underset{R}{\Rightarrow} ... \underset{R}{\Rightarrow} H_n \underset{R}{\Rightarrow} H_{n+1}$$

Les graphes engendrés par R sont isomorphes, on note R^ω la famille des graphes engendrés par R. L'ensemble des graphes engendrés par des grammaires déterministes de graphes à récriture d'hyperarc est l'ensemble des *graphes réguliers*. Étant donné un sommet v, on appelle *niveau* de v, noté $\ell(v)$ l'entier tel que $v \in H_{\ell(v)}$ et $v \notin H_{\ell(v)-1}$.

On peut observer que les graphes réguliers sont des graphes étiquetés et colorés. La sémantique d'une grammaire est donnée par le graphe régulier, engendré par celle-ci.

Quelques propriétés des graphes réguliers

Proposition 4.5. *Dans un graphe régulier, l'ensemble des traces depuis une couleur vers une autre couleur, forme un langage algébrique.*

Cette proposition établit que l'expressivité des graphes réguliers est la même que celle des automates à pile. Cependant, les graphes réguliers ne sont pas isomorphes aux graphes de transition des automates à pile. En effet, dans un graphe régulier, certains sommets peuvent avoir un degré sortant, ou entrant, infini, ce qui n'est pas possible dans les graphes de transition des automates à pile.

Par la suite, afin d'utiliser les graphes réguliers comme des DES infinis, nous introduirons un ensemble d'état initiaux. Ces états initiaux serons désignés par une couleur particulière `init`.

La proposition 4.5 nous assure que l'ensemble des résultats qui existe sur les langages algébriques peut être transférés aux graphes réguliers. C'est le cas en particulier de l'accessibilité, de l'union, de la concaténation et du produit avec un automate fini.

4.2.3 Automates à pile d'ordre supérieur

Nous présentons à présent un modèle plus général de la récursivité : les *automates à pile d'ordre supérieur*. Ces derniers ont été définis à la fin des années 69, [**?**, **?**, 52], et ont connu récemment des développements importants, [47, 19, 18]. Notre objectif ici est de réaliser une

simple introduction informelle, par souci de complétude, car nous ne considérerons pas cette famille dans nos travaux.

Cette famille est une hiérarchie. A l'ordre 1, se trouvent les automates à pile ordinaires. Un automate d'ordre 2 dispose d'une pile d'ordre 2, dans laquelle il peut empiler des piles d'ordre 1. Il peut également agir sur la pile d'ordre 1 qui est au sommet de la pile d'ordre 2, en utilisant les opérations habituelles. Pour ce qui est de la pile d'ordre 2, il dispose de deux opérations : $push_2$, qui crée une copie de la pile d'ordre 1 qui est au sommet, et pop_2 qui la détruit (il existe de nombreuses variantes ayant des expressivités similaires). Ce qui est fait à l'ordre 2 se généralise à l'ordre n, pour tout n supérieur à 1.

Il a été montré, [31], que la hiérarchie de famille s de langages ainsi engendrée est stricte. Chacun des niveaux ne partage pas les propriétés des langages algébriques (par exemple, il n'est pas établi qu'il soit possible de supprimer les transitions étiquetées par τ d'un automate d'ordre k, en restant à l'ordre k). En revanche la théorie monadique du second ordre est décidable pour tous les graphes des transitions de ces automates.

4.3 Modèles restreints de la récursivité

Le but de ce travail est d'utiliser des modèles de la récursivité afin de résoudre des problèmes de diagnostic, d'opacité ou de produire des cas de tests. Pour apporter des solutions à ces questions, certaines opérations sont nécessaires, en particulier la déterminisation ou le produit. Comme toutes ces transformations ne sont pas effectives pour les automates à pile (ou leurs extensions), nous présentons ici certaines restrictions à ces systèmes, pour lesquelles ces opérations sont effectives.

4.3.1 Automates à pile visible

Les automates à pile visibles (VPA) ont été introduits par Alur et Madusudan [6]. Le but de cette restriction est de synchroniser les opérations sur la pile avec l'alphabet d'événements. Ainsi pour un mot d'entrée donné, la pile est de hauteur fixée.

Définition 4.4. Un *automate à pile visible* $\mathcal{A}$ est un automate à pile $(Q_\mathcal{A}, \Sigma_\mathcal{A}, \Gamma_\mathcal{A}, Q^0_\mathcal{A}, \perp_\mathcal{A}, \Delta_\mathcal{A}, F_\mathcal{A})$ tel que $\Sigma_\mathcal{A}$ est une partition $\Sigma_\mathcal{A} = \Sigma^{push}_\mathcal{A} \cup \Sigma^{int}_\mathcal{A} \cup \Sigma^{pop}_\mathcal{A}$. Les transitions sont de trois sortes :
- interne (n'affecte pas la pile), et étiquetées par $\Sigma^{int}_\mathcal{A}$,
- empile (ajoute un symbole sur la pile), étiquetées par $\Sigma^{push}_\mathcal{A}$,
- dépile (retire un symbole de la pile), étiquetées par $\Sigma^{pop}_\mathcal{A}$.

Les automates à pile visibles nous autorisent à utiliser les opérations de produit et de déterminisation.

Proposition 4.6. *([6]) Soient $\mathcal{A}$ et $\mathcal{B}$ deux VPA tels que $\Sigma^{push}_\mathcal{A} = \Sigma^{push}_\mathcal{B}$, $\Sigma^{pop}_\mathcal{A} = \Sigma^{pop}_\mathcal{B}$ et $\Sigma^{int}_\mathcal{A} = \Sigma^{int}_\mathcal{B}$. Alors l'intersection entre les langages de ces deux automates $L(\mathcal{A}) \cap L(\mathcal{B})$ peut-être reconnue par un automate à pile visible. Cet automate peut être construit de façon effective.*

Proposition 4.7. *([6]) Soit $\mathcal{A}$ un VPA. Le langage $L(\mathcal{A})$ reconnu par $\mathcal{A}$ peut être reconnu par un automate à pile visible déterministe. Cet automate peut être construit de manière effective.*

Théorème 3. *([6]) Les problèmes suivants sont* EXPTIME*-complets pour les VPA :*
- *le problème de l'universalité.*
- *le problème de l'inclusion.*

4.3.2 Grammaires synchronisées et pondérées

Les bonnes propriétés des VPA proviennent de la forte contrainte de la pile vis-à-vis de la trace d'exécution. Cependant, il est aisé d'imaginer certains cas de langages algébriques qui ne peuvent être reconnus par un automate à pile visible mais dont la pile est fortement contrainte. Par exemple, le langage composé des mots de la forme $a^n b a^n$, pour tout $n \in \mathbb{N}$. Nous étudions maintenant une restriction des HR-grammaires qui permet de prendre en compte cette contrainte. Dans un premier temps, nous présentons la notion de synchronisation de grammaires introduite par Caucal et Hassen dans [23].

Grammaires pondérées

Définition 4.5. Une grammaire R est dite *pondérée* si pour tout mot w de Σ_R^* :

$$\left| \left\{ \ell(q) \mid \exists q_0 \in V_{R^\omega}, (\texttt{init}, q_0) \in R^\omega \wedge (q_0 \underset{R^\omega}{\overset{w}{\Longrightarrow}} q) \right\} \right| \leq 1$$

La notion de grammaire pondérée signifie que le niveau de l'extrémité finale d'une exécution ne dépend que de la trace de celle-ci.

Proposition 4.8. *[23] Pour toute grammaire pondérée R, il existe une grammaire R' telle que $L(R) = L(R')$, et telle que R'^ω est un graphe déterministe. La construction de R' est effective.*

Synchronisation de grammaires pondérées

Étant données deux grammaires pondérées, on peut définir une notion d'inclusion en terme de contraintes imposées sur le niveau. Cela correspond à la notion de synchronisation.

Définition 4.6. Une grammaire pondérée R_1 *synchronize* une grammaire pondérée R_2 si pour tout mot w de Σ_R^*, il existe q_0 et q dans $V_{R_1^\omega}$ tels que :

$$((\texttt{init}, q_0) \in R_1^\omega \wedge q_0 \underset{R_1^\omega}{\overset{w}{\Longrightarrow}} q) \implies (\exists q_0', q' \in V_{R_2^\omega}, (\texttt{init}, q_0') \in R_2^\omega \wedge q_0' \underset{R_2^\omega}{\overset{w}{\Longrightarrow}} q' \wedge \ell(q) = \ell(q'))$$

On note $R_1 \rhd R_2$.

On note $Sync(R) = \{L(R) \cap L(R') \mid R \rhd R'\}$ l'ensemble des langages synchronisés par la grammaire R.

Proposition 4.9. *Pour toute grammaire pondérée R, $Sync(R)$ est clos par intersection. Pour R_1, R_2 tels que $L(R_1), L(R_2) \in Sync(R)$, la grammaire $R_{1\times 2}$ qui engendre le produit des sémantiques de R_1 et de R_2 peut être construite.*

Pour le problème de la diagnosticabilité, nous utiliserons la proposition 4.9. En effet, par définition $R \rhd R$, donc $L(R) \in Sync(R)$ ce qui permet de construire l'auto-produit.

Dans ce chapitre, nous avons exploré diverses constructions, toutes destinées à modéliser des systèmes récursifs. Nous nous sommes plus particulièrement intéressés aux HR-grammaires qui généralisent un petit peu les automates à pile. Les restrictions étudiées, et en particulier les grammaires pondérées, nous serons très utiles par la suite afin de résoudre les problèmes de la diagnosticabilité, de l'opacité, et afin de générer des cas de test pour des systèmes récursifs.

Chapitre 5

Systèmes récursifs de tuiles

Dans ce chapitre, nous présentons notre propre modèle : les systèmes récursifs de tuiles. Ils servent à modéliser les systèmes récursifs en restant proches des langages de programmation. Tout comme les grammaires déterministes de graphes, il s'agit de générateurs fini des graphes réguliers. Les systèmes récursifs de tuiles, sont un ensemble de tuiles dont chacune est un graphe fini, et qui peuvent s'assembler entre elles afin de former des graphes de plus grande taille.

Les systèmes récursifs de tuiles (RTS) sont fortement inspirés des grammaires déterministes de graphes à récriture d'hyperarcs. On ne considère que des terminaux d'arité 2, et les graphes engendrés peuvent contenir des états de degré infini. Il s'agit essentiellement d'une reformulation syntaxique des grammaires de graphes. La motivation de cette reformulation provient du fait que les grammaires de graphes sont utilisées dans de très nombreux contextes, le plus souvent pour définir des ensembles de graphes finis. Ainsi, pour lever toute ambiguïté, il a été choisi, dans ce manuscrit, d'utiliser une formulation légèrement différente qui porte l'accent sur les tuiles, qui sont de simples graphes finis, et les supports et frontières qui permettent de les composer. Cependant, les résultats connus sur les graphes réguliers peuvent être simplement adaptés sur les systèmes récursifs de tuiles.

5.1 Définition des RTS

Définition 5.1. Un *système récursif de tuiles* (RTS) est un quadruplet $\mathcal{M} = ((\Sigma_\mathcal{M}, \Lambda_\mathcal{M}), \mathcal{T}_\mathcal{M}, t^0_\mathcal{M}, Q^0_\mathcal{M})$ où :

- $\Sigma_\mathcal{M} = \Sigma^o_\mathcal{M} \cup \{\tau\}$ est un alphabet fini d'événements qui forme une partition entre les événements observables $\Sigma^o_\mathcal{M}$ et un événement non observable τ,
- $\Lambda_\mathcal{M}$ est un ensemble fini de *couleurs*,
- $\mathcal{T}_\mathcal{M}$ est l'ensemble des *tuiles* dans lequel chaque tuile est donnée par un sixtuplet $t = ((\Sigma_\mathcal{M}, \Lambda_\mathcal{M}), Q_t, \Delta_t, col_t, S_t, F_t)$ avec :
  - $Q_t \subseteq \mathbb{N}$ est un ensemble fini de *sommets*,
  - $\Delta_t \subseteq Q_t \times \Sigma_\mathcal{M} \times Q_t$ est un ensemble fini d'*arcs*,

- $col_t \subseteq Q_t \times \Lambda_t$ est une relation finie de *coloriage*,
- $S_t \subseteq Q_t$ est le *support* de t, (il servira pour fusionner t avec d'autres tuiles de $\mathcal{T}_\mathcal{M}$),
- $F_t \subseteq \mathcal{T}_\mathcal{M} \times 2^{\mathbb{N}\times Q_t}$, est la *frontière* de t. Un couple (t', f') appartient à F_t signifie que la tuile t' peut être fusionnée avec la tuile t en identifiant les sommets du support de t' avec des sommets de t (voir définition 5.3). De plus on impose à f' d'être une application $f' : S_{t'} \to Q_t$,

- $t^0_\mathcal{M} \in \mathcal{T}_\mathcal{M}$ est la tuile initiale, l'*axiome*,
- $Q^0_\mathcal{M} \subseteq Q_{t^0_\mathcal{M}}$ est l'ensemble des sommets initiaux.

On utilise ici les dénominations *sommets* et *arcs* afin de souligner la différence entre les éléments d'une tuile et les éléments d'un DES. De plus, nous noterons $|Q_t|$ le nombre de sommets d'une tuile et $max(Q_t)$ l'entier maximum de Q_t. On note aussi $\rho(t) = |S_t|$ l'*arité* de la tuile t et $deg(t) = |F_t|$ le nombre de tuiles qui appartiennent à la frontière de t. Par ailleurs, on dira, par abus de langage, qu'une tuile t' est dans la frontière d'une tuile t s'il existe f' telle que (t', f') est dans la frontière de t ; on dira également qu'un sommet est dans la frontière d'une tuile t s'il appartient à $Im(Im(F_t))$. On note $Front(s)$ l'ensemble $\{(t, f) \mid s \in Im(f)\}$.

Exemple 17. *Nous présentons ici un RTS* $\mathcal{M} = ((\Sigma_\mathcal{M}, \Lambda_\mathcal{M}), \mathcal{T}_\mathcal{M}, t^0_\mathcal{M}, Q^0_\mathcal{M})$ *à trois tuiles formellement défini par :*

- $\Sigma_\mathcal{M} = \{a, b, c\}$,
- $\Lambda_\mathcal{M} = \{\texttt{init}, \lambda\}$,
- $\mathcal{T}_\mathcal{M} = \{t^0_\mathcal{M}, A, B\}$,
- $Q^0_\mathcal{M} = \{1\}$.

où tuile $t^0_\mathcal{M}$ *est la tuile définie par :*

- $Q_{t^0_\mathcal{M}} = \{1, 2, 3, 4, 5, 6, 7\}$,
- $\Delta_{t^0_\mathcal{M}} = \{(1, a, 2), (1, b, 3), (2, b, 5), (3, a, 6), (7, b, 4)\}$,
- $col_{t^0_\mathcal{M}} = \{(1, \texttt{init}), (4, \lambda)\}$,
- $S_{t^0_\mathcal{M}} = \emptyset$,
- $F_{t^0_\mathcal{M}} = \{(A, \{(1, 5), (2, 6), (3, 7)\})\}$

où tuile A *est définie par :*

- $Q_A = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$,
- $\Delta_A = \{(1, a, 4), (2, a, 5), (5, b, 2), (6, b, 3), (4, c, 5), (5, c, 6), (4, a, 7), (5, a, 8), (8, b, 5),$
  $(9, b, 6), (5, a, 10), (11, b, 6)\}$,
- $col_A = \emptyset$,
- $S_A = \{1, 2, 3\}$,
- $F_A = \{(A, \{(1, 7), (2, 8), (3, 9)\}), (B, \{(1, 10), (2, 11)\})\}$

où B *est la tuile définie par :*

- $Q_B = \{1, 2, 3, 4, 5\}$,
- $\Delta_B = \{(1, a, 3), (5, b, 2)\}$
- $col_B = \emptyset$,
- $S_B = \{1, 2\}$,
- $F_B = \{(A, \{(1, 3), (2, 4), (3, 5)\})\}$.

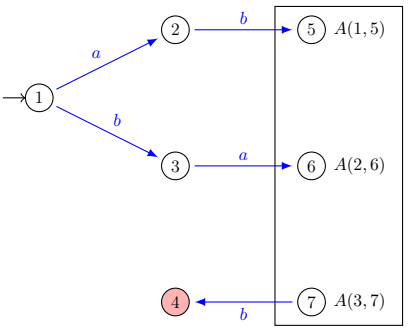


FIGURE 5.1 – Axiome du RTS de l'exemple 17.

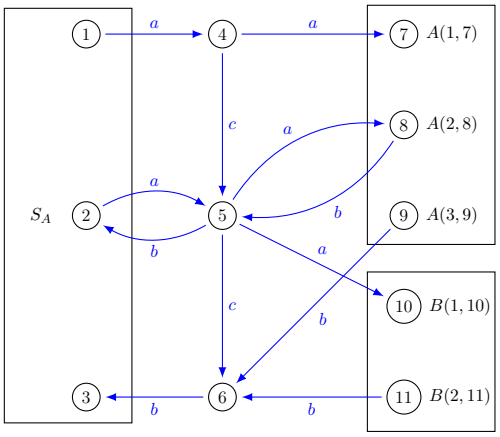


FIGURE 5.2 – Tuile A du RTS de l'exemple 17

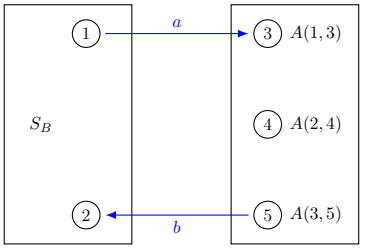


FIGURE 5.3 – Tuile B du RTS de l'exemple 17

Nous représentons ces trois tuiles par des graphes finis. La couleur `init` *est représentée par une flèche entrante et la couleur* λ *pour la couleur rouge. La notation* $(A,(1,7))$ *signifie qu'il existe une frontière* (A, f_A) *et que* $f_A(1) = 7$.

Dans la suite, nous aurons besoin de considérer une tuile comme un DES.

Définition 5.2. Étant donnée une tuile $t = ((\Sigma_{\mathcal{M}}, \Lambda_{\mathcal{M}}), Q_t, \Delta_t, col_t, S_t, F_t)$ d'un RTS $\mathcal{M}$, et un ensemble d'états $Q^0 \subseteq Q_t$, on appelle *DES sous-jacent* de t, le graphe DES :

$$[t]_{Q^0} = (Q_t, \Sigma_{\mathcal{M}}, Q^0, \Lambda_{\mathcal{M}}, \Delta_t, col_t)$$

Les tuiles prises indépendamment ne sont que des DES finis. Nous allons maintenant présenter comment il est possible de fusionner plusieurs tuiles en utilisant la frontière de l'une d'elles.

Définition 5.3. Soient t_A et t_B deux tuiles sur (Σ, Λ). On dit que t_A peut être *étendue* par t_B si F_{t_A} contient un couple (t_B, f_B). Dans ce cas, on peut alors définir l'*extension* de t_A par t_B, qui est la tuile $t_A \bullet_{f_B} t_B = ((\Sigma, \Lambda), Q, \Delta, col, S_{t_A}, F)$, obtenue de la manière suivante :

– $\Sigma = \Sigma_{t_A} \cup \Sigma_{t_B}$,

- $\Lambda = \Lambda_{t_A} \cup \Lambda_{t_B}$,
- $Q = Q_{t_A} \cup \{i \in \mathbb{N} \mid \exists q \in (Q_{t_B} \setminus S_{t_B}), i = q + max(Q_{t_A}) + 1\}$,
- $\Delta = \left\{ \begin{array}{ll} & \Delta_{t_A} \\ \cup & \{(s,a,s') \mid \exists (q,a,q') \in \Delta_{t_B}, q,q' \in S_{t_B} \wedge (f_B(q) = s) \wedge (f_B(q') = s')\} \\ \cup & \{(s,a,s') \mid \exists (q,a,q') \in \Delta_{t_B}, q \in S_{t_B} \wedge (f_B(q) = s) \wedge (s' = q' + max(Q_{t_A}) + 1)\} \\ \cup & \{(s,a,s') \mid \exists (q,a,q') \in \Delta_{t_B}, q' \in S_{t_B} \wedge (f_B(q') = s') \wedge (s = q + max(Q_{t_A}) + 1)\} \\ \cup & \{(s,a,s') \mid \exists (q,a,q') \in \Delta_{t_B} \wedge (s = q + max(Q_{t_A}) + 1) \wedge (s' = q' + max(Q_{t_A}) + 1)\} \end{array} \right.$
- $col = col_{t_A} \cup \left\{ \begin{array}{ll} & \{(s,\lambda) \mid \exists q \in Q_{t_B}, (s = q + max(Q_{t_A}) + 1) \wedge (q,\lambda) \in col_{t_B}\} \\ \cup & \{(s,\lambda) \mid \exists q \in S_{t_B}, f_B(q) = s \wedge (q,\lambda) \in col_{t_B}\} \end{array} \right.$
- $(t_C, f_C) \in F$ si $\left\{ \begin{array}{ll} & (t_C, f_C) \in (F_{t_A} \setminus (t_B, f_B)) \\ ou & \exists (t_C, f'_C) \in F_{t_B}, f_C = \{(q,s) \mid \exists (q,s') \in f'_C, s = s' + max(Q_{t_A}) + 1\} \end{array} \right.$

L'opération d'extension permet de construire une seule tuile à partir de deux tuiles en étendant une partie de la frontière. Intuitivement, cela revient à copier la tuile t_B et à coller cette copie sur la tuile t_A de manière à ce que chaque sommet du support de t_B se raccroche à un sommet de la frontière de t_A.

Lemme 3. *Soit t une tuile et soient (t_A, f_A) et (t_B, f_B) deux éléments de la frontière de t. Alors $t \bullet_{f_B} t_B \bullet_{f_A} t_A$ et $t \bullet_{f_B} t_B \bullet_{f_A} t_A$ sont isomorphes.*

L'ordre dans lequel on procède pour les extensions n'est donc pas important. Cela nous permet d'étendre l'opération d'extension à un ensemble de tuiles.

Définition 5.4. Étant donnés un ensemble de tuiles $\mathcal{T}$ et une tuile t de $\mathcal{T}$, l'*extension complète* de la tuile t par $\mathcal{T}$ est la tuile $\mathcal{T}(t)$, obtenue par extension de tous les éléments de la frontière :

$$\mathcal{T}(t) = t \left\{ \bullet_{f'} t' \right\}_{(t',f') \in F_t}$$

Remarque 5.1. On peut noter que pour toute tuile t de $\mathcal{M}$, $Q_t \subseteq Q_{\mathcal{T}_\mathcal{M}(t)}$, $\Delta_t \subseteq \Delta_{\mathcal{T}_\mathcal{M}(t)}$ et $col_t \subseteq col_{\mathcal{T}_\mathcal{M}(t)}$ et que les alphabets Σ, Λ sont les mêmes. On écrit alors que $[t] \subseteq [\mathcal{T}_\mathcal{M}(t)]$.

Enfin, on définit la *sémantique* d'un RTS $\mathcal{M}$ comme le DES obtenu par l'union de tous les DES de toutes les tuiles résultantes des opérations d'extension successives.

Définition 5.5. Soit $\mathcal{M}$ un RTS. La *sémantique* de $\mathcal{M}$ est définie à isomorphisme près, par le DES suivant :

$$[\![\mathcal{M}]\!] = \bigcup_{k \in \mathbb{N}} [\mathcal{T}^k_\mathcal{M}(t^0_\mathcal{M})]_{Q^0_\mathcal{M}}$$

L'union infinie de la définition 5.5 est valide car, par construction, pour tout $k \geq 0$: $[\mathcal{T}^k_\mathcal{M}(t)] \subseteq [\mathcal{T}^{k+1}_\mathcal{M}(t)]$ (comme noté dans la remarque 5.1). Par la suite, il sera ponctuellement utile de considérer cette construction pour une tuile initiale t différente de $t^0_\mathcal{M}$, on notera alors :

$$\mathcal{T}^\omega_\mathcal{M}(t) = \bigcup_{k \in \mathbb{N}} [\mathcal{T}^k_\mathcal{M}(t)]$$

On réutilise la notion de *niveau* introduite précédemment pour les HR-grammaires. Le niveau $\ell(q)$ d'un état q de $[\![\mathcal{M}]\!]$ est le plus petit entier $k \in \mathbb{N}$ tel que q est un état de $[\mathcal{T}^k_{\mathcal{M}}(t)]$.

La tuile $t(q)$ représente la tuile de $\mathcal{T}_{\mathcal{M}}$ qui engendre q. Le sommet $v(q)$ représente le sommet de $t(q)$ dont q est une copie. Pour un sommet s d'une tuile de $\mathcal{M}$, $Q(s) = \{q \in Q_{[\![\mathcal{M}]\!]} \mid s = v(q)\}$ représente l'ensemble des états de $[\![\mathcal{M}]\!]$ qui sont des copies de s.

Exemple 18. *Nous illustrons le principe d'extension en utilisant le RTS défini précédemment dans l'exemple 17. Les figures 5.4 et 5.5 représentent respectivement l'extension complète de l'axiome $\mathcal{T}_{\mathcal{M}}(t^0_{\mathcal{M}})$ et la double extension $\mathcal{T}_{\mathcal{M}}{}^2(t^0_{\mathcal{M}})$*

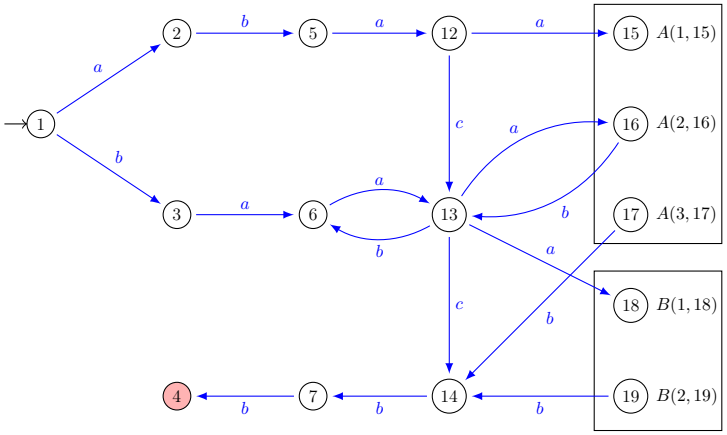


FIGURE 5.4 – La tuile $\mathcal{T}_{\mathcal{M}}(t^0_{\mathcal{M}})$

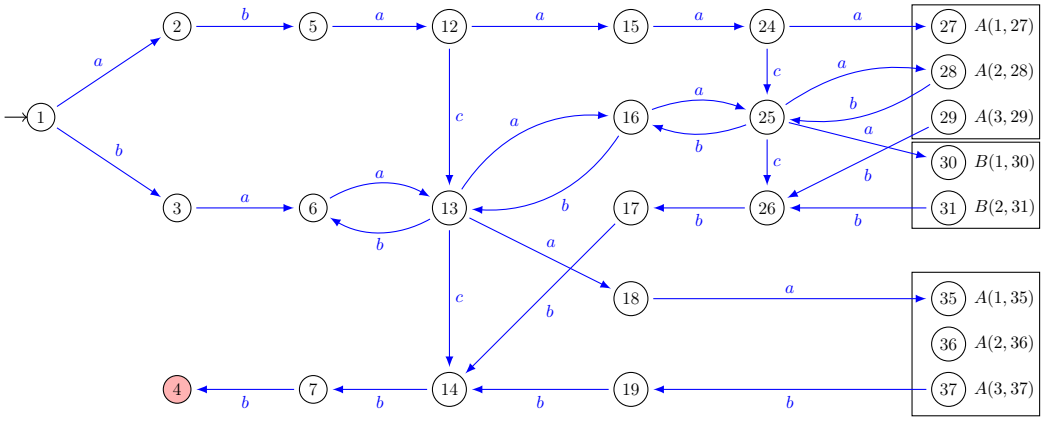


FIGURE 5.5 – La tuile $\mathcal{T}_{\mathcal{M}}{}^2(t^0_{\mathcal{M}})$

5.2 Tuiles de chemins

Les tuiles de chemin sont une forme normale des RTS qui se concentre sur les chemins. Cette présentation a été introduite dans [22] et est utilisée par Stéphane Hassen sur les grammaires déterministes de graphes sous le nom de *grammaire de chemins* ([38, 23]). La transformation

en grammaire de chemins, agit sur le DES engendré en dupliquant certains chemins tout en conservant les langages. Elle est utilisée dans [38, 23] pour déterminiser des graphes réguliers dans certains cas. Ici, elle permet un calcul simple de la clôture en supposant qu'il n'y a pas de chemin traversant inobservable.

Définition 5.6. Une *tuile de chemin* est une tuile dont le support est composé de deux sommets, **0** et **1**, et où le sommet **0** a un degré entrant nul et le sommet **1** a un degré sortant nul.

L'objectif en utilisant les tuiles de chemin est de décomposer les chemins. Ainsi, si un chemin passe par le sommet **0**, alors, soit il passe aussi par le sommet **1**, soit le chemin se termine « au delà » de la tuile. Ce qui peut être traduit dans la sémantique : si on passe par un état $q \in Q(\mathbf{0})$, alors, soit on ne traversera par la suite que des états de niveau supérieur à $\ell(q)$, soit on passe par l'état de $Q(\mathbf{1})$ de niveau $\ell(q)$.

On appelle *RTS de chemin*, un RTS dont toutes les tuiles sont des tuiles de chemin.

Proposition 5.1. *Pour tout RTS $\mathcal{M}$, il est possible de construire un RTS de chemin $chem(\mathcal{M})$ qui reconnaît les mêmes langages couleur à couleur. La construction est effective.*

A défaut de donner une démonstration formelle de la proposition 5.1, on en donne la construction.

Construction Pour chaque tuile t de $\mathcal{T}_\mathcal{M}$, on suppose que $\mathbf{0}, \mathbf{1} \notin Q_t$ et pour chaque couple de sommets du support $p, q \in S_t$ (non nécessairement distincts) on construit la tuile $t_{p,q}$ de la façon suivante :

- $\Sigma_{t_{p,q}} = \Sigma_t$ et $\Lambda_{t_{p,q}} = \Lambda_t$,
- $Q_{t_{p,q}} = (Q_t \setminus S_t) \cup \{\mathbf{0}, \mathbf{1}\}$,
- $\Delta_{t_{p,q}} = \left\{ \begin{array}{ll} & \Delta_t \\ \cup & \{(\mathbf{0}, a, q') \mid (p, a, q') \in \Delta_\mathcal{M}\} \\ \cup & \{(q', a, \mathbf{1}) \mid (q', a, q) \in \Delta_\mathcal{M}\} \end{array} \right.$
- $col_{t_{p,q}} = col_t \cup \{(\mathbf{0}, \lambda) \mid (p, \lambda) \in \Lambda_t\} \cup \{(\mathbf{1}, \lambda) \mid (q, \lambda) \in \Lambda_t\}$,
- $S_{t_{p,q}} = \{\mathbf{0}, \mathbf{1}\}$,
- $(t'_{p_1,p_2}, f) \in F_{t_{p,q}}$ si il existe $(t', f') \in F_t$ et si $f = \{(\mathbf{0}, q_1), (\mathbf{1}, q_2)\}$ tels que $(p_1, q_1) \in f'$ et $(p_2, q_2) \in f'$.

Ainsi, si $\mathcal{M} = ((\Sigma_\mathcal{M}, \Lambda_\mathcal{M}), \mathcal{T}_\mathcal{M}, t^0_\mathcal{M}, Q^0_\mathcal{M})$ est un RTS, le RTS de chemin reconnaissant les mêmes langages, est le RTS $chem(\mathcal{M}) \stackrel{\text{def}}{=} ((\Sigma_\mathcal{M}, \Lambda_\mathcal{M}), \mathcal{T}_{chem(\mathcal{M})}, t^0_\mathcal{M}, Q^0_\mathcal{M})$, où $\mathcal{T}_{chem(\mathcal{M})}$ est l'ensemble $\{t_{q_1,q_2} \mid t \in \mathcal{T}_\mathcal{M} \wedge q_1 \in S_t \wedge q_2 \in S_t\}$. On peut noter que $S_{t^0_\mathcal{M}} = \emptyset$

Cette transformation duplique une grande partie des états de $\mathcal{M}$ mais permet de préserver les langages. On peut noter que si $[\![\mathcal{M}]\!]$ est déterministe, $[\![chem(\mathcal{M})]\!]$ ne l'est pas nécessairement. Enfin, le nombre de tuiles engendrées à partir d'une tuile t est quadratique en fonction de $\rho(t)$, le nombre total de tuiles de $chem(\mathcal{M})$ est donc borné par $\max_{t \in \mathcal{T}_\mathcal{M}}(\rho(t)^2)$.

Exemple 19. *Les figures 5.6 à 5.14 représentent les tuiles de chemins obtenues à partir de la tuile A de l'exemple 17. Les frontières ne sont pas représentées par souci de concision. En effet, toutes les tuiles représentées appartiennent au domaine de la frontière.*

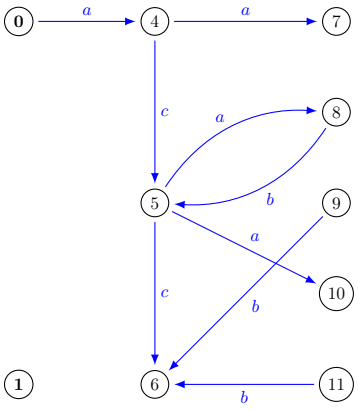


FIGURE 5.6 – La tuile A_{11}

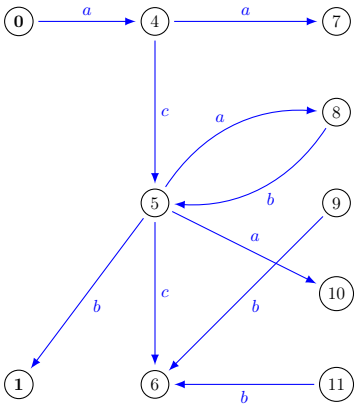


FIGURE 5.7 – La tuile A_{12}

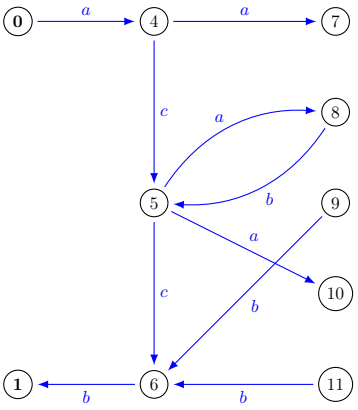


FIGURE 5.8 – La tuile A_{13}

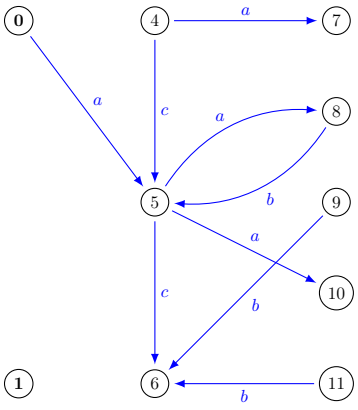


FIGURE 5.9 – La tuile A_{21}

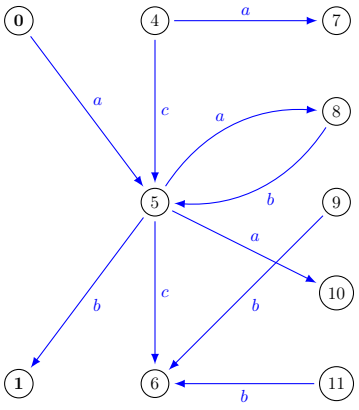


FIGURE 5.10 – La tuile A_{22}

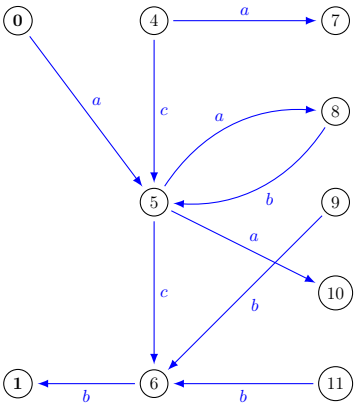


FIGURE 5.11 – La tuile A_{23}

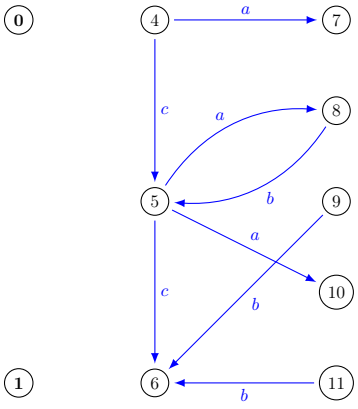


FIGURE 5.12 – La tuile A_{31}

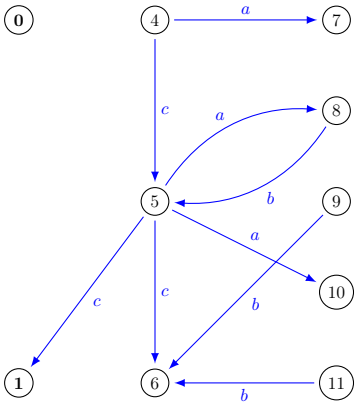


FIGURE 5.13 – La tuile A_{32}

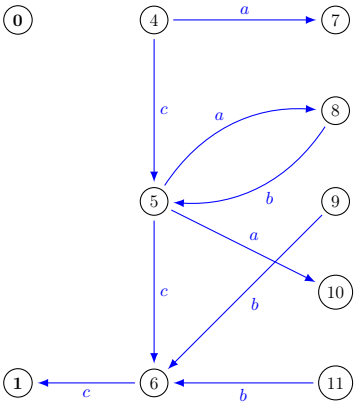


FIGURE 5.14 – La tuile A_{33}

5.3 Propriétés élémentaires héritées des grammaires de graphe

Dans ce paragraphe, nous allons considérer dans un premier temps le problème du calcul de l'ensemble des états accessibles depuis une couleur dans un RTS, puis l'opération de produit entre un RTS et un graphe fini.

5.3.1 Accessibilité

Le calcul des ensembles d'états accessibles est central dans les problèmes de vérification. Nous l'avons vu précédemment, il intervient à divers niveaux, aussi bien pour le test que pour la diagnosticabilité et l'opacité.

Ensemble des états accessibles depuis une couleur.

Proposition 5.2. *Pour un RTS donné $\mathcal{M}$, un sous alphabet $\Sigma^{\subseteq} \subseteq \Sigma_{\mathcal{M}}$ et une couleur $\lambda \in \Lambda_{\mathcal{M}}$, il est possible de construire de façon effective un RTS $Acc(\mathcal{M}, \lambda, \Sigma^{\subseteq})$, tel que, dans la sémantique $[\![\mathcal{M}]\!]$, les états accessibles depuis un état coloré par λ via un chemin de $\Sigma^{\subseteq}$ sont colorés par $\# \notin \Lambda_{\mathcal{M}}$ une nouvelle couleur. Cette construction est exponentielle si on impose que $[\![\mathcal{M}]\!]$ et $[\![Acc(\mathcal{M}, \lambda, \Sigma^{\subseteq})]\!]$ sont isomorphes.*

Construction Nous allons construire une fonction h, qui pour chaque tuile, colore une partie des accessibles, puis nous montrerons que le point fixe de cette fonction peut être atteint en un nombre fini d'étapes. Pour chaque tuile $t \in \mathcal{T}_{\mathcal{M}}$ et pour chaque sous-ensemble de sommets $P \subseteq S_t$ du support de t, on construit la tuile t_P dans laquelle les états de P sont colorés par $\#$.

On construit l'image $h(t_P)$ de t_P par h de la manière suivante :

- $\Sigma_{h(t_P)} = \Sigma_{t_P}$ et $\Lambda_{h(t_P)} = \Lambda_{t_P}$,
- $Q_{h(t_P)} = Q_{t_P}$,
- $\Delta_{h(t_P)} = \Delta_{t_P}$,
- $(q, \#) \in col_{h(t_P)}$ si $\begin{cases} & (q, \lambda) \in col_{t_P} \\ \vee & q \in P \\ \vee & \exists p \in Q_{t_P}, (p, \#) \in col_{t_P}, \exists a \in \Sigma^{\subseteq}, p \xrightarrow[t]{a} q \\ \vee & \exists (t'_{P'}, f) \in F_{t_P}, \exists p \in S_{t'_{P'}}, (p, q) \in f \wedge (p, \#) \in col_{t'_{P'}} \end{cases}$
- $S_{h(t_P)} = S_{t_P}$
- $(t'_{P'}, f) \in F_{h(t_P)}$ si $\begin{cases} & (t', f) \in F_t \\ \wedge & P' = \left\{ q \in Q_{t_P} \mid (p, q) \in f \wedge (q, \#) \in col_{h(t_P)} \right\} \end{cases}$

Comme seul le coloriage et la frontière sont modifiés, le domaine et l'image de h sont des ensembles finis. On peut observer que la fonction h est croisante pour l'inclusion des sous-structures étant donné qu'on ne fait que colorer par $\#$ des états qui ne l'étaient pas. La fonction h converge donc en un nombre fini d'étapes, on note $H(t_P)$ cette limite.

Le RTS $Acc(\mathcal{M}, \lambda, \Sigma^{\subseteq})$ est le RTS $((\Sigma_{\mathcal{M}}, \Lambda_{\mathcal{M}} \cup \{\#\}), \mathcal{T}, H(t^0_{\mathcal{M}}))$ où $\mathcal{T}$ est l'ensemble $\{H(t_P) \mid t \in \mathcal{T}_{\mathcal{M}} \wedge P \subseteq S_t\}$. On note que comme le support de $t^0_{\mathcal{M}}$ est vide, il n'est pas nécessaire de définir d'ensemble P d'état pré-colorés par $\#$.

Exemple 20. *Les figures 5.15 à 5.18 représentent les tuiles de chemins obtenues à partir du RTS de l'exemple 17 en colorant les états accessibles depuis le sommet* 3 *de l'axiome par la couleur bleue. Nous n'avons représenté ici que les tuiles qui servent effectivement lors de l'extension de l'axiome.*

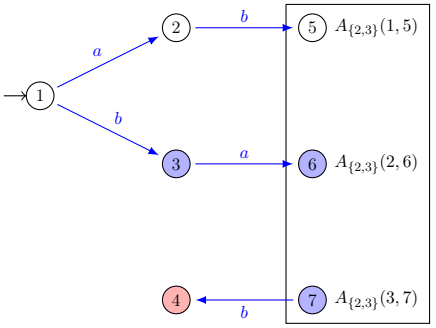


FIGURE 5.15 – L'axiome

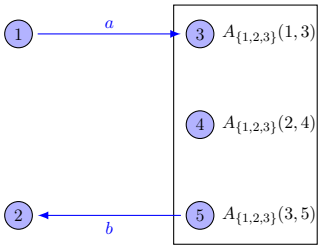


FIGURE 5.16 – La tuile $B_{\{1,2\}}$

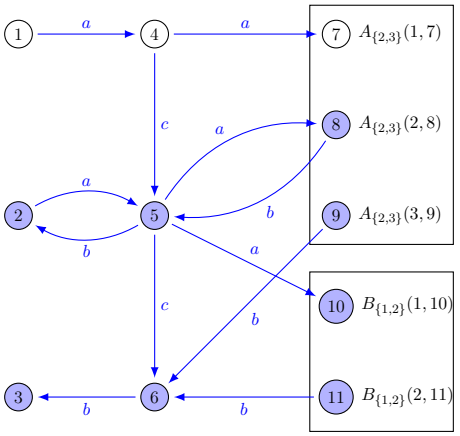


FIGURE 5.17 – La tuile $A_{\{2,3\}}$

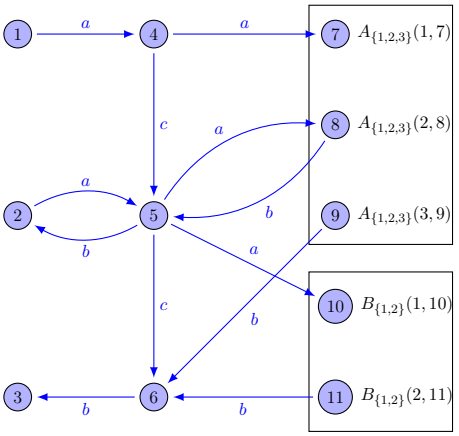


FIGURE 5.18 – La tuile $A_{\{1,2,3\}}$

Détection d'un chemin infini

Pour la diagnosticabilité, nous avons besoin de détecter les chemins infinis formés de transitions non observables.

Il existe deux types de chemins infinis, ceux qui sont composés de circuits, et ceux qui traversent un nombre infini d'états distincts que nous appellerons *chemins divergents*, et qui peuvent poser problème si on cherche à représenter la clôture de manière finie. Pour la diagnosticabilité, il est nécessaire de considérer les deux types de chemin infinis.

Nous utilisons pour cela le résultat suivant :

Proposition 5.3. *Pour tout RTS $\mathcal{M}$, il existe un circuit ou un chemin divergent dans $[\![\mathcal{M}]\!]$ si, et seulement si, il existe un sommet s et deux états $p_1, p_2 \in Q(s)$ avec $\ell(p_1) \leq \ell(p_2)$ tels que $p_1 \overset{\sigma}{\underset{\mathcal{M}}{\Longrightarrow}} p_2$ pour $\sigma \in (\Sigma_{\mathcal{M}})^+$ et pour tout état q sur ce chemin, $\ell(p_1) \leq \ell(q)$.*

Démonstration. ($\Rightarrow$) Soit $\gamma = q_0 \overset{a_1}{\underset{[\![\mathcal{M}]\!]}{\Longrightarrow}} q_1 \overset{a_2}{\underset{[\![\mathcal{M}]\!]}{\Longrightarrow}} q_2 ...$ un chemin infini de $[\![\mathcal{M}]\!]$, avec $\forall k \in \mathbb{N}, a_k \in \Sigma_{\mathcal{M}}$. Si γ contient un circuit, alors il existe un état p avec un niveau minimal dans ce circuit. En posant $p_1 = p_2 = p$, nous obtenons deux états qui vérifient la proposition 5.3.

Maintenant, considérons un chemin élémentaire. Comme chaque état est traversé une unique fois, on peut construire une séquence d'états q_{i_k} tel que $\forall i_k \leq j, \ell(q_{i_k}) \leq \ell(q_j)$. Comme il n'y a qu'un nombre fini de sommets, il y a un plus petit s tel que deux états de $Q(s)$ apparaissent dans ce chemin. Soient p_1 et p_2 ces deux états.

($\Leftarrow$) Supposons maintenant qu'il existe un sommet s et deux états $p_1, p_2 \in Q(s)$ tels que $p_1 \overset{\sigma}{\underset{[\![\mathcal{M}]\!]}{\Longrightarrow}} p_2$ pour $\sigma \in \Sigma^+_{\mathcal{M}}$, et que pour tout état q de ce chemin, alors $\ell(p_1) \leq \ell(q)$. On distingue deux cas. Si $\ell(p_1) = \ell(p_2)$ alors, comme tout chemin depuis impliquant deux occurrences distinctes de la même tuile implique une tuile de niveau inférieur, et que $\ell(p_1)$ est le niveau minimal, $p_1 = p_2$, et ce chemin est un circuit.

Sinon, $\ell(p_1) < \ell(p_2)$, soit $\gamma_0 = p_1 \overset{\sigma}{\underset{[\![\mathcal{M}]\!]}{\Longrightarrow}} p_2$ un chemin de $[\![\mathcal{M}]\!]$. Par définition d'un RTS, un chemin similaire, γ_1, peut être construit depuis p_2 vers un état $p_3 \in Q(s)$, avec, $\gamma_1 = q_2 \overset{\sigma}{\underset{[\![\mathcal{M}]\!]}{\Longrightarrow}} q_3$, $\ell(p_2) < \ell(p_3)$, et $\ell(p_2) \leq \ell(q)$ pour tout q de ce chemin. Répéter ce processus permet de construire un chemin infini d'actions inobservables dans $[\![\mathcal{M}]\!]$: un chemin divergent. □

Remarque 5.2. Il est assez simple d'identifier si le sommet s d'une tuile t de $\mathcal{M}$ peut être impliqué dans un chemin infini en utilisant la proposition 5.3. Il suffit de considérer la tuile t comme une axiome, de colorer le sommet s par une couleur λ. Ensuite, en utilisant la proposition 5.2, on peut colorer par une autre couleur #, les sommets accessibles depuis λ. Si il existe un sommet coloré à la fois par λ et #, alors s est impliqué dans un chemin infini .

5.3.2 Produit avec un graphe fini

Comme il a été vu dans le chapitre 4, le produit de deux graphes réguliers n'est pas un graphe régulier en général. En revanche, le produit d'un RTS avec un DES fini est un RTS.

Proposition 5.4. *Le produit d'un RTS avec un DES fini est un RTS. La construction est effective.*

Construction. Étant donné un RTS $\mathcal{M} = ((\Sigma_{\mathcal{M}}, \Lambda_{\mathcal{M}}), \mathcal{T}_{\mathcal{M}}, t^0_{\mathcal{M}}, Q^0_{\mathcal{M}})$ et un graphe fini, étiqueté et coloré $\mathcal{A} = (Q_{\mathcal{A}}, \Sigma_{\mathcal{A}}, \Lambda_{\mathcal{A}}, Q^0_{\mathcal{A}}, \Delta_{\mathcal{A}}, col_{\mathcal{A}})$, on construit le RTS $\mathcal{M}_{\times\mathcal{A}}$ de la façon suivante :

- $\Sigma_{\mathcal{M}_{\times\mathcal{A}}} = \Sigma_{\mathcal{M}} \cap \Sigma_{\mathcal{A}}$,
- $\Lambda_{\mathcal{M}_{\times\mathcal{A}}} = \Lambda_{\mathcal{M}} \cup \Lambda_{\mathcal{A}}$,
- $\mathcal{T}_{\mathcal{M}_{\times\mathcal{A}}} = \{t \times \mathcal{A} \mid t \in \mathcal{T}_{\mathcal{M}}\}$,

– $t^0_{\mathcal{M}_{\times\mathcal{A}}} = t^0_{\mathcal{M}} \times \mathcal{A}$,
– $Q^0_{\mathcal{M}_{\times\mathcal{A}}} = Q^0_{\mathcal{M}} \times Q^0_{\mathcal{A}}$
avec $t \times \mathcal{A}$ la tuile définie à partir de t et de $\mathcal{A}$ de la manière suivante :
– $Q_{t\times\mathcal{A}} = Q_t \times Q_{\mathcal{A}}$,
– $\Delta_{t\times\mathcal{A}} = \{((s,q),a,(s',q')) \mid (s,a,s') \in \Delta_t \wedge (q,a,q') \in \Delta_{\mathcal{A}}\}$,
– $col_{t\times\mathcal{A}} = \{((s,q),\lambda) \mid (s,\lambda) \in col_t \vee (q,\lambda) \in col_{\mathcal{A}}\}$,
– $S_{t\times\mathcal{A}} = S_t \times Q_{\mathcal{A}}$,
– $(t',f') \in F_{t\times\mathcal{A}}$ si il existe $(t',f) \in F_t$ tel que $f' = \{((s,q),(s',q)) \mid (s,s') \in f \wedge q \in Q_{\mathcal{A}}\}$

Comme nous l'avons vu au chapitre 1, un mot w appartient à $L_{\mathcal{M}}(\lambda) \cap L_{\mathcal{A}}(\lambda')$ si il existe une exécution dans $\mathcal{M}_{\times\mathcal{A}}$ dont la trace est w et qui atteint un état coloré à la fois par λ et par λ'.

5.4 Clôture de la sémantique d'un RTS

Le calcul de la clôture pour les DES est habituellement concentré sur la préservation des traces, ce qui peut être obtenu soit par une τ^*a-simulation, soit par $a\tau^*$-simulation (voir § 1.4.1). Dans le cas qui nous intéresse, les états d'un DES ont des couleurs, et de telles transformations peuvent aboutir à la suppression d'informations importantes. En effet, ces couleurs représentent des fautes, des secrets, ou dans le cas du test, différents types de blocage ou bien encore les échecs ou les succès de tests. Pour préserver ces informations, nous introduisons la notion de *traces colorées* afin d'obtenir une équivalence plus précise que l'équivalence de traces.

Traces colorées et coloration équivalente. Une *trace colorée* dans un DES $\mathcal{M}$ pour un alphabet Σ^o d'événements observables est un mot de $(\Lambda.\Sigma^{o+})^+.\Lambda$. Une trace colorée $\lambda_1 w_1 \lambda_2 w_2 ... \lambda_n$ est *reconnue* par $\mathcal{M}$ si il existe n états $q_1, q_2, ..., q_n$ tels que pour tout $1 \leq k \leq n-1$, $q_k \xRightarrow[\mathcal{M}]{w_k} q_{k+1}$, et $(q_k, \lambda_k) \in col_{\mathcal{M}}$.

Pour cette notion de reconnaissance, on n'exige pas que les chemins débutent nécessairement par un état initial. De plus, étant donné un chemin dans un DES, il est possible de définir plusieurs traces colorées distinctes pour un même chemin. Enfin, le mot vide n'est l'étiquette d'aucune trace colorée, il s'agit là d'un choix arbitraire décidé pour des raisons techniques : préserver les traces colorées étiquetées par τ, tout en supprimant les transitions avec cette étiquette, imposerait une représentation plus complexe des couleurs dans les états.

Définition 5.7 (TC-équivalence)**.** Deux DES $\mathcal{M}$ et $\mathcal{M}'$ sont *TC-équivalents* s'ils reconnaissent les mêmes traces colorées.

La TC-équivalence est plus précise que l'équivalence de trace, étant donné que deux systèmes TC-équivalents ont les mêmes traces (à l'exception éventuelle du mot vide) tandis que la réciproque est en générale fausse. En revanche, la TC-équivalence est moins précise que la bissimulation.

Proposition 5.5. *Pour tout RTS $\mathcal{M}$, il est possible de construire un RTS $clo(\mathcal{M})$, qui ne contient pas de τ-transitions, tel que $[\![\mathcal{M}]\!]$ et $[\![clo(\mathcal{M})]\!]$ sont TC-équivalents, et tel que $[\![clo(\mathcal{M})]\!]$ ne contienne pas de degré infini. La construction est effective.*

A présent, nous allons établir ce résultat en donnant une construction pour $clo(M)$.

L'objectif recherché est de calculer la clôture, par rapport aux événements non-observables, d'un DES défini par un RTS. Une approche naïve afin d'obtenir un tel résultat pourrait être de calculer cette clôture sur chacune des tuiles. Malheureusement, aussi bien la clôture avec les τ devant que celle avec les τ derrière ne peuvent traiter convenablement le cas des transitions dont la source ou la cible appartiennent au support ou à la frontière. Pour ces raisons nous utilisons la clôture mixte qui a deux avantages : elle se manipule mieux avec la structure de RTS et elle préserve les traces colorées.

La clôture est effectuée en cinq étapes :

1. Pour chaque tuile, on sépare les sommets du support de ceux de la frontière.
2. Le RTS est mis sous forme d'un RTS de chemin.
3. Les chemins de τ-transitions entre le support et la frontière sont identifiés.
4. Le RTS est transformé pour que chaque tuile soit appelée dans un unique contexte.
5. Les τ-transitions internes sont supprimées en utilisant la clôture mixte présentée sur les DES finis.

Séparation des sommets du support et de la frontière

Dans un RTS, il est possible qu'un sommet appartienne à la fois au support d'une tuile et à un des éléments de la frontière. Ce genre de cas se rencontre nécessairement quand un état de degré infini existe dans la sémantique, mais il peut aussi se produire ponctuellement.

La première étape de la clôture va donc être d'éliminer le degré infini en supprimant les sommets qui appartiennent à la fois au support et à un élément de la frontière. Pour cela, on duplique le sommet et on ajoute deux transitions étiquetées par τ entre ces deux sommets, l'original reste dans le support, et la copie est utilisée dans la frontière. Plus formellement, pour une tuile t_A, on définit la fonction $h_A : Q_{t_A} \to \mathbb{N}$ telle que :

$$\begin{cases} h_A(s) = s + \max(Q_{t_A}) + 1 & si \qquad s \in S_{t_A} \\ h_A(s) = s & sinon \end{cases}$$

et on construit la tuile t'_A de la façon suivante :

- $\Sigma_{t'_A} = \Sigma_{t_A}$, $\Lambda_{t'_A} = \Lambda_{t_A}$, $S_{t'_A} = S_{t_A}$
- $Q_{t'_A} = Q_{t_A} \cup \{h_A(s) \mid s \in S_{t_A}\}$,
- $\Delta_{t'_A} = \Delta_{t_A} \cup \{(s, \tau, h_A(s)) \mid s \in S_{t_A}\} \cup \{(h_A(s), \tau, s) \mid s \in S_{t_A}\}$,
- $F_{t'_A} = \{(t_B, h_A \circ f_B) \mid (t_B, f_B) \in F_{t_A}\}$.

Mise sous forme de RTS de chemin

On utilise la proposition 5.1 sur le RTS obtenu par la construction précédente afin d'obtenir un RTS de chemin reconnaissant les mêmes traces colorées.

Identification des chemins de τ-transitions entre le support et la frontière

Il y a deux opérations symétriques. Nous présentons uniquement celle qui permet d'identifier les chemins depuis le support vers la frontière. La transformation inverse est effectuée de la même manière, en considérant que les transitions sont inversées.

Cette transformation est effectuée sur chacune des tuiles en deux étapes. La première consiste à étendre chaque tuile de façon à ce que, soit, elle ne contient plus de chemin de τ depuis le support vers la frontière, soit, un tel chemin satisfait une certaine condition (qu'on définira). La seconde consiste à transformer la tuile précédente en ajoutant certains arcs pour contourner les trajectoires problématiques.

Pour toute tuile t de $\mathcal{T}_{\mathcal{M}}$, on note $\hat{t}$ la tuile $\mathcal{T}_{\mathcal{M}}^N(t)$ avec $N = |\mathcal{T}_{\mathcal{M}}|$, la frontière est donnée par $F_{\hat{t}} = \left\{(\hat{t_2}, f_2) \mid (t_2, f_2) \in F_{\mathcal{T}_{\mathcal{M}}^N(t)}\right\}$. Comme les supports ne sont pas modifiés, cette définition de la frontière est cohérente.

Pour tout τ-chemin γ, avec $init(\gamma) \in S_{\hat{t}}$ et $fin(\gamma) \in Im(Im(F_{\hat{t}}))$ depuis le support vers la frontière, il existe $s_1, s_2 \in Q_{\hat{t}}$ deux sommets distincts vérifiant les conditions suivantes :

1. $\gamma = init(\gamma) \overset{w_1}{\underset{\hat{t}}{\Longrightarrow}} s_1 \overset{w_2}{\underset{\hat{t}}{\Longrightarrow}} s_2 \overset{w_3}{\underset{\hat{t}}{\Longrightarrow}} fin(\gamma)$, avec $w_1, w_2, w_3 \in \tau^*$,
2. il existe k tel que $s_1 \in \mathcal{T}_{\mathcal{M}}^k(t) \setminus \mathcal{T}_{\mathcal{M}}^{k-1}(t)$ tel que quel que soit s successeur de s_1 sur le chemin γ, $s \in \mathcal{T}_{\mathcal{M}}^N(t) \setminus \mathcal{T}_{\mathcal{M}}^k(t)$,
3. il existe une tuile $t_{\substack{\bullet\bullet'\\ss}}$ telle que s_1 et s_2 appartiennent à $Q(\overset{\bullet}{s})$.

En effet, considérons la sous-suite de $N+1$ sommets de γ, $(u_k)_{0\leq k\leq N}$ telle que $u_0 = init(\gamma)$ et u_k est le dernier sommet de γ appartenant à $\mathcal{T}_{\mathcal{M}}^{k-1}(t)$ pour $1 \leq k$. Par construction, tous les éléments de la suite $(u_k)_{0\leq k\leq N}$ vérifient les conditions 1 et 2. Comme il existe seulement N tuiles et qu'elles sont sous forme de tuile de chemin, il existe dans la suite $(u_k)_{0\leq k\leq N}$, deux sommets vérifiant la condition 3. On note alors s_1' et s_2' les sommets de $Q(\overset{\bullet'}{s})$ engendrés par les extensions qui ont engendré respectivement s_1 et s_2.

À présent, il s'agit d'ajouter un nouvel arc τ de γ de façon à coutourner ce chemin traversant, en conservant les traces colorées. On note s_3 (non nécessairement distinct de s_1) le prédécesseur de s_2 le long du chemin γ. On construit ensuite la tuile t' à partir de $\hat{t}$ de la façon suivante :

- $\Sigma_{t'} = \Sigma_{\hat{t}}$, $\Lambda_{t'} = \Lambda_{\hat{t}}$,
- $Q_{t'} = Q_{\hat{t}}$,
- $\Delta_{t'} = \Delta_{\hat{t}} \cup \{(s, a, s_2') \mid (s, a, s_1') \in \Delta_{\hat{t}}\}$,
- $col_{t'} = col_{\hat{t}}$,

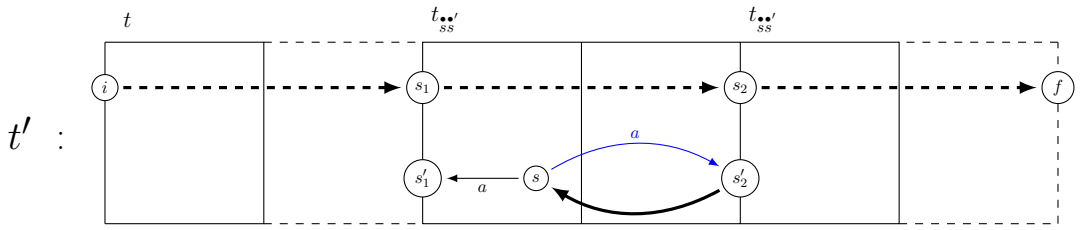


FIGURE 5.19 – Représentation schématique de la troisième étape de la clôture

- $S_{t'} = S_{\hat{t}}$,
- $F_{t'} = F_{\hat{t}}$.

Remarque 5.3. La figure 5.19 est une représentation schématique de la troisième étape de la clôture. Les arcs en gras représentent des chemins, c'est-à-dire des suites de transitions. Les arcs en gras et pointillés représentent les chemins de transitions étiquetées par τ. Les sommets i et f représentent respectivement les extrémités initiales et finales d'un chemin γ. L'arc ajouté est représenté en bleu.

Le RTS $\mathcal{M}'$ obtenu à partir du RTS $\mathcal{M}$ en appliquant la transformation ci-dessus a les mêmes traces colorées que $\mathcal{M}$.

Tuiles à contexte unique

Préalablement à la suppression des arcs étiquetés par τ nous allons avoir besoin d'identifier pour chaque tuile le contexte de son extension. Pour cela, nous transformons le RTS afin que chaque tuile ne soit présente que dans la frontière d'une unique autre tuile. Pour un RTS $\mathcal{M}$, on construit l'ensemble $\mathcal{T}$ à partir de $\mathcal{T}_{\mathcal{M}}$ de la manière suivante. Une tuile de $\mathcal{T}$ est une tuile t_{AB} la duplication d'une tuile t_A de $\mathcal{T}_{\mathcal{M}}$ étendue dans le contexte de la tuile t_B.

On note $t_{AB} \in \mathcal{T}$ la tuile dont les états, couleurs, arcs, coloriages et supports sont les mêmes que la tuile t_A. En revanche la frontière $F_{t_{AB}}$ est l'ensemble $\{(t_{CA}, f_C) \mid (t_C, f_C) \in F_{t_A}\}$.

Suppression des arcs étiquetées par τ dans un RTS

Étant donné un RTS $\mathcal{M}$, on appelle $\mathcal{M}'$ le RTS obtenu par l'enchaînement des quatre transformations précédentes. On peut remarquer que chaque transformation n'altère pas les propriétés de la précédente. Ainsi $\mathcal{M}'$ est un RTS de chemin, dont le support et la frontière sont séparés, qui vérifient le lemme **??** et dont chaque tuile est appellée dans un contexte unique. De plus, $\mathcal{M}'$ engendre les mêmes traces colorées que $\mathcal{M}$.

Étant donnée une tuile t_{AB} de $\mathcal{M}'$, on définit les quatre ensembles de sommets suivants reliés à d'autres tuiles par des chemins de τ.

- ceux qui vont vers la frontière d'une tuile t_{CA} :
  $Q_A^{succ}(C) = \left\{ s \in Q_{t_{AB}} \mid \exists (t_{CA}, f_C) \in F_{t_{AB}}, \exists (s', s_2) \in f_C, s \xRightarrow[t_{AB}]{\tau^*} s_2 \right\} \setminus S_{t_{AB}}$

– ceux qui viennent de la frontière d'une tuile t_{CA} :
$Q_A^{pred}(C) = \left\{ s \in Q_{t_{AB}} \mid \exists (t_{CA}, f_C) \in F_{t_{AB}}, \exists (s', s_2) \in f_C, s_2 \xRightarrow[t_{AB}]{\tau^*} s \right\} \setminus S_{t_{AB}}$
– ceux qui vont vers le support :
$Q_A^{sortants} = \left\{ s \in Q_{t_{AB}} \mid \exists s' \in S_{t_{AB}}, s \xRightarrow[t_{AB}]{\tau^*} s' \right\} \setminus Im(Im(F_{t_{AB}}))$
– ceux qui viennent du support :
$Q_A^{entrants} = \left\{ s \in Q_{t_{AB}} \mid \exists s' \in S_{t_{AB}}, s' \xRightarrow[t_{AB}]{\tau^*} s \right\} \setminus Im(Im(F_{t_{AB}}))$

On peut remarquer que pour toute tuile ces notations ne dépendent pas du contexte dans lequel est étendue t_{AB}, c'est pourquoi B n'apparaît pas dans la notation.

On peut aussi remarquer que nous avons fait en sorte que les sommets du support et ceux de la frontière ne soient pas dans les mêmes ensembles.

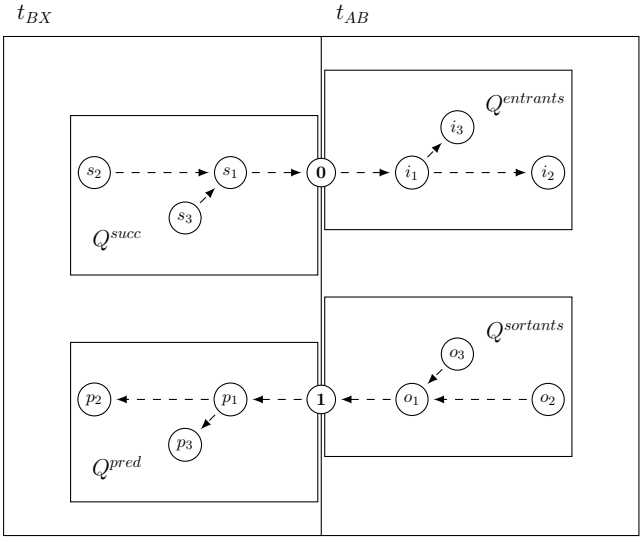


FIGURE 5.20 – Représentation des quatre ensembles d'états décrits pour la cinquième étape de la clôture en § 5.4.

Maintenant, nous pouvons construire le RTS $clo(\mathcal{M})$ comme étant l'ensemble de tuiles $\mathcal{T}_{clo(\mathcal{M})}$ où $t_{clo} \in \mathcal{T}_{clo(M)}$ est obtenue à partir de la tuile t_{AB} de $\mathcal{T}_{\mathcal{M}'}$ de la manière suivante :
– $\Sigma_{t_{clo(AB)}} = \Sigma_{\mathcal{M}} \setminus \{\tau\}$,
– $\Lambda_{t_{clo(AB)}} = \Lambda_{\mathcal{M}}$,
– $Q_{t_{clo(AB)}} = \left\{ \begin{array}{ll} \cup & \left\{ (s_1, s_2) \in Q_t \times Q_t \mid s_1 \xRightarrow[t]{\tau^*} s_2 \right\} \\ \cup & Q_B^{succ}(A) \times Q_A^{entrants} \\ \cup & Q_A^{sortants} \times Q_B^{pred}(A) \\ \cup & \left\{ Q_A^{succ}(C) \times Q_C^{entrants} \mid t_{CA} \in Dom(F_{t_{AB}}) \right\} \\ \cup & \left\{ Q_C^{sortants} \times Q_A^{pred}(C) \mid t_{CA} \in Dom(F_{t_{AB}}) \right\} \end{array} \right.$
– $\Delta_{t_{clo(AB)}} = \left\{ ((s_1, s_2), a, (s_1', s_2')) \mid a \in \Sigma_{\mathcal{M}}^o, (s_1, s_2), (s_1', s_2') \in Q_{t_{clo(AB)}}, (s_2, a, s_1') \in \Delta_{t_{AB}} \right\}$

- $col_{t_{clo(AB)}} = \{((s_1, s_2), \lambda) \mid (s_1, \lambda) \in col_t \vee (s_2, \lambda) \in col_t\}$
- $S_{t_{clo(AB)}} = (Q_B^{succ}(A) \times Q_A^{entrants}) \cup (Q_A^{sortants} \times Q_B^{pred}(A))$,
- $(t_{clo(CA)}, f) \in F_{t_{clo(AB)}}$ si $(t_{CA}, f') \in F_{t_{AB}}$ et f est l'identité sur $(Q_A^{succ}(C) \times Q_C^{entrants}) \cup (Q_C^{sortants} \times Q_A^{pred}(C))$.

Il reste maintenant à montrer que les traces colorées de $clo(\mathcal{M})$ sont les mêmes que celles de $\mathcal{M}$.

5.5 RTS pondérées

5.5.1 Définition

Comme cela a été vu pour les graphes réguliers, les systèmes engendrés par des RTS ne peuvent être déterminisés en général (il n'existe pas de RTS reconnaissant les mêmes traces engendrant un DES déterministe). Nous proposons donc, une définition des RTS pondérés analogue à celle des HR-grammaires pondérées, qui permettra de disposer d'opérations effectives pour la déterminisation et certains cas de produits.

Définition 5.8. Un RTS $\mathcal{M}$ est *pondéré pour* $Q^0_{\mathcal{M}}$, si $Q^0_{\mathcal{M}} \subseteq Q_{t^0_{\mathcal{M}}}$ et :

$$\forall w \in \Sigma^*_{\mathcal{M}}, (\forall q_0, q'_0 \in Q^0_{\mathcal{M}}, \forall q, q' \in Q_{[\![\mathcal{M}]\!]}, (q_0 \underset{[\![\mathcal{M}]\!]}{\overset{w}{\Longrightarrow}} q \wedge q'_0 \underset{[\![\mathcal{M}]\!]}{\overset{w}{\Longrightarrow}} q') \implies \ell(q) = \ell(q')$$

Quand l'ensemble des état initiaux est évident (défini par une couleur par exemple), on dit simplement que $\mathcal{M}$ est un RTS pondéré.

Intuitivement, la propriété de RTS pondéré consiste à dire que tous les états atteints au terme d'une exécution ont le même niveau.

Remarque 5.4. Déterminer si un RTS est pondéré pour un ensemble d'états donné est décidable en temps polynomial en utilisant un algorithme de [23].

5.5.2 Déterminisation

Une propriété majeure pour les RTS pondérés est d'être déterminisables.

Proposition 5.6 ([23])**.** *Tout RTS pondéré $\mathcal{M}$ peut être transformé en un RTS déterministe $det(\mathcal{M})$ reconnaissant les mêmes langages. Le RTS $det(\mathcal{M})$ peut être construit de manière effective, il est de taille exponentielle en fonction du nombre de sommets de $\mathcal{M}$.*

Construction. La construction de $det(\mathcal{M})$ est décrite en détail dans la thèse de Stéphane Hassen [38]. On peut résumer le processus en deux étapes clés :

- Construire $chem(\mathcal{M})$, ce qui est possible pour n'importe quel RTS.
- A partir de $chem(\mathcal{M})$, construire un RTS qui engendre un graphe déterministe, ce qui nécessite l'hypothèse de pondération.

5.5.3 Produit de RTS

Les RTS reconnaissent les langages algébriques forment un ensemble non clos par intersection. On en déduit que le produit de deux RTS n'est pas un RTS. Cependant, on peut étendre la notion de synchronisation définie sur les HR-grammaires aux RTS pondérés, ce qui nous donne des sous ensembles de RTS clos par produit.

Définition 5.9. Un RTS pondéré $\mathcal{M}_1$ *synchronise* un RTS pondéré $\mathcal{M}_2$ si pour tout mot w de $\Sigma^*_{\mathcal{M}}$:

$$(q_0 \underset{[\![\mathcal{M}_1]\!]}{\overset{w}{\Longrightarrow}} q \wedge q_0 \in Q^0_{[\![\mathcal{M}_1]\!]}) \implies (\exists q'_0 \in Q^0_{[\![\mathcal{M}_2]\!]}, \exists q' \in Q_{[\![\mathcal{M}_2]\!]}, q'_0 \underset{[\![\mathcal{M}_2]\!]}{\overset{w}{\Longrightarrow}} q' \wedge \ell(q) = \ell(q'))$$

On note $\mathcal{M}_1 \rhd \mathcal{M}_2$.

Proposition 5.7. *Soient deux RTS pondérés $\mathcal{M}_1$ et $\mathcal{M}_2$. Si $\mathcal{M}_1 \rhd \mathcal{M}_2$ et $\mathcal{M}_2 \rhd \mathcal{M}_1$ alors le produit $\mathcal{M}_{1\times 2}$ de $\mathcal{M}_1$ et $\mathcal{M}_2$ est un RTS pondéré.*

Construction Soient $\mathcal{M}_1$ et $\mathcal{M}_2$ deux RTS. Pour chaque couple de tuiles $(t_A, t_B) \in \mathcal{T}_{\mathcal{M}_1} \times \mathcal{T}_{\mathcal{M}_2}$, on construit la tuile t_{AB} telle que :

- $\Sigma_{t_{AB}} = \Sigma_{\mathcal{M}_1} \cap \Sigma_{\mathcal{M}_2}$ et $\Lambda_{t_{AB}} = \Lambda_{\mathcal{M}_1} \cup \Lambda_{\mathcal{M}_2}$,
- $Q_{t_{AB}} = Q_{t_A} \times Q_{t_B}$,
- $\Delta_{t_{AB}} = \{((p_A, p_B), a, (q_A, q_B)) \mid (p_A, a, q_A) \in \Delta_{t_A} \wedge (p_B, a, q_B) \in \Delta_{t_B}\}$,
- $col_{t_{AB}} = \{((q_A, q_B), \lambda) \mid (q_A, \lambda) \in col_{t_A} \vee (q_B, \lambda) \in col_{t_B}\}$,
- $S_{t_{AB}} = S_{t_A} \times S_{t_B}$,
- $(t_{CD}, f_{CD}) \in F_{t_{AB}}$ si $\begin{cases} & (t_C, f_C) \in F_{t_A} \wedge (t_D, f_D) \in F_{t_B} \\ \wedge & f_{CD} = \{((q_C, q_D), (q_A, q_B)) \mid (q_C, q_A) \in f_C \wedge (q_D, q_B) \in f_D\} \end{cases}$

Le RTS $\mathcal{M}_{1\times 2} = ((\Sigma, \Lambda), \mathcal{T}, t^0, Q^0)$ défini de la manière suivante :

- $\Sigma = \Sigma_{\mathcal{M}_1} \cap \Sigma_{\mathcal{M}_2}$,
- $\Lambda = \Lambda_{\mathcal{M}_1} \cup \Lambda_{\mathcal{M}_2}$,
- $\mathcal{T} = \{t_{AB} \mid t_A \in \mathcal{T}_{\mathcal{M}_1} \wedge t_B \in \mathcal{T}_{\mathcal{M}_2}\}$,
- t^0 est le produit comme défini ci-dessus entre $t^0_{\mathcal{M}_1}$ et $t^0_{\mathcal{M}_2}$,
- $Q^0 = Q^0_{\mathcal{M}_1} \times Q^0_{\mathcal{M}_2}$.

Pour finir, on peut observer que ce calcul peut être effectué pour tout couple de RTS $\mathcal{M}_1$, $\mathcal{M}_2$. Mais, à moins que ceux-ci ne soient pondérés, il n'y a aucune garantie que la sémantique du RTS obtenu soit le produit entre les sémantiques de $\mathcal{M}_1$ et $\mathcal{M}_2$, $[\![\mathcal{M}_{1\times 2}]\!] = [\![\mathcal{M}_1]\!] \times [\![\mathcal{M}_2]\!]$. Plus précisément, la propriété de RTS pondéré assure que deux chemins de $[\![\mathcal{R}]\!]$, avec des étiquettes identiques, atteignent le même niveau, et donc qu'il en est de même dans $[\![\mathcal{R}_2]\!]$. Donc, toute exécution dans la sémantique $[\![\mathcal{M}_{1\times 2}]\!]$ peut être effectuée dans $[\![\mathcal{M}_1]\!] \times [\![\mathcal{M}_2]\!]$ (la réciproque est toujours vraie).

On peut observer qu'un RTS pondéré $\mathcal{M}$ est toujours synchronisé avec lui même, $\mathcal{M} \rhd \mathcal{M}$, ce qui permet d'obtenir le corollaire suivant.

Corollaire 1. *Soit $\mathcal{M}$ un RTS pondéré, alors l'auto-produit $\mathcal{M}^2$ est un RTS pondéré qu'on peut construire de façon effective.*

L'auto-produit sera utilisé par la suite pour vérifier la diagnosticabilité. Le RTS obtenu est de taille quadratique en fonction du nombre de sommets total du RTS initial.

Dans la suite de ce travail, nous aurons besoin de calculer la clôture avant d'effectuer d'autres opérations qui ne sont effectives que sur les RTS pondérés. Pour cela nous introduisons la classe suivante :

Définition 5.10. Un RTS $\mathcal{M}$ est dit *à clôture pondérée* si $clo(\mathcal{M})$ est pondéré. On note cette classe les $\mathcal{C}$wRTS.

Chapitre 6

Diagnosticabilité et opacité pour les systèmes récursifs de tuiles

Dans ce chapitre, nous aborderons différents problèmes liés à l'observation partielle, pour les RTS. Plus précisément, il s'agit des problèmes de diagnosticabilité et d'opacité déjà abordés au chapitre 2 et considérés ici sur des DES infinis modélisés par des RTS. Nous verrons que ces problèmes sont indécidables dans le cas général, mais qu'ils deviennent décidables pour une sous-classe décidable de RTS : les $\mathcal{C}$wRTS(introduite au chapitre 5). Nous considérerons aussi quelques unes des extensions au problème de diagnosticabilité proposées au chapitre 2 dans le contexte des RTS.

6.1 Résultats de diagnosticabilité et d'opacité pour les systèmes récursifs de tuiles

6.1.1 Diagnosticabilité

Pour résoudre le problème de la diagnosticabilité, la méthode utilisée dans le cas fini utilise l'auto-produit. Cette construction n'est pas effective pour les RTS. On peut donc se poser la question de la décidabilité.

Proposition 6.1. *Le problème de la diagnosticabilité pour les RTS est indécidable.*

Ce résultat est une conséquence de [55] puisque la sous-classe des automates à pile visible est une sous-classes de RTS. Nous en redonnons ici la preuve.

Démonstration. Nous allons réduire le problème du vide de l'intersection de deux langages algébriques au problème de la diagnosticabilité des RTS.

Considérons deux langages algébriques déterministes L_1 et L_2 sur un même alphabet Σ et $\# \notin \Sigma$ un nouveau symbole observable. On note alors $\Sigma_\# = \Sigma \cup \{\#\}$. Il est alors possible de construire les RTS $\mathcal{M}_1 = ((\Sigma_\#, \Lambda_{\mathcal{M}_1}), \mathcal{T}_{\mathcal{M}_1}, t^0_{\mathcal{M}_1}, Q^0_{\mathcal{M}_1})$ et $\mathcal{M}_2 = ((\Sigma_\#, \Lambda_{\mathcal{M}_2}), \mathcal{T}_{\mathcal{M}_2}, t^0_{\mathcal{M}_2}, Q^0_{\mathcal{M}_2})$ qui reconnaissent respectivement $L_1 \cdot \# \cdot \Sigma^*$ et $L_2 \cdot \# \cdot \Sigma^*$.

On suppose que $Q_{t^0_{\mathcal{M}_1}}$ et $Q_{t^0_{\mathcal{M}_2}}$ sont disjoints et on construit le RTS $\mathcal{M} = ((\Sigma_{\mathcal{M}}, \Lambda_{\mathcal{M}}), \mathcal{T}_{\mathcal{M}}, t^0_{\mathcal{M}}, Q^0_{\mathcal{M}})$ avec :

- $\Sigma_{\mathcal{M}} = \Sigma_{\#}$,
- $\Lambda_{\mathcal{M}} = \{\lambda\}$,
- $\mathcal{T}_{\mathcal{M}} = (\mathcal{T}_{\mathcal{M}_1} \setminus t^0_{\mathcal{M}_1}) \cup (\mathcal{T}_{\mathcal{M}_2} \setminus t^0_{\mathcal{M}_2}) \cup \{t^0_{\mathcal{M}}\}$, où les sommets accepteurs de $\mathcal{M}_1$ sont colorés par λ,
- $t^0_{\mathcal{M}}$ définie ci-dessous,
- $Q^0_{\mathcal{M}} = Q^0_{\mathcal{M}_1} \cup Q^0_{\mathcal{M}_2}$.

La tuile initiale $t^0_{\mathcal{M}}$ est définie par la fusion de $t^0_{\mathcal{M}_1}$ et $t^0_{\mathcal{M}_2}$ en ce qui concerne, les ensembles de sommets, les transitions et la frontière (le support de l'axiome est vide). Si l'un des sommets accepteurs de $\mathcal{M}_1$ est dans $t^0_{\mathcal{M}_1}$, alors il sera coloré par λ dans $t^0_{\mathcal{M}}$.

Par construction, l'ensemble des sommets colorés par λ est un piège, car le langage $L_1 \cdot \# \cdot \Sigma^*$ est clos par extension. Nous allons montrer que $L_1 \cap L_2 \neq \emptyset$ si, et seulement si, λ n'est pas diagnosticable dans $\mathcal{M}$.

Si λ n'est pas diagnosticable dans $\mathcal{M}$, alors il existe deux exécutions de même observation $u\#w$, telle que l'une aboutit dans λ et l'autre non. Par construction, si une exécution aboutit à un sommet coloré par λ, alors c'est une exécution de $\mathcal{M}_1$. Si elle n'aboutit pas dans λ après un $\#$, alors, c'est aussi une exécution de $\mathcal{M}_2$. Il existe donc une exécution d'observation $u\#w$ dans $\mathcal{M}_1$ et dans $\mathcal{M}_2$, donc $u \in L_1 \cap L_2$.

Réciproquement, si il existe $u \in L_1 \cap L_2$, alors par construction, il existe une exécution dans $\mathcal{M}_1$ qui aboutit en λ d'observation $u\#$ et une exécution dans $\mathcal{M}_2$ qui n'aboutit pas dans λ de même observation.

Par construction, ces deux exécutions pourront perpétuellement être prolongées d'où la non-diagnosticabilité. □

Théorème 4. *Le problème de la diagnosticabilité sur la classe des RTS à clôture pondérée (CwRTS) est décidable.*

Démonstration. Soit $\mathcal{M} \in \mathcal{C}\text{wRTS}$, et λ la couleur que l'on cherche à diagnostiquer. On suppose que les états qui ne sont pas colorés par λ sont colorés par $\overline{\lambda}$. Comme $clo(\mathcal{M})$ est pondéré, en utilisant la proposition 5.7, on peut construire le RTS $\mathcal{M}_2$ tel que $[\![\mathcal{M}_2]\!] = [\![clo(\mathcal{M})]\!]^2$ (auto-produit de $[\![clo(\mathcal{M})]\!]$).

Maintenant, grâce au lemme 1, on sait que la non diagnosticabilité est équivalente à la détection d'un chemin infini dont les états portent à la fois λ et $\overline{\lambda}$ dans $[\![clo(\mathcal{R})]\!]^2$. Ceci qui peut être décidé d'après la proposition 5.3. □

On peut noter que même lorsque le système n'est pas diagnosticable, le RTS $det(clo(\mathcal{M}))$ est un objet intéressant. En effet, $det(clo(\mathcal{M}))$ peut alors être utilisé pour superviser le système $[\![\mathcal{M}]\!]$ et détecter l'appartenance éventuelle à λ. Cependant, si $\mathcal{M}$ n'est pas diagnosticable nous n'avons aucune garantie sur le fait que le verdict **oui** sera renvoyé lorsqu'une exécution traversera λ.

6.1.2 Opacité

Dans le cas fini, on peut réduire le problème de l'universalité au problème de l'opacité. Il est donc intéressant de se poser la même question pour les RTS.

Proposition 6.2. *Le problème de l'opacité est indécidable pour les RTS.*

Démonstration. Pour prouver ce résultat, nous allons réduire le problème de l'inclusion de deux langages algébriques au problème de l'opacité. Considérons deux langages algébriques L_1 et L_2 sur un même alphabet Σ et $\# \notin \Sigma$ un nouveau symbole observable. On note alors $\Sigma_\# = \Sigma \cup \{\#\}$. Il est donc possible de construire les RTS $\mathcal{M}_1 = ((\Sigma_\#, \Lambda_{\mathcal{M}_1}), \mathcal{T}_{\mathcal{M}_1}, t^0_{\mathcal{M}_1}, Q^0_{\mathcal{M}_1})$ et $\mathcal{M}_2 = ((\Sigma_\#, \Lambda_{\mathcal{M}_2}), \mathcal{T}_{\mathcal{M}_2}, t^0_{\mathcal{M}_2}, Q^0_{\mathcal{M}_2})$ qui reconnaissent respectivement $L_1 \cdot \#$ et $L_2 \cdot \#$. Les sommets accepteurs de $\mathcal{M}_1$ portent la couleur λ et tous les autres sommets (de $\mathcal{M}_1$ et ceux de $\mathcal{M}_2$) portent la couleur $\overline{\lambda}$.

On utilise alors la même construction que pour la diagnosticabilité. On suppose que $Q_{t^0_{\mathcal{M}_1}}$ et $Q_{t^0_{\mathcal{M}_2}}$ sont disjoints et on construit le RTS $\mathcal{M} = ((\Sigma_\mathcal{M}, \Lambda_\mathcal{M}), \mathcal{T}_\mathcal{M}, t^0_\mathcal{M}, Q^0_\mathcal{M})$ avec :

- $\Sigma_\mathcal{M} = \Sigma_\#$,
- $\Lambda_\mathcal{M} = \{\lambda, \overline{\lambda}\}$,
- $\mathcal{T}_\mathcal{M} = (\mathcal{T}_{\mathcal{M}_1} \setminus t^0_{\mathcal{M}_1}) \cup (\mathcal{T}_{\mathcal{M}_2} \setminus t^0_{\mathcal{M}_2}) \cup \{t^0_\mathcal{M}\}$, ou les sommets accepteurs de $\mathcal{M}_1$ sont colorés par λ et les autres par $\overline{\lambda}$,
- $t^0_\mathcal{M}$ définie ci-dessous,
- $Q^0_\mathcal{M} = Q^0_{\mathcal{M}_1} \cup Q^0_{\mathcal{M}_2}$.

La tuile initiale $t^0_\mathcal{M}$ est définie par la fusion de $t^0_{\mathcal{M}_1}$ et $t^0_{\mathcal{M}_2}$ en ce qui concerne, les ensembles de sommets, les transitions et la frontière (le support de l'axiome est vide). Si l'un des sommets accepteurs de $\mathcal{M}_1$ est dans $t^0_{\mathcal{M}_1}$, alors il sera coloré par λ dans $t^0_\mathcal{M}$.

Nous allons montrer que $L_1 \subseteq L_2$ si, et seulement si, λ est opaque dans $\mathcal{M}$.

Supposons que λ soit opaque dans $\mathcal{M}$. Soit $u \in L_1$, alors $u\#$ aboutit dans λ. Comme λ est opaque, il existe une exécution de même observation qui aboutit dans $\overline{\lambda}$. Les seules exécutions qui aboutissent dans $\overline{\lambda}$ suite à un $\#$ sont les exécutions de $\mathcal{M}_2$ donc ce qui prouve $u \in \mathcal{L}_2$

Réciproquement, supposons que $L_1 \subseteq L_2$. Tout exécution qui aboutit en λ a une observation de la forme $u\#$ avec $u \in L_1$. Comme $L_1 \subseteq L_2$, il existe une exécution d'observation $u\#$ dans $\mathcal{M}_2$ et donc pas construction, qui aboutit dans $\overline{\lambda}$. Donc λ est opaque dans $\mathcal{M}$. □

Il est possible d'imiter la procédure de décision classique qui résout le problème de l'opacité sur les système finis. Il s'agit de construire un moniteur correct et précis de λ et de vérifier l'accessibilité du verdict **oui**.

Théorème 5. *Le problème de l'opacité sur la classe des $\mathcal{C}$wRTS est décidable.*

Démonstration. Nous donnons ici la procédure de décision. Soit $\mathcal{M} \in \mathcal{C}$wRTS, et λ une couleur. Comme $clo(\mathcal{M})$ est pondéré, nous appliquons la proposition 5.6 afin d'obtenir le RTS $det(clo(\mathcal{M}))$. On en déduit alors le RTS $\mathcal{M}on$ dans lequel un sommet s d'une tuile t est coloré par le verdict :

– **oui** si $(s,\lambda) \in col_t \wedge (s,\overline{\lambda}) \notin col_t$
– **non** si $(s,\lambda) \notin col_t \wedge (s,\overline{\lambda}) \in col_t$
– **equi** si $(s,\lambda) \in col_t \wedge (s,\overline{\lambda}) \in col_t$

Il suffit alors d'appliquer la proposition 5.2 pour vérifier l'accessibilité d'un sommet coloré par **oui**. □

6.2 Extensions des problèmes d'observation partielle.

Nous nous intéressons à présent aux extensions vues au chapitre 2 des problèmes d'observation partielle.

6.2.1 Opacité bornée synchrone

Pour rappel, le problème de l'opacité bornée synchrone d'une couleur λ à l'ordre k (k-s-opacité), consiste à déterminer si on peut trouver une observation $\sigma = \sigma_1\sigma_2$, avec $|\sigma_2|$ plus petite que k, et telle que l'ensemble des états atteints après σ_1 soient colorés par λ.

Pour résoudre ce problème, il est possible d'adapter directement la construction du cas fini (voir § 2.4.1). Étant donné un RTS $\mathcal{M}$ on construit pour chaque tuile t de $\mathcal{T}_\mathcal{M}$ la tuile $t_{\|k}$ de la façon suivante :

– $\Sigma_{t_{\|k}} = \Sigma_\mathcal{M}$,
– $\Lambda_{t_{\|k}} = 2^{[k+1]}$,
– $Q_{t_{\|k}} = Q_t \times 2^{[k+1]}$,
– $((s,K),a,(s',K')) \in \Delta_{t_{\|k}}$
avec $a \in \Sigma^o_\mathcal{M}$, $s \overset{a}{\underset{t}{\rightsquigarrow}} s'$ et $\begin{cases} K' = \{i \leq k \mid i-1 \in K\} \cup \{0\} \text{ si } (s',\lambda) \in col_t \\ K' = \{i \leq k \mid i-1 \in K\} \text{ si } (s,\lambda) \notin col_t \end{cases}$
– $col_{t_{\|k}} = \left\{((s,K),K) \mid s \in Q_\mathcal{M}, K \in 2^{[k+1]}\right\}$,
– $S_{t_{\|k}} = S_t \times 2^{[k+1]}$,
– $(t',f') \in F_{t_{\|k}}$ si $\exists (t',f) \in F_t$ et $f' = \{((s,K),(s',K)) \mid (s,s') \in f\}$.

A partir de l'ensemble de tuiles $\mathcal{T}$ construit précédemment, on peut alors définir le RTS $\mathcal{M}_{\|k} = ((\Sigma_\mathcal{M},\{\lambda\}),\mathcal{T},t^0_{\mathcal{M}_{\|k}},Q^0)$ avec

$$Q^0 = \left\{(s,\emptyset) \mid s \in Q^0_\mathcal{M}, (s,\lambda) \notin col_{t^0_\mathcal{M}}\right\} \cup \left\{(s,\{0\}) \mid s \in Q^0_\mathcal{M}, (s,\lambda) \in col_{t^0_\mathcal{M}}\right\}$$

Proposition 6.3. *La couleur λ est k-s-opaque dans le RTS $\mathcal{M}$ si, et seulement si, toutes les couleurs de $[k+1]$ sont opaques dans $\mathcal{M}_{\|k}$.*

En effet, on peut construire un moniteur $\mathcal{M}on$ à partir de $\mathcal{M}_{\|k}$. Cependant, il n'y a plus une unique couleur, mais $2^{[k+1]}$ couleurs, il faut donc définir précisément à quels états sont associés les couleurs **oui**, **non** et **equi**. Le DES $\mathcal{M}on$ est construit à partir de $\mathcal{M}_{\|k}$ en utilisant

la construction de § 5.6. Le coloriage est alors défini de la manière suivante :

$$\begin{array}{ll}(P, \mathbf{oui}) \in col_{\mathcal{M}on} & \text{si } \exists i \in \mathbb{N}, \forall (p,K) \in P, i \in K \\ (P, \mathbf{non}) \in col_{\mathcal{M}on} & \text{si } \forall (p,K) \in P, K = \emptyset \\ (P, \mathbf{equi}) \in col_{\mathcal{M}on} & \text{sinon}\end{array}$$

6.2.2 Opacité bornée asynchrone

Pour rappel, le problème de l'opacité bornée asynchrone d'une couleur λ à l'ordre k (k-a-opacité), consiste à déterminer si étant donnée une observation, quelque soit l'exécution correspondant à cette observation, un état coloré par λ a été visité dans les k derniers événements.

Pour résoudre ce problème, il est possible d'adapter directement la construction du cas fini (voir § 2.4.2). Étant donné un RTS $\mathcal{M}$ on construit pour chaque tuile t de $\mathcal{T}_{\mathcal{M}}$ la tuile $t_{\tilde{k}}$ de la façon suivante :

- $\Sigma_{t_{\tilde{k}}} = \Sigma_{\mathcal{M}}$,
- $\Lambda_{t_{\tilde{k}}} = \{\lambda\}$,
- $Q_{t_{\tilde{k}}} = Q_t \times \{i \in \mathbb{N} \mid i \leq k+1\}$,
- $((s,i), a, (s',i')) \in \Delta_{t_{\tilde{k}}}$

  si $a \in \Sigma^o_{\mathcal{M}}$, $s \overset{a}{\underset{t}{\rightsquigarrow}} s'$ et $\begin{cases} i' = 1 & \text{si } (s',\lambda) \in col_t \\ \text{si } (s',\lambda) \notin col_t & \text{alors } \begin{cases} \text{si } 1 \leq i \leq k & \text{alors } i+1 = i' \\ \text{sinon} & i' = 0 \end{cases} \end{cases}$
- $col_{t_{\tilde{k}}} = \{((s,i),\lambda) \mid i \geq 1\}$,
- $S_{t_{\tilde{k}}} = S_t \times \{i \in \mathbb{N} \mid i \leq k+1\}$,
- $(t', f') \in F_{t_{\tilde{k}}}$ si $\exists (t', f) \in F_t$ et $f' = \{((s,i),(s',i)) \mid (s,s') \in f\}$.

A partir de l'ensemble de tuiles $\mathcal{T}$ construit précédemment, on peut alors définir le RTS $\mathcal{M}_{\tilde{k}} = ((\Sigma_{\mathcal{M}}, \{\lambda\}), \mathcal{T}, t^0_{\mathcal{M}\tilde{k}}, Q^0)$ avec

$$Q^0 = \left\{(s,0) \mid s \in Q^0_{\mathcal{M}}, (s,\lambda) \notin col_{t^0_{\mathcal{M}}}\right\} \cup \left\{(s,1) \mid s \in Q^0_{\mathcal{M}}, (s,\lambda) \in col_{t^0_{\mathcal{M}}}\right\}$$

Proposition 6.4. *La couleur λ est k-a-opaque dans le RTS $\mathcal{M}$ si, et seulement si, λ est opaque dans $\mathcal{M}_{\tilde{k}}$.*

6.2.3 Opacité infinie synchrone

Pour rappel, le problème de l'opacité infinie synchrone d'une couleur λ, consiste à déterminer si étant donnée une observation, il existe un préfixe quelconque de cette observation, tel que tous les états atteints après ce préfixe sont colorés par λ.

Pour les DES finis, la résolution du problème de l'opacité infinie synchrone nécessite de construire l'estimateur d'état courant et l'estimateur d'état initial. Si l'estimateur d'état courant est similaire à une déterminisation en utilisant la construction par parties, et pourrait donc être adaptée, la construction de l'estimateur d'état initial est plus complexe. Nous n'avons pas réalisé l'adaptation de cette transformation, le problème résulte de [60].

6.2.4 Opacité infinie asynchrone

Pour rappel, le problème de l'opacité infinie asynchrone d'une couleur λ, consiste à déterminer si étant donnée une observation, il existe une exécution compatible qui ne passe jamais par λ.

Pour résoudre ce problème, il est possible d'adapter directement la construction du cas fini (voir § 2.4.4). Étant donné un RTS $\mathcal{M}$ on construit pour chaque tuile t de $\mathcal{T}_\mathcal{M}$ la tuile $t_{\tilde{\omega}}$ de la façon suivante :

- $\Sigma_{t_{\tilde{\omega}}} = \Sigma_\mathcal{M}$,
- $\Lambda_{t_{\tilde{\omega}}} = \{\lambda\}$,
- $Q_{t_{\tilde{\omega}}} = Q_t \times \{0,1\}$,
- $((s,i),a,(s',i')) \in \Delta_{t_{\tilde{\omega}}}$ si $\exists a \in \Sigma^o_\mathcal{M}$, $s \overset{a}{\underset{t}{\rightsquigarrow}} s'$ et $\begin{cases} si (s',\lambda) \in col_t alors & i' = 1 \\ sinon & i' = i \end{cases}$
- $col_{t_{\tilde{\omega}}} = \{((s,1),\lambda) \mid s \in Q_t\}$,
- $S_{t_{\tilde{\omega}}} = S_t \times \{0,1\}$,
- $(t',f') \in F_{t_{\tilde{\omega}}}$ si il existe $(t',f) \in F_t$ et $f' = \{((s,i),(s',i)) \mid (s,s') \in f\}$.

A partir de l'ensemble de tuile $\mathcal{T}$ construit précédemment, on peut alors définir le RTS $\mathcal{M}_{\tilde{\omega}} = ((\Sigma_\mathcal{M},\{\lambda\}),\mathcal{T},t^0_{\mathcal{M}\tilde{\omega}},Q^0)$ avec

$$Q^0 = \left\{(s,0) \mid s \in Q^0_\mathcal{M}, (s,\lambda) \notin col_{t^0_\mathcal{M}}\right\} \cup \left\{(s,1) \mid s \in Q^0_\mathcal{M}, (s,\lambda) \in col_{t^0_\mathcal{M}}\right\}$$

Proposition 6.5. *La couleur λ est k-∞-opaque dans le RTS $\mathcal{M}$ si, et seulement si, λ est opaque dans $\mathcal{M}_{\tilde{\omega}}$.*

6.2.5 Diagnostic d'une fuite d'information

Si une propriété λ n'est pas opaque dans un système $\mathcal{M}$, il peut alors être utile de superviser son exécution afin de déterminer en temps réel si λ est vérifié ou non. Nous avons vu diverses extensions sur les durées de détection de l'opacité, mais comme au § 2.4.5, on peut aussi considérer un attaquant et un défenseur ayant chacun des alphabets distincts.

On peut alors tenter d'adapter la construction utilisée au § 2.4.5 pour les DES finis. On considère donc maintenant que l'on a l'alphabet $\Sigma^a_\mathcal{M} \subseteq \Sigma_\mathcal{M}$ observé par l'attaquant et l'alphabet $\Sigma^d_\mathcal{M} \subseteq \Sigma_\mathcal{M}$ observé par le défenseur, ainsi qu'une couleur λ. Nous cherchons donc à

diagnostiquer la propriété donnée par le langage suivant :

$$L_\lambda = \pi^{-1}_{\Sigma^d_\mathcal{M}} L_{[\![\mathcal{D}(clo(\mathcal{R}))]\!]}(\lambda) \cdot \Sigma^*$$

Plus précisément, si $\mathcal{M}$ est un $\mathcal{C}$wRTS pour $\Sigma^a_\mathcal{M}$, alors on peut modéliser la propriété, que cherche à détecter le défenseur, par un RTS $\mathcal{M}'$, défini de la manière suivante :

1. Faire la clôture de $\mathcal{M}$ pour $\Sigma^a_\mathcal{M}$,
2. Déterminiser $clo(\mathcal{M})$, ce qui donne $det(clo(\mathcal{M}))$, qu'on note D.
3. Transformer λ en un piège : dans chaque tuile de D, ajouter un sommet puits s_p coloré par λ (bouclant sur tous les événements de $\Sigma_\mathcal{M}$) et pour tout sommet s coloré par λ, rediriger tous les arc sortant vers s_p.
4. Enfin, ajouter une auto-boucle étiquetée par les événements non observables du défenseur $(\Sigma_\mathcal{M} \setminus \Sigma^d_\mathcal{M})$ dans tous les sommets.

Cette construction assure que $L_\lambda = L_{[\![\mathcal{M}']\!]}(\lambda)$. Si $\mathcal{M}'$ est lui aussi un $\mathcal{C}$wRTS, on peut alors appliquer la procédure de décision de la diagnosticabilité présentée en § 4.

Si $\mathcal{M}$ est un $\mathcal{C}$wRTS, alors $\mathcal{M}'$ l'est aussi. Cependant, ce n'est pas le résultat dont nous avons besoin. Pour appliquer la procédure de décision de la diagnosticabilité, il suffit d'être en mesure de construire l'auto-produit. Nous n'avons actuellement pas trouvé d'exemple pour lequel la construction complète est impossible quand il est possible de construire $\mathcal{M}'$. Ce problème reste donc ouvert pour les RTS.

Chapitre 7

Test de systèmes modélisés par les RTS

Dans ce chapitre, nous considérons la génération de cas de tests pour les RTS. Nous nous concentrons en particulier sur les RTS pondérés, qui sont déterminisables, et nous proposons un algorithme de génération hors-ligne qui fonctionne par la sélection de comportements spécifiés par un objectif de test décrit par un DES fini. Comme nous l'avons vu au chapitre 3 pour les DES finis, la génération de tests, hors-ligne, conduite par des objectifs, consiste en une succession d'opérations élémentaires : la suspension, puis la clôture, la complétion en sortie et enfin la déterminisation permettent d'obtenir un testeur canonique.

Par la suite, dans le contexte de la sélection par objectif, deux autres opérations sont utilisées : le produit avec l'objectif de test et l'analyse de co-accessibilité.

En outre, comme l'opération de déterminisation n'est pas toujours effective, nous proposons aussi une approche en ligne en décrivant un algorithme à la volée.

7.1 Exemple

Dans la suite de ce chapitre, nous utiliserons un exemple constitué d'un programme récursif. En effet, un des champs d'application possible pour nos travaux est la vérification de conformité de programmes.

Exemple 21. *Le programme qui suit est présenté dans une syntaxe proche de celle de Java. Nous avons choisi ce langage, car il autorise les exceptions. Ces exceptions vont se traduire par l'apparition dans le graphe de sommets de degré (entrant) infini : retour dans l'axiome quel que soit le nombre d'appels récursifs.*

Le comportement du programme est le suivant : l'utilisateur choisit un entier (bloc 1) puis le programme appelle la fonction "comp" avec la valeur choisie. Dans "comp" un booléen est demandé à l'utilisateur pour tenter de poursuivre le calcul. S'il poursuit et que la valeur d'appel est 0*, une exception est levée provoquant le retour au bloc* 3 *dans le programme principal. Sinon, un appel récursif est effectué avec une valeur décrémentée de* 1*.*

Au terme des appels récursifs (en l'absence d'exception) le système exécute le bloc 6 *autant de fois que nécessaire pour atteindre le bloc* 2

```
static void main(String [] args){
    try{
        // Bloc 1 (entrée)
        int k =in.readInt() ;
        comp(k) ;
        // Bloc 2 (sortie)
        System.out.println("Fin") ;
    }
    catch (Exception e){
        // Bloc 3 (sortie)
        System.out.println(e.getMessage()) ;
    }
}
void comp (int x){
    // Bloc 4 (entrée)
    int res =1 ;
    boolean cont=in.readBoolean() ;
    if (cont){
        if (x==0)throw new Exception("Une erreur s'est produite") ;
        // Bloc 5 (interne)
        res=x*comp(x-1) ;
        // Bloc 6 (sortie)
        System.out.println("Un affichage") ;
        return res ;
    }
    else {
        // Bloc 7 (sortie)
        system.out.println("Vous avez arrêté le programme") ;
        return res ;
    }
}
```

L'exemple qui suit propose une modélisation du programme de l'exemple 21 par un RTS.

Exemple 22. *Le programme défini dans l'exemple 21 peut se représenter par le RTS ci-dessous, où chaque tuile représente une fonction.*

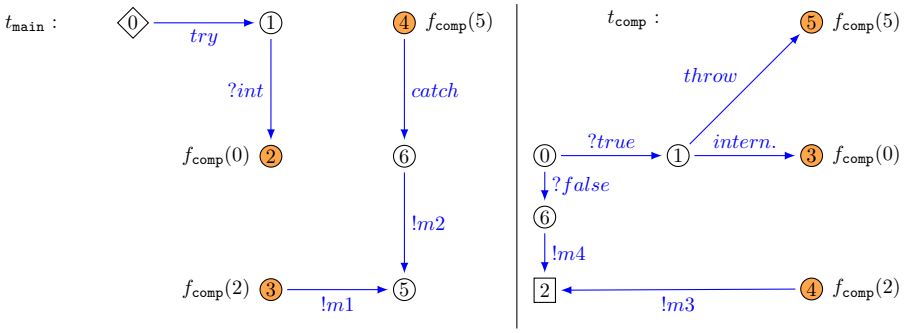


FIGURE 7.1 – Le RTS modélisant le programme de l'exemple 21

Dans le RTS présenté à la figure 7.1, les messages des blocs 2*,* 3*,* 6 *et* 7 *sont représentés respectivement pas* $m1$*,* $m2$*,* $m3$ *et* $m4$*.*

7.2 Construction du testeur canonique

Comme nous l'avons indiqué au chapitre 3, les blocages représentent l'absence de réaction visible dans la spécification. Étant donnée une spécification définie par un RTS $\mathcal{M}$, le problème est de détecter les sommets dans lesquels les blocages sont autorisés afin de construire la spécification suspendue, $Susp(\mathcal{M})$.

Pour les DES à états finis, les blocages vivants proviennent des circuits. Cependant, pour les DES infinis (en l'occurrence définis par des RTS), les blocages vivants peuvent aussi venir de chemins infinis d'événements inobservables traversant un nombre infini d'états, appelés, ici, *chemins divergents*.

7.2.1 Identification des événements admissibles depuis un sommet

Avant d'aborder la suspension, nous allons établir un lemme préalable qui concerne le calcul des événements admissibles depuis un sommet. En effet, nous avons besoin de cette information pour détecter les blocages de sortie.

Lemme 4. *Pour tout RTS* $\mathcal{M}$*, pour toute tuile* t *de* $\mathcal{T}_{\mathcal{M}}$ *et pour tout sommet* $s \in Q_t \setminus S_t$*, il est possible de construire l'ensemble* $Out(s) = \left\{ a \in \Sigma_{\mathcal{M}} \mid \exists q \in Q(s), \exists q' \in Q_{[\![\mathcal{M}]\!]}, q \xrightarrow[[\![\mathcal{M}]\!]]{a} q' \right\}$.

Démonstration. On distingue deux cas : soit s est dans la frontiere de t, soit il ne l'est pas.

Si s n'appartient pas à la frontière de t (par hypothèse, il n'appartient pas au support), alors tous les successeurs de s appartiennent à la tuile t. Il est donc facile de vérifier si il existe une transition étiquetée par un élément de $Out(s)$ ayant pour source s, $Out(s) = \left\{ a \in \Sigma_{\mathcal{M}} \mid \exists s' \in Q_t, s \xrightarrow[t]{a} s' \right\}$.

Si s appartient à la frontière de t, les successeurs peuvent être dans l'extension immédiate de t ou dans plusieurs extensions de t. On construit d'abord l'ensemble des sommets qui sont identifiés avec s lors des extensions de t par récurrence :

Initialement : $Som_0 = \{(t, s)\}$

Récurrence : $Som_{n+1} = Som_n \cup \{(t_2, s_2) \mid \exists (t_1, s_1) \in Som_n, \exists (t_2, f_2) \in F_{t_1} \wedge (s_2, s_1) \in f_2\}$

Cette récurrence converge puisque pour tout $n \in \mathbb{N}$, $Som_n \subseteq \{(t', s') \mid t \in \mathcal{T}_{\mathcal{M}}, s' \in Q_t\}$ qui est un ensemble fini. On note $Som(s)$ la limite de Som_n.

Dans ce deuxième cas, on a $Out(s) = \left\{ a \in \Sigma_{\mathcal{M}} \mid \exists (t', s') \in Som(s), s'' \in Q_{t'}, s' \xrightarrow[t']{a} s'' \right\}$. □

7.2.2 Calcul de la suspension d'un système engendré par un RTS

Comme nous l'avons vu en 5.3, un circuit ou un chemin divergent se traduisent tous les deux par un sommet auto-accessible dans une des tuiles de $\mathcal{M}$. De plus, ces sommets peuvent être détectés et colorés.

Proposition 7.1. *Pour tout RTS $\mathcal{M}$, il est effectif de construire un RTS noté $Susp(\mathcal{M})$ tel que* $\mathsf{Obs}_{[\![\mathsf{Susp}(\mathcal{M})]\!]} = \mathsf{SObs}_{[\![\mathcal{M}]\!]}$.

Démonstration. Soit $\mathcal{M}$ un RTS, on ajoute des auto-boucles étiquetées par δ dans les cas de blocages de sorties ou dans les cas de blocages vivant.

En ce qui concerne les blocages de sortie (c'est-à-dire les blocages complets ou les absences de sortie), nous utilisons le lemme 4 afin d'identifier l'ensemble des sorties d'un sommet s. Si $\Sigma_s \subseteq \Sigma^?_{\mathcal{M}}$, alors on ajoute une auto-boucle d'étiquette δ.

En utilisant la proposition 5.3, on peut colorer par une nouvelle couleur $\lambda \notin \Lambda_{\mathcal{M}}$ les sommets auto-accessibles dans chaque tuile. Il suffit alors d'ajouter une auto-boucle à chacun des sommets colorés par λ. □

7.2.3 Clôture de la sémantique d'un RTS

La seconde opération pour réaliser le testeur canonique est la clôture des transitions étiquetées par τ.

Le calcul de la clôture est effectuée en utilisant la proposition 5.5, à partir de $Susp(\mathcal{M})$. Il est alors possible de construire un RTS $clo(Susp(\mathcal{M}))$, dont la sémantique $[\![clo(Susp(\mathcal{M}))]\!]$ ne possède aucune action non observable et a les mêmes observations suspendues, ce qui se traduit par $\mathsf{SObs}_{[\![\mathsf{Susp}(\mathcal{M})]\!]} = \mathsf{SObs}_{[\![\mathsf{clo}(\mathsf{Susp}(\mathcal{M}))]\!]}$.

7.2.4 Complétion en sortie de la sémantique d'un RTS

Après avoir calculé la clôture de la suspension, $clo(Susp(\mathcal{M}))$, l'étape suivante consiste à compléter $clo(Susp(\mathcal{M}))$ afin de mettre en évidence les observations qui se terminent par une sortie non spécifiée. La spécification complétée et suspendue, $CS(\mathcal{M})$, est calculée depuis

$clo(Susp(\mathcal{M}))$ en envoyant toutes les sorties non-spécifiées vers de nouveaux sommets. Pour chaque sommet s d'une tuile t de $clo(Susp(\mathcal{M}))$, on note $Out(s)$ l'ensemble des étiquettes des transitions de source s obtenues en appliquant le lemme 4.

Étant donné un RTS $\mathcal{M}$, il est possible d'ajouter un nouvel état à chaque tuile afin de toujours pouvoir réaliser l'opération de complétion. Le RTS $CS(\mathcal{M})$ est alors $((\Sigma_{\mathcal{M}}, \Lambda_{\mathcal{M}} \cup \{NS, \overline{NS}\}), \mathcal{T}_{CS(\mathcal{M})}, t^0_{CS(\mathcal{M})}, Q^0_{\mathcal{M}})$, avec NS et $\overline{NS}$ deux nouvelles couleurs n'appartenant pas à $\Lambda_{\mathcal{M}}$ et $\mathcal{T}_{CS(\mathcal{M})}$ l'ensemble des tuiles définies comme suit. Pour chaque tuile t de $\mathcal{T}_{clo(Susp(\mathcal{M}))}$ on construit la tuile $t_{NS} = ((\Sigma_{\mathcal{M}}, \Lambda_{t_{NS}}), Q_{t_{NS}}, \Delta_{t_{NS}}, col_{t_{NS}}, S_t, F_t)$ avec :

- $\Lambda_{t_{NS}} = \Lambda_{\mathcal{M}} \cup \{NS, \overline{NS}\}$,
- $Q_{t_{NS}} = Q_t \cup \{s_{NS}\}$ avec $s_{NS} \notin Q_t$,
- $\Delta_{t_{NS}} = \Delta_t \cup \{(s, a, s_{NS}) \mid a \in \Sigma^{!\delta}_{\mathcal{M}} \wedge a \notin Out(s)\}$,
- $col_{t_{NS}} = col_t \cup \{(s_{NS}, NS)\} \cup \{(s, \overline{NS}) \mid s \in Q_t\}$.

Afin de bien identifier les sommets qui ne sont pas colorés par NS par la suite, les sommets autres que s_{NS} possèdent la couleur $\overline{NS}$.

Par construction, nous obtenons alors :

$$\mathsf{Obs}_{[\![\mathsf{CS}(\mathcal{M})]\!]}(NS) \subseteq \mathsf{SObs}_{[\![\mathcal{M}]\!]}.\Sigma^{!\delta}_{\mathcal{M}} \tag{7.1}$$

$$\mathsf{Obs}_{[\![\mathsf{CS}(\mathcal{M})]\!]}(\overline{NS}) = \mathsf{SObs}_{[\![\mathcal{M}]\!]} \tag{7.2}$$

$$\mathsf{Obs}_{[\![\mathsf{CS}(\mathcal{M})]\!]} = \mathsf{SObs}_{[\![\mathcal{M}]\!]}.\Sigma^{!\delta}_{\mathcal{M}} \cup \mathsf{SObs}_{[\![\mathcal{M}]\!]} \tag{7.3}$$

L'inégalité (7.1) dit simplement que les traces des séquences reconnues par NS sont incluses dans les traces suspendues de $\mathcal{M}$ prolongées par des sorties.

L'inégalité (7.2) est vérifiée car $\mathsf{Obs}_{[\![\mathsf{CS}(\mathcal{M})]\!]}(\overline{NS})$, qui sont les traces des séquences menant ailleurs que dans la couleur NS, sont aussi les traces d'origine de la suspension de $clo(Susp(\mathcal{M}))$, donc $\mathsf{SObs}_{[\![\mathcal{M}]\!]}$.

L'égalité (7.3) est obtenue par union de (7.1) et (7.2). On peut noter qu'il ne s'agit pas d'une union disjointe : une trace peut à la fois être dans $\mathsf{Obs}_{[\![\mathsf{CS}(\mathcal{M})]\!]}(\overline{NS})$ et $\mathsf{Obs}_{[\![\mathsf{CS}(\mathcal{M})]\!]}(NS))$, car $[\![CS(\mathcal{M})]\!]$ n'est pas déterministe en général.

En étendant la définition de $\mathsf{MinFObs}([\![\mathcal{M}]\!])$, nous obtenons

$$\mathsf{MinFObs}([\![\mathcal{M}]\!]) = \mathsf{Obs}_{[\![\mathsf{CS}(\mathcal{M})]\!]}(NS) \setminus \mathsf{Obs}_{[\![\mathsf{CS}(\mathcal{M})]\!]}(\overline{NS}) \tag{7.4}$$

7.2.5 Déterminisation

Quand $CS(\mathcal{M})$ est pondéré, la proposition 5.6 permet de le déterminiser pour obtenir $det(CS(\mathcal{M}))$. Depuis $det(CS(\mathcal{M}))$, on construit alors le testeur canonique comme un nouveau RTS $Can_{\mathcal{M}}$. Chaque tuile $t_{Can} = ((\Sigma_{\mathcal{M}}, \Lambda_{t_{Can}}), Q_t, \Delta_t, col_{t_{Can}}, S_t, F_t)$ de Can_M est obtenue à partir d'une tuile $t = ((\Sigma_{\mathcal{M}}, \Lambda_{\mathcal{M}}), Q_t, \Delta_t, col_t, S_t, F_t)$ de $det(CS(\mathcal{M}))$ avec :

- $\Lambda_{t_{Can}} = \Lambda_{\mathcal{M}} \cup \left\{\acute{E}chec, \overline{\acute{E}chec}\right\}$,

– $col_{t_{Can}} = col_{\mathcal{M}} \left\{ \begin{array}{l} \cup \left\{(s, \acute{E}chec) \mid (s, NS) \in col_t \wedge (s, \overline{NS}) \notin col_t\right\} \\ \cup \left\{(s, \overline{\acute{E}chec}) \mid (s, \overline{NS}) \in col_t\right\} \end{array} \right.$

En utilisant cette construction ainsi que l'égalité (7.4), on peut déduire que

$$\mathsf{Obs}_{[\![Can_{\mathcal{M}}]\!]}(\acute{E}chec) = \mathsf{MinFObs}([\![\mathcal{M}]\!]) \quad (7.5)$$

$$\mathsf{Obs}_{[\![Can_{\mathcal{M}}]\!]}(\overline{\acute{E}chec}) = \mathsf{SObs}_{[\![\mathcal{M}]\!]} \quad (7.6)$$

et donc

$$\mathsf{Obs}_{[\![Can_{\mathcal{M}}]\!]} = \mathsf{SObs}_{[\![\mathcal{M}]\!]} \cup \mathsf{MinFObs}([\![\mathcal{M}]\!]) \quad (7.7)$$

où l'union est, ici, disjointe.

En fait, on a :

$$\mathsf{Obs}_{[\![Can_{\mathcal{M}}]\!]}(\overline{\acute{E}chec}) = \mathsf{Obs}_{[\![\mathsf{CS}(\mathcal{M})]\!]}(\overline{NS}) = \mathsf{SObs}_{[\![\mathcal{M}]\!]} \text{ d'après l'égalité (7.2)}$$

et on a également :

$$\begin{array}{rcl} \mathsf{Obs}_{[\![Can_{\mathcal{M}}]\!]}(\acute{E}chec) & = & \mathsf{Obs}_{[\![\mathsf{CS}(\mathcal{M})]\!]}(NS) \setminus \mathsf{Obs}_{[\![\mathsf{CS}(\mathcal{M})]\!]}(\overline{NS}) \\ & = & \mathsf{MinFObs}([\![\mathcal{M}]\!]) \text{ d'après l'égalité (7.4)} \end{array}$$

De l'égalité (7.5) on peut déduire immédiatement que la suite de test $\mathcal{TS}$ réduite au seul testeur canonique, $\mathcal{TS} = \{Can_{\mathcal{M}}\}$, est correcte et exhaustive (voir § 3.3). La suite $\mathcal{TS}$ est stricte, ce qui est établi par l'égalité suivante :

$$\mathsf{Obs}_{[\![Can_{\mathcal{M}}]\!]} \cap \mathsf{MinFObs}([\![\mathcal{M}]\!]) = \mathsf{Obs}_{[\![Can(\mathcal{R})]\!]}(\acute{E}chec)$$

qui repose sur le fait que (7.7) est une union disjointe ainsi que sur l'égalité (7.6).

Exemple 23. *La figure 7.2, représente le testeur canonique obtenu depuis l'exemple 22. Le sommet F est celui coloré par Échec (représenté par la forme triangulaire).*

7.3 Sélection par objectifs

Le testeur canonique possède quelques propriétés importantes, mais si on veut se concentrer sur un comportement particulier, il est plus efficace de le composer à un objectif de test. Dans notre cadre formel, un objectif de test est défini par un DES fini, et repose sur le fait que le produit entre un RTS et un DES fini est un RTS.

Définition 7.1. Un *objectif de test* pour un alphabet $\Sigma^{o\delta}$ est un DES fini, déterministe et complet sur $\Sigma^{o\delta}$, avec une couleur particulière *Accept*, telle que les états colorés par *Accept* n'ont pas de successeurs autres que eux-même.

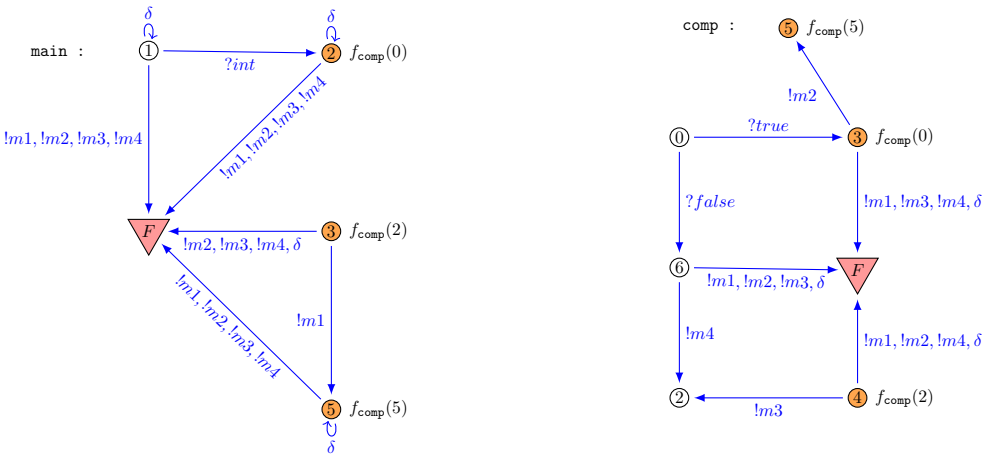


FIGURE 7.2 – Exemple d'un testeur canonique.

Comme nous l'avons vu dans le chapitre 5, le produit entre un DES fini et un RTS est un RTS. Comme $\mathcal{TP}$ est un DES fini, on peut calculer de façon effective le produit noté $S_{\times}TP$ entre $Can_{\mathcal{M}}$ et $\mathcal{TP}$, le coloriage d'un couple de sommets étant l'union des coloriages des deux sommets.

L'étape suivante est la coloration des états co-accessibles depuis *Accept* dans le produit. La proposition 5.2, construit un RTS dans lequel les sommets accessibles depuis une couleur donnée sont colorés par une autre couleur. Il est possible d'adapter cette transformation (en considérant les transitions dans le sens inverse) afin de colorer l'ensemble des états co-accessibles. On note alors $Coreach(\mathcal{M}, \lambda)$ le RTS isomorphe à $\mathcal{M}$ dans lequel les états co-accessibles de λ sont mis en évidence.

Ainsi, la propriété 5.2 permet de construire le RTS, $Coreach(S_{\times}TP, Accept)$, dans lequel les états co-accessibles de *Accept* sont colorés par une couleur particulière #.

Enfin, on construit le RTS $\mathcal{TC}$ à partir de $Coreach(S_{\times}TP, Accept)$. Chaque tuile t' de $\mathcal{TC}$ est obtenue à partir d'une tuile t de $Coreach(S_{\times}TP, Accept)$ avec les couleurs définies de la façon suivante :

- $(s, \acute{E}chec) \in col_{t'}$ si $(s, \acute{E}chec) \in col_t$,
- $(s, Succès) \in col_{t'}$ si $(s, \acute{E}chec) \notin col_t$ et $(s, Accept) \in col_t$,
- $(s, Aucun) \in col_{t'}$ si $(s, \acute{E}chec) \notin col_t$ et $(s, Accept) \notin col_t$ et $(s, \#) \in col_t$,
- $(s, Inconc) \in col_{t'}$ si $(s, \acute{E}chec) \notin col_t$ et $(s, Accept) \notin col_t$ et $(s, \#) \notin col_t$.

On peut noter que, par construction, les couleurs *Échec*, *Succès*, *Aucun* et *Inconc* induisent une partition de l'ensemble des sommets. Les sommets colorés par *Échec* n'ont pas de successeurs autres que eux-mêmes, les successeurs des sommets colorés par *Succès* sont colorés par *Succès* ou *Échec*, et les sommets colorés par *Inconc* ont uniquement des sommets colorés par *Échec* ou *Inconc* comme successeurs.

Afin d'éviter les sommets colorés par *Inconc*, où l'objectif de test ne peut plus être satisfait, les transitions étiquetées par une entrée et menant à un sommet coloré par *Inconc* peuvent être élaguées, de la même manière que celles qui quittent *Inconc*. En conséquence, les exécutions menant aux sommets colorés par *Inconc* finissent nécessairement avec une action sortie.

Exemple 24. *En utilisant le testeur canonique (fig. 7.2) qui provient de l'exemple 22, la figure 7.3 représente le cas de test obtenu avec l'objectif de test $\mathcal{TP}$ et acceptant les traces de $(\Sigma^{o\delta}{}^*.?true.?true(\Sigma^{o\delta})^*.!m1$. Le DES $\mathcal{TP}$ a quatre états q_1, q_2, q_3 et q_4 (le seul état coloré par Accept), avec des boucles sur toutes les actions à l'exception d'une en q_1 et q_3, et des transitions étiquetées respectivement par $?true$ et $!m1$, de q_1 vers q_2 et de q_3 vers q_4. Le produit entre le testeur canonique et $\mathcal{TP}$ est calculé dans chaque tuile ce qui abouti aux deux tuiles représentées à la figure 7.3. Pour une meilleure lisibilité, les seuls sommets représentés sont ceux accessibles depuis $(1, q_1)$ (de la tuile* `main`*) et, dans chaque tuile, les sommets colorés par Échec* (resp., *Inconc) sont fusionnés.*

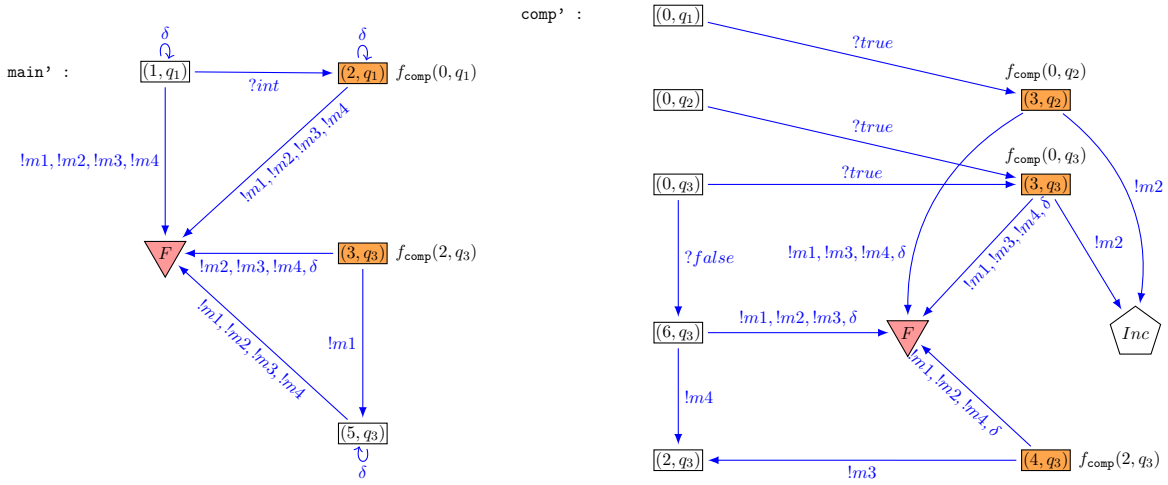


FIGURE 7.3 – Exemple d'un cas de test

7.4 Propriétés des cas de test générés

Nous allons maintenant prouver les propriétés requises sur les cas de tests définies au chapitre 3, qui mettent en relation l'échec d'un cas de test et la non-conformité, ainsi que la précision qui parle des cas de tests qui aboutissent (verdict *Succès*) à la satisfaction de l'objectif de test.

Correction et sévérité

D'après la construction de $S_\times TP = Can_\mathcal{M} \times \mathcal{TP}$, la définition de $\mathsf{Obs}_{\mathcal{TP}}(\acute{E}chec)$, et l'élagage, la sélection par $\mathcal{TP}$ n'ajoute aucun coloriage par *Échec* par rapport au testeur canonique $Can_\mathcal{M}$, donc $\mathsf{Obs}_{[\![\mathcal{TC}]\!]}(\acute{E}chec) = \mathsf{Obs}_{[\![\mathcal{TC}]\!]} \cap \mathsf{Obs}_{[\![Can_\mathcal{M}]\!]}(\acute{E}chec)$.

D'après l'équation 7.5, on peut déduire l'égalité $\mathsf{Obs}_{[\![\mathcal{TC}]\!]}(\acute{E}chec) = \mathsf{Obs}_{[\![\mathcal{TC}]\!]} \cap \mathsf{MinFObs}([\![\mathcal{M}]\!])$ et l'inclusion $\mathsf{Obs}_{[\![\mathcal{TC}]\!]}(\acute{E}chec) \subseteq \mathsf{MinFObs}([\![\mathcal{M}]\!]) \cdot {\Sigma_{\mathcal{M}}}^*$ ce qui établit respectivement la sévérité et la correction.

Exhaustivité

Nous allons prouver que la suite de test $\mathcal{TS}$ composée de tous les tests qui peuvent être engendrés à partir d'objectifs de test arbitraires $\mathcal{TP}$ est exhaustive. Nous avons donc besoin d'établir l'inclusion $\mathsf{MinFObs}([\![\mathcal{M}]\!]) \subseteq \bigcup_{\mathcal{TC} \in \mathcal{TS}} \mathsf{Obs}_{[\![\mathcal{TC}]\!]}(\acute{E}chec) \cdot {\Sigma_{\mathcal{M}}}^*$.

Soit $\sigma' = \sigma.a \in \mathsf{MinFObs}([\![\mathcal{M}]\!]) = \mathsf{Obs}_{[\![Can_{\mathcal{M}}]\!]}(\acute{E}chec)$ une trace minimale et non-conforme de $\mathcal{M}$. Par définition de $\mathsf{MinFObs}([\![\mathcal{M}]\!])$, $\sigma \in \mathsf{SObs}_{[\![\mathcal{M}]\!]}$ et il existe $b \in \Sigma^{!\delta}_{\mathcal{M}}$ tel que $\sigma.b \in \mathsf{SObs}_{[\![\mathcal{M}]\!]}$ (si aucune sortie b appartenant à $\Sigma^{!\delta}_{\mathcal{M}}$ ne prolonge σ, alors un blocage δ devrait être une sortie qui prolonge σ).

Maintenant, il reste à définir un objectif de test $\mathcal{TP}$ tel que $\sigma.b \subseteq \mathsf{Obs}_{\mathcal{TP}}(Accept)$. Soit $\mathcal{TC}$ un cas de test généré depuis $\mathcal{M}$ et $\mathcal{TP}$. Par construction de $\mathcal{TC}$ depuis $\mathcal{M}$ et $\mathcal{TP}$, nous avons à la fois $\sigma.b \in \mathsf{Obs}_{[\![\mathcal{TC}]\!]}$ et $\sigma' \in \mathsf{Obs}_{[\![\mathcal{TC}]\!]}(\acute{E}chec)$. Ce qui permet d'obtenir l'inclusion nécessaire.

Précision

En complément des propriétés ci-dessus, la *précision* est liée aux objectifs de tests. Elle énonce que le verdict *Succès* doit être renvoyé le plus tôt possible, une fois que l'objectif de test a été satisfait. Plus formellement,

Définition 7.2. Un cas de test $\mathcal{TC}$ est *précis* par rapport à une spécification donnée par un DES $\mathcal{M}$ et un objectif de test $\mathcal{TP}$ si $\mathsf{Obs}_{[\![\mathcal{TC}]\!]}(Succès) = \mathsf{SObs}_{\mathcal{M}} \cap \mathsf{Obs}_{[\![\mathcal{TC}]\!]}$.

Il est facile de prouver que les tests générés depuis le RTS $\mathcal{M}$ et l'objectif de test $\mathcal{TP}$ sont précis. Par construction, les états colorés par *Succès* sont ceux colorés par *Accept* dans $\mathcal{TP}$ et qui ne sont pas colorés par *Échec* dans $Can_{\mathcal{M}}$. Donc $\mathsf{Obs}_{[\![\mathcal{TC}]\!]}(Succès) = \mathsf{Obs}_{\mathcal{TP}}(Accept) \cap \mathsf{SObs}_{[\![\mathcal{M}]\!]}$, ce qui (étant donné que $\mathsf{Obs}_{\mathcal{TC}}(Succès) \subseteq \mathsf{Obs}_{[\![\mathcal{TC}]\!]}$) entraine la précision.

7.5 Génération à la volée

Comme tous les modèles caractérisant les langages algébriques, les RTS ne sont pas déterminisables en général. Il n'est donc pas possible de construire le testeur canonique. Afin de contourner cette difficulté, Tretmans [64] a proposé de générer les tests à la volée. Ce processus permet de construire des cas de tests, sans construire le testeur canonique, en utilisant l'algorithme d'exploration à la volée. Cette technique est bien entendu applicable aux RTS.

7.5.1 Génération des cas de test

Étant donné que la seule opération, qui ne soit pas réalisable algorithmiquement, est la déterminisation, les premières étapes de génération restent les mêmes. Nous supposons donc que pour un RTS $\mathcal{M}$, nous avons construit $CS(\mathcal{M})$ en utilisant la construction vue précédemment dans ce chapitre.

Rappelons que ce RTS vérifie les équations 7.1, 7.2, 7.3 et 7.4 :

$$\begin{aligned}
\mathsf{Obs}_{[\![\mathsf{CS}(\mathcal{M})]\!]}(NS) &\subseteq \mathsf{SObs}_{[\![\mathcal{M}]\!]}.\Sigma_{\mathcal{M}}^{!\delta} \quad (7.1)\\
\mathsf{Obs}_{[\![\mathsf{CS}(\mathcal{M})]\!]}(\overline{NS}) &= \mathsf{SObs}_{[\![\mathcal{M}]\!]} \quad (7.2)\\
\mathsf{Obs}_{[\![\mathsf{CS}(\mathcal{M})]\!]} &= \mathsf{SObs}_{[\![\mathcal{M}]\!]}.\Sigma_{\mathcal{M}}^{!\delta} \cup \mathsf{SObs}_{[\![\mathcal{M}]\!]} \quad (7.3)
\end{aligned}$$

et

$$\mathsf{MinFObs}([\![\mathcal{M}]\!]) = \mathsf{Obs}_{[\![\mathsf{CS}(\mathcal{M})]\!]}(NS) \setminus \mathsf{Obs}_{[\![\mathsf{CS}(\mathcal{M})]\!]}(\overline{NS}) \quad (7.4)$$

Produit avec l'objectif de test

L'étape suivante consiste à calculer le produit entre $CS(\mathcal{M})$ et un objectif de test $\mathcal{TP}$ défini par un DES fini ; on obtient alors $S_{\times}TP$. On construit ensuite $Coreach(S_{\times}TP, Accept)$, où les états co-accessibles sont colorés par #.

Exécution des cas de tests

La dernière étape consiste à construire un cas de test à partir de $Coreach(S_{\times}TP, Accept)$. Il s'agit d'un DES qui peut être construit au cours de l'exécution, à partir de la sémantique de $Coreach(S_{\times}TP, Accept)$, d'une façon analogue à celle de l'algorithme présenté par Tretmans [64].

Ces cas de test sont modélisés par des arbres finis, construits à partir de l'alternance des choix entre les entrées et les réponses possibles de l'implémentation. Chaque nœud de l'arbre porte un verdict.

Formellement, un tel arbre est un ensemble de mots sur $(\Sigma_{\mathcal{M}}^{o\delta})^*$ clos par préfixe. Les arbres sont définis par récurrence à l'aide des opérations de concaténation et union suivantes. Étant donné un arbre θ, et pour un symbole $a \in \Sigma_{\mathcal{M}}^{o\delta}$, nous introduisons la notation $a;\theta \stackrel{\mathbf{def}}{=} \{au \mid u \in \theta\}$. De plus, étant donnés deux arbres, θ, θ', l'arbre formé par l'union de ces deux arbres est noté $\theta + \theta'$.

Un cas de test $\mathcal{TC}$ est un arbre construit par une succession d'opérations élémentaires définies ci-dessous, à partir de $Coreach(S_{\times}TP, Accept)$ en utilisant en argument un ensemble d'états PS de $Coreach(S_{\times}TP, Accept)$. L'ensemble PS est initialisé avec les états initiaux de $[\![Coreach(S_{\times}TP, Accept)]\!]$.

De plus, pour un ensemble PS d'états de $[\![\mathcal{M}]\!]$ et un événement $a \in \Sigma_{\mathcal{M}}$, on note PS `after` a l'ensemble $\left\{ q' \in Q_{[\![\mathcal{M}]\!]} \mid \exists q \in PS, q \xrightarrow[[\![\mathcal{M}]\!]]{a} q' \right\}$. Cet ensemble est calculable en utilisant la même technique que le lemme 4.

Choisir de façon non déterministe entre les opérations suivantes :

1. (* Terminer le cas de test *)
   $\theta := \{None\}$
2. (* Donner une entrée à l'implémentation *)
   Choisir un $a \in Out(PS)$ tel que
   $(PS \texttt{ after } a) \cap \left\{ q \mid (q, \#) \in col_{Coreach(S_\times TP, Accept)} \right\} \neq \emptyset$
   $\theta := a; \theta'$
   où θ' est obtenu récursivement en appliquant l'algorithme sur $PS' = PS$ `after` a
3. (* Vérifier la prochaine sortie de l'implémentation *)

$$\theta := \sum_{a \in X_1} a; \acute{E}chec + \sum_{a \in X_2} a; Succès + \sum_{a \in X_3} a; Inconc + \sum_{a \in X_4} a; \theta'$$

avec :
- $X_1 = \left\{ a \in \Sigma^{!\delta}_{\mathcal{M}} \mid \forall q \in PS \texttt{ after } a, (q, NS) \in col_{Coreach(S_\times TP, Accept)} \right\}$,
- $X_2 = \left\{ a \in \Sigma^{!\delta}_{\mathcal{M}} \mid \exists q \in PS \texttt{ after } a, (q, Accept) \in col_{Coreach(S_\times TP, Accept)} \right\} \setminus X_1$,
- $X_4 = \left\{ a \in \Sigma^{!\delta}_{\mathcal{M}} \mid \exists q \in PS \texttt{ after } a, (q, \#) \in col_{Coreach(S_\times TP, Accept)} \right\} \setminus (X_1 \cup X_2)$,
- $X_3 = \Sigma^{!\delta}_{\mathcal{M}} \setminus (X_1 \cup X_2 \cup X_4)$,
- θ' est obtenu en appliquant l'algorithme récursivement avec $PS' = (PS \texttt{ after } a)$.

7.5.2 Propriétés des cas de test générés à la volée

Les propriétés qui sont vérifiées par les cas de test générés hors-ligne, le sont également pour les cas de test générés à la volée. Les preuves sont très similaires.

Correction et sévérité

Par définition de l'ensemble X_1, les observations de $\mathcal{TC}$ qui aboutissent dans un état coloré par *Échec* sont $\mathsf{Obs}_{[\![\mathsf{CS}(\mathcal{M})]\!]} \setminus \mathsf{Obs}_{[\![\mathsf{CS}(\mathcal{M})]\!]}(\overline{NS}) = \mathsf{MinFObs}([\![\mathcal{M}]\!])$. Donc $\mathsf{Obs}_{\mathcal{TC}}(\acute{E}chec) = \mathsf{MinFObs}([\![\mathcal{M}]\!]) \cap \mathsf{Obs}_{\mathcal{TC}}$ ce qui prouve à la fois la correction et la sévérité, de la même manière que pour les cas de test hors-ligne.

Exhaustivité

La preuve d'exhaustivité est similaire à celle présentée en 7.4 et consiste à construire un objectif de test $\mathcal{TP}$ pour chaque trace non-conforme. Le cas de test ainsi généré peut renvoyer le verdict *Échec* après cette trace.

Précision

A partir de la construction de $\mathcal{TC}$, en particulier l'ensemble X_2, on obtient que

$$\mathsf{Obs}_{\mathcal{TC}}(\mathit{Succès}) = \mathsf{Obs}_{\mathsf{CS}(\mathcal{M})}(\mathit{Succès}) \cap \mathsf{Obs}_{\mathcal{TC}}$$

Donc, par définition des couleurs, on a :

$$\mathsf{Obs}_{\mathcal{TC}}(\mathit{Succès}) = \mathsf{Obs}_{\mathsf{CS}(\mathcal{M})}(\overline{NS}) \cap \mathsf{Obs}_{\mathcal{TP}}(\mathit{Accept})) \cap \mathsf{Obs}_{\mathcal{TC}}$$

Ce qui permet de conclure sur la précision : $\mathsf{Obs}_{\mathcal{TC}}(\mathit{Succès}) = \mathsf{SObs}_{\mathcal{M}} \cap \mathsf{Obs}_{\mathcal{TP}}(\mathit{Accept}) \cap \mathsf{Obs}_{\mathcal{TC}}$.

7.5.3 Application de la génération de cas de test à la volée

Nous venons de voir comment produire des cas de test à la volée, sans construire le testeur canonique.

A partir de ces algorithmes, nous calculons un ensemble de chemins symboliques : chacun de ces chemins est mémorisé par une paire formée du sommet et d'un mot sur $\mathcal{T}$ représentant les tuiles traversées.

Exemple 25. *On applique maintenant les constructions précédentes sur le RTS de l'exemple 22 (qui est déterministe, mais sera suffisant pour illustrer le procédé). On définit l'ensemble* $\mathcal{T} = \{m, c\}$ *dont les symboles représentent respectivement les tuiles* `main` *et* `comp`.

Supposons que l'observation courante soit $8, true, true, true$. *Il est alors possible de vérifier que la paire* $(1, mccc)$ *est atteinte. En conséquence, si l'implémentation renvoie une sortie, le test devra échouer. En revanche, si par exemple, le cas de test choisit de façon non-deterministe l'entrée* $faux$, *la paire* $(6, mcccc)$ *est alors atteinte, et la sortie attendue est* `"Vous avez arrêté le programme"` *(le message* `m4`*), ce qui fait passer le système dans l'état* $(2, mcccc)$.

Par la suite, le cas de test est en attente du message `"Un affichage"` *(message* `m3`*), qui à chaque étape retire un* c. *L'état* $(5, m)$ *est alors atteint après la réception de* `"Fin"` *(message* `m1`*). Ceci termine le test.*

Conclusion et perspectives

Synthèse

Dans ce document, nous avons présenté les systèmes récursifs de tuiles. Ce modèle offre une expressivité équivalente à celle des grammaires déterministes de graphes, et légèrement supérieure à celle des automates à pile (en autorisant le degré infini). Nous avons présenté plusieurs transformations permettant de mettre en évidence l'accessibilité de certains sommets (à partir de certains autres), de réaliser la suppression d'événements inobservables, ou encore de déterminiser les membres d'une sous-famille. Ces transformations ont été mises au service de la résolution d'un ensemble de problèmes.

Plus précisément, nous avons considéré trois domaines d'application : le diagnostic, l'opacité ainsi que la génération de tests de conformité. Pour le diagnostic et l'opacité, nous avons établi l'indécidabilité du problème dans toute sa généralité pour les RTS. Nous avons tiré profit de la notion de système pondéré pour établir la décidabilité dans le cas où la clôture d'un système se révèle être pondérée. Nous avons également décrit quelques variantes du problème d'opacité. Nous avons réduit certains de ces cas au problème classique d'opacité sur des modèles obtenus après transformation de la spécification originale du système. Enfin, nous avons abordé le diagnostic (et la diagnosticabilité) de l'opacité. Ceci consiste à observer l'exécution d'un système non opaque, et à lever une alarme lorsque de l'information est révélée.

Pour ce qui est du test, lorsque la clôture de la spécification est un RTS pondéré, nous avons été en mesure de produire un testeur canonique sous forme d'un RTS. Cet objet permet d'employer les mêmes techniques que dans le cas fini pour réaliser des suites de tests ayant les propriétés de correction, de sévérité et d'exhaustivité à la limite. Il permet également de guider les tests vers des objectifs particuliers en modélisant ces derniers par des automates finis. Lorsqu'on utilise de tels objectifs, nous avons montré que les suites de tests engendrées avaient la propriété de précision. Ce travail a, en outre, abordé les conditions d'exécution de RTS, de façon à conduire des tests à la volée. Dans ce contexte, il s'agit de comprendre comment conserver le moins d'information possible tout en offrant une garantie vis-à-vis du test poursuivi. Cette approche peut être utilisée lorsque le testeur canonique n'est pas un RTS ou que sa construction n'est pas réalisable en pratique (par exemple pour des questions de durée).

Perspectives

Nous entrevoyons plusieurs continuations possibles à ce travail. La prolongation la plus naturelle serait certainement d'approfondir nos résultats vis-à-vis des différentes formes booléennes d'opacité, ou du diagnostic d'opacité.

Aspects quantitatifs. Dans la vérification formelle, les éléments quantitatifs prennent une place de plus en plus importante. Pour les modèles récursifs, les travaux de Kucera (en particulier [32]) ont posé de solides bases pour aborder des problèmes plus appliqués. Une partie de ces résultats a été présentée dans le contexte des grammaires déterministes de graphes [13]. En prenant appui sur ces résultats, il serait intéressant de chercher à étendre les travaux de Thorsley et Teneketzis, [63], qui définissent plusieurs notions quantitatives de diagnostic, et les étudient dans le cas fini. De façon analogue il est utilisé en [11] une modélisation probabiliste finie pour quantifier l'opacité d'un système. Il est envisageable d'étendre ces travaux dans les cas infinis, même si les difficultés sont nombreuses. En particulier, la notion de pondération est très fortement liée aux étiquettes du système, et ne révèle rien de la *structure* du système, à savoir un RTS quelconque est isomorphe à un RTS pondéré si on fait abstraction des étiquettes. Ainsi, cet élément ne peut être utilisé pour définir des restrictions pertinentes des RTS probabilistes.

Sur le plan du test, nous pensons qu'il serait possible d'utiliser ces modèles de façon à proposer des éléments de couverture. Néanmoins, ces approches n'ayant pas encore été conduites dans le cas fini, il est certainement plus judicieux de traiter cette situation dans une première phase.

Supervision. Les problèmes de diagnosticabilité ou d'opacité peuvent être interprétés comme deux cas particuliers de la classe des problèmes de supervision. Globalement, cette classe concerne des systèmes qui s'exécutent sous la tutelle d'un superviseur dont le rôle est de fournir des signaux reflétant l'état du système où les propriétés de l'exécution en cours. Souvent, aucune supposition n'est faite sur le système qui s'exécute. Le superviseur a pour rôle d'établir des éléments relatifs à l'observation (voir, par exemple [39, 40]). Dans ce contexte, les RTS disposent d'une expressivité bien supérieure à celle des systèmes finis tout en se prêtant bien à l'implémentation. Dans ce registre, les travaux conduits en [34] semblent particulièrement prometteurs ; ils s'intéressent à une propriété légèrement différente qui est de savoir s'il est possible de traiter des informations en flux (c'est-à-dire sans avoir à stocker l'intégralité de l'exécution).

Modèles. Une extension naturelle des automates à pile est constituée des automates à pile de pile [1, 2, 52]. Pour les graphes des transitions de ces automates, il existe une caractérisation externe qui est fondée sur la *hiérarchie de Caucal* [19]. En revanche, il n'existe pas de caractérisation à l'aide de grammaires de graphes. Il serait certainement intéressant de disposer d'une telle caractérisation. L'obstacle vient du fait qu'une telle représentation serait très probablement d'une grande complexité.

Bibliographie

[1] A. Aho. Indexed grammars – an extension of context-free grammars. *J. ACM*, 15(4) :647–671, 1968.

[2] A. Aho. Nested stack automata. *J. ACM*, 16(3) :383–406, 1969.

[3] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126 :183–235, 1994.

[4] R. Alur, K. Etessami, and M. Yannakakis. Analysis of recursive state machines. In *CAV*, volume 2102 of *LNCS*, pages 207–220, 2001.

[5] R. Alur, L. Fix, and T. A. Henzinger. Event-clock automata : a determinizable class of timed automata. *Theoretical Computer Science*, 211(1 2) :253 – 273, 1999.

[6] R. Alur and P. Madhusudan. Visibly pushdown languages. In *STOC 04*, pages 202–211. ACM, 2004.

[7] J.-M. Autebert. *Théorie des langages et des automates*. Masson, 1994.

[8] J.-M. Autebert, J. Berstel, and L. Boasson. Context-free languages and pushdown automata. In *Handbook of formal languages, Vol. 1*, pages 111–174. Springer-Verlag, 1997.

[9] E. Badouel, M. Bednarczyk, A. Borzyszkowski, B. Caillaud, and P. Darondeau. Concurrent secrets. *Discrete Event Dynamic Systems*, 17 :425–446, 2007.

[10] P. Baldan, Th. Chatain, S. Haar, and B. König. Unfolding-based diagnosis of systems with an evolving topology. *Information and Computation*, 208(10) :1169–1192, October 2010.

[11] B. Bérard, J. Mullins, and M. Sassolas. Quantifying opacity. In *QEST*, pages 263–272. IEEE Computer Society, 2010.

[12] N. Bertrand, T. Jéron, A. Stainer, and M. Krichen. Off-line test selection with test purposes for non-deterministic timed automata. *Logical Methods in Computer Science*, 8(4), 2012.

[13] N. Bertrand and C. Morvan. Probabilistic regular graphs. In *12th International Workshop on Verification of Infinite-State Systems, INFINITY'10*, volume 39 of *EPTCS*, pages 77–90, Singapour, August 2010.

[14] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata : Application to model-checking. In *CONCUR : 8th International Conference on Concurrency Theory*, pages 135–150. LNCS, Springer-Verlag, 1997.

[15] P. Bouyer, F. Chevalier, and D. D'Souza. Fault diagnosis using timed automata. In *FoSSaCS'05)*, volume 3441 of *LNCS*, pages 219–233, Edinburgh, U.K., April 2005.

[16] E. Brinskma, A. Alderen, R. Langerak, J. van de Laagemat, and J. Tretmans. A formal approach to conformance testing. In *Protocol Specification, Testing and Verification (PSTV'90)*, pages 349–363, 1990.

[17] J. Bryans, M. Koutny, L. Mazaré, and P. Y. A. Ryan. Opacity generalised to transition systems. *Int. J. Inf. Sec.*, 7(6) :421–435, 2008.

[18] A. Carayol. Regular sets of higher-order pushdown stacks. In J. Jedrzejowicz and A. Szepietowski, editors, *MFCS*, volume 3618 of *Lecture Notes in Computer Science*, pages 168–179, 2005.

[19] A. Carayol and S. Wöhrle. The caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In Paritosh K. Pandya and Jaikumar Radhakrishnan, editors, FSTTCS 03, volume 2914 of *LNCS*, pages 112–123, 2003.

[20] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Springer, 1999.

[21] F. Cassez. The Dark Side of Timed Opacity. In *Proc. of the 3rd International Conference on Information Security and Assurance (ISA'09)*, volume 5576 of *LNCS*, pages 21–30, Seoul, Korea, June 2009.

[22] D. Caucal. Deterministic graph grammars. In *Texts in logics and games 2*, pages 169–250, 2007.

[23] D. Caucal and S. Hassen. Synchronization of grammars. In E. Hirsch, A. Razborov, A. Semenov, and A. Slissenko, editors, *CSR*, volume 5010 of *LNCS*, pages 110–121. Springer, 2008.

[24] S. Chédor, T Jéron, and C. Morvan. Test generation from recursive tiles systems. In A.D. Brucker and J. Julliand, editors, *6th International Conference on Tests and Proofs*, volume 7305 of *LNCS*, pages 99–114, Prague, Czech Republic, June 2012.

[25] S. Chédor, T Jéron, and C. Morvan. Test generation from recursive tile systems. *Journal on Software Testing, Verification & Reliability*, 2013. Article accepté (mai 2013), version étendue de [24].

[26] S. Chédor, C. Morvan, S. Pinchinat, and H. Marchand. Analysis of partially observed recursive tile systems. In *11th edition of Workshop on Discrete Event Systems*, pages 265–271, Guadalajara, Mexico, September 2012.

[27] S. Chédor, C. Morvan, S. Pinchinat, and H. Marchand. Diagnosis and opacity problems for infinite state systems modeled by recursive tile systems. *Discrete Event Dynamic Systems*, 2013. Article accepté (juin 2013), version étendue de [26].

[28] C. Constant, B. Jeannet, and T. Jéron. Automatic test generation from interprocedural specifications. In *TestCom/FATES'07*, volume 4581 of *LNCS*, pages 41–57, 2007.

[29] B. Courcelle. *Handbook of Theoretical Computer Science*, chapter Graph rewriting : an algebraic and logic approach. Elsevier, 1990.

[30] J. Dubreil, T. Jéron, and H. Marchand. Monitoring confidentiality by diagnosis techniques. In *ECC*, pages 2584–2590, Budapest, Hungary, August 2009.

[31] Joost Engelfriet. Iterated pushdown automata and complexity classes. In *STOC '83 : Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 365–373, New York, NY, USA, 1983. ACM.

[32] J. Esparza, A. Kučera, and R. Mayr. Model checking probabilistic pushdown automata. *Logical Methods in Computer Science*, 2(1), 2006.

[33] R.J. Evey. The theory and application of pushdown store machines. Mathematical linguistic and automatic translation report NSF-10, The computation laboratory of Harvard University, 1963.

[34] E. Filiot, O. Gauwin, P.-A. Reynier, and F. Servais. Streamability of Nested Word Transductions. In Supratik Chakraborty and Amit Kumar, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011)*, volume 13 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 312–324, Dagstuhl, Germany, 2011. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[35] L. Frantzen, J. Tretmans, and T.A.C. Willemse. A symbolic framework for model-based testing. In *FATES 2006 and RV 2006, Revised Selected Papers*, volume 4262 of *LNCS*, pages 40–54, 2006.

[36] C. Hadjicostis and A. Saboori. Notions of security and opacity in discrete event systems. In *Decision and Control, 46th IEEE Conference on*, pages 5056–5061, 2007.

[37] J. Harrison. Formal verification at intel. In *LICS*, pages 45–. IEEE Computer Society, 2003.

[38] S. Hassen. *Synchronisation de grammaires de graphes.* Phd thesis, Université de la Réunion, 2008.

[39] K. Havelund and G. Rosu. Monitoring java programs with java pathexplorer. *Electr. Notes Theor. Comput. Sci.*, 55(2) :200–217, 2001.

[40] K. Havelund and G. Rosu. Efficient monitoring of safety properties. *STTT*, 6(2) :158–173, 2004.

[41] J. E. Hopcroft, R. Motwani, and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation, Second Edition.* Addison-Wesley, 2001.

[42] C. Jard and T. Jéron. TGV : theory, principles and algorithms. *Software Tools for Technology Transfer (STTT)*, 7(4) :297–315, 2005.

[43] B. Jeannet, T. Jéron, and V. Rusu. Model-based test selection for infinite-state reactive systems. In *5th International Symposium on Formal Methods for Components and Objects (FMCO'06), Revised Lectures*, volume 4709 of *LNCS*, pages 47–69, 2006.

[44] T. Jéron, H. Marchand, S. Pinchinat, and M-O. Cordier. Supervision patterns in discrete event systems diagnosis. In *WODES'06*, pages 262–268, July 2006.

[45] T. Jéron. *Contribution à la génération automatique de tests pour les systèmes réactifs.* PhD thesis, Université de Rennes 1, March 2004.

[46] T. Jéron, H. Marchand, S. Pinchinat, and M-O. Cordier. Supervision patterns in discrete event systems diagnosis. In *Workshop on Discrete Event Systems, WODES'06*, pages 262–268, Ann-Arbor (MI, USA), July 2006.

[47] T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-order pushdown trees are easy. In M. Nielsen, editor, *Foundations of Software Science and Computation Structures*, volume 2303 of *LNCS*, pages 205–222, 2002.

[48] K. Kobayashi and K. Hiraishi. Verification of opacity and diagnosability for pushdown systems. *Journal of Applied Mathematics*, 2013.

[49] M. Krichen and S. Tripakis. Conformance testing for real-time systems. *Formal Methods in System Design*, 34(3) :238–304, 2009.

[50] S.-Y. Kuroda. Classes of languages and linear-bounded automata. *Information and Control*, 7(2) :207–223, June 1964.

[51] K.G. Larsen, M. Mikucionis, and B. Nielsen. Online testing of real-time systems using Uppaal. In *Formal Approaches to Software Testing (FATES'04)*, volume 3395 of *LNCS*, pages 79–94, 2005.

[52] A. Maslov. Multilevel stack automata. *Problems of Information Transmission*, 12 :38–43, 1976.

[53] L. Mazaré. Using unification for opacity properties. In *In Proceedings of the 4th Workshop on Issues in the Theory of Security, WITS 04*, pages 165–176, 2004.

[54] R. Milner. *Communication and concurency*. Prentice Hall, 1989.

[55] C. Morvan and S. Pinchinat. Diagnosability of pushdown systems. In *HVC2009, Haifa Verification Conference*, volume 6405 of *LNCS*, pages 21–33, Haifa, Israel, October 2009.

[56] D. Muller and P. Schupp. The theory of ends, pushdown automata, and second-order logic. *Theoretical Computer Science*, 37 :51–75, 1985.

[57] D. Park. Concurrency and automata on infinite sequences. In 5th GIConference, volume 104 of *LNCS*, pages 167–183, 1981.

[58] D. Peled. *Software Reliability Methods*. Springer, 2001.

[59] C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, University of Bonn, 1962.

[60] A. Saboori and C. Hadjicostis. Verification of infinite-step opacity and complexity considerations. *IEEE Trans. Automat. Contr.*, 57(5) :1265–1269, 2012.

[61] M. Sampath, R. Sengupta, S. Lafortune, K. Sinaamohideen, and D. Teneketzis. Diagnosability of discrete event systems. *IEEE Trans. on Automatic Control*, 40(9) :1555–1575, 1995.

[62] M. Sampath, R. Sengupta, S. Lafortune, K. Sinaamohideen, and D. Teneketzis. Failure diagnosis using discrete event models. *IEEE Transactions on Control Systems Technology*, 4(2) :105–124, March 1996.

[63] D. Thorsley and D. Teneketzis. Diagnosability of stochastic discrete-event systems. *IEEE Trans. Automat. Contr.*, 50(4) :476–492, 2005.

[64] J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Software - Concepts and Tools*, 17(3) :103–120, 1996.

[65] J. Tretmans and H. Brinksma. Torx : Automated model-based testing. In A. Hartman and K. Dussa-Ziegler, editors, *First European Conference on Model-Driven Software Engineering*, pages 31–43, December 2003.

[66] S. Tripakis. Fault diagnosis for timed automata. In Werner Damm and Ernst-Rüdiger Olderog, editors, *FTRTFT*, volume 2469 of *LNCS*, pages 205–224. Springer, 2002.

[67] T. Ushio, I. Onishi, and K. Okuda. Fault detection based on petri net models with faulty behaviors. *IEEE Int. Conf. on Systems, Man, and Cybernetics.*, 1 :113–118 vol.1, Oct 1998.

[68] Igor Walukiewicz. Model checking CTL properties of pushdown systems. In Sanjiv Kapoor and Sanjiva Prasad, editors, *FSTTCS*, volume 1974 of *Lecture Notes in Computer Science*, pages 127–138. Springer, 2000.

[69] T-S . Yoo and S. Lafortune. Polynomial-time verification of diagnosability of partially-observed discrete event systems. *IEEE Trans. on Automatic Control*, 47(3) :1491–1495, 2002.

Résumé

L'une des façons les plus efficace de s'assurer du bon fonctionnement d'un système informatique est de les représenter par des modèles mathématiques. De nombreux travaux ont été réalisés en utilisant des automates finis comme modèles, nous essayons ici d'étendre ces travaux à des modèles infinis.

Dans cette thèse, nous nous intéressons à quelques problèmes dans lesquels un système est observé de façon incomplète. Dans ce cas, il est impossible d'accéder à certaines informations internes. La diagnosticabilité d'une propriété donnée consiste à vérifier qu'à l'exécution du système, un observateur sera en mesure de déterminer avec certitude que la propriété est vérifiée par le système. L'opacité consiste, réciproquement, à déterminer qu'un doute existera toujours.

Une autre application concerne la génération de cas de test. Une fois encore, on considère qu'un observateur n'accède qu'à une partie des événements se produisant dans le système (en général les entrées et les sorties). À partir d'une spécification, on produit automatiquement des cas de test, qui ont pour but de détecter des non-conformités (elles même formalisées de façon précise).

Ces trois problèmes ont été étudiés pour des modèles finis. Dans cette thèse, nous étendons leur étude aux modèles récursifs, pour cela nous avons introduit notre propre modèle, les RTS, qui sont une généralisation des automates à pile, et d'autres modèles de la récursivité. Nous adaptons ensuite les techniques utilisées sur des modèles finis, qui servent à résoudre les problèmes qui nous intéressent.

Mots clés : Diagnostic, Opacité, Test de conformité, IOCO, système récursifs, systèmes récursifs de tuiles.

Abstract

An effective way to ensure the proper functioning of a computer system is to represent it by using mathematical models . Many studies have been conducted using finite automata as models, in this thesis we try to extend these works to infinite models.

We focus on three problems in which a system is partially observed. In this case, it is impossible to access certain internal informations. Diagnosability of a given property consist in checking, that, during the execution of the system, an observer will be able to determine with certainty that the property is verified by the system. Conversely, the opacity consists in determining if a doubt will always exist. Another application is the generation of test cases. Once again, we consider that an observer accesses only some events of the system (typically the inputs and outputs) : from a specification, we automatically generate test cases, which are designed to detect non-conformance.

These three problems have been studied for finite models. In this thesis, we extend their study to recursive models. For this purpose, we have introduced a new model, the RTS, which are a generalization of pushdown automata and other models of recursion. In order to solve problems of interest, we adapt the techniques used in finite models.

Keywords : Diagnosis, Opacity, Compliance Testing, IOCO, recursive system, recursive tile systems.