Thèse

présentée à

L'UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

spécialité : Informatique

par

Sébastien SALVA

Étude de la qualité de test des systèmes temporisés

---

soutenue à Reims le 2001 devant la Commission d'Examen formée de :

Président :

Rapporteur :

J. P. Thomesse, LORIA, Université de Nancy

N. Debnath, Winona State University, USA

Examinateurs :

K. Drira, LAAS, Université de Toulouse

J. Zaytoon, Université de Reims

Directeur de thèse :

S. Bloch, Université de Reims

Codirecteur de thèse :

H. Fouchal, Université de Reims

-2001-

"Don't get involved in partial problems, but always take flight to where there is a free view over the whole single great problem, even if this view is still not a clear one."
Ludwig Wittgenstein (1889-1951), Austrian philosopher

Remerciements

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

Il est certainement trop tôt pour que les sociologues qualifient et analysent le monde d'aujourd'hui. Pourtant, un fait incontestable est que nous sommes dans une nouvelle ère, l'ère de l'informatique.

En effet, là où de nombreux processus étaient encore manuels, ne serait-ce qu'une décennie, une automatisation plus ou moins complète est apparue. Les industries fonctionnent presque seules, les avions volent presque sans aide humaine, on visite même l'espace tout en restant sur la planète. En fait, que nous le voulions ou pas, nous vivons au milieu de l'informatique et de l'électronique, et nous en dépendons. Il suffit de couper l'électricité pour le remarquer.

Ainsi, proportionnellement à l'évolution spectaculaire de la rapidité de calcul des ordinateurs, nous construisons des systèmes, et dans un sens plus large des logiciels ou des matériels, de plus en plus complexes. Nous y introduisons diverses notions difficiles telles que le multimedia, la distribution, le parallélisme, le temps... Ils deviennent tellement complexes qu'ils nécessitent parfois plusieurs années pour être conçus. Citons par

exemple les protocoles de télécommunications, les systèmes de contrôle de trafic aérien... La complexité croissante de ces systèmes a comme conséquence directe l'augmentation des risques de défaillances. En effet, celles-ci, communément appelées "bug" font partie courante du monde informatique.

Par exemple, le 15 janvier 1990, une panne téléphonique a bloqué tous les appels longue distance aux États-Unis pendant un après-midi. Une instruction d'une ancienne version du programme de gestion des centraux d'AT&T se trouva placée dans une nouvelle version du programme. Le logiciel de test n'avait pas exploré cette partie du programme, qui n'intervenait qu'accidentellement en cas d'arrêt d'urgence d'un central. Le résultat de l'erreur fut que le programme avait fonctionné correctement jusqu'à ce qu'intervienne l'arrêt accidentel d'un central. Celui-ci, à cause de l'erreur, se mit à avertir aussitôt tous ses centraux voisins, pour leur dire d'appliquer aussi la procédure d'arrêt d'urgence. Comme ces autres centraux avaient tous la nouvelle version du programme, ils se mirent à avertir les centraux proches. Et tout le système s'écroula en quelques minutes. Personne ne s'était rendu compte de cette erreur, puisqu'on n'utilisait jamais la procédure d'urgence et, typiquement, ce programme s'est effondré brutalement.

Mais ne soyons pas surpris et ne blâmons personne ! Car, comme le dit le fameux proverbe, l'erreur est humaine. Ainsi, il est difficile voire impossible d'obtenir des systèmes fiables, c'est-à-dire ne contenant pas d'erreurs, sans passer par une phase de vérification de leurs fonctionnements. Ainsi, des méthodes permettant de rechercher des défaillances potentielles ont été mises au point.

La première et la plus répandue est sans aucun doute le Beta-Test. Celle-ci est la plus naturelle des solutions car elle consiste à donner un produit beta à plusieurs utilisateurs qui vérifient sa fiabilité. Mais cette solution est loin d'être sans défaut. Premièrement, elle est très coûteuse, car le nombre d'utilisateurs et le nombre de beta produits sont lourds à gérer. L'handicap essentiel de cette méthode réside dans la détection des erreurs. En effet, rien ne permet d'affirmer que les utilisateurs feront appel à tous les services du système ou de l'application. En fait, on remarque que l'ensemble des services proposés sont rarement, voire jamais, entièrement testés. Certains le sont aussi beaucoup plus que d'autres. Ceci a pour conséquence qu'une grande source d'erreurs potentielles risque de ne pas être découverte.

Ainsi, une seconde solution, basée sur le cycle de vie du logiciel, a été proposée. Celle-ci, plus coûteuse et plus complexe, est aussi plus fiable. Elle intéresse d'ailleurs de plus en plus les industriels. Ces techniques, normalisées et composées de nombreuses règles, sont généralement réunies en un cycle qui débute par l'élaboration d'un cahier des charges spécifiant le produit et qui se termine par la génération d'un produit fini validé. Ce cycle est appelé **Cycle de vie ou de développement du logiciel**. Nous en détaillons son fonctionnement dans ce qui suit.

1.1 Cycle de vie du logiciel

Le cycle de vie de logiciel est un modèle de développement composé de méthodes de conception, d'analyse de codage et de test, permettant de produire une implantation conforme à un cahier des charges initial réunissant des informations informelles sur le fonctionnement voulu du produit fini. Les différentes phases du cycle de vie sont généralement séquentielles : chacune exploite le résultat délivré par la phase précédente, le traite et l'offre à la phase suivante. Ainsi, ce cycle peut être vu comme un processus de raffinement permettant de passer d'une spécification décrite avec un haut niveau d'abstraction à un produit final conforme à ce que le concepteur attendait. Lors de chaque phase, divers critères du produit, tels que la complexité, la performance, la robustesse, la facilité à tester, peuvent être analysés avant l'implantation du produit, afin de juger si ces critères sont suffisants ou non. Ainsi, à tout moment, il est possible de connaître la qualité du système.

Les différentes phases du cycle de vie d'un logiciel sont illustrées en Figure 1.1. Elles sont détaillées comme suit :

- **Phase d'analyse des besoins :** elle permet l'analyse des besoins qui sont ensuite réunis dans un cahier des charges. La description de ces besoins est généralement exprimée en langage naturel.

- **Phase de conception :** le langage naturel reste ambigu et difficilement analysable. Cette phase vise alors à produire une spécification détaillée et parfois formelle à l'aide de langages de description formelle (FDT Formal Description Technique). Grâce à une suite de raffinements, la spécification décrit en détail les structures de données, l'architecture du système et l'interface liant ces différents modules.

- **Phase de Validation de la spécification :** cette opération, aussi appelée Vérification, a pour but de vérifier la cohérence de la spécification. Plus précisément, elle assure une correction et vérifie certaines propriétés comme l'existence des services proposés, l'admissibilité de ces services et leurs bons fonctionnements. Un certain nombre d'erreurs dues à la conception de la spécification peuvent ainsi être détectées.

- **Phase d'implantation :** la spécification détaillée et vérifiée est, dans cette étape, traduite sous forme de langage de programmation exécutable. Cette transformation tient compte de plusieurs paramètres tels que l'environnement du système, les interactions avec celui-ci...

- **Phase de Validation de l'implantation :** cette étape aussi appelée la phase de Test, permet de vérifier que l'implantation satisfait un certain nombre de propriétés de la spécification. Elle permet donc la détection de défectuosités du fonctionnement de l'implantation du système. Ces tests peuvent être divers : ils peuvent s'attacher aux performances, à la robustesse, ainsi qu'à la conformité de l'implantation par rapport à sa spécification. C'est dans le cadre de ce dernier type de test que s'inscrit cette thèse.

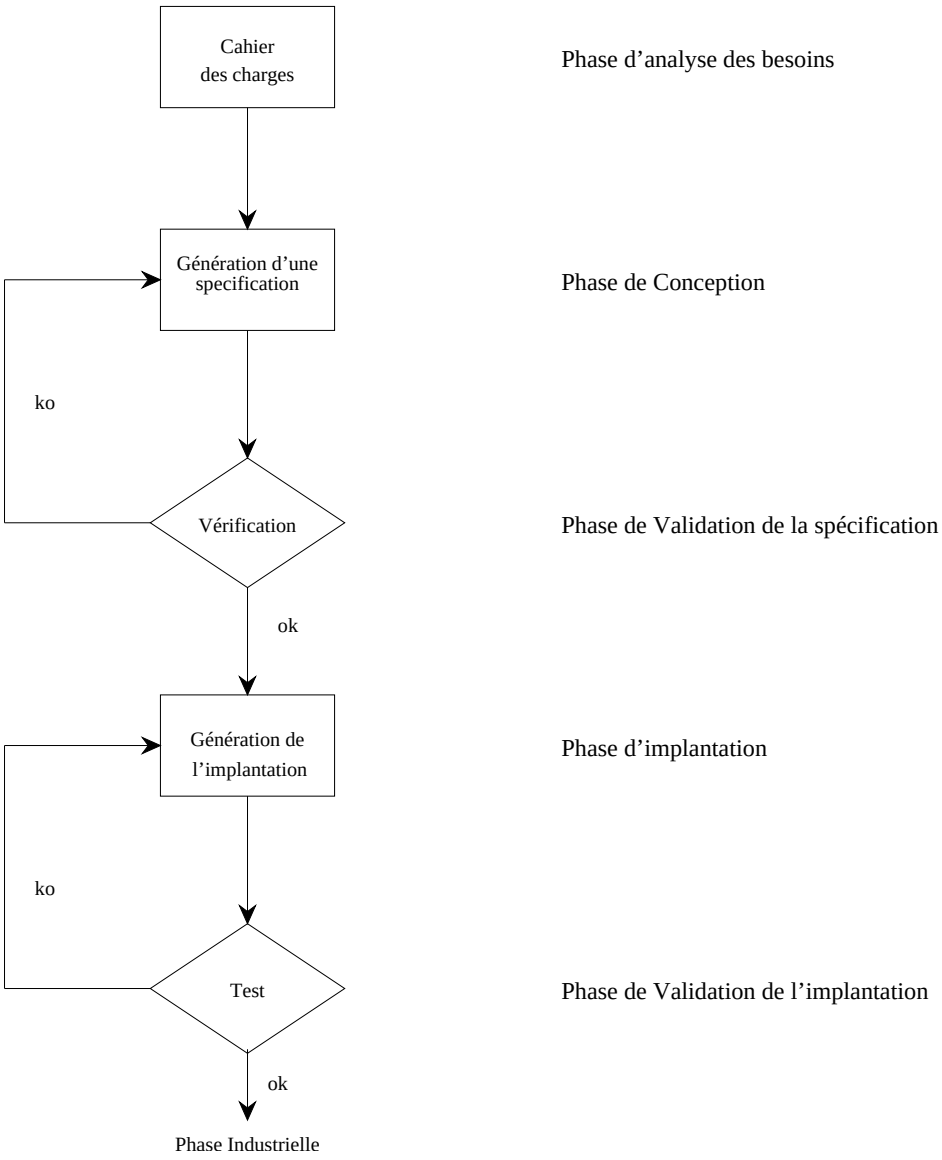


FIG. 1.1 – Cycle de vie du logiciel

1.2 Le test

Le cycle de vie est loin d'être complètement automatisé et se compose donc de nombreuses activités manuelles. Ces dernières entraînent inévitablement un risque d'erreurs. C'est pour cette raison que les tests restent incontournables. Ceux-ci visent à stimuler le système dans le but d'observer les éventuelles erreurs de dysfonctionnement, de les localiser puis de les corriger. Ces erreurs peuvent être locales ou affecter plusieurs modules du système. Elles peuvent être d'origine humaine ou physique. Elles peuvent aussi être liées à d'autres erreurs.

Par conséquent, différents types de test ont été définis visant à traiter des aspects plus ou moins indépendants. Par exemple, citons celui qui offre tout notre attention dans cette thèse, le test de conformité du logiciel qui vise à montrer la présence, et non pas l'absence, d'erreurs. Ces types de test peuvent se regrouper en deux catégories :

- **les tests en Boite Blanche :** Ceux-ci, aussi appelés tests structurels, sont effectués sur des systèmes dont la structure interne est connue et observable. Citons par exemple le test de boucle qui vise à valider toutes les boucles d'un programme en détectant les erreurs d'initialisation, d'indexage et de conditions d'arrêt.

- **les tests en Boite Noire :** Comme son nom l'indique, cette catégorie rassemble les tests, aussi appelés tests fonctionnels, qui sont appliqués sur des systèmes où la structure interne est inconnue. Seules les interfaces reliant le système avec l'environnement extérieur sont connues.

Citons quelques exemples de tests en Boite Noire.

1.2.1 Quelques types de test

Tests de Performance

Ces tests ont pour but de vérifier un certain nombre de paramètres liés aux performances du système, tels que le débit, le temps de réponse, le nombre de connections. D'une manière simpliste, la performance est mesurée en soumettant le système à certaines conditions d'utilisation et en observant ses réactions. Ces conditions d'utilisation peuvent être minimales, moyennes ou extrêmes. Ce genre de test est souvent appliqué aux systèmes temps réel.

Tests de Robustesse

Ces tests consistent à vérifier la réaction d'un système dans des conditions d'utilisations extrêmes ou bien son exécution dans un environnement dit hostile. Ces conditions ne sont généralement pas prévues dans la spécification, cette dernière référençant des conditions de fonctionnement normales. Ces tests permettent ainsi de vérifier si d'autres erreurs existent telles que des fraudes ou des erreurs d'utilisation du système.

Tests d'Interopérabilité

Ces tests vérifient si le système développé interagit d'une façon correcte avec d'autres systèmes extérieurs en observant les fonctionnements des différents systèmes et des com-

munications engendrées. Ces tests permettent de vérifier un service plus global fourni aux utilisateurs.

Tests de l'usager

Ces tests sont effectués au niveau de l'usager qui manipule le système pour vérifier si ce dernier répond bien à ses besoins. Ces tests, communément appelés beta-tests, permettent de vérifier les services les plus demandés.

Tests de Conformité

Ces tests occupent une place prépondérante dans le domaine de l'ingénierie des protocoles. Ils ont de ce fait été normalisés ISO dans [ISO91a]. Ceux-ci consistent à vérifier que le comportement de l'implantation est conforme à la spécification de référence. Ces tests s'effectuent par l'intermédiaire de deux étapes.

- **Recherche de propriétés significatives de la spécification :** Elle consiste en la génération de suites de propriétés, qui sont autorisées par la spécification, et qui devront ensuite être observées à partir de l'implantation. Les éléments constitutifs de cette suite sont appelés des cas de test ou **séquences de test**. Ces dernières sont obtenues par l'application sur la spécification de méthodes de génération de séquences de test.

- **Application des séquences de test :** cette étape est réellement l'étape de test. Elle consiste à stimuler l'implantation par les séquences de test, dans le but de vérifier si leurs réactions sont identiques à celles de la spécification. En appliquant ces séquences de test, un **Verdict** est alors énoncé.

Définition 1.2.1 (Verdict du test) *L'application de séquences de test sur l'implantation retourne trois types de verdict :*

- ***PASS*** *si l'application des séquences de test permet d'observer sur l'implantation des réactions identiques à celles de la spécification.*
- ***INCONCLUANT*** *si l'application des séquences de test ne permet pas de conclure sur la conformité de l'implantation.*
- ***FAIL*** *autrement. Ce dernier verdict montre la présence d'erreurs dans l'implantation.*

Ce type de test tend aujourd'hui à être de plus en plus étudié. En effet, à cause de la complexité croissante des systèmes et de l'ajout de nouvelles notions telles que le temps, il est naturel de s'intéresser à des méthodes de test plus formelles. Cette formalisation induit de plus une automatisation possible de la tâche, ce qui augmente encore son intérêt par les industriels.

1.3 Les problèmes ouverts

Le coût de la validation (Vérification et Test) d'un système dépasse souvent 70% du coût total de son développement, et ne cesse de croître de par la complexité de plus en plus accrue de ces systèmes et l'ajout de notions difficiles à manipuler telles que le temps.

Il est par conséquent nécessaire de proposer des méthodes permettant de réduire ce coût tout en obtenant des systèmes pouvant être complètement testés. Pour ce faire, il est nécessaire d'évaluer ce coût et d'évaluer les parties du système qui pourront être testées. Cette évaluation est appelée la **Qualité de test** ou **Testabilité** du système.

Les systèmes non temporisés ont été les premiers étudiés [VM95, oSET90, Fre91, Kar97, DAdSS01] dans le but de déterminer une façon de mesurer cette qualité de test. Ainsi, ces travaux ont montré que certaines propriétés de la spécification, après avoir été implantées, impliquaient un test coûteux voire impossible de l'implantation. Le fait de répertorier ces propriétés permet d'évaluer la qualité de test de ces systèmes, de les améliorer pour réduire le coût du test et d'obtenir un système qui puisse être mieux testé.

En ce qui concerne les systèmes temporisés, plusieurs méthodes ont été proposées pour les tester [SVD01, AKA00, ENDKE98, Ost92, PLC98, FPS00], sans que celles-ci ne prennent en compte cette notion de qualité de test. Il n'est donc pas surprenant que différents problèmes soient apparus, sans qu'ils aient été au préalable soulevés : par exemple, des parties de l'implantation ne peuvent pas toujours être testées, le test peut être anormalement long voire bloqué.

Cette thèse a donc pour but d'étudier principalement ces questions :

- *Peut-on définir et ensuite évaluer la qualité de test des systèmes temporisés ? Peut-on améliorer cette qualité de test ?* La notion de temps est une notion abstraite et qui ne peut être contrôlée dans un système. Or pouvoir définir la qualité de test des systèmes temporisés permettrait de connaître la cause des problèmes soulevés ci-dessus. Ensuite, elle permettrait éventuellement de les résoudre. Mais sachant qu'un système temporisé est plus complexe qu'un système où la notion de temps n'apparaît pas, il peut être plus difficile de trouver une solution permettant d'évaluer la qualité du test et de modifier le système afin de l'améliorer.

- *Peut-on réduire le coût du test et améliorer la détection des erreurs, non pas cette fois en modifiant le système, mais en améliorant les méthodes de test ?* En effet, il peut être aussi nécessaire de devoir modifier ou proposer d'autres méthodes de test pour d'une part, prendre en compte la qualité de test pendant la génération des séquences de test et d'autre part, pour réduire le coût global du test.

1.4 Contribution et choix apportés par cette thèse

Cette thèse a pour but d'apporter des réponses aux questions précédentes.
Ainsi, les chapitres 2 et 3 rappellent dans un premier temps comment modéliser et

tester des systèmes non temporisés et ensuite des systèmes temporisés. Ceux-ci montrent les avantages et les inconvénients de plusieurs modèles et méthodes de test.

Ensuite, le chapitre 4 a pour but de montrer l'importance et l'état de l'art de la qualité de test des systèmes non temporisés, puis de définir et d'évaluer cette qualité sur les systèmes temporisés, modélisés par des automates temporisés et des graphes des régions. Les automates temporisés et les graphes des régions sont au coeur de nombreuses méthodes de vérification et de test et sont, sans aucun doute, les modèles les plus utilisés actuellement. Après avoir défini cette qualité de test, nous montrons comment modifier les systèmes en vue d'obtenir une meilleure qualité, et donc en vue de réduire les coûts de test et d'améliorer la détection des erreurs. De façon plus précise :

1. nous rappelons l'état de l'art de la qualité de test des systèmes non temporisés. Nous apportons un choix sur les critères qui pourront nous être utiles pour en partie évaluer la qualité de test des systèmes temporisés, par rapport à leurs propriétés comportementales.
2. ensuite, nous analysons les propriétés des automates temporisés puis des graphes des régions. Pour chacun de ceux-ci nous trouvons quatre critères prédominants influençant la qualité de test. Pour évaluer la qualité d'une manière globale, nous mesurons dans un premier temps l'impact de chacun de ces critères par des facteurs que nous appelons Degré de qualité.
3. nous montrons ensuite comment améliorer ces degrés et donc la qualité de test des automates temporisés et des graphes des régions.

Une seconde partie est composée des chapitres 5 et 6. Celle-ci a pour but de montrer comment réduire le coût du test, non pas en modifiant la spécification mais en proposant de nouvelles méthodes de test. Ainsi :

1. le chapitre 5 décrit deux méthodes de génération de séquences de test basées sur les automates temporisés. La première est une méthode orientée Objectif de test. Elle permet de tester une suite de propriétés comportementales et temporelles quelconque, données par le concepteur, sur l'implantation. Cette méthode prend en compte la qualité de test et permet de vérifier n'importe quelle propriété sur l'implantation.

2. La seconde méthode est une approche basée sur la caractérisation d'états de graphe des régions. L'originalité de cette méthode est de caractériser (sous certaines hypothèses) des états temporisés sans apporter aucune modification de la spécification, en tirant partie des propriétés comportementales mais aussi temporelles de cette dernière. Son utilisation permet donc de réduire les coûts du test, tout en obtenant une couverture de fautes maximale.

3. Pour chacune de ces méthodes, nous montrons aussi comment exécuter les séquences de test en prenant en compte les propriétés temporelles. Une dernière partie du chapitre illustre le passage des séquences de test en TTCN [ISO91a] qui est un

langage normalisé modélisant, dans un faible niveau d'abstraction, les séquences de test. Cette transformation offre un intérêt dans le sens où TTCN est un langage reconnu et utilisé en tant que tel par les industriels.

4. Le chapitre 6 apporte une solution, encore embryonnaire au problème de la génération du graphe des régions, qui peut engendrer un nombre d'états exponentiels. En effet, les méthodes de test et de vérification (ainsi que celles décrites dans le chapitre 5), basées sur le modèle du graphe des régions, construisent un premier graphe des régions à partir de la spécification puis le minimisent afin de réduire, parfois considérablement, la représentation du système. Ainsi, l'application de méthodes de vérification et de test demande un coût réduit. Mais la construction de ce premier graphe intermédiaire, d'une part augmente considérablement le coût total du test, et d'autre part réduit l'engouement de ces méthodologies à cause de l'exponentialité de ses états.
   La méthode que nous décrivons, permet de transformer la spécification en graphe des régions, tout en évitant la génération d'un premier graphe pouvant contenir ce nombre exponentiel d'états.

Ensuite, nous appliquons les résultats de ces travaux sur une partie d'un protocole réel, le MAP-DSM. C'est-à-dire que nous évaluons sa qualité de test, une première fois sur un automate temporisé le modélisant et une seconde fois sur un graphe des régions. Les résultats obtenus sont ensuite commentés pour montrer les améliorations à apporter aux deux spécifications. Puis, nous appliquons les deux méthodes présentées au chapitre 5 et décrivons comment sont générées les séquences de test.

Enfin, nous concluons sur les travaux futurs et leurs perspectives .

Chapitre 2

Modélisation et test des systèmes non temporisés

Nous rappelons dans ce chapitre, une formalisation de la modélisation des systèmes non temporisés et de leurs tests de conformité.

Une première partie est donc vouée à la présentation de langages formels qui permettent la représentation non ambiguë de spécifications non temporisées. Dans un premier temps, seront exposés quelques langages ayant un niveau d'abstraction élevé, tels que LOTOS, ESTELLE, LDS, et les Réseaux de Petri. Puis, nous décrirons des langages de plus faible niveau d'abstraction, qui sont habituellement générés depuis l'un des précédents langages.

Une deuxième partie présente quelques méthodologies de génération de séquences de test basées sur certains de ces langages formels. Ces méthodes sont l'essence des travaux

sur le test temporisé et nous seront utiles aussi à de nombreuses reprises au cours de cette thèse. Quelques outils de génération de séquences de test, utilisant l'une de ces méthodologies, seront ensuite décrits.

2.1 Modélisation des systèmes non temporisés

Diverses modélisations sont proposées pour traduire un cahier des charges, représentant une spécification, en une description plus détaillée et plus formelle. Le but de cette traduction en un langage de description, aussi appelé Technique de Description Formelle (TDF), est de lever toute ambiguïté sur la spécification, et permet aussi d'y appliquer plus facilement des analyses, des algorithmes et notamment des méthodes de test.

Pour décrire avec précision et forme une spécification, un tel langage doit être construit sur des règles strictes, puis doit être normalisé pour garder une unicité de traduction. Les langages de description formelle actuellement normalisés et définis peuvent être placés dans deux catégories :

- La première rassemble des langages que nous qualifions de haut niveau, car ceux-ci modélisent une spécification avec un degré d'abstraction élevé. Certains de ces langages sont les logiques temporelles, LOTOS, ESTELLE, LDS et les Réseaux de Petri.

- La deuxième rassemble des langages que nous qualifions de bas niveau. Ils sont généralement obtenus automatiquement depuis un des langages précédents, et décrivent de façon plus détaillée mais plus lourde la même spécification. Étant d'un faible degré d'abstraction, ils ne peuvent décrire des notions telles que le processus ou les priorités. Ce sont ces modèles sémantiques qui sont généralement utilisés par les méthodes de test. Parmi les plus connues, citons la Machine à États Finis et les Systèmes de Transitions Étiquetées.

Les deux sections suivantes ont pour but de présenter et définir certains de ces langages qui sont les plus employés dans le domaine des protocoles de télécommunications.

2.1.1 Langages de description formelle de haut niveau

Les Logiques temporelles

Bien que cela semble paradoxal vu leurs noms, les logiques temporelles sont des formalismes exprimant, non pas des aspects temporels, mais l'ordonnancement d'actions.

Ainsi, de nombreuses logiques temporelles ont été proposées dans le but de modéliser des propriétés de protocoles et traiter des problèmes aussi divers que variés. Ce nombre peut s'expliquer par le fait que les logiques temporelles sont des extensions de logiques propositionnelles ou des logiques du premier ordre, auxquelles ont été ajoutés des opérateurs modaux tels que : *toujours, une fois*... Ainsi, les logiques temporelles sont généralement répertoriées selon ces critères :

- Une logique temporelle est propositionnelle ou du premier ordre.

– Une logique temporelle est soit linéaire soit arborescente : une logique dite linéaire est telle qu'une propriété se vérifie sur une branche ($\Box(p \rightarrow \Diamond\ q)$), tandis qu'une propriété d'une logique dite arborescente se vérifie sur plusieurs branches ($\forall\Box(p \rightarrow \exists\Diamond\ q)$).

Citons comme logique temporelle classique, exprimant l'ordonnancement des actions, CTL (Computation Tree Logic) [CE81].

Certaines logiques, considèrent cependant quelques aspects temporels. Nous pouvons citer : MTL (Metric Temporal Logic) [AH90] qui est une logique propositionnelle linéaire et RTTL (Real-Time Temporal Logic) [Ost92] qui est du premier ordre aussi linéaire. Une logique temporelle non linéaire, vivement utilisée en model-checking (vérification) est TCTL [ACD90, Yov93].

LOTOS

Le langage LOTOS (Language Of Temporal Ordering of Specification), a été conçu entre 1980 et 1988, et a été normalisé ISO 8807 ([ISO87]) (présentation du langage dans [BB88]).

LOTOS est adéquat pour modéliser des architectures et systèmes distribués et parallèles. Une spécification, décrite en LOTOS, est composée de deux parties orthogonales, qui sont :

– la **Structure de contrôle**, inspirée des recherches sur les algèbres de processus, dont les plus connues sont CCS (Calculus of Communicating Systems) [Mil80] et CSP (Communicating Sequential Processes) [Hoa85].
– la **Structure de donnée**, inspirée des recherches sur les types abstraits algébriques ([EFH83, EM85]).

Un processus de base d'une spécification est modélisé par un ensemble d'actions (appelées PORTE) qui symbolisent ce qui peut être transmis depuis le système et ce qui peut être reçu. Par la suite, il est possible de spécifier des comportements plus complexes en combinant ces actions élémentaires avec divers opérateurs, comme par exemple la Composition Parallèle qui permet d'exécuter parallèlement des comportements avec synchronisation sur un ensemble de portes.

Une grande différence avec d'autres modèles, est la communication entre deux processus de la spécification qui s'effectue au moyen de points de rendez-vous symétriques qui permettent l'échange de données.

Bien qu'aux premiers abords, la syntaxe de LOTOS semble complexe, LOTOS propose une sémantique forte et simplifiée, avec peu d'opérateurs. LOTOS permet donc de décrire un système d'une manière plus concise et allégée, d'où sa puissance et ses avantages.

LDS (Specification and Description Language)

LDS (ou SDL pour Specification and Description Language) est un langage, standardisé par le CCITT, pour la spécification et la description de systèmes. De nombreuses versions furent conduites depuis 1972 jusqú'à la dernière, appelée SDL88 (Définition dans [Z.198]). De nombreuses extensions ont été développées pendant ces années et ont amené LDS à un grand degré de maturité.

Il propose d'ailleurs deux syntaxes de description de systèmes. L'une, appelée LDS/GR, est une représentation graphique qui fournit une vue d'ensemble du système. L'autre, appelée LDS/PR, décrit le système textuellement, ce qui complète la première représentation. La complétude de ces deux représentations a en partie offert un grand succès à ce langage.

Le comportement d'un système est constitué par la combinaison de BLOCS (représentant des modules), contenant un certain nombre de processus possédant chacun une adresse unique. Un processus est une machine à états finis étendue fonctionnant de façon autonome en parallèle à d'autres processus. Un processus est composé d'actions qui s'exécutent. Mais il peut aussi réagir à des stimuli externes (sous forme de signaux) en accord avec sa description.

La communication entre ces processus s'effectue de manière asynchrone par des messages discrets appelés signaux qui partent d'un processus spécifié par son adresse vers un autre. Un processus peut de plus échanger des données avec un processus d'un autre bloc du système grâce à un autre type de signaux appelés SIGNALROUTE, circulant dans des canaux pré-déclarés. Chaque processus possède une file d'attente où les signaux arrivant sont stockés. Il peut alors, soit attendre l'arrivée d'un signal, soit passer une transition, modélisant une action, entre deux états. Une transition est déclenchée par le premier signal arrivé, ce qui permet, par exemple, la manipulation de variables.

Outre le fait de décrire un système grâce à deux représentations, LDS utilise des notions connues, telles que les signaux, les états, les processus. Ceci offre une prise en main rapide du langage, à l'instar du langage LOTOS, présenté précédemment. Par contre, décrire un système avec LDS peut être long, du fait qu'il faut décrire chaque canal, chaque signal, chaque processus et chaque bloc.

Estelle

Le langage ESTELLE a aussi été développé par le CCITT, et a été spécialement étudié pour modéliser des protocoles de communications. Développé dans les années 80, il a ensuite été normalisé ISO 9074 ([ISO88]). Une présentation générale du langage peut être trouvée dans [BD88].

Un système, décrit en Estelle, est composé de modules organisés hiérarchiquement et exécutés en parallèle de manière asynchrone qui s'échangent des données. Le comportement d'un module est décrit grâce à un automate à états finis, tandis que les données sont décrites en langage PASCAL.

L'automate est, quant à lui, composé de transitions partant d'un état initial, noté FROM, et arrivant à un état final, noté TO. Une telle transition symbolise un ensemble d'actions qui sont exécutées si et seulement si :

- un prédicat PROVIDED, modélisant une condition sur des données, est satisfait,
- une garde WHEN, modélisant une condition sur la réception de données d'un autre module est satisfaite.

Une priorité (PRIORITY) peut, de plus, être définie sur chaque transition.

Les modules communiquent par des points d'interaction, connectés entre eux par des canaux, qui peuvent être construits et détruits dynamiquement, qui autorisent des échanges bi-directionnels de données. Ces canaux permettent de faire des échanges de services et de protocoles.

L'avantage de ce langage est qu'il permet une description d'un protocole de communication d'une manière plus simple qu'avec, par exemple, LDS. En effet, il est possible de différencier la notion de service de la notion de protocole grâce aux canaux, et ceux-ci peuvent être mis à disposition dynamiquement à l'instar de LDS. Son niveau d'abstraction est par contre plus faible que LOTOS car il est très proche du formalisme de l'automate.

Réseaux de Petri

Les réseaux de Petri (ou RdP), ont été définis initialement dans les années 60 par C. Petri, puis développés au MIT. Les réseaux de Petri sont très adéquats pour modéliser des systèmes car ils permettent de représenter des notions telles que le parallélisme, la synchronisation et le partage de ressources.
Un réseau de Petri est défini par un tuple $C =< P, T, I, O >$ où P est un ensemble de places, T un ensemble de transitions, I une fonction associant à chaque état un nombre fini de transitions, et O une fonction associant à chaque transition un nombre fini d'états. Le fonctionnement d'un RdP peut se résumer par ce qui suit : Chaque place du réseau de Petri contient un nombre entier de jetons, qui sera amené à varier pendant le fonctionnement du système. A l'instar des automates, une transition est segmentée en arcs entrants et sortants. Chaque arc sortant est valué par un nombre entier représentant le nombre de jetons, que la place de départ doit au minimum contenir pour que cette transition soit franchie. Lors du franchissement, la place de départ perd ce nombre de jetons. Puis, chaque place d'arrivée reçoit un nombre de jetons égal à la valuation de l'arc entrant. L'ensemble des nombres de jetons associés à chaque état à un instant donné est appelé marquage du RdP.

Ce modèle détient sans aucun doute un niveau d'abstraction plus faible que les précédents. Bien qu'à première vue sa prise en main soit plus difficile à cause de sa grande diversité de notions (jeton, place, transition multiple), les RdP étalent tous leurs intérêts grâce à la théorie mathématique et aux outils déjà mis en oeuvre, qui permettent de vérifier la cohérence d'un système même complexe (vérification, évaluation de performance, robustesse).

2.1.2 Langages de description formelle de bas niveau

Ces modélisations sont généralement générées à partir d'un TDF de haut niveau et décrivent la même spécification sans l'utilisation de notions telles que la priorité, les canaux, les processus...

Dans les langages de haut niveau, il est souvent question de communication entre les modules ou entre un module et l'environnement extérieur. La notion de communication n'est pas décrite explicitement dans les modèles suivants mais est représentée par la notion de symboles d'entrée et de sortie.

Définition 2.1.1 (Ensemble de symboles d'entrée et de sortie) *Un symbole d'en-*

trée est un symbole d'un alphabet commençant par "?", modélisant un stimulus ou un message appliqué au système.

Un symbole de sortie est un symbole d'un l'alphabet commençant par "!", modélisant une observation faite depuis le système ou un message reçu.

Dans la suite, nous présentons quelques modèles connus qui nous seront utiles tout au long de cette thèse.

Modèle Relationnel

Le modèle relationnel, décrit dans [Tar41, Mil90] définit une spécification comme étant composée d'un espace E de variables manipulées par un programme et d'une relation R définie sur le domaine (ensemble des symboles d'entrée) et l'image (ensemble des symboles de sortie).

Une spécification S, décrite grâce à une relation $R : I \rightarrow O$, avec I l'ensemble des symboles d'entrée de cette relation et O l'ensemble des symboles de sortie, permet de représenter une spécification à un niveau d'abstraction supérieur aux automates : en effet, seuls les symboles sont mis en évidence. Ainsi, utiliser une relation permet de décrire une spécification comme une boîte de laquelle sont émis et reçus des symboles, ce qui est très proche de la représentation de l'implantation, étant donné qu'elle est vue par le test comme une boîte noire.

Machine à états finis

Une machine à états finis est un graphe décrivant les actions de la spécification. Les états de cet automate décrivent les états internes du système. Un ensemble de symboles d'entrée et un ensemble de symboles sortie sont définis pour modéliser respectivement les signaux émis vers la spécification et reçus depuis cette dernière. Comparée au modèle relationnel, cette représentation détaille la spécification en illustrant ses états ainsi que les interactions qu'il est possible d'effectuer.

Une FSM (Finite State Machine) est un graphe défini comme suit :

Définition 2.1.2 (FSM) *Une FSM A est un quintuplet $< s_0, S, I, O, T >$ où :*

- *$S_0 \in S$ est l'état initial,*
- *S est un ensemble non vide d'états,*
- *I est l'ensemble des entrées,*
- *O est l'ensemble des sorties,*
- *T est un ensemble non vide de transitions tel que $T \in S \times I \times O \times S$. Le tuple <s,i,o,s'> représente une transition qui part de l'état s et va vers l'état s'. Celle-ci est étiquetée par une entrée et une sortie qui symbolisent le fait que pour passer à l'état s', il est nécessaire d'envoyer l'interaction i et de recevoir l'interaction o. Une telle transition est communément notée par $s \xrightarrow{i/o} s'$.*

Nous allons maintenant clarifier certaines propriétés souvent employées dans ce manuscrit qui sont le déterminisme et la complétude.

Définition 2.1.3 *FSM déterministe*

Soit la FSM $A =< s_0, S, I, O, T >$. *A est déterministe ssi :*
$\forall s \in S, \forall t = s \xrightarrow{A/B} s' \in T$ *avec* $A \in I, B \in O, s' \in S$,
$\forall t' = s \xrightarrow{C/D} s'' \in T$, *avec* $C \in I, D \in O, s'' \in S$,
$A = C$ *implique* $B = D$ *et* $s' = s''$.

Définition 2.1.4 *FSM complète*

Une FSM $< s_0, S, I, O, T >$ *est complète si pour chaque état* $s \in S$ *et pour chaque entrée* $a \in I$, *il existe une transition correspondant à la réception de a partant de s. On dit que la FSM est incomplète sinon.*

Système de transition étiqueté (LTS)

Le modèle de la machine à états finis, bien que très utilisé comporte une grande restriction quant à la description d'une spécification. En effet, il est impossible de séparer émission d'un symbole d'entrée (stimulus) et réception d'un symbole de sortie (observation). Or, bien que le fait de combiner sur une seule transition les symboles d'entrée et de sortie simplifie les calculs et les méthodes de test, il serait intéressant de pouvoir modéliser un protocole de communication, ou plus globalement une application, grâce à un automate composé de transitions étiquetées par un symbole unique.

Définition 2.1.5 *Système de Transitions Étiquetées (LTS)*

Un système de transitions étiquetées (LTS pour Labeled Transition System) A est un quadruplet $< S, L, s_0, T >$ *où :*

- *S est un ensemble fini non vide d'états,*
- *L est un ensemble fini non vide d'interactions,*
- s_0 *est l'état initial du système de transitions étiquetées.*
- $T \subseteq S \times (L \cup \{\tau\}) \times S$ *est la relation de transition. <s,a,s'> représente une transition qui part de l'état s et va vers l'état s'. Une telle transition est communément notée par* $s \xrightarrow{a} s'$.

Certaines extensions aux LTS (par exemple les IOSM [Pha94]) ont été proposées dans le but de répartir les interactions en deux sous ensembles, qui sont là aussi l'ensemble des symboles d'entrée et l'ensemble des symboles de sortie.

L'évolution du temps n'est pas précisée. La décision d'émettre peut se faire dans un temps quelconque, et il en est de même pour les réceptions qui peuvent subvenir à n'importe quel moment.

Comme précédemment, nous allons définir ce qu'est un LTS déterministe et complet.

Définition 2.1.6 (LTS déterministe) *Soit le LTS* $A =< S, L, s_0, T >$. *A est déterministe ssi :*
$\forall s \in S, \forall t = s \xrightarrow{A} s' \in T$ *avec* $A \neq \tau \in L, s' \in S$,
$\forall t' = s \xrightarrow{B} s'' \in T$, *avec* $B \neq \tau \in L, s'' \in S$,
$A \neq B$ *ou* $s' = s''$.

La notion de complétude est la même que celle d'une FSM. C'est-à-dire qu'à chaque état il existe une transition permettant d'accepter tout message de réception.

2.2 Méthodologies de génération de test de systèmes non temporisés

Nous rappelons dans cette section, la définition et le fonctionnement de quelques méthodologies de génération de séquences de test qui sont fortement étudiées et qui nous ont inspirés dans cette thèse.

Celles-ci sont présentées sous forme de trois catégories :

- la première rappelle le fonctionnement des méthodologies de test orientées Objectif de test qui vérifient si uniquement un ensemble de propriétés appartenant ou non à la spécification peut être vérifié sur l'implantation,

- la deuxième décrit quelques méthodologies basées sur le modèle FSM (TT, UIO, W, Wp, HSI), qui consistent à vérifier que l'implantation ne contient pas de fautes, en caractérisant, d'une manière plus ou moins forte, l'ensemble du comportement de la spécification et en vérifiant que la signature obtenue existe dans l'implantation,

- la dernière catégorie décrit une méthodologie basée sur le modèle LTS utilisant l'approche du Testeur Canonique, qui détermine si une relation, appelée relation d'implantation, entre la spécification et l'implantation est vérifiée.

Chacune de ces méthodologies permet, suivant un degré plus ou moins élevé, la détection de fautes sur l'implantation sous test. Les fautes qui ont la possibilité d'être relevées, sur les modèles FSM et LTS, sont réunis dans un modèle de fautes qui est décrit ci-dessous.

2.2.1 Modèle de fautes

L'ensemble des fautes que les méthodologies suivantes sont susceptibles de détecter, correspond à une combinaison des fautes, qui sont réunies dans ce qui est appelé un modèle de fautes [BDDD91].

1. *erreur de sortie* : une implantation produit une erreur de sortie, s'il existe une transition qui ne porte pas le même symbole dans la spécification et l'implantation.
2. *erreur de transfert* : une implantation produit une erreur de transfert s'il existe une transition qui ne conduit pas au même état dans la spécification et l'implantation.
3. *Extra état (état manquant)* : une implantation est dite avoir un état extra (état manquant) si le nombre d'états de cette implantation doit être diminué (augmenté) pour obtenir le même nombre d'états que la spécification.

2.2.2 Méthodes orientées Objectif de test

Les méthodes orientées Objectif de test [Bri87, Tre96] sont spécifiquement désignées pour ne tester qu'une partie du comportement de la spécification. Le concepteur choisit

des propriétés comportementales, appelées Objectif de test, qu'il souhaite vérifier sur l'implantation.

Ces objectifs peuvent être générés pour satisfaire deux buts : vérifier que des propriétés de la spécification existent dans l'implantation, ou que d'autres propriétés n'existent pas. Ceci conduit à la Définition suivante :

Définition 2.2.1 *Un objectif de test est un automate déterministe et acyclique contenant des états particuliers.*

- *Un objectif de test acceptant est composé d'un ensemble de propriétés appartenant à la spécification. Chaque état de l'objectif appartient à un ensemble d'états noté Accept, montrant que l'objectif de test doit être entièrement satisfait.*
- *Un objectif de test refusant est composé d'un ensemble de propriétés appartenant ou non à la spécification. Les états appartiennent soit à l'ensemble Accept soit à l'ensemble Refuse. Le fait qu'un état soit étiqueté par Accept montre que la propriété précédant cet état doit se vérifier dans l'implantation. S'il est étiqueté par Refuse, la propriété ne doit pas se vérifier.*

Ces objectifs de test sont généralement créés à la main, mais certains travaux proposent une automatisation de leurs choix [FJJV96].

Les séquences de test, sont extraites à partir d'un objectif de test qui peut être combiné avec la spécification. Dans ce dernier cas, une sorte de synchronisation entre la spécification et l'objectif est obtenue, en parcourant chaque chemin de la spécification pour vérifier si une suite de propriétés de l'objectif de test y est inclue. Si un chemin satisfait cette condition, il correspond à une séquence de test. Plusieurs méthodes sont proposées pour arriver à ce but, notamment par parcours en profondeur ou largeur de la spécification et par simulation. Chacune de ces méthodes possède ses propres avantages et inconvénients suivant la spécification. Une fois les séquences de test générées, celles-ci sont appliquées sur l'implantation, lors de la phase de test. Un verdict est ensuite obtenu : si les propriétés acceptantes de l'objectif de test sont vérifiées et si les propriétés refusantes ne le sont pas alors l'objectif de test est satisfait.

Prenons l'exemple d'une spécification illustrée en Figure 2.2, décrivant un distributeur à café. Si le concepteur veut juste être conforté sur le fait que lorsqu'il met un jeton, il obtient bien son café, l'objectif de test décrit par le LTS de la Figure 2.2 lui permet de vérifier cette propriété fondamentale pour le système (et le concepteur).

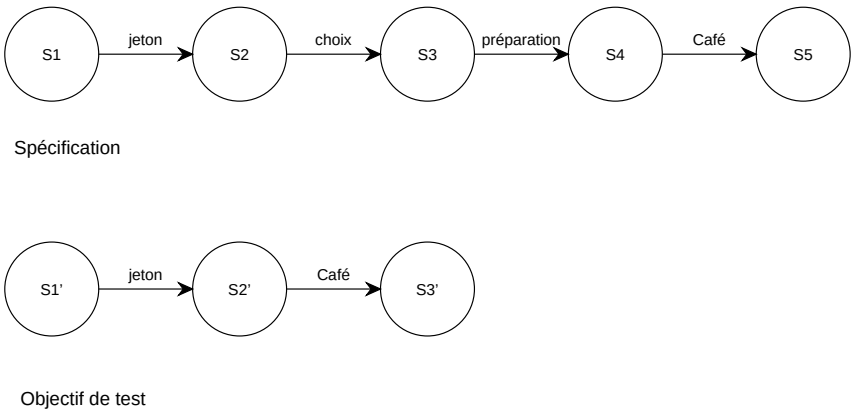


FIG. 2.2 – Une spécification LTS décrivant un distributeur à Café, et un objectif de test

2.2.3 Méthodes basées sur le modèle FSM.

Les méthodes suivantes produisent des séquences de test qui permettent de couvrir toutes les propriétés de la spécification. Selon les résultats des observations de l'application de ces séquences, nous pouvons conclure sur la nature de la conformité de l'implantation par rapport à la spécification. Cette nature de conformité est formalisée par un ensemble de relations d'équivalences, aussi appelées relations de conformité.

Définition 2.2.2 (Relation de Conformité) *Soit S une spécification décrite par une FSM et I une implantation supposée décrite par le même formalisme. I est conforme à S ssi :*

- *I et S acceptent le même ensemble de symboles d'entrée E,*
- *I et S réagissent de la même manière face aux stimuli induits par E, c'est-à-dire si le même ensemble de symboles de sortie est observé depuis S et I, en appliquant E.*

Notons de plus, que la méthode W et ses extensions proposent et prouvent, une plus forte équivalence entre S et I qui est basée sur un isomorphisme entre S et I. Cette équivalence décrite avec la méthodologie est la relation la plus forte, proposée entre une spécification et une implantation.

Plusieurs travaux sur les méthodologies de test basées sur le modèle FSM ont été proposés. Les plus connues sont la méthode TT [NT81](Transition Tour), la méthode DS [Gon80](Distinguish Sequence), la méthode UIO [SD88b] et ses extensions UIOv [VCI89], UIOp [CA92b], UIOs [SLD92] (Unique Input Output), la méthode W [Cho78] et ses extensions Wp [FBK+91], HSI [LPvB94].
Le but de ces méthodes est de rechercher une sorte de signature de toutes les propriétés comportementales de la spécification et de l'intégrer dans les séquences de test. Lors de la phase de test, l'application de ces séquences permet de vérifier l'existence de cette signature dans l'implantation.

Ces méthodes peuvent détecter un éventail variable de fautes dans l'implantation. Cet éventail, ou couverture de fautes, varie suivant la méthode utilisée et influence la

longueur des séquences de test. Les auteurs de [FBK+91] ont présenté une comparaison de couverture de fautes de quelques unes des méthodes précédemment citées. La Figure 2.3 illustre cette comparaison.

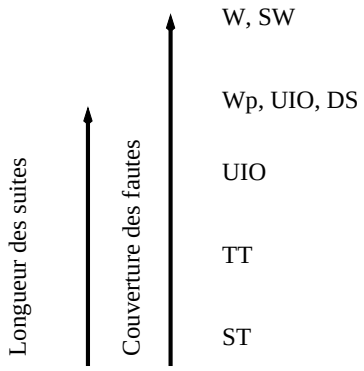


FIG. 2.3 – Couverture de fautes de quelques méthodes de dérivation de test.

Certaines de ces méthodes utilisent un ensemble de séquences de transitions, qui sont intégrées dans les séquences de test. Celui-ci est appelé l'ensemble des préambules et est défini ci-dessous.

Définition 2.2.3 *Ensemble des préambules Q. Soit un FSM* $A < s_0, S, I, O, T >$. *L'ensemble des préambules Q est l'ensemble des séquences de symboles d'entrée qui permettent d'arriver à tout état de S en partant de l'état initial* S_0, *tel que :*

$\forall s_i \in S, \exists p_i \in Q$ *tel que* $s_0 \xrightarrow{p_i} s_i$. *Pour l'état initial* s_0, *nous avons* $s_0 \xrightarrow{\epsilon} s_0$. ϵ *appartient donc toujours à Q.*

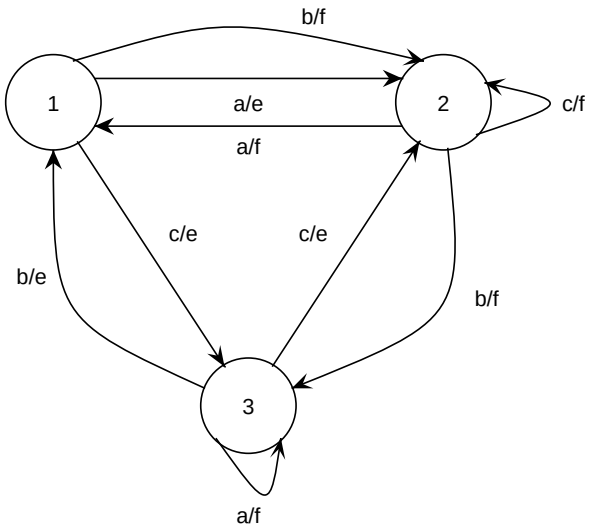


FIG. 2.4 – Exemple de FSM

Un ensemble de préambules obtenu depuis la FSM de la figure 2.4 est le suivant : $\{\epsilon, a, c\}$.

La méthode TT.

La méthode Tour de Transition consiste à trouver une séquence minimale, parcourant au moins une fois chaque transition de la FSM. Il en résulte une génération de séquence de test extrêmement simple mais qui ne permet pas la détection de toutes les erreurs de l'implantation. En particulier, aucune erreur de transfert ne peut être détectée. Grâce à son faible coût, cette méthode peut être prise en compte pour une première recherche de fautes flagrantes avant la mise en oeuvre de méthodes plus fines mais plus onéreuses de détection.

Une séquence de test obtenue grâce à cette méthode sur la spécification de la Figure 2.4 peut être :
{ c/e a/f b/e a/e b/f c/e c/f a/f b/f }

La méthode UIO.

Le but de cette méthode [SD88a] est de caractériser les états de la spécification en trouvant une unique séquence de symboles d'entrée / sortie pour chaque état. Ensuite, il ne reste plus qu'à concaténer chaque séquence obtenue depuis l'état s_i au préambule p_i permettant d'arriver à cet état, pour enfin obtenir une des séquences de test. La spécification doit être déterministe et fortement connexe pour lui appliquer cette méthode. Cependant, en plus de ces contraintes, elle n'est pas toujours applicable. Sa couverture de fautes n'est pas non plus optimale, car certaines fautes de transfert ne peuvent être détectées. Son temps de calcul reste faible ($n^2 * d_{max}^{2n^2+2}$, avec n le nombre d'états de la spécification et d_{max} le nombre maximal de transitions sortantes). De nombreuses améliorations (UIOv, UIOp), ont été proposées pour l'optimiser, notamment en ce qui concerne la détection des fautes de transfert.

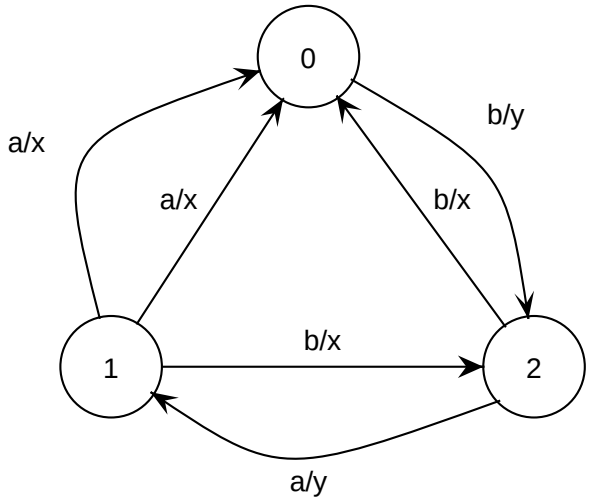


FIG. 2.5 – Nouvel exemple de FSM.

La spécification modélisée dans la figure 2.5 appliquée à la méthode UIO nous donne les séquences de test suivantes : { b, b.a.a, b.a }

La méthode W.

La méthode W [Cho78] est inspirée de la distinction de deux états d'une FSM présentée dans [Gil62]. Cette méthode utilise un ensemble particulier appelé Ensemble de Caractérisation d'états noté W.

Définition 2.2.4 *Ensemble de caractérisation d'états W.*
Soit $A < s_0, S, I, O, T >$ une FSM déterministe et minimal. Un ensemble de caractérisation W est composé de séquences d'entrée qui permettent la distinction du comportement de chaque paire d'états d'un automate minimal, en observant leurs réactions.

Cet ensemble permet d'obtenir la signature de la spécification la plus précise. De plus, cet ensemble existe toujours, ce qui n'est pas le cas des séquences UIO. Notons que pour lui appliquer la méthode, la spécification doit être déterministe, minimale et complètement spécifiée sur l'ensemble des symboles d'entrée.

Cette méthode et ses extensions utilisent un autre ensemble de séquences, appelé Ensemble de couverture de transitions, lors de la génération des séquences de test.

Définition 2.2.5 *Ensemble de couverture de transitions P.*
Soit une FSM $A < s_0, S, I, O, T >$. L'ensemble de couverture de transitions P est un ensemble de séquences de symboles d'entrée tel que :
$\forall s_i \xrightarrow{x/y} s_j \in T, \exists p, p.x \in P$ *tel que* $s_0 \xrightarrow{p} s_i$ *et* $s_0 \xrightarrow{p.x} s_j$.

La construction peut être faite en générant un arbre couvrant de l'automate, l'ensemble P étant tous les chemins partiels de cet arbre.

Les séquences de test obtenues par la méthode W sont : $P \times \{W \cup W.I \ ... \ \cup I^{m_n}.W\}$, avec I l'ensemble des symboles d'entrée de la spécification S, n le nombre d'états de S, m le nombre d'états de l'implantation, et P l'ensemble de couverture de transitions.

La relation de conformité, entre la spécification S et l'implantation I , est définie pour la méthode W par une équivalence, appelée W-équivalence. En résumé, cette équivalence, basée sur un isomorphisme de S vers I (qui correspond à une bissimulation), prodigue que pour chaque état E de S, il doit exister un unique état E' dans I tel que E et E' réagissent de la même manière en leur appliquant l'ensemble W. En appliquant les séquences de test sur I, si celle-ci réagit en donnant les mêmes symboles de sortie que S, alors I est W-équivalent à S. Chow a montré avec cette équivalence [Cho78], que toute erreur de transfert et de sortie sera détectée dans l'implantation.

Son temps de calcul est $n^2 * k$ (automate comportant n états et k transitions), ce qui est bien supérieur à celui de la méthode UIO, mais elle prodigue la couverture de fautes la plus grande.

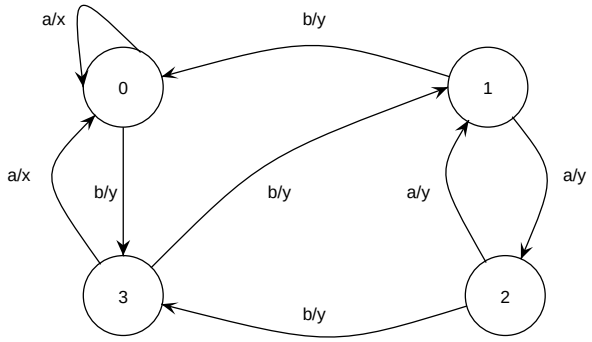


FIG. 2.6 – Autre exemple de FSM.

L'implantation de la figure 2.6 nous donne les suites de test suivantes : { b.b.a.b.a.a.a, b.b.a.b.b.b.a, b.b.a.a.a.a, b.b.a.a.b.b.a, b.b.b.a.a.a, b.a.a.a.a, b.b.b.b.b.a, b.a.b.b.a, b.b.a.a.a, b.b.b.b.a, a.a.a.a, a.b.b.a, b.a.a.a, b.b.b.a, a.a.a, b.b.a }

La Méthode Wp

La méthode Wp [FBK+91] est une amélioration de la méthode W, qui permet d'obtenir moins de séquences de test et donc de réduire le coût du test, tout en gardant la même couverture de fautes. Celle-ci est constituée de deux phases.

1. L'obtention des premières séquences de test est effectuée par la concaténation $Q.W = \bigcup_{s_0 \xrightarrow{p/q} S_i \text{ et } p \in Q.(\{\epsilon\} \cup I \ldots \cup I^{m-n})} \{p\}.W$,
   où n est le nombre d'états de la spécification et m celui de l'implantation.
2. La deuxième phase utilise une partition $\{W_1 \ldots W_n\}$ de l'ensemble W. L'ensemble W_i est l'ensemble nécessaire et suffisant caractérisant l'état s_i des autres états de la spécification. Le reste des séquences est donc construit de la manière suivante
   $(P - Q) \otimes W = \bigcup_{s_0 \xrightarrow{p/q} S_i \text{ et } p \in (PQ).(\{\epsilon\} \cup I \ldots \cup I^{m-n})} \{p\}.W_i$

Le temps de calcul nécessaire pour générer les séquences de test est $2 * n^2 * k - n * k + n$ (FSM comportant n états et k transitions), ce qui est encore supérieur à celui de la méthode W.
En reprenant la figure 2.6, nous obtenons les suites de test suivantes : Partie 1 : { b.a.a.a.a, b.b.a.b.b.a, b.b.a.a.a, b.b.b.b.a, b.a.a.a, b.b.b.a, a.a.a, b.b.a },
Partie 2 : { a.b.b.a, a.a.a.a }.

La Méthode HSI

Estelle, LOTOS et SDL peuvent décrire des spécifications non déterministes et partiellement spécifiées. Or, toutes les méthodes précédentes imposent que la spécification soit déterministe, et certaines imposent que la spécification soit complètement spécifiée. La méthode HSI [LPvB94], autre extension de la méthode W, permet donc de générer des séquences de test, à partir de spécifications ne satisfaisant pas ces propriétés.

Elle commence donc par construire un automate équivalent à la spécification qui est observable, non partiellement défini et déterministe (OPNFSM), en utilisant un algorithme de déterminisation. Puis les séquences de tests sont générées grâce aux points suivants :

1. Déterminer le degré de flou δ de la spécification S. Celui-ci permet de trouver un ensemble de caractérisation d'états même si certains de ceux-ci ne sont pas distinguables.
2. si n est le nombre d'états de S, alors trouver une partition $\{D_0, D_1, ...D_{n-1}\}$ permettant de caractériser tous les états de S.
3. Trouver l'ensemble des préambules.
4. Générer les suites de test : $\Pi = \bigcup_{s_0 \xrightarrow{p/q} S_i \text{ et } p \in Q.(\{\epsilon\} \cup I ... \cup I^{\delta * m - n + 1})} \{p\}.D_i$.

La clé réside essentiellement dans le degré de flou δ. En effet, lors de la construction des suites de test, on considère que l'implantation peut avoir des états non distinguables que l'on ne peut observer. Nous avons donc réellement $\delta * m$ états dans l'implantation, mais m sont distinguables. La prise en compte des états non observables, permettant le test des automates partiellement définis, est faite en concaténant à la suite de l'ensemble Q l'ensemble de toutes les séquences de symboles d'entrée possibles de longueur $\delta * m - n + 1$. Ceci garantit l'accès à tous les états non distinguables.

Cette méthode crée les suites de test avec un temps de calcul encore plus long que les méthodes précédentes. Son plus grand intérêt réside dans la prise en compte de tout type de spécifications.

2.2.4 Testeur Canonique

Un testeur canonique est un automate dérivé de la spécification qui permet le test de l'implantation. Celui-ci est construit par rapport à la relation décrite entre la spécification et l'implantation, qui est appelée Relation d'implantation.

Définition 2.2.6 *Relation d'implantation*
Une relation d'implantation est une relation R sur $LTS \times LTS$. Soient $I, S \in LTS$. Si $R(I, S)$ on dit que I est une implantation conforme de S.

Plusieurs relations d'implantations peuvent être trouvées, notamment dans [Bri88, Pha94]. Toutes permettent d'obtenir une plus ou moins grande couverture de fautes.

Par exemple, nous pouvons trouver dans [Pha94] des relations basées sur les traces d'un IOSM et de l'implantation. Une trace d'un IOSM S $< S_s, L_s, T, s_0 >$, notée $Tr(S)$, est donc une séquence d'entrées et de sorties observable qui, à partir de l'état initial, nous amène à un état quelconque de S. Pour $\sigma \in Tr(S)$, notons $O(\sigma, S) = \{a \in L_s \mid \sigma!a \in Tr(S)\}$ les sorties autorisées par la spécification après la trace σ.

Soit donc l'exemple de relation d'implantation décrite dans [Pha94] :

Définition 2.2.7 *Relation d'implantation R_1.*
$R_1(I, S)$ ***ssi*** $(\forall \sigma \in Tr(S))(\sigma \in Tr(I) \Rightarrow O(\sigma, I) \subseteq O(\sigma, S))$

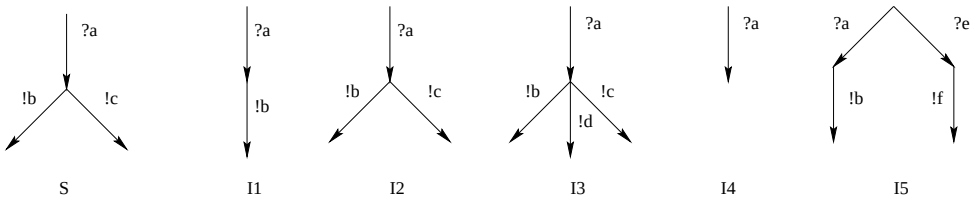


FIG. 2.7 – Une spécification S et des implantations.

D'après la figure 2.7, nous pouvons voir que toutes les implantations sont conformes par rapport à S sauf I3. R_1 garantit que tout ce qui peut sortir de l'implantation en réponse à une entrée donnée a été prévu par la spécification.

Le testeur canonique obtenu grâce à la relation R_1 est appelé testeur canonique asynchrone [Pha94]. D'autres testeurs canoniques peuvent être obtenus suivant la relation d'implantation choisie. D'autres peuvent être trouvés dans [Bri88, Pha94].

Définition 2.2.8 *Testeur canonique asynchrone[Pha94]*
Soit S une spécification et TM(S)=$< S_s, L_s, T_s, S_{0s} >$ *son automate de traces. On appelle testeur canonique asynchrone de S, une IOSM notée TCA(S) =* $T = < S_t, L_s, T_t, S_{0t} >$ *telle que :*

1. $S_t = S_s \cup \{fail\}$
2. $L_t = L_s$
3. $s_0 t = s_0 s$
4. *les transitions du testeur sont exactement celles obtenues par les règles suivantes :*

   a) $(\forall \mu \in \{!, ?\} \times L_s)(\forall s, s' \in S_s)((s, \mu, s') \in T_s \Leftrightarrow (s, inv(\mu), s') \in T_t)$

   b) $(\forall s \in S_s)(\forall a \in L_s)(\neg(\exists s')((s, !a, s') \in T_s) \Rightarrow (s, ?a, fail) \in T_t)$

   avec $inv(\mu)$ *la séquence obtenue en inversant les réceptions (? en !) et les émissions (! en ?).*

L'application du testeur canonique sur l'implantation fournit le verdict de test suivant :

Définition 2.2.9 *Verdict du test. FAIL(T,I) ssi* $(\exists \sigma \in Tr(T))(inv(\sigma) \in Tr(I) \wedge (T, \sigma, fail))$. *Succ*$(T, I)$ *ssi* $\neg$ *FAIL(T,I).*

Le testeur canonique asynchrone de S est chargé de vérifier que rien de contraire à ce qui est prévu ne peut arriver. Il fournit donc une image miroir des traces de la spécification. On rajoute un mécanisme de détection des erreurs, c'est-à-dire que si le testeur reçoit une interaction qui n'est pas censée pouvoir arriver dans un état donné, il passe dans l'état **fail**.

Voici un exemple du fonctionnement du testeur canonique basé sur la spécification de la figure 2.7.

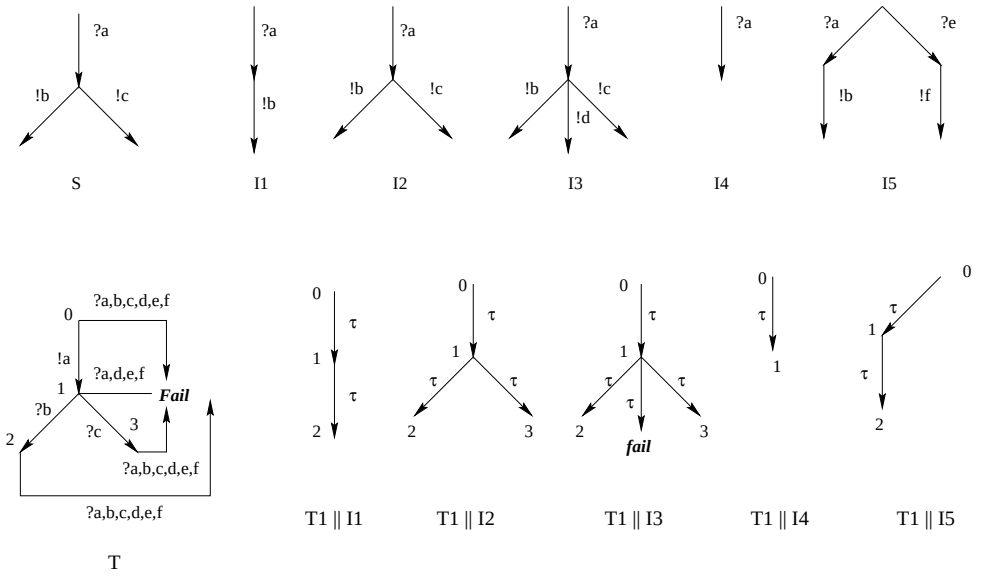


FIG. 2.8 – Testeur canonique asynchrone.

Cet exemple montre une spécification S et ses cinq implantations. Le testeur canonique asynchrone obtenu est $T1$. Et l'application de ce dernier sur les implantations sont notées $T1||I1, ..., T1||I5$. La détermination du verdict revient à savoir si un état de la forme (fail,s_i) se trouve dans le graphe de la composition du testeur $T1$ et de l'implantation.

2.3 Architectures de test

Une fois les séquences de test générées, celles-ci sont appliquées à l'implantation sous test au moyen d'une architecture de test. La norme ISO 9646 [ISO91b] décrit quelques architectures, qui sont utilisées dans le test de protocoles de télécommunications. Certaines ont été précédemment présentées dans [Ray87]. Celles-ci sont les architectures locales et distribuées.

2.3.1 Architecture Locale

Comme son nom l'indique, cette architecture réunit ses éléments dans un système interne. Elle permet le test d'une couche N d'un protocole, ou d'une implantation considérée comme telle. Deux testeurs y sont considérés : l'un, appelé testeur supérieur (TS) interagit directement avec le service de niveau N par émission et réception de primitives de services (ASP(N)). L'autre, appelé Testeur inférieur (TI), interagit avec le service de niveau N-1 par émission et réception de primitives de services (ASP(N-1)) qui contiennent des PDU(N) de niveau N. Les points d'accès à l'implantation par les testeurs sont appelés des PCO (Point de Contrôle et d'Observation). La Figure 2.9 illustre et résume cette architecture.

L'avantage de cette architecture est la communication directe qu'il existe entre l'implantation et les testeurs. Or un tel cas de figure semble rare et souvent irréalisable. Voilà pourquoi des architectures non locales ont été proposées.

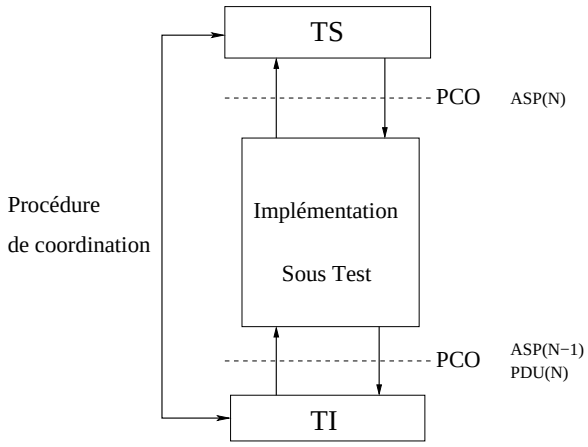


FIG. 2.9 – Architecture locale

2.3.2 Architecture distante

Plusieurs architectures non locales ont été proposées, dont l'architecture distante qui est la plus généraliste de toutes. Le testeur inférieur devient cette fois externe au système testé, avec lequel il communique grâce au service fourni par la couche N-1, qui est supposée ne contenir aucune erreur.

Par contre aucune exigence n'est promulguée au niveau du testeur supérieur ou sur les procédures de communication entre les deux testeurs. Au moins un PCO subsiste entre le testeur inférieur et la couche N. La Figure 2.10 illustre cette autre architecture.

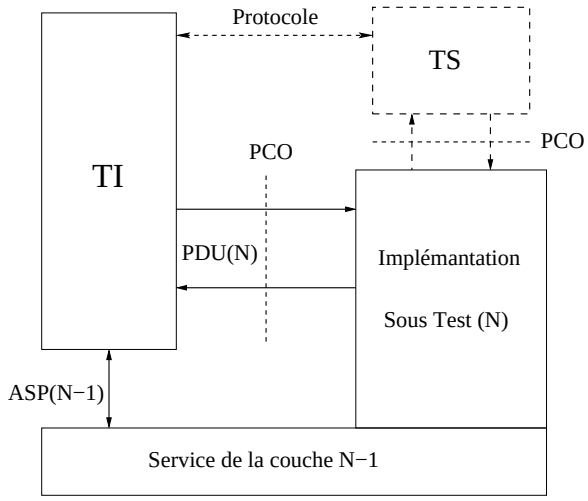


FIG. 2.10 – Architecture distante

Cette fois, le concepteur possède un choix plus libre sur la manière dont il souhaite tester son implantation. Le fait d'avoir des éléments externes permet aussi de se rapprocher de cas plus réels. Mais elle implique une communication indirecte entre le testeur inférieur et l'implantation, ce qui demande une prise en compte d'un temps de latence, lors du test

temporisé.

2.4 Outils de Génération de test

Cette section est vouée à la description de quelques outils de génération de séquences de test, basés sur une des méthodologies précédentes.

2.4.1 TVEDA

TVEDA [Pha91] est un outil de génération de séquences de test, élaboré au sein du CNET. Celui-ci implante, entre autre, la méthodologie de génération de séquences de test, décrite en section 2.2.4. Cet outil est composé d'une interface qui accepte les langages Estelle et LDS. A partir de ces langages, un automate est généré sur lequel est appliqué la méthodologie. Le résultat obtenu réside en des séquences de test, décrites grâce au langage TTCN [ISO91a] qui représente des séquences de test avec un faible niveau d'abstraction.

2.4.2 TGV

TGV (Test Generation with Verification technology, Verimag [FJJV96]) est un outil de génération de séquences de test élaboré utilisant une approche orientée objectif de test.

TGV accepte une spécification dont le comportement est décrit par un IOSM. Il combine la spécification avec un objectif de test donné sous forme d'un automate pour générer des séquences de test décrites sous forme TTCN. Dans une première étape, il construit un graphe modélisant le comportement observable de la spécification dans un environnement de test. Puis une synchronisation entre la spécification et l'objectif de test est produite, grâce à un parcours en profondeur de la spécification. Le résultat est ensuite transcrit en TTCN.

2.4.3 TTGgeN

TTCgeN (Verilog) est un autre outil de génération de séquences de test élaboré utilisant une approche orientée objectif de test, et correspond plus à une extension d'Object Geode

Il accepte donc des spécifications décrites en LDS et construit des séquences de test décrites par TTCN. A la différence de TGV, les séquences de test sont construites grâce à une simulation : la spécification LDS est exécutée, ce qui permet l'exploration exhaustive de ces états qui est elle-même guidée par les états de l'objectif de test. Le graphe obtenu est ensuite simplifié pour être transformé en TTCN.

2.4.4 TorX

L'environnement TorX, développé par l'université de Twente, l'université de Eindhoven et les laboratoires de recherche de Philips permet, à la volée, la génération de séquences de test et leurs exécutions à partir de spécifications décrites par plusieurs formalismes tels

que LOTOS, PROMELA et LDS. Cette génération à la volée s'effectue à partir d'objectifs de test donnés par l'utilisateur.

L'originalité de TorX réside en sa composition faite par plusieurs modules qui lui permettent d'être flexible et d'accepter des spécifications décrites en plusieurs langages. Par exemple, un module appelé Primer offre d'utiliser les fonctionnalités d'OPEN/CAESAR, ce qui apporte l'utilisation de spécifications décrites en LOTOS. Une description de la méthode de génération des séquences de test de TorX est donnée dans [TB99].

2.4.5 TAG

Une équipe du département IRO (Université de Montréal) a aussi proposé un outil de génération de séquences de test, appelé TAG [TPB96]. Son originalité réside dans le choix de la méthode de test : l'utilisateur peut proposer des objectifs de test (composés d'une transition) pour tester quelques propriétés ou bien peut demander la génération de séquences permettant le test de toute l'implantation.

La méthode prise en compte pour générer ces séquences de test est HSI (section 2.2.3). L'avantage d'utiliser HSI est que cette méthode garantie de trouver toutes les erreurs de sortie et de transfert. Ainsi, l'outil accepte en entrée des FSM déterministes et connexes qui peuvent être partiellement spécifiés. TAG intègre des algorithmes permettant de les minimiser et de les compléter. Les séquences de test obtenues sont sous forme d'un squelette LDS.

2.5 Conclusion

Concluons ce chapitre sur un aspect du test de conformité limitant et fondamental, qui se résume par l'affirmation très connue de Dijstra : *Testing shows the presence, not the absence od bugs*. En effet, quelquesoit le formalisme d'une spécification et la relation de conformité employée, il est difficile de montrer que l'implantation et la spécification sont complètement équivalents, par le fait que d'un coté nous observons un graphe et de l'autre une boite noire. C'est pourquoi une panoplie d'autres relations, plus ou moins fortes (relation de conformité, relation d'implantation) ont été proposées.

Concluons aussi sur le fait que la myriade de modèles proposés ne sont pas spécifiquement en concurrence, mais se complètent et chacun vise un type de spécification et de test. Par exemple, les logiques temporelles sont généralement utilisées au niveau de la vérification de la spécification, alors que les graphes (LTS, FSM) sont très utilisés au niveau du test. Il est aussi a moitié vrai d'affirmer que la notion de temps n'est pas considérée dans ces formalismes. En effet, il est toujours sous-entendu qu'une action doit se produire immédiatement. Mais bien entendu, cette notion est très insuffisante pour modéliser des systèmes temporisés. Voila pourquoi d'autres modèles (et d'autres tests) ont été définis. Certains sont présentés dans le chapitre suivant.

Chapitre 3

Modélisation et test des systèmes temporisés

De même que précédemment, nous rappelons dans ce chapitre l'état de l'art de la modélisation et du test de systèmes temporisés. La notion de temps a été représentée par de nombreuses façons, ce qui a conduit à plusieurs nouveaux modèles. Nous en décrirons donc quelques uns, dans ce qui suit, ainsi que quelques méthodes de génération de séquences de test, qui ont servies à l'aboutissement de cette thèse.

La première partie est donc vouée à la présentation de langages formels qui permettent de décrire des spécifications temporisées. Tout comme dans le chapitre précédant, nous exposerons, dans un premier temps, quelques langages ayant un haut niveau d'abstraction, tels que des extensions de LOTOS, Estelle ou des Réseaux de Petri. Puis, nous décrirons des langages de plus faible niveau d'abstraction, comme les automates temporisés d'Alur et Dill et les graphes des régions, qui sont souvent utilisés

par des techniques de vérification [MY96, SY96, AD92, BFG+00],des méthodes de test [LC97, RNHW98, SVD01, AKA00, NS01] et des outils [DOTY95, BL95]. C'est d'ailleurs ces deux modèles que nous utiliserons tout au long de ce manuscrit. Nous traiterons aussi d'autres formalismes tels que les automates à événements d'horloges et TTM (Timed transition Model).

Quelques méthodologies de génération de séquences de test basées sur le modèle de l'automate temporisé seront par la suite présentées. L'étude de celles-ci permettra de conclure sur ce qui peut être amélioré dans le but d'obtenir un test de meilleure qualité (réduction du coût du test et une meilleure détection des fautes).

A l'heure actuelle, peu d'outils, considérant des systèmes temporisés, ont été implantés. En fait, seuls des outils de vérification académiques sont en ce moment disponibles. Nous en décrirons les plus connus pour terminer ce chapitre.

3.1 Modélisation des systèmes temporisés

Plusieurs langages de description formelle incluent la notion du temps. Celui-ci est représenté de diverses façons : certains ont voulu l'implanter d'une manière globale et fictive, d'autres ont souhaité manipuler plusieurs horloges, certains encore ont représenté le temps d'une manière continue, d'autres d'une manière discrète. Le choix de la partie intégrant le temps varie aussi et dépend du langage (état, transition, opérateur sur des objets).

Ainsi, dans ce qui suit, nous montrons plusieurs de ces langages intégrant des fonctionnalités différentes. Comme pour les modélisations de systèmes non temporisés, nous déclinons deux sous-ensembles de langages : le premier rassemblant des langages dits de haut niveau, le second rassemblant des langages dits de faible niveau, généralement utilisés par les méthodes de test temporisé.

3.1.1 Langages de description formelle de haut niveau

Extensions du langage LOTOS

Le langage LOTOS n'intègre pas à la base la notion du temps. Ainsi, une révision du langage, nommée E-LOTOS "Extended LOTOS", est en cours de standardisation par l'ISO depuis 1993 ; l'objectif fixé par cette révision est d'intégrer les constructions des langages de programmation fonctionnels au langage LOTOS dans le but de faciliter son utilisation et d'augmenter la concision et la clarté des spécifications produites. La révision tend aussi à intégrer la notion de temps quantitatif (le type "time" et l'action "wait" ont été définis, avec ajout d'opérateurs d'addition et de comparaison sur le temps global), afin d'adapter LOTOS aux systèmes multi-média et à temps réel.
Bien que la standardisation de E-LOTOS soit toujours en cours, plusieurs variantes ont déjà été proposées parmi lesquelles on retrouve :

- LOTOS-TP "Timed Probalistic LOTOS" qui supporte les comportements temporisés et probalistes,

- RT-LOTOS "Real Time LOTOS" [CdO95], qui ajoute trois nouveaux opérateurs de délais (opérateur de délais sur l'exécution d'une action, opérateur de latence exprimant un temps de latence non déterministe, opérateur de restriction sur le temps exprimant une limite durant laquelle une action observable est offerte à son environnement). Une description détaillée est donnée dans [CSLO99],
- ET-LOTOS "Time Extended LOTOS",[LLdFQ95] proposé pour la modélisation des comportements à temps réel. ET-LOTOS est très proche de RT-LOTOS et permet de décrire les mêmes comportements. La différence majeure réside dans l'opérateur de latence temporel non déterministe de RT-LOTOS qui est remplacé par un opérateur déterministe de plus simple utilisation.
- TI-LOTOS "Time Interval LOTOS" [RBC92], qui propose de modéliser le temps par des intervalles. Par exemple, l'instruction $a[t_{min}t_{max}]; B$ illustre que a doit s'exécuter dans l'intervalle $[t_{min}t_{max}]$.

Extensions du langage Estelle

Le standard Estelle propose d'associer une clause de délais, de la forme $delay(E1, E2)$ avec une transition t pour exprimer les contraintes d'horloges. $delay(E1, E2)$ indique que la transition peut être franchie si l'horloge globale du système prend une valeur comprise entre $E1$ et $E2$.

Dans [CA92a], les auteurs ont montré que cette clause présente quelques limites. En effet, chaque transition est considérée comme étant immédiate. Il peut donc arriver qu'une action moins prioritaire qu'une autre, qui doive vérifier une clause delay, soit mise en attente et dépasse la borne supérieure. Ceci induit de gros problèmes de fonctionnement car dans ce cas certaines actions ne pourront être exécutées. Pour palier à ce problème, les auteurs ont proposé une nouvelle clause DOBY, qui au lieu de contraindre un délai d'attente, impose une contrainte sur le temps au bout duquel la transition doit avoir été tirée. En d'autres termes, elle impose une durée maximale d'exécution.

Une autre extension à Estelle a été décrite dans [Fis96]. Fisher propose l'ajout d'opérateurs formant une relation temporelle entre les états, tels que *HENCEFOR* et *ENVENTUALLY*. Par exemple, *HENCEFOR p* signifie qu'à partir de maintenant p est vrai. *ENVENTUALLY p* signifie que dans un futur état p est vrai. Plusieurs horloges peuvent aussi être instanciées par une variable, et les opérateurs $<, >, =\leq, \geq$ peuvent être utilisés pour construire des contraintes sur ces horloges.

LDS et la notion de temps

La notion de temps est intégrée de base dans LDS. Celle-ci est exprimée grâce à un mécanisme asynchrone appelé un *timer*. Ce constructeur appartient à un processus et est capable d'engendrer un signal, puis de le placer dans la queue d'entrée du processus. Tout comme l'opérateur delay d'Estelle, celui-ci exprime un intervalle $[E1E2]$, tel que $E1$ est le moment où est donné le signal à la queue et $E2$ le moment avant lequel il doit être consommé.

Lors du franchissement d'une transition, un timer peut être activé par la fonction *set* qui contient deux arguments : le premier concerne la date d'expiration du timer, tandis que le second lui assigne un nom. La date d'expiration peut être spécifiée grâce à l'expression

NOW qui renvoie la date (énoncée par un réel) courante du système. Un timer peut aussi être désactivé par la fonction *reset*.

Les Réseaux de Petri temporisés

Plusieurs extensions aux RdP ont été proposées afin de modéliser des systèmes temps réels. Les plus connues de ces extensions sont sans aucun doute les Réseaux de Petri Temporisés (Timed Petri Nets) [Ram74], qui ne sont pas à confondre avec les Réseaux de Petri Temporels (Time Petri Nets) [MF76] puis les Réseaux de Petri Stochastiques Temporisés [GJ95]. Pour chacune de ces extensions, la notion de temps est introduite au niveau des transitions, soit par un intervalle durant lequel la transition peut être franchie, soit par une valeur représentant une valeur de franchissement. L'ajout d'une probabilité sur ces intervalles, par les Réseaux de Petri Stochastiques Temporisés, permet d'introduire un indéterminisme temporel qui ajoute un degré fort de réalisme au système.

D'autres extensions [Sif80], approximativement équivalentes, sont aussi proposées afin d'introduire la notion de temps sur les places.

3.1.2 Langages de description formelle de bas niveau

TTM (Timed Transition Model)

Le modèle à transition temporisé (TTM), défini dans [Ost92, Ost94], est un automate où le temps est modélisé d'une manière discrète, c'est-à-dire que l'exécution de chaque action du système est régie par une horloge globale fictive avançant par ticks (en général en seconde) successifs, et tel qu'aucune exécution n'est permise entre deux ticks. La définition d'un TTM est la suivante :

Définition 3.1.1 *Un TTM (Timed Transition Model) est un triplet (V, I, E) tel que :*

- *V est un ensemble fini de variables contenant deux variables spéciales : t qui représente la valeur d'horloges globale et ϵ, appelée la variable d'états, qui permet de spécifier l'occurrence d'une transition dans un état particulier. D'autres variables peuvent être ajoutées pour exprimer diverses activités,*
- *I est l'état initial,*
- *E est l'ensemble des transitions décrites par le tuple (e, h, d, f, inf, sup), où e est un ensemble de conditions sur les variables, h un ensemble de transformations sur les variables, d l'état de départ, f l'état d'arrivée et $[inf sup]$ un intervalle de temps. Pour franchir une transition, il faut que les conditions de e soient satisfaites et que le temps passé dans l'état d soit compris entre inf et sup.*

L'inconvénient majeur de ce modèle concerne la discrétisation du temps. En effet, cette représentation du temps réduit considérablement le nombre de systèmes temps réels pouvant être construits du fait qu'aucune réaction n'est permise entre deux ticks d'horloges. De plus, représenter le temps par une horloge globale est restrictif : il est alors impossible d'exprimer qu'une action puisse être exécutée un certain laps de temps après l'exécution d'une ou plusieurs autres. Cependant, ce langage reste fort adapté pour faire du model-checking conjointement à la logique temporelle RTTL car cette dernière utilise aussi le temps discret.

Les Automates Temporisés et les Graphes de Régions

Alur et Dill ont proposé dans [AD94] une autre modélisation des systèmes temporisés qui décrit le temps d'une manière continue grâce à un ensemble d'horloges. A la différence des TTM, cette modélisation, appelée l'automate temporisé, permet de décrire la plupart des système temporisés. Il reste d'ailleurs le modèle le plus utilisé actuellement, tant pour la vérification que pour le test.

Un automate temporisé n'est autre qu'un LTS (Définition 2.1.2) auquel est associé un ensemble d'horloges. Chaque horloge est évaluée par un réel (représentation dense du temps). Elles sont initialisées à 0 dans l'état initial de l'automate, mais peuvent être réinitialisées avec chaque transition. Ces transitions peuvent de plus contenir des contraintes sur ces horloges. Ainsi, pour qu'une transition soit franchie, toutes les horloges du système doivent satisfaire ces contraintes.

Définition 3.1.2 (Contrainte d'horloges [AD94]) *Soit $C_{\mathcal{A}}$ l'ensemble fini des horloges du système $\mathcal{A}$, et $x_i \in C_{\mathcal{A}}$. Une contrainte d'horloges δ sur x_i est une expression booléenne de la forme $\delta = x_i \leq c \mid c \geq x_i \mid \neg\delta \mid \delta_1 \wedge \delta_2$ où $c \in \mathbb{Q}$.*

$\Phi(C_{\mathcal{A}})$ représente l'ensemble des contraintes d'horloges sur $C_{\mathcal{A}}$.

La définition de l'automate temporisé est donc la suivante :

Définition 3.1.3 (Automate temporisé) *Un automate temporisé $\mathcal{A}$ est défini comme un tuple $< \Sigma_{\mathcal{A}}, S_{\mathcal{A}}, s^0_{\mathcal{A}}, C_{\mathcal{A}}, E_{\mathcal{A}} >$, tel que :*

- *$\Sigma_{\mathcal{A}}$ est un alphabet fini,*
- *$S_{\mathcal{A}}$ est un ensemble fini d'états,*
- *$s^0_{\mathcal{A}}$ est l'état initial,*
- *$C_{\mathcal{A}}$ est un ensemble fini d'horloges,*
- *$E_{\mathcal{A}} \times S_{\mathcal{A}} \times \Sigma_{\mathcal{A}} \times 2^{C_{\mathcal{A}}} \times \Phi(C_{\mathcal{A}})$ est un ensemble fini de transitions. Un tuple $< s, s', a, \lambda, G >$ représente une transition de l'état s vers l'état s' avec le symbole a. L'ensemble $\lambda \subseteq C_{\mathcal{A}}$ correspond aux horloges réinitialisées avec cette transition, et G est une contrainte d'horloges sur $C_{\mathcal{A}}$.*

Pour obtenir la valeur d'une ou plusieurs horloges, il est nécessaire de les interpréter par des fonctions appelées valuations d'horloges.

Définition 3.1.4 (Valuation d'horloges [AD94]) *Une valuation d'horloges sur un ensemble d'horloges $C_{\mathcal{A}}$ est une fonction v qui assigne à chaque horloge $x \in C_{\mathcal{A}}$ une valeur dans $\mathbb{R}+$. Nous notons l'ensemble des valuations d'horloges $V(C_{\mathcal{A}})$.*

Une valuation d'horloges satisfaisant une contrainte d'horloges G, sera notée $v \models G$.

Pour $d \in \mathbb{R}+$, $v + d$ dénote la valuation d'horloges qui assigne une valeur $v(x) + d$ à chaque horloge x. Pour $X \subseteq C_{\mathcal{A}}$, $[X \rightarrow d]$ dénote la valuation d'horloges sur $C_{\mathcal{A}}$ qui assigne d à chaque horloge $x \in X$.

Ainsi, si x et y sont deux horloges, $v(x)$ donne la valeur de l'horloge x, $v(y)$ donne celle de l'horloge y, $v(x, y)$ donne le couple de valeurs d'horloges.

Il résulte, d'après les définitions précédentes, que le comportement d'un système temporisé dépend de ses états et des valeurs d'horloges, obtenues par les valuations de $V(C_{\mathcal{A}})$.

Cependant, cet ensemble de valeurs d'horloges est infini. Une relation d'équivalence sur les valuations d'horloges est donc définie dans [AD94]. Celle-ci décrit les classes d'équivalences qui rassemblent les valeurs d'horloges possédant la même partie entière, en régions d'horloges.

Pour $t \in \mathbb{R}+$, la partie fractionnelle de t est notée $fract(t)$ et sa partie entière est notée $\lfloor t \rfloor$.

Définition 3.1.5 (Région d'horloges [AD94]) *Soit $\mathcal{A} =< \Sigma_\mathcal{A}, S_\mathcal{A}, s^0_\mathcal{A}, C_\mathcal{A}, E_\mathcal{A} >$ un automate temporisé. Pour chaque $x \in C_\mathcal{A}$, soit c_x le plus grand entier c tel que $(x \leq c)$ où $(c \leq x)$ est une sous formule de contraintes d'horloges appartenant à $E_\mathcal{A}$.*

La relation d'équivalence $\sim$ est définie sur l'ensemble de toutes les valuations d'horloges sur $C_\mathcal{A}$; $v \sim v'$ ssi toutes les conditions suivantes sont vérifiées :

- *Pour tout $x \in C_\mathcal{A}$, soit $\lfloor v(x) \rfloor$ et $\lfloor v'(x) \rfloor$ sont les mêmes, soit $v(x)$ et $v'(x)$ sont supérieurs à c_x.*
- *Pour tout $x, y \in C_\mathcal{A}$ avec $v(x) \leq c_x$ et $v(y) \leq c_y$, $fract(v(x)) \leq fract(v(y))$ ssi $fract(v'(x)) \leq fract(v'(y))$.*
- *Pour tout $x \in C_\mathcal{A}$ avec $v(x) \leq c_x$, $fract(v(x)) = 0$ ssi $fract(v'(x)) = 0$.*

Une région d'horloges pour $\mathcal{A}$ est une classe d'équivalence de valuations d'horloges induites par $\sim$.

Donc, plus simplement, une région d'horloges est une sorte d'intervalle de temps dépendant de plusieurs horloges. Elle peut être représentée par un polyhèdre ayant plusieurs sommets.

Un graphe des régions est une représentation équivalente d'un automate temporisé où les contraintes d'horloges ne sont pas décrites sur les transitions mais dans les états. Ainsi, un état d'un graphe des régions, est composé d'un état de l'automate temporisé et d'une région d'horloges. Cette représentation permet de distinguer, avec certitude, chaque intervalle de temps pendant lequel une action peut être exécutée. Un algorithme de transformation d'un automate temporisé en graphe des régions est donné dans [AD94].

Définition 3.1.6 (Graphe des régions) *Soit $\mathcal{A} = (\Sigma_\mathcal{A}, S_\mathcal{A}, s^0_\mathcal{A}, C_\mathcal{A}, E_\mathcal{A})$ un automate temporisé. Un graphe des régions de $\mathcal{A}$ est un automate $\mathcal{RA} = (\Sigma_\mathcal{RA}, S_\mathcal{RA}, s^0_\mathcal{RA}, E_\mathcal{RA})$ tel que :*
$\Sigma_\mathcal{RA} = \Sigma_\mathcal{A} \cup \delta$, où δ représente l'écoulement du temps.
$S_\mathcal{RA} \subseteq \{\langle s, [v]\rangle \mid s \in S_\mathcal{A} \wedge v \in V(C_\mathcal{A})\}$.
$s^0_\mathcal{RA} = \langle s^0_\mathcal{A}, [v_0]\rangle$ où $v_0(x) = 0 \ \forall x \in C_\mathcal{A}$.
$\mathcal{RA}$ possède comme transition $q \xrightarrow{a}_\mathcal{RA} q'$, qui part de l'état $q = \langle s, [v]\rangle$ vers $q' = \langle s', [v']\rangle$ avec le symbole a, ssi

- *soit $a \neq \delta$ $\exists (s, s', a, \lambda, G) \in E_\mathcal{A}$, $d \in \mathbf{R}^+$ tq $(v + d) \models G$, $v' = [\lambda \mapsto 0](v + d)$,*
- *soit $a = \delta$, $s = s'$ et $\exists d \in \mathbf{R}^+$ tq $v' = v + d$.*

Notre définition est légèrement différente de celle qui peut être trouvée dans [AD94], car nous prenons en compte les laps de temps nécessaires pour atteindre une région d'horloges depuis une autre, qui sont modélisés par des transitions étiquetées par δ. Comme les automates temporisés, les graphes des régions sont souvent utilisés dans les techniques de vérification [AD94, DY96, SFB00] et de test de systèmes temporisés [ENDKE98, ENFDE97, FPS00].

Notons que cette transformation demande un coût : ainsi pour un automate temporisé $\mathcal{A} =< \Sigma_\mathcal{A}, S_\mathcal{A}, s^0_\mathcal{A}, C_\mathcal{A}, E_\mathcal{A} >$, celui-ci est proportionnel à $(card(S_\mathcal{A}) + card(E_\mathcal{A})).2^{\delta(A)}$, avec $\delta(A)$ la longueur de l'encodage binaire des constantes de chaque contrainte d'horloges [AD94].

L'inconvénient majeur de ce modèle est sans aucun doute le nombre de régions d'horloges engendrées et donc le nombre d'états qui peut être exponentiel. Pour éviter ce nombre exponentiel d'états, l'algorithme décrit dans [ACH+92] génère le graphe des régions minimisé sur son ensemble d'états à partir de l'automate temporisé. D'après les auteurs, ceci permet, dans la plupart des cas, d'éviter l'obtention d'un ensemble d'états exponentiel. Cette minimisation est utile tant pour la phase de vérification que pour celle du test.

L'automate temporisé tel qu'il est décrit dans [AD94] ne permet pas de faire la distinction entre les symboles d'entrée et de sortie. Or ces symboles sont nécessaires pour faire la distinction entre ce qui peut être émis et observé depuis un système. De plus, il ne permet pas de comparer les horloges entre elles dans les contraintes d'horloges. Pourtant ceci permettrait de modéliser plus de systèmes. Ainsi, dans ce qui suit, nous utiliserons un automate étendu à l'automate temporisé, qui se distingue par son alphabet et par l'ajout de contraintes d'horloges.

Définition 3.1.7 (Automate Temporisé Étendu) *Un Automate Temporisé Étendu est un automate temporisé* $\mathcal{A} =< \Sigma_\mathcal{A}, S_\mathcal{A}, s^0_\mathcal{A}, C_\mathcal{A}, E_\mathcal{A} >$, *où :*

- $\Sigma_\mathcal{A}$ *contient des symboles d'entrées (commençant par "?") et des symboles de sorties (commençant par "!").*
- $\Phi(C_\mathcal{A})$ *est l'ensemble des contraintes d'horloges sur* $C_\mathcal{A}$ *tel que chaque contrainte d'horloges* δ *soit de la forme* $\delta = x_i \ op \ c \ (op \in \{\leq, \geq, <, >\}) \mid x_i \ op \ x_j + c \ (op \in \{\leq, \geq, <, >\}) \mid \neg\delta \mid \delta_1 \wedge \delta_2$ *où* $c \in \mathbb{Q}, x_i \in C_\mathcal{A}, x_j \in C_\mathcal{A}$.

Dans la suite de ce manuscrit, dans un souci de simplification des notations, nous appellerons automate temporisé un automate temporisé étendu.

Un exemple d'automate temporisé étendu, construit avec deux horloges, est représenté en figure 3.11.

Cet automate peut ensuite être traduit en un graphe des régions minimisé, qui est illustré en Figure 3.12. Les régions d'horloges associées à ce graphe sont illustrées en Figure 3.13. Celles-ci peuvent aussi être représentées par les systèmes d'inéquations suivants :

Région d'horloges	Système d'inéquations
R_1	$0 \leq x \leq 2$ et $y \leq x$
R_2	$x = 2$ et $0 \leq y \leq 2$
R_3	$y < 2$ et $y \geq 2$
R_4	$y > 2$ et $0 \leq x \leq 2$
R_5	$x > 2$ et $y < 2$
R_6	$x > 2$ et $y > 2$

TAB. 3.1 – Liste des régions d'horloges

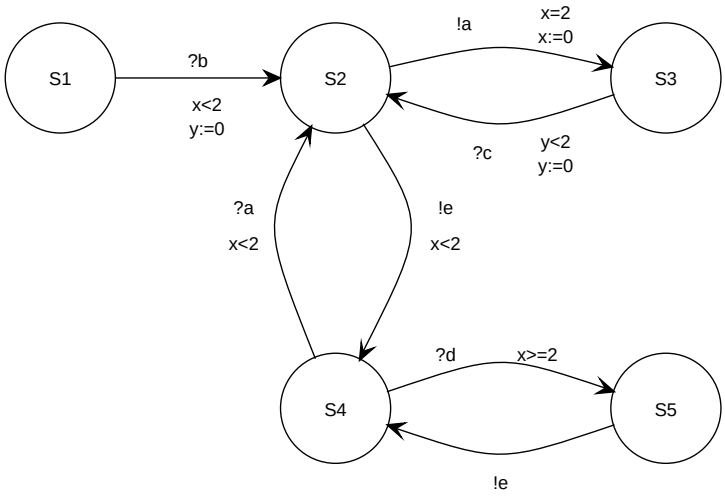


FIG. 3.11 – Automate temporisé

Les automates à événements d'horloges(Event Clock Automata)

Les auteurs de [AFH99] ont montré les deux propriétés suivantes :

1. La transformation d'un automate non déterministe en un automate déterministe équivalent est indécidable.
2. L'inclusion de langage de deux automates temporisés est indécidable.

Pour satisfaire ces deux propriétés, les auteurs ont proposé un autre modèle, appelé l'automate à événements d'horloges (Event Clock Automata) comme étant une sous-classe des automates temporisés.

Définition 3.1.8 *Un automate M à événements d'horloges est un tuple $< Act, N, l_0, E >$ tel que Act est l'ensemble des actions, N est un ensemble fini d'états, l_0 est l'état initial, et $E \subseteq N \times G(X) \times Act \times N$ est l'ensemble des transitions.*

$X = \{x_a \mid a \in Act\}$ est l'ensemble des horloges. La garde $G(X)$ est construite par $G = \gamma \mid g \wedge g$, avec γ une contrainte d'horloges.

Tout comme les automates temporisés, les automates à événements d'horloges modélisent la notion de temps grâce aux contraintes d'horloges sur les transitions. La grande différence avec ces derniers, est que chaque action a est associée à une seule horloge appelée l'horloge d'événements de a. De plus, cette horloge est automatiquement réinitialisée au moment de l'exécution de a. Elle permet ainsi d'enregistrer la valeur de temps écoulé entre l'initialisation du système et l'exécution de a, ou entre deux exécutions de a.

Bien que satisfaisant les deux propriétés d'indécidabilité précédentes, ce modèle n'est pas exempt de défauts. En effet, d'après la définition, une horloge doit être modélisée pour chaque action. Cette propriété peut donc induire un nombre d'horloges beaucoup plus important que si le système avait été décrit par un automate temporisé. Et ceci, peut augmenter le coût d'une analyse ou de l'application d'algorithmes sur le système [DY96].

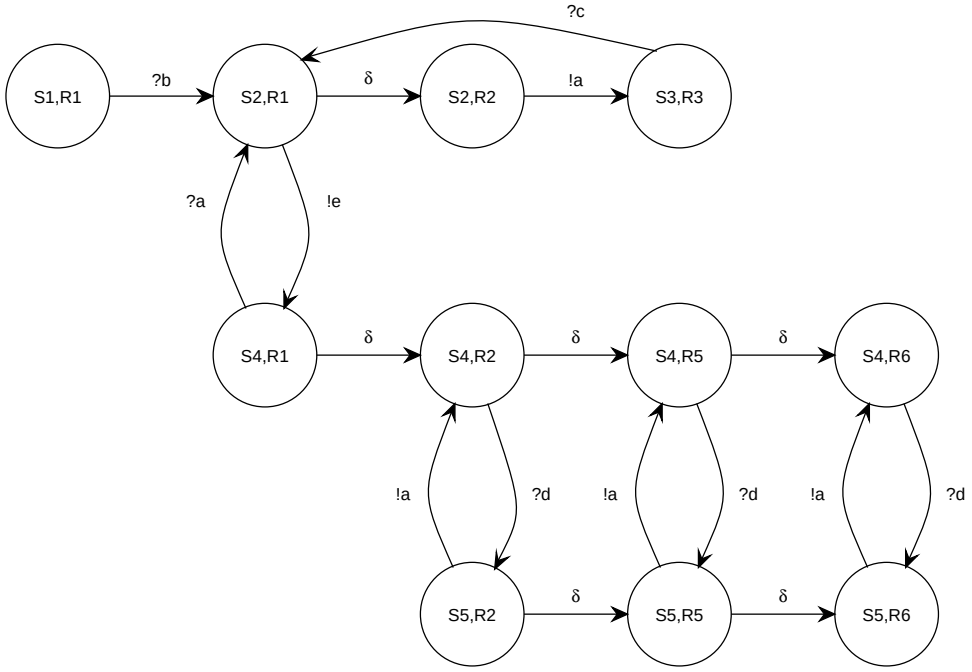


FIG. 3.12 – Un graphe des régions minimisé

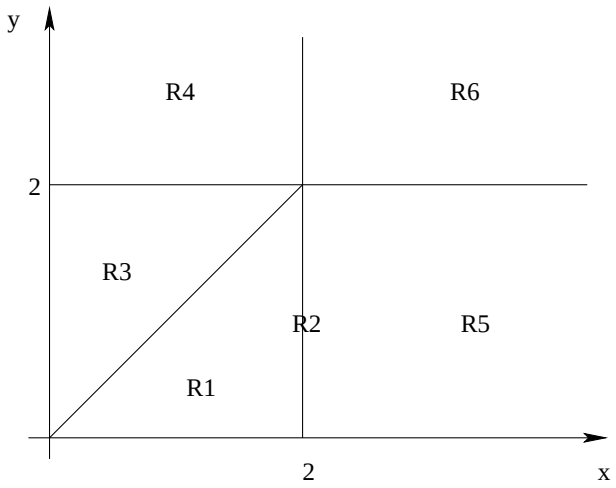


FIG. 3.13 – Régions d'Horloges

De plus, lors de l'exécution du test, pour chaque action, chaque horloge référencée dans les contraintes d'horloges doit être interrogée. Si le nombre d'horloges utilisées pour construire ces contraintes est important, il n'est plus possible d'interroger ces horloges dans un temps négligeable. De plus, les automates à événements d'horloges ne peuvent pas modéliser tout type de système temporisé.

Automate Parallèle

Les applications distribuées ou parallèles sont généralement modélisées par plusieurs automates qui décrivent chaque trace d'exécution ou chaque processus. Dans ce cas, la complexité à exprimer la synchronisation entre les automates devient un problème majeur, sans compter ensuite la complexité à vérifier et à tester de tels systèmes.

Dans [HM01], les auteurs ont proposé une alternative qui consiste à réunir en un seul automate, appelé l'Automate Parallèle (Parallel automaton), chaque trace d'exécution du système. Pour ce faire, les auteurs ont introduit la notion d'états privés, nécessaires pour différencier ces traces. D'où la définition suivante :

Définition 3.1.9 *Soit un ensemble fini d'événements ou conditions, notamment temporelles, qui peuvent être produits en parallèle et soit un ensemble d'actions qui peuvent être exécutées en parallèle. A chaque action et chaque événement, est associé un état privé, représentant leur nombre d'occurrence dans le système. Ce nombre d'états est fini. L'automate résultant est un automate parallèle.*

L'exécution d'un automate parallèle peut être vue comme l'exécution de plusieurs processus, chacun d'eux ayant un ensemble d'actions et d'événements à exécuter. Chaque processus est de la forme :
$act1 \rightarrow < cond1, PS1 > \rightarrow act2 \rightarrow < cond2, PS2 > \rightarrow act3$, ce qui signifie qu'une action $act1$ produit une condition $cond1$ associée à l'état $PS1$ et que l'occurrence de cette condition et de cet état produit l'action $act2$. Chaque processus est, quant à lui, synchronisé grâce à ces couples $< condition, etat >$.

Il semble intéressant de pouvoir rassembler tout un système distribué en un seul automate. Cette vue générale de la spécification peut faciliter le test et la définition d'une méthode. Mais avant de parler de test, il est nécessaire qu'un automate parallèle puisse être vérifié. Or aucune méthode ne le permet à l'heure actuelle.

3.2 Méthodes de génération de test de systèmes temporisés

Nous rappelons dans cette section, la description de quelques méthodes de génération de séquences de test basées sur des systèmes temporisés. Pour chacune des méthodes, nous expliciterons ses avantages et ses inconvénients. Ceci nous permettra de conclure sur ce qui peut être amélioré dans le but de réduire le coût du test et de mieux couvrir le système par les séquences de test.

Dans le domaine académique, depuis une dizaine d'année, l'automate temporisé (et ses extensions) est le modèle le plus employé au niveau du test. Nous présentons donc des méthodes de génération de séquences de test, basées sur ce dernier. Pour faire une analogie avec le chapitre précédent, nous réunissons ces méthodes en catégories : une première rappelle le fonctionnement de méthodes orientées objectif de test, une deuxième illustre des méthodologies utilisant des méthodes de test non temporisé (telle que la méthode Wp), une troisième décrit une méthode orientée testeur canonique temporisé, et nous finirons par une méthode basée sur l'automate à événements d'horloges (qui, rappelons-le, est un sous ensemble des automates temporisés).

Les méthodes de test temporisé ayant pour but de vérifier des propriétés comportementales mais aussi temporelles dans l'implantation, celles-ci peuvent détectées de nouvelles erreurs temporelles. Un nouveau modèle de fautes a donc été défini. Celui-ci est décrit ci-après.

3.2.1 Modèle de fautes

Au niveau des modèles temporisés, de nouvelles fautes liées à cet aspect peuvent figurer dans une implantation sous test. Le modèle de fautes des systèmes non temporisés (section 2.2.1) a donc été étendu dans [PF00, Pet00] sur le modèle des automates temporisés.

Le nouveau modèle de fautes obtenu est donc le suivant :

1. *erreur de sortie, erreur de transfert, état extra* qui sont les erreurs détectables sur les systèmes non temporisés, (voir section 2.2.1).
2. *erreur de dépassement de délai*, qui se produit lorsque l'implantation ne renvoie pas de symbole de sortie alors que l'un d'eux est attendu dans un certain délai. Dans ce cas, une source d'erreur peut être une contrainte d'horloges plus grande sur l'émission de ce symbole par l'implantation.
3. *erreur d'entrée temporelle*, qui se produit si l'implantation refuse un symbole d'entrée pour une valuation d'horloges décrite comme valide pour la spécification, c'est à dire satisfaisant l'émission de ce symbole. L'une des causes de cette erreur peut être une contrainte d'horloges plus forte sur la réception de ce symbole dans l'implantation. Ce type d'erreur, difficilement détectable, peut se manifester par le blocage du système.
4. *erreur de sortie temporelle*, qui se produit lorsque l'implantation émet un symbole de sortie à un moment reconnu comme non valide dans la spécification. Cette erreur, proche de l'erreur de dépassement de délai peut être induite pour les mêmes raisons que cette dernière.

Au niveau du modèle du graphe des régions, une autre erreur, concernant l'incapacité à atteindre certaines valuations d'horloges, peut être détectée. Celle-ci, appelée erreur d'inaccessibilité à une région d'horloges, se produit si lors du franchissement d'une transition, les horloges ne peuvent prendre une valeur de la région d'horloges étiquetée dans l'état d'arrivée.

3.2.2 Méthodes orientées Objectif de test

Étant donné que les systèmes temporisés comportent des propriétés comportementales et temporelles, il serait intéressant de pouvoir vérifier qu'un sous ensemble de ces dernières existe dans l'implantation sous test, par l'intermédiaire d'un objectif de test temporisé. Quelques méthodologies [PLC98, FPS00] ont été proposées dans cette optique.

La méthodologie décrite dans [PLC98] permet de vérifier la satisfaction d'un objectif de test, modélisé par un automate temporisé. L'objectif de test peut être composé de toute propriété inclue dans la spécification : c'est-à-dire qu'il peut être composé de tout symbole inclus dans l'alphabet de la spécification et que chacune de ses contraintes d'horloges Ct est telle que toute valeur mettant à vrai Ct met aussi à vrai une contrainte d'horloges de la spécification. Pour générer les séquences de test, une synchronisation est effectuée entre la spécification et l'objectif : cette synchronisation se réalise premièrement sur les actions, puis sur les contraintes d'horloges. Des intervalles de temps synchronisés, satisfaisant une action de la spécification et une action de l'objectif de test sont alors construits. Ainsi, lors de la phase de test, si chaque action d'une séquence de test s'exécute dans son intervalle respectif alors l'implantation satisfait l'objectif de test. Bien qu'elle soit rapide à mettre en oeuvre, cette méthode ne permet pas l'utilisation de contraintes permettant la comparaison d'horloges [PLC98], ce qui réduit le nombre de systèmes pouvant être testés par cette méthode.

La méthodologie décrite dans [FPS00] donne une solution à ce problème. L'objectif de test et la spécification sont toujours modélisés par des automates temporisés. Avant de générer les séquences de test, l'objectif de test et chaque chemin de la spécification incluant cet objectif sont transformés en graphe des régions. Puis les graphes des régions obtenus à partir de la spécification sont synchronisés avec l'objectif de test. Aucune région d'horloges synchronisée n'est générée car, par hypothèse, les contraintes d'horloges de l'objectif de test doivent exactement se trouver dans la spécification. Ainsi, il est impossible de spécifier une quelconque contrainte d'horloges dans l'objectif de test. C'est d'ailleurs l'inconvénient majeur de cette autre méthode.

En plus des inconvénients cités, ces méthodologies ne permettent pas d'utiliser des objectifs de test refusant. Ainsi, il est impossible de vérifier la non existence d'une ou plusieurs propriétés comportementales ou temporelles, n'appartenant pas à la spécification, dans l'implantation sous test. Une solution à ces inconvénients est présentée dans le chapitre 5.

3.2.3 Méthodes basées sur la caractérisation d'états

Plusieurs méthodologies [ENFDE97, PF00, ENDKE98, Pet00] ont proposé d'utiliser la méthode Wp, sans modification, afin de générer les séquences de test sur un automate temporisé. La première question que le lecteur peut se poser est la suivante : de quelle manière le temps peut-il être pris en compte sachant que par définition la méthode Wp prend en entrée des automates où la notion de temps n'est pas considérée ?

En fait, ces méthodologies traduisent cet automate temporisé en un autre automate dans lequel le temps n'est pas vu comme une notion propre mais comme une donnée quel-

conque. C'est lors de l'exécution du test, que les propriétés temporelles seront considérées et testées. Une fois l'automate temporisé transformé, il ne reste plus qu'à appliquer une méthode de test classique, comme par exemple la méthode Wp, pour obtenir les séquences de test.

Dans [Pet00], la transformation de l'automate temporisé en automate pouvant être accepté par les méthodes de test non temporisé, est composée des quatre phases suivantes :

1. La spécification est premièrement transformée en un graphe des régions minimisé sur l'ensemble de ces états, grâce à l'algorithme décrit dans [ACH+92]. Ceci a pour but de déplacer les contraintes d'horloges des transitions vers les états. Le graphe des régions obtenu est, bien sûr, équivalent à l'automate précédent.
2. Le graphe des régions minimal est ensuite traduit en un automate plat (flatten automaton). Cette transformation est nécessaire pour que l'automate soit par la suite accepté, par une méthode de test non temporisé (W, W_p, UIO). Celle-ci est obtenue en renommant les étiquettes du graphe selon les règles suivantes :
   - si la transition est étiquetée par le symbole δ, représentant le passage du temps, la référence de la région d'horloges de l'état d'arrivée est étiquetée à la transition,
   - si la transition est étiquetée par un symbole d'entrée, la référence de la région d'horloges de l'état de départ est étiquetée à la transition,
   - si la transition est étiquetée par un symbole de sortie, les références des deux régions d'horloges (états de départ et d'arrivée) sont étiquetées à la transition.
3. L'automate obtenu est maintenant très proche d'un LTS. Or la méthode Wp n'accepte que des FSM. Ainsi, une dernière transformation consiste à réunir les symboles d'entrée et de sortie en couple entrée/sortie.

Prenons l'exemple d'une transition $(s_i, z_k) \xrightarrow{!B} (s_j, z_l)$. Celle-ci, d'après la troisième règle, devient donc $(s_k, z_k) \xrightarrow{k!Bl} (s_j, z_l)$.

Dans [ENDKE98], les auteurs proposent une autre transformation de l'automate temporisé. Ce dernier est traduit en un Grid Automaton qui est une version sémantiquement réduite des automates temporisés. Des règles, proches de celles utilisées par la méthode précédente, permettent ensuite de traduire ce grid automaton en une FSM.

L'avantage principal de ces méthodes est l'utilisation d'une autre méthode classique , la Wp, qui est déjà connue et utilisée, et qui permet la plus grande détection des fautes non temporelles. Ainsi, il est assuré que les fautes de transfert et de sortie seront détectées. De plus, la relation de conformité ne nécessite pas d'être redéfinie. Cette adaptation permet donc de réutiliser ce qui est connu et déjà implanté dans des outils. Cependant, le fait d'appliquer plusieurs transformations alourdit, parfois considérablement, le coût de test. D'autant plus que certaines méthodologies [ENDKE98] apportent d'autres transformations. Nous proposerons ainsi dans le chapitre 5 une nouvelle méthodologie utilisant la caractérisation d'états temporisés, qui évite toutes ces transformations et réduit le coût de la génération des séquences de test (sous certaines hypothèses).

La méthodologie décrite dans [COG98] utilise aussi la caractérisation d'états pour générer des séquences de test. Les systèmes sont modélisés par des systèmes de transitions temporisés qui consistent en des règles de la forme : "If C Then B Between L and U",

avec C une contrainte sur des variables, B une action, L et U des bornes de temps. Le temps est représenté par une horloge globale qui ne peut être reinitialisée. Ces règles sont ensuite transformées en des séquences de transitions, étiquetées par un symbole d'entrée ou de sortie, un écoulement de temps et une action modélisant une règle.

La caractérisation des états de ces chemins s'effectue en considérant les transitions sortantes de chaque paire d'états. Cette caractérisation offre l'avantage d'une utilisation directe de la spécification mais ne permet pas d'être appliquée sur des automates temporisés. En effet, ces derniers autorisent les réinitialisations d'horloges, modélisent le temps par des contraintes sur plusieurs horloges, et décrivent des spécifications par un graphe au lieu d'un ensemble de séquences distinctes. Et ces propriétés apportent de nombreux cas qui ne peuvent être pris en compte par cette caractérisation.

3.2.4 Méthodologie orientée testeur canonique temporisé.

Cette méthode de test d'implantations temporisées décrite dans [LC97], est basée sur le testeur canonique classique. Le but est d'ajouter la notion de temps dans le testeur canonique et de vérifier qu'une relation d'implantation est valide, tout en respectant des contraintes d'horloges.

Cette méthodologie utilise, pour modéliser les spécifications temporisées, des ETIOSM (Extended Timed Input Output State Machine). Il s'agit en fait d'une autre extension des automates temporisés où les symboles d'entrée et de sortie sont séparés et à laquelle est ajouté un ensemble de variables non temporelles sur lesquelles peuvent être appliquées des fonctions quelconques.

Cette méthode de test utilisant une approche par testeur canonique, une relation d'implantation $R(I, S)$ est définie, de telle manière que lorsqu'elle est respectée, l'implantation I est conforme à la spécification S. Pour cette méthode, la relation proposée est basée sur les traces des automates :

$$R(I, S) \Leftrightarrow Trace(I) = Trace(S)$$

Le testeur canonique généré doit intégrer les aspects temporels. Pour ce faire, les auteurs de [LC97] proposent de découper la spécification en sous automates ne contenant qu'une horloge, puis de générer des intervalles de temps satisfaisant les contraintes de chaque transition obtenue. Ceci est résumé avec les étapes suivantes :

1. Des sous automates, qui ne prennent en considération qu'une seule horloge chacun, sont extraits de la spécification. Ainsi, autant d'automates que d'horloges sont donc créés. Le fait d'extraire ces automates permet d'en obtenir des plus petits qui, selon les auteurs, sont plus simples à analyser.
2. Un testeur canonique est généré pour chaque sous automate. A chaque transition sont construits :
   - un symbole d'entrée (si nous avons une réception dans l'automate), un symbole de sortie (si nous avons une émission),
   - la même condition concernant les variables non temporelles,
   - la valeur de temps minimum pour atteindre la transition, ainsi que la valeur de temps maximum.

3. Les petits testeurs canoniques sont ensuite fusionnés pour obtenir le testeur canonique temporisé.

Les valeurs de temps minimum (Tmin) étiquetées sur chaque arc sont calculées grâce à l'algorithme de *Dijkstra* [GFS90]. Pour cela, chaque arc est au préalable valué par une borne de temps minimale satisfaisant la contrainte d'horloges de l'arc. Puis, l'algorithme de *Dijkstra* recherche le chemin donnant un temps minimal Tmin pour atteindre l'arc.
Quant aux valeurs de temps maximal (Tmax), elles sont calculées en partant de la transition ayant la plus grande borne supérieure (max) dans sa contrainte d'horloges (c'est généralement l'état terminal, s'il existe). Puis, il faut parcourir dans le sens inverse les chemins de l'automate et à chaque transition, non encore parcourue, il faut lui donner un temps maximal qui dépend du temps maximal de la transition suivante et de la borne supérieure de la contrainte de cette transition.
Une fois le testeur canonique généré, il ne reste plus qu'à construire les tests envoyés à l'implantation. Dans cette méthode, ils sont écrits dans le langage TTCN (Tree and Tabular Combined Notation) [ISO91a].

Le premier intérêt de cette méthode est qu'elle est complètement implantable et utilisable en pratique. De plus, le temps de calcul du testeur canonique reste très inférieur au temps qu'il faut pour générer des séquences de test avec une méthode W. Cependant, elle possède de nombreux défauts. En premier lieu, et quelque soit la relation d'implantation prise en compte, sa couverture de faute est plus faible que celle obtenue avec une méthode W, Wp. Il n'est donc pas garanti que toutes les fautes soient découvertes. Ensuite, il est impossible de comparer les horloges entre elles dans les contraintes d'horloges. En effet, il est alors impossible de calculer l'intervalle de temps maximum (Tmax) ou minimum (Tmin) pendant lequel une interaction peut se produire. Seul le calcul des régions d'horloges nous permet de répondre à de telles contraintes.

3.2.5 Méthode de test basée sur les automates à événements d'horloges

Les auteurs de [NS01] présentent une autre technique de génération de séquences de test depuis une spécification modélisée par un automate à événements d'horloges. Pour ce faire, une relation de conformité est définie sur la théorie des tests d'Hennessy [DNH84].

Les tests d'Hennessy supposent que l'implantation I (et la spécification S) peut être observée par un ensemble fini de tests T via une séquence de communications synchrones de type CCS (algèbre de processus, Calculus of Communicating Systems). Ainsi, l'exécution d'un test est composée d'une séquence finie de communications formant une *Computation*, notée $Comp(T||I)$ (ou $Comp(T||S)$). Un verdict est attribué à une exécution de test, et une computation est satisfaite si elle se termine après une observation donnant le verdict Pass. La relation de conformité est définie suivant ces computations et est la suivante :

Définition 3.2.1 (Relation de conformité $\sqsubseteq_{te}$) *1. S must T ssi $\forall \Sigma \in Comp(T||S), \Sigma$ est satisfaite*
2. S may T ssi $\exists \Sigma \in Comp(T||S), \Sigma$ est satisfaite
3. $S \sqsubseteq_{must} I$ ssi $\forall T, S$ must $T \Rightarrow I$ must T

4. $S \sqsubseteq_{may} I$ *ssi* $\forall T, S$ *may* $T \Rightarrow I$ *may* T
5. $S \sqsubseteq_{te} I$ *ssi* $S \sqsubseteq_{must} I$ *et* $S \sqsubseteq_{may} I$

Plus simplement, I est conforme à S (noté $S \sqsubseteq_{te} I$) ssi pour une trace Σ de $Comp(T||S)$ avec T l'ensemble des séquences de test, si Σ peut toujours être obtenue à partir de S, alors Σ doit être obtenue à partir de I, et si Σ peut être obtenue à partir de S, alors Σ peut être obtenue à partir de I.

Pour générer les séquences de test, permettant de vérifier cette relation de conformité, les auteurs proposent les phases suivantes :

1. l'automate de départ est traduit en un graphe, très proche des graphes des régions. Pour cela, une classe d'équivalence est au préalable définie : celle-ci rassemble les valuations d'horloges satisfaisant les mêmes franchissements de transitions. Les ensembles de ces valuations d'horloges, décrits par des systèmes d'inéquations, sont ensuite étiquetés dans les états de l'automate, ces derniers pouvant être divisés si plusieurs ensembles existent.
2. A partir de ce graphe, un autre graphe, appelé graphe d'accessibilité est construit. Celui-ci ne rassemble que les valuations d'horloges qui peuvent être atteintes suivant les exécutions de la spécification.
3. Enfin, les séquences de test sont extraites de ce graphe en accord avec les tests d'Hennessy [DNH84].

L'avantage d'utiliser cette méthodologie est en fait extrêmement lié au modèle de l'automate à événements d'horloges. En effet, le fait d'implanter une horloge unique par action, permet une génération simplifiée et moins lourde du graphe de la phase 1, par rapport à la génération du graphe des régions. Il est possible de plus de spécifier un système non déterministe, ce que peu de méthodes de génération de séquences de test temporisé permettent. Cependant, comme nous l'avons précédemment dit, ce langage ne permet pas de spécifier tout type de système temporisé. Il ne permet pas non plus de comparer les horloges entre elles dans les contraintes d'horloges. Ainsi, si le système ne peut être décrit avec ce type d'automate, alors un autre modèle ainsi qu'une autre méthodologie devront être utilisés.

3.3 Architecture de test

Tester des systèmes temporisés implique aussi de prendre en compte la notion du temps lors de l'exécution des séquences de test. Il était donc impératif d'inclure le facteur temps au niveau des architectures de test. En effet, connaître l'état des horloges, lors de la phase de test, est essentiel pour valider la conformité de l'implantation.

Une première architecture a été proposée dans [ENFDE97]. Celle-ci, s'inspirant de l'architecture définie dans [ISO91a], suppose que l'implantation est composée de deux parties : une partie de contrôle gérant le flot de données du système (émission et réception de symboles) et une autre partie gérant les horloges. Puis, il est supposé que le testeur peut avoir un accès direct à chaque partie, notamment aux horloges. Or, dans [Pet00], les auteurs ont montré qu'une telle architecture n'est pas utilisable si la gestion

du temps dans l'implantation doit être réalisée exactement de la même manière que dans la spécification. De plus, l'implantation ne peut plus être vue comme une boîte noire ce qui est en contradiction avec le test.

Ainsi, une seconde architecture a été décrite dans [PF99b]. Celle-ci considère ne rien connaître de la modélisation du temps à l'intérieur de l'implantation, ce qui offre un plus grand degré de réalisme. Celle-ci devient donc une véritable boîte noire. Cette fois, le testeur est divisé en deux parties communiquant l'une avec l'autre : une partie de contrôle et une partie horloges composée du même nombre d'horloges inclues dans la spécification. Cette dernière partie a pour but de simuler les horloges de l'implantation. Ceci implique que le testeur doit de plus gérer les horloges lors de la phase de test, c'est-à-dire qu'il doit y appliquer les réinitialisations éventuelles. Quelques hypothèses supplémentaires sont cependant nécessaires. Ainsi, les horloges du testeur sont supposées être synchronisées sur celles de l'implantation et sont supposées croître de la même manière que ces dernières. La Figure 3.14 illustre cette architecture.

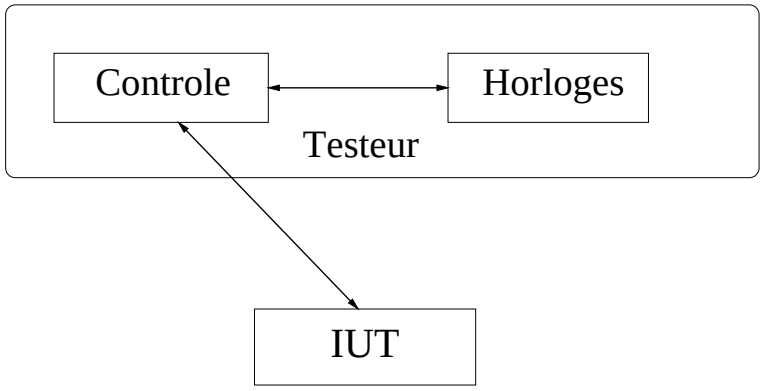


FIG. 3.14 – Architecture de test temporisée

3.4 Outils de Vérification

Cette section est vouée à la description de quelques outils de vérification de systèmes temporisés.

3.4.1 KRONOS

KRONOS est un outil développé par Verimag (Grenoble, France) procédant à la vérification des systèmes temps réel complexes. Dans KRONOS, les composantes du système temps réel sont modélisées par des automates temporisés. Les propriétés à vérifier sont, quant à elles, décrites par la logique temporelle TCTL qui représente le temps d'une manière continue (section 2.1.1).

KRONOS vérifie si un automate temporisé satisfait ou non une formule décrite en TCTL contenant des propriétés comportementales et temporelles. L'algorithme de model-checking est basé sur une représentation symbolique de l'espace infini d'états par des

ensembles linéaires de contraintes. Un guide complet de son utilisation est décrit dans [OY93].

De nombreux outils gravitent autour de KRONOS. Par exemple, OptiKron implante une méthode permettant d'optimiser les automates temporisés en réduisant leurs nombres d'horloges [DY96]. MINIM est un autre outil qui permet de minimiser un automate temporisé avec respect de la bi-simulation à temps abstrait.

3.4.2 UPPAAL

UPPAAL est un outil de vérification, résultant d'une collaboration entre l'équipe BRICS (Université d'Aalborg, Danmark) et le Département of Computing Systems (Université d'Uppsala, Suède). Il permet de modéliser, simuler et vérifier des systèmes temps réel, en prenant partie de la théorie des automates temporisés. Plus précisément, il est approprié pour des systèmes qui peuvent être modélisés comme une collection de processus non déterministe composé de structures de contrôle finies et d'horloges prenant des valeurs réelles, chaque processus pouvant communiquer par des actions de synchronisation.

UPPAAL est composé de trois parties principales :

- une partie de description de langage, implantée par une interface graphique permettant de décrire un système.
- une partie de simulation, qui est un outil de validation permettant d'examiner les exécutions dynamiques possibles pendant la phase de modélisation et fournissant un moyen de choix de propriétés à vérifier dans le système.
- une partie de vérification, offrant la vérification de propriétés invariantes et accessibles en explorant l'espace des états du système.

Une description plus détaillée d'UPPAAL peut se trouver dans [BLL+95, RLP+98].

3.4.3 HyTech

HyTech est un autre outil de vérification de systèmes hybrides développé par le Département of Electrical Engineering and Computer Sciences (Université de Berkeley, Californie). Les systèmes hybrides sont des automates modélisant des systèmes temps réel avec représentation du temps continue, où les conditions sur les horloges sont étiquetées dans les états.

HyTech prend en entrée un système distribué temporisé décrit par des systèmes hybrides. A partir de ces derniers, il forme une composition parallèle ne donnant qu'un seul automate. HyTech est aussi composé d'un langage de description d'analyse qui permet d'écrire simplement des programmes itératifs pour produire des tâches telles que l'analyse d'accessibilité de composantes ou de parties de temps du système ou la génération de traces d'erreurs.

HyTech en est à sa troisième version, et selon les comparaisons décrites par ses auteurs serait l'un des outils de vérification les plus rapides actuellement. Une description détaillée de HyTech est donnée dans [HHWT95].

3.4.4 REAL-TIME SPIN

Real-Time Spin (RT-Spin)[TC96] (développé à Verimag) est une extension de l'outil de vérification Spin (Laboratoires Bell [Hol90]) avec l'ajout de la notion du temps continu.

Le langage accepté par T-Spin est Real-Time Promela qui lui-même est une extension de Promela. Promela est proche du langage C auquel sont ajoutés plusieurs opérateurs exprimant la communication, et la synchronisation de modules du système.

Comme Spin, RT-Spin fonctionne comme un générateur de code : il prend donc en entrée une spécification écrite en RT-Promela et génère un ensemble de fichiers C qui sont ensuite compilés dans un analyseur. Dans le cas de RT-Spin, les fichiers C compilés doivent aussi être liés avec plusieurs objets qui implantent des opérations symboliques sur des DBM (matrices représentant l'information du temps). L'analyseur est ensuite exécuté avec des options variées (Spin est sans doute l'outil offrant le plus d'options) pour vérifier la spécification. Les expressions à vérifier sont décrites par la logique LTL qui est une logique linéaire temporelle.

3.5 Conclusion

On remarque que le test temporisé est difficile à réaliser, aussi bien pour un néophyte que pour un expert.

D'une part, le choix d'une modélisation s'impose, celle-ci devant intégrer une représentation adéquate du temps par rapport au système. D'autre part, la génération et l'application de séquences de test sont plus complexes à mettre en oeuvre à cause de l'ajout de propriétés temporelles. On pourrait penser qu'après tout, les horloges ne sont que des variables, et qu'il est possible de les manipuler comme telles. Or, elles sont indépendantes, progressent inlassablement, elles se réinitialisent, se comparent. Et en appliquant ces propriétés au comportement d'un système, nous obtenons un maelstrom de possibilités de comportements souvent indiscernables par une observation externe à l'implantation. C'est pourquoi le test est difficile, et c'est pourquoi, affirmer que l'ensemble d'un système est couvert par les tests, demande une étude sérieuse et approfondie.

Dans cette optique, le chapitre 4 définit et évalue la qualité de test des automates temporisés et des graphes des régions pour étudier la partie du système qui a la possibilité d'être testée et mesurer un coût moyen du test. Le chapitre 5, quant à lui, présente deux nouvelles méthodologies de génération de séquences de test qui apporteront une réponse à certains manques et problèmes rencontrés.

Chapitre 4

Qualité de Test des Systèmes Temporisés

Le coût de la validation d'un système dépasse souvent 70% du coût total de son développement, et ne cesse de croître de par la complexité de plus en plus accrue de ces systèmes et l'ajout de notions telles que le temps et le multimedia. Il est, par conséquent, nécessaire de proposer des méthodes permettant de réduire ce coût tout en obtenant des systèmes validés. Lorsque nous parlons de qualité pour un système, nous sous-entendons généralement la robustesse, la flexibilité, les performances,... Or un autre critère de qualité, essentiel pour réduire le coût de la validation est la qualité de test, aussi communément appelée, testabilité. Cette notion repose sur une analyse et une évaluation du système qui permettent d'orienter les opérations de test, d'en réduire le coût, la durée et d'en assurer l'efficacité. Plus précisément, la qualité de test permet, d'une part, une aide à la modification du système en vue d'être plus testable, et d'autre part, de choisir la méthodologie de test la plus adaptée, ce qui globalement facilite le cycle de validation.

La qualité des systèmes non temporisés a inspiré de nombreux travaux qui ont permis une évaluation fine de cette qualité et ont contribué à la création d'outils. L'évaluation de cette qualité a aussi permis d'éradiquer plusieurs problèmes liés au test, comme son blocage. D'autres problèmes, sous-jacents au test de systèmes temporisés ont été soulevés, comme d'autres blocages du test ou une couverture partielle du système. Ceux-ci motivent l'étude, la définition et l'évaluation de la qualité des systèmes temporisés. En vue de ces considérations, nous proposons d'étudier la qualité de test de deux modèles temporisés [SFB00, SF01b, SF01a], qui sont souvent utilisés par des méthodologies de vérification et de test : les automates temporisés et les graphes des régions [AD94] (définis en section 3.1.2). Après avoir décrit un état de l'art de la qualité de test, nous proposerons, pour chacun de ces modèles, plusieurs facteurs évaluant leur qualité de test. Ces facteurs peuvent être évalués avant la génération des séquences de test et sont indépendants de toute méthodologie. Nous montrerons cependant qu'un raffinement peut être apporté si cette méthodologie est prise en compte. Nous développerons aussi, la complexité du calcul des facteurs et l'amélioration qui peut être apportée à la qualité de test. Nous conclurons sur une discussion portant sur les avantages et inconvénients du modèle de l'automate temporisé et celui du graphe des régions, par rapport à la représentation de leurs contraintes temporelles.

4.1 La Qualité de Test : état de l'art

4.1.1 Définition de la Qualité de Test

De nombreuses définitions de la qualité de test peuvent être trouvées dans la littérature, tant pour les logiciels que pour les composants (carte, circuit...). Ayant pris petit à petit de l'importance dans la validation des systèmes, cette notion s'est vue attribuée de nouvelles propriétés dans le but essentiel de réduire les coûts de la validation. Quelques unes des plus pertinentes sont décrites ci-dessous :

Définition 4.1.1 *La testabilité est l'aptitude à générer, évaluer et appliquer les tests de façon à satisfaire les objectifs prédéfinis des tests (couverture de fautes, isolation de fautes,...) qui sont sujets à des contraintes de coût (argent et temps) [Ben94].*

Définition 4.1.2 *La testabilité est la capacité du logiciel à révéler ses fautes lors de l'étape de test [VM95].*

Définition 4.1.3 *La testabilité est :*

(1) *la facilité d'établir des critères de test d'un système ou d'une composante de système et la capacité à déterminer si les critères sont satisfaits,*

(2) *la facilité avec laquelle on peut exprimer les objectifs de test et la capacité des tests à déterminer si de tels objectifs sont satisfaits [oSET90].*

D'autres définitions peuvent être trouvées dans [Kar97]. Parmi ces définitions, nous avons noté que certains points restaient obscurs ou inexistants. Parmi ceux-ci :

- la qualité est avant tout une notion qui est quantifiable. Or, dans les définitions précédentes, les auteurs ne traitent que de qualité optimale.

- le fait que le système relève ses fautes lors du test, sous-entend la prise en compte de la notion d'accès de l'ensemble des actions ou composantes du système.

Ainsi, nous proposons une définition de la qualité de test, qui prend en compte les principes des définitions précédentes et nos extensions. Cette définition peut aussi bien s'appliquer aux logiciels qu'aux composants hardware.

Définition 4.1.4 (Qualité de Test et Système Testable) *La qualité de test d'un système est une quantification évaluant :*
- *le coût du test du système,*
- *l'accessibilité des composantes ou des parties de comportements du système,*
- *la capacité du système à relever ses fautes d'une manière non ambiguë, grâce aux stimulations (contrôles) et aux observations,*
- *la facilité d'établir des critères de test du système et de déterminer si, par l'application du test, ces critères sont satisfaits*

Le système est testable, (la qualité de test du système est optimale) si :
- *le coût du test du système est le meilleur,*
- *toutes les fautes du système peuvent être relevées par stimulation et observation,*
- *les critères de test peuvent être facilement définis et peuvent être vérifiés .*

4.1.2 Cycle de vie du logiciel et qualité de test

L'évaluation de la qualité de test, étant nécessaire pour faciliter le test de systèmes, il est donc logique qu'elle prenne place dans le cycle de vie du logiciel. Un cycle de vie étendu a d'ailleurs été proposé dans [KDC96]. Celui-ci est résumé et illustré en Figure 4.15.

D'après ce cycle de vie, le concepteur peut évaluer la qualité de test du système, à chaque nouvelle étape de sa création. Ainsi, à chacune de celles-ci, il peut avoir connaissance du comportement qui pourra ou ne pourra pas être testé, et d'une idée sur le coût de ce test. Ainsi, il a la possibilité d'améliorer la spécification pour n'obtenir que des propriétés qui puissent être testées, de réduire le coût de la validation du système et d'apprécier la qualité des améliorations apportées. Il peut être aussi satisfait de la qualité obtenue et continuer le processus de conception. Ces mesures peuvent éviter la génération d'implantations qui ne pourront être entièrement testées et qui demanderont une modification de la spécification puis une nouvelle génération d'implantation. Donc ces mesures évitent une perte de temps et d'argent.

La mesure de qualité de test peut aussi être pertinente pour choisir une spécification parmi un ensemble de candidates, décrivant le même comportement.

4.1.3 Degré de qualité de test des systèmes non temporisés

La qualité de test des automates non temporisés dépend de trois notions fondamentales qui sont l'observabilité, la contrôlabilité et le coût du test. Ainsi, Freeman, dans [Fre91], décrit un système testable, comme étant un système observable et contrôlable.

Définition 4.1.5 (Observabilité [Fre91]) *Un système est dit observable si pour tout symbole d'entrée appliqué au système, un symbole de sortie différent est observé*

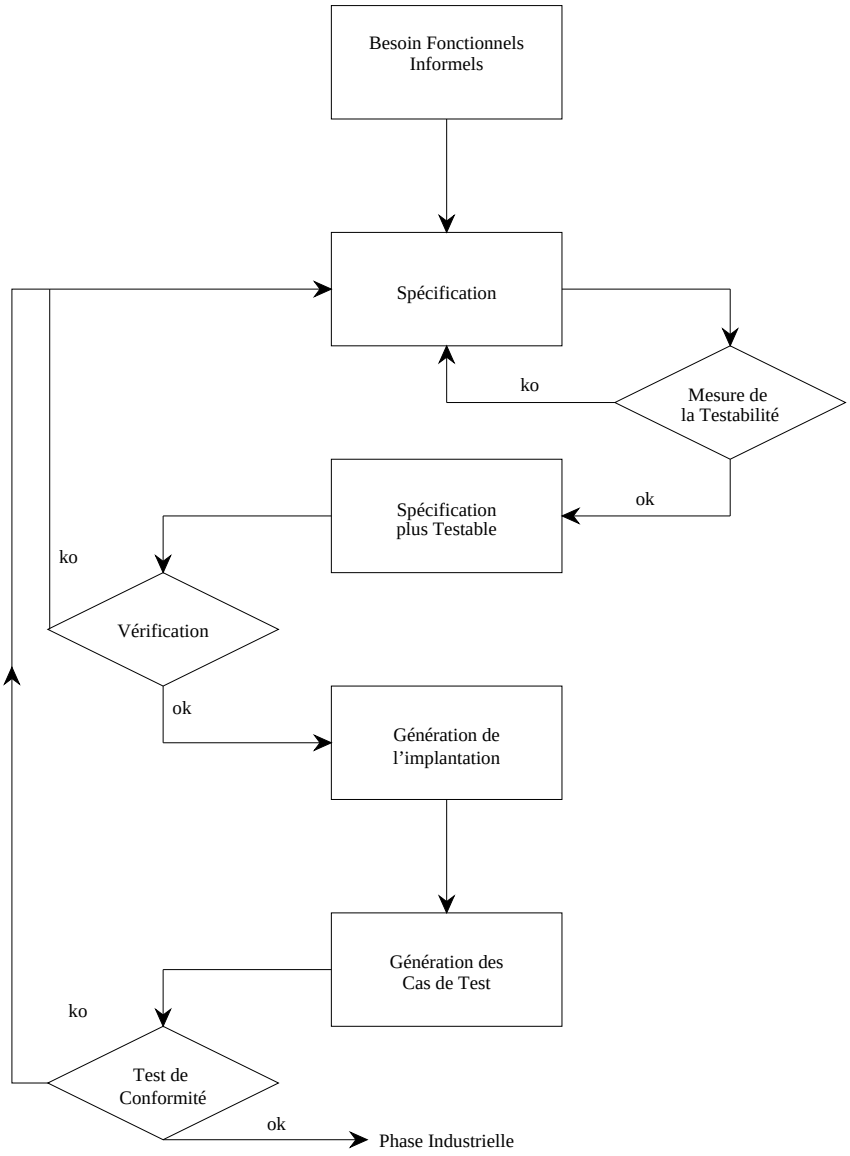


FIG. 4.15 – Cycle de vie étendu

Définition 4.1.6 (Contrôlabilité [Fre91]) *Un système est dit contrôlable si pour chaque symbole de sortie observé, il existe un symbole d'entrée, qui, une fois appliqué au système, force cette observation.*

Ces notions sont généralement dépendantes de plusieurs propriétés du système. Celles-ci ont été analysées dans [KDCK94, KGD96] et ont permis de définir plusieurs facteurs, appelés degrés de qualité, mesurant l'influence de ces propriétés sur la qualité de test du système. L'évaluation des degrés permet ainsi de mesurer l'observabilité et la contrôlabilité de la spécification. Ces propriétés étant dépendantes du modèle choisi pour représenter la spécification, nous rappelons dans ce qui suit, les degrés de qualité obtenus depuis le modèle relationnel et le modèle FSM, (définis en section 2.1.2 et section 2.1.2). Ceux-ci sont résumés ci-dessous :

Degrés de qualité de test basés sur le modèle relationnel

La Définissabilité

Un modèle relationnel est dit complètement défini, si pour chaque symbole d'entrée différent de ϵ, il existe au moins un symbole de sortie différent de ϵ. Un symbole d'entrée non défini provoque une interaction interne qui n'est pas observable. Cette propriété contribue à l'observabilité et peut être évaluée sur chaque symbole d'entrée du système par un degré, noté Def.

L'Informabilité

Un modèle relationnel est dit complètement informé, si pour chaque symbole de sortie différent de ϵ, il existe au moins un symbole d'entrée différent de ϵ qui, une fois appliqué au système, donne ce symbole de sortie. Un symbole de sortie est dit non informé si aucun symbole d'entrée n'est appliqué avant observation. Cette propriété influence donc la contrôlabilité du système et peut être évaluée sur chaque symbole de sortie par un facteur, noté Inf.

Le Déterminisme

Un modèle relationnel est dit déterministe, si à chaque symbole d'entrée est associé un seul symbole de sortie. Si plusieurs symboles de sortie y sont associés, alors le symbole d'entrée est dit non déterministe. Cette propriété influence la contrôlabilité et peut être aussi évaluée par un facteur, noté D.

La Convergence

Un modèle relationnel est dit convergent, si à chaque symbole de sortie est associé un unique symbole d'entrée. Cette propriété influence l'observabilité. Les symboles de sortie qui sont associés à un symbole d'entrée peuvent masquer le fait qu'un état interne de l'implantation soit atteint à la place d'un autre, ce qui diminue l'observation. Le facteur, noté D^{-1} évalue l'influence de la convergence.

Degrés de qualité de test basés sur le modèle FSM

Le Degré de Contrôle

Le Degré de Contrôle C évalue les efforts nécessaires pour atteindre chaque état de la spécification. Ces efforts dépendent de l'ensemble des préambules P, qui n'est généralement pas unique, et de leurs longueurs. Ce facteur mesure ainsi le coût d'une partie de l'exécution de la phase de test, car l'ensemble des préambules constitue un squelette des séquences de test. Donc plus les préambules sont longs, plus le test sera lent.

Le Degré de Flou

Si une spécification n'est pas réduite, c'est-à-dire si elle contient des états non distinguables, alors celle-ci va demander un coût supplémentaire pour être testée (voir [LPvB94]. De plus, le test ne pourra être effectué que grâce à des méthodologies particulières, comme la méthode HSI [LPvB94] (voir section 2.2.3). Soit une FSM M avec lequel est construit une partition $\prod = \{Q_1, ..., Q_k\}$ de l'ensemble des états S contenant les sous-ensembles d'états distinguables deux à deux. Le nombre de sous-ensembles dans la partition minimale $\prod_{min}$ est appelé le degré de flou de M. Ce facteur représente la difficulté à distinguer les états de M. Plus ce facteur est grand, plus les séquences de test seront longues et plus le coût du test sera élevé.

Le Degré de Distinction

Certaines méthodologies de test utilisent un ensemble de caractérisation d'états W pour distinguer les états de la spécification [Cho78, FBK+91, LPvB94]. La longueur et le nombre de séquences de cet ensemble influencent, une fois de plus, le coût du test. Le degré de distinction permet de quantifier ce coût.

Le Degré d'Abstraction

La spécification et l'implantation peuvent être vues comme des représentations différentes du même comportement. Elles appartiennent à différents niveaux d'abstraction. Une implantation est moins abstraite que sa spécification, et comporte généralement plus d'états. Ce nombre d'états influence le coût du test. Le degré d'abstraction, noté AB a pour but d'évaluer ce coût.

Quelques remarques concernant ces facteurs

Les précédents degrés permettent d'évaluer la qualité de test de systèmes par rapport à leurs propriétés comportementales. Certains seront donc repris pour évaluer une partie de la qualité de test des systèmes temporisés. D'autres ne sont pas nécessaires, sachant que nous souhaitons évaluer la qualité de test des systèmes temporisés, indépendamment des méthodologies de test. En effet, le degré de distinction et le degré de flou sont liés aux méthodologies de test utilisant un ensemble de caractérisation. Ces degrés n'ont aucun intérêt pour des méthodologies, par exemple, basées sur les objectifs de test. Nous montrerons aussi, par la suite, que le degré de contrôle ne peut être utilisé tel quel pour les systèmes temporisés. Par conséquent, nous reprenons la Définissabilité, l'Informabilité, le

Déterminisme, la Convergence et le degré d'Abstraction pour évaluer, sur les systèmes temporisés, l'influence de leurs propriétés comportementales.

Ces paramètres sont regroupés dans un vecteur, appelé Vecteur de testabilité, noté $TV =< Def, Inf, D, D^{-1}, AB >$. Ainsi, les quatres premiers degrés mesureront l'observabilité et la contrôlabilité des systèmes temporisés, et le degré d'abstraction mesurera une partie du coût de leurs tests.

Bien que ces facteurs soient proposés pour d'autres modèles, ceux-ci peuvent être facilement utilisés sur les automates temporisés et les graphes des régions. En effet, les quatre premiers prennent en compte les symboles d'entrée et les symboles de sortie. Comme nous l'avons vu en section 3.1.2, les automates temporisés et les graphes des régions ont des transitions étiquetées soit par un symbole d'entrée soit par un symbole de sortie. Donc les facteurs peuvent être mesurés directement. Il en est de même pour le degré d'abstraction qui prend en compte le nombre d'états de la spécification et de l'implantation.

4.1.4 Outils d'évaluation de la qualité de test

Plusieurs outils d'évaluation de la qualité de test de composants (cartes, circuits,...) et de logiciels ont été récemment développés. Généralement, ceux-ci proposent, non seulement le calcul de plusieurs facteurs, mais aussi une aide au diagnostic, une aide à la modification du système. Parmi ces outils, nous en présenterons trois, SATAN, PLATON et eXpress. Les deux premiers mesurent la testabilité de logiciels, quant à eXpress il mesure, la testabilité de composants. Aux premiers abords, il est intéressant de noter que ceux-ci font appel à des notions précédentes comme l'observabilité et la contrôlabilité. Au niveau des composants, d'autres notions, proches du matériel, sont aussi prises en compte, telles que l'isolation ou l'utilisation de sondes.

SATAN : Systems Automatic Testability ANalysis

L'équipe Valsys a développé cet outil pour évaluer la qualité de test de systèmes conçus selon des méthodes flot de données. Cette analyse de testabilité comporte deux volets :

- Le premier définit des mesures d'effort de génération des jeux d'essai et d'interprétation des résultats (contrôlabilité et observabilité) ; ces mesures permettent de comparer plusieurs architectures logicielles, de déterminer les parties les moins testables, de guider le concepteur dans la modification de sa spécification pour améliorer la testabilité.
- Le second analyse et qualifie la spécification en déterminant les composants critiques qui s'avéreront difficiles à tester. Ainsi le concepteur peut-il apprécier la nécessité d'améliorer la spécification.

De même que les facteurs définis précédemment, les résultats de l'analyse se traduisent par des valeurs comprises entre 0 et 1, 1 exprimant une bonne qualité de test et 0 une mauvaise. Cet outil a été utilisé par SCHLUMBERGER, AéROSPATIALE et par le CNES.

PLATON

PLATON est un autre outil de l'équipe VALSYS, évaluant la testabilité des logiciels procéduraux rédigés en langage C. Celui-ci permet :

- d'une part d'observer l'évolution de certains paramètres au cours des tests (éléments activés, fréquence d'activation,...),
- d'autre part, de programmer des mesures du logiciel et particulièrement des mesures de testabilité grâce à une API (Application Programming Interface).

eXpress

eXpress est un outil d'évaluation de la testabilité, développé par l'industriel SERIEM, appliqué au matériel (circuit, carte,...). La représentation utilisée par cet outil est une modélisation de Dépendance de Premier Ordre. Elle représente une conception basée sur les relations de cause à effet, ce qui semble proche du modèle relationnel.

eXpress traite de certains aspects de la qualité de test, pris en compte pour les logiciels, comme l'observabilité. Cependant, étant axé vers le matériel, il propose aussi l'utilisation de sondes, et prend en compte d'autres aspects tels que l'isolation. A noter aussi, que l'étude de la qualité de test n'est pas, dans ce cas, précoce à la génération des tests mais suit cette génération.

eXpress offre 5 types différents de test, comme par exemple la couverture complète du matériel ou l'observation de certains points. Une fois les tests créés, eXpress donne des informations concernant les composants et fonctions testées, la liste des stimuli pris en compte, la couverture du test. Puis eXpress propose le calcul de probabilités pour offrir une aide à la décision au concepteur sur le choix des tests à appliquer.

4.1.5 Amélioration de la qualité de test

De nombreuses méthodes diverses et variées permettent une amélioration de la qualité de test de système. Plusieurs méthodes sont décrites dans [Kar97], et nous n'en présenterons donc que quelques unes. Les méthodes les plus exploitées sont celles faisant appel soit à la modification de la spécification soit à l'instrumentation.

- **Applications des facteurs et modifications de la spécification :** de nombreux travaux ont prodigué l'amélioration de la qualité de test par une phase de calcul de facteur puis une phase de modification de la spécification [KDC96, YPKP95, KDCK94, VM95, PDK93]. Les facteurs détectent les parties de la spécification à modifier pour obtenir un système plus testable. Dans [KDCK94], les auteurs détaillent comment améliorer un système, au préalable évalué par les quatre premiers degrés énoncés en section 4.1.3.
- **Instrumentation :** l'instrumentation, plus généralement employée dans le test de composants matériels, vise à améliorer l'observabilité et la contrôlabilité en ajoutant au niveau de l'implantation divers mécanismes, tels que des sondes ou des primitives [KC95]. Dans [Pro82], l'auteur propose, quant à lui, l'application des tests en *boîte grise*, ce qui implique le droit à l'observation des composantes normalement non observables.
- **Approche modulaire de développement :** cette méthode [KC95, Pro82] vise à générer une spécification module par module, ce qui a pour but de clarifier, de simplifier et de rendre plus flexible le système. L'inconvénient majeur de cette méthode, est que l'utilisation de modules induit une phase de test différente et plus complexe.

- **Génération de spécifications à partir de services testables :** cette méthode consiste à générer une spécification à partir de services testables, et de primitives d'observations et de contrôle.

4.2 Qualité de test des Automates Temporisés

Dans cette section, nous étudierons les contraintes temporelles des automates temporisés et leurs influences sur le test. Ensuite nous définirons quatre facteurs, évaluant la qualité de test des automates temporisés. Ceux-ci sont, le **Degré de Forme Temporelle des Automates Temporisés** $\mathcal{S}_t$, le **Degré d'Accessibilité Temporelle des Automates Temporisés** $\mathcal{R}_t$, le **Degré de Contrôlabilité des Automates Temporisés** $\mathcal{C}_t$, et le **Degré d'Indépendance Temporelle des Automates Temporisés** $\mathcal{IT}_t$.

Les deux premiers évaluent la couverture possible du test sur une implantation et permettent donc de quantifier la part du système qui sera testée. Les deux derniers facteurs, quant à eux, mesurent le coût du test du système. Comme ces facteurs sont calculés avant la génération des séquences de test, nous ne pouvons évaluer, avec précision, ce coût de test, mais nous proposons des facteurs qui l'influencent principalement et qui donnent une idée sur les futurs efforts nécessaires au test.

Par la suite, ces degrés sont regroupés dans un vecteur , appelé le **Vecteur de Testabilité Temporelle**, qui est noté pour un automate temporisé par :

$$TTV_t =< \mathcal{S}_t, \mathcal{R}_t, \mathcal{C}_t, \mathcal{IT}_t >$$

C'est par le calcul de ce vecteur, et par le calcul du vecteur de testabilité $TV =< Def, Inf, D, D_{-1}, AB >$, décrit en section 4.1.3, que nous obtenons la qualité de test complète de l'automate temporisé. Le nombre de facteurs obtenus permet une aide à la décision précise pour le concepteur mais aussi un libre choix quant aux modifications des propriétés de la spécification à apporter pour obtenir un système présentant une meilleure qualité de test.

4.2.1 Définitions préliminaires

Comme nous l'avons vu en section 3.1.2, la modélisation du temps dense est obtenue par un ensemble d'horloges prenant des valeurs réelles et des contraintes d'horloges, qui sont ajoutées à des transitions de l'automate. Tester des propriétés temporelles, sur une implantation, avec ces systèmes d'inéquations, est difficile : en effet, sans calcul préalable, l'ensemble des moments satisfaisant l'exécution d'une action peut être difficilement connu car il dépend de plusieurs contraintes d'horloges qui ne sont pas forcément étiquetées sur la même transition. C'est pourquoi la plupart des méthodes de dérivation de test, utilisant le modèle des automates temporisés [LC97, PLC98, AKA00], traduisent ces systèmes d'inéquations en diverses notations, comme les intervalles de temps.

Ainsi, pour étudier la qualité de test des automates temporisés, nous considérerons que les contraintes temporelles sont d'abord traduites en intervalles de temps, selon la méthode décrite dans [LC97]. C'est-à-dire, que pour un automate temporisé $\mathcal{A} =<$

$\Sigma_{\mathcal{A}}, S_{\mathcal{A}}, s^0_{\mathcal{A}}, C_{\mathcal{A}}, E_{\mathcal{A}} >$, pour chaque transition $t \in E_{\mathcal{A}}$, et pour chaque horloge $x_i \in C_{\mathcal{A}}$, est calculé un intervalle de temps $[T^i_1 T^i_2]$, avec T^i_1 la valeur minimale et T^i_2 la valeur maximale, satisfaisant les contraintes d'horloges de t. En résumé, les valeurs de temps minimales sont obtenues en utilisant l'algorithme de *Dijkstra*[GFS90] qui calcule le plus court chemin dans l'automate temporisé où chaque transition est préalablement valuée par une contrainte représentant le temps minimal nécessaire pour la franchir. Les valeurs maximales sont obtenues en parcourant dans le sens inverse ces mêmes chemins et en utilisant un algorithme décrit dans [LC97] (voir section 3.2.4).

Soit, par exemple, l'automate temporisé de la Figure 4.16. L'automate traduit, contenant les intervalles de temps, est illustré en Figure 4.17

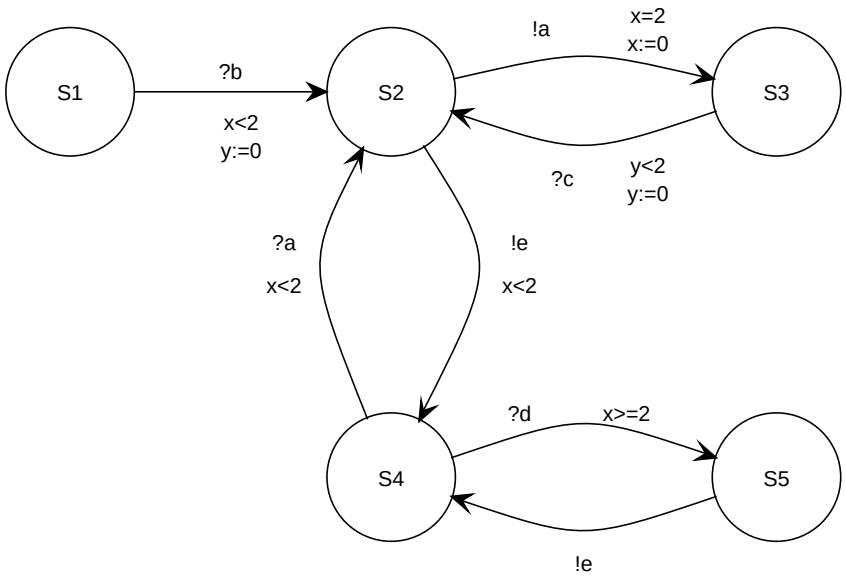


FIG. 4.16 – Automate temporisé

De plus, pour un automate temporisé $\mathcal{A} =< \Sigma_{\mathcal{A}}, S_{\mathcal{A}}, s^0_{\mathcal{A}}, C_{\mathcal{A}}, E_{\mathcal{A}} >$, nous définissons quelques ensembles, utiles pour le calcul des degrés de qualité de test présentés par la suite.

- $I_{\mathcal{A}}$ est l'ensemble de tous les intervalles de temps de $\mathcal{A}$ obtenus par les précédents algorithmes.
- $Q_{\mathcal{A}} = \{$séquence $\sigma \in E^*_{\mathcal{A}} \mid \forall$ transition $t \in \sigma, \sigma$ il n'existe qu'une occurrence de t dans $\sigma\}$ est l'ensemble de couverture des séquences de $\mathcal{A}$.
- $Exec(Seq)$ est l'ensemble non dénombrable d'exécutions d'une séquence d'actions de $E^*_{\mathcal{A}}$. Une exécution est une suite de stimuli donnés à des moments déterminés, d'observations faites à des moments déterminés et d'intervalles de temps d'attente entre stimuli et observations.

4.2.2 Degré de Forme temporelle des Automates Temporisés

D'après la définition de l'automate temporisé (section 3.1.2), un concepteur a la possibilité d'adjoindre à tout automate temporisé des contraintes temporelles du type :

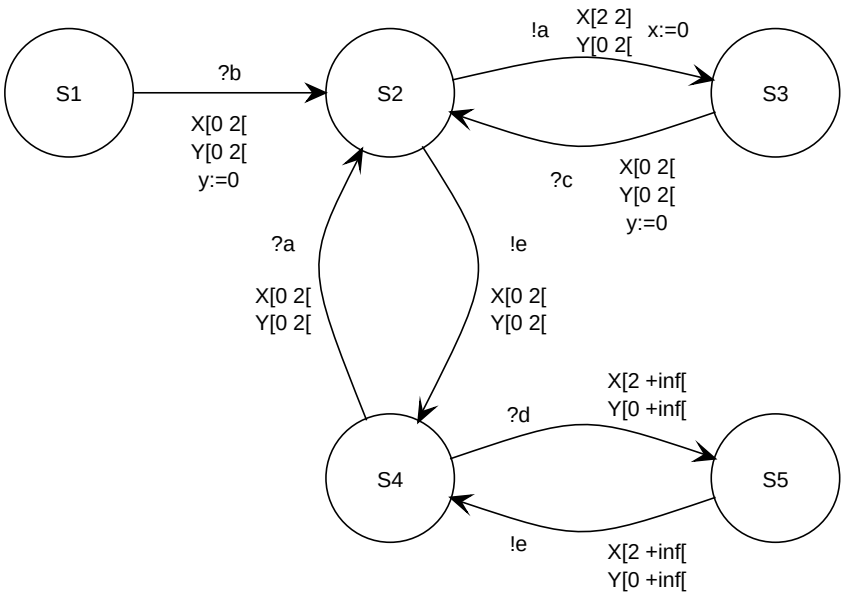


FIG. 4.17 – Contraintes d'horloges modélisées par des intervalles de temps

- $x_i \geq c$ ou $x_i > c$, avec $x_i \in C_{\mathcal{A}}$ et $c \in \mathbb{R}$,
- x_i op x_j avec $x_i \in C_{\mathcal{A}}, x_j \in C_{\mathcal{A}}$ et $op \in \{<, >, =, \leq, \geq\}$
- contrainte vide

Or, chacun de ces cas peut générer des intervalles de temps infinis si aucune autre contrainte temporelle ne les borne. Le problème que posent ces intervalles est le suivant : comment tester le comportement infini d'un système, et plus précisément, comment tester une action dans un intervalle de temps infini ? En effet, si l'on considère des transitions étiquetées par un symbole d'entrée et de telles contraintes d'horloges, le choix du moment de la stimulation de l'implantation, c'est-à-dire le choix de la valeur d'horloge utilisée lors de l'envoi du symbole, est difficile par le nombre infini de possibilités. Actuellement, les méthodes de test basées sur le modèle de l'automate temporisé [AKA00, LC97, PLC98], choisissent comme valeur soit le point milieu de l'intervalle soit les points extrêmes. Or comment obtenir de telles valeurs dans un intervalle infini ? D'autre part, des transitions étiquetées par un symbole de sortie demandent, lors du test, l'attente de la réception du symbole, qui peut être dans ce cas infinie !

Par conséquent, la solution généralement employée par les méthodes de test de systèmes temporisés, est de borner ces intervalles de temps. Cependant, uniquement une partie du comportement du système sera alors vérifiée par le test, et donc, des erreurs potentielles ne pourront être détectées. Ainsi, le système est partiellement testé et la détection des fautes est réduite, ce qui implique une perte de qualité de test.

Étant donné que ces intervalles de temps sont infinis et que leur nombre influence la qualité du test, nous proposons un nouveau facteur, appelé **Degré de Forme Temporelle** et noté $\mathcal{S}_t$, dont le but est d'évaluer cette influence sur l'automate temporisé. Soit $INF_{\mathcal{A}} = \{I \in I_{\mathcal{A}} \mid I = [a, \infty], a \in \mathbb{R}^+\}$, l'ensemble des intervalles infinis d'un automate

temporisé $\mathcal{A}$. Le Degré de Forme Temporelle est obtenu par :

$$\mathcal{S}_t(\mathcal{A}) = 1 - \frac{card(INF_{\mathcal{A}})}{card(I_{\mathcal{A}})} \text{ et } 0 \leq \mathcal{S}_t(\mathcal{A}) \leq 1$$

Par conséquent, plus $\mathcal{S}_t(\mathcal{A})$ est grand et proche de 1, moins le système contient d'intervalles infinis, et plus son comportement sera couvert par le test. Si $\mathcal{S}_t(\mathcal{A})$ est égal à 1, $\mathcal{A}$ ne contient pas d'intervalles de temps infinis. Nous dirons que tout intervalle de temps de $\mathcal{A}$ est formé. A l'inverse, si $\mathcal{S}_t(\mathcal{A})$ est égal à 0, tout intervalle de temps est infini : ce cas demande d'abord un choix sur les bornes à appliquer pour le test, et offre une couverture du système qui ne peut être que très réduite. Ainsi, la détection des fautes, lors du test, ne pourra être suffisante pour assurer la validation du système complet.

Illustrons ce facteur par l'exemple de son application sur l'automate temporisé, modélisé en figure 4.17, et ses intervalles de temps spécifiques. En mesurant le Degré de Forme Temporelle sur cet automate, nous obtenons $\mathcal{S}_t(\mathcal{A}) = \frac{5}{7}$. En effet, les transitions $S_4 \xrightarrow{?d} S_5$, et $S_5 \xrightarrow{!e} S_4$ sont étiquetées par des intervalles de temps infinis et $Card(I_{\mathcal{A}}) = 14$. Nous pouvons conclure que le système a peu d'intervalles infinis mais qu'il sera, malgré cela, partiellement testé et que l'exécution des actions précédentes pourra générer des erreurs non détectables par le test.

4.2.3 Degré d'Accessibilité Temporelle des Automates Temporisés

Lors de la phase de test, le comportement de l'implantation est difficile à contrôler, c'est-à-dire qu'il est difficile de forcer l'exécution d'une séquence d'actions à des valeurs de temps spécifiques.

Cependant, si nous souhaitons tester une action, une condition nécessaire et non suffisante à ce test, consiste à ce que chaque horloge de l'implantation x_i prenne au moins une valeur T dans l'intervalle de temps $[T_1^i T_2^i] \in I$. Nous dirons que les horloges doivent entrer dans leurs intervalles de temps correspondants, c'est-à-dire que ces intervalles doivent être complètement accessibles.

Définition 4.2.1 (Accessibilité d'intervalles de temps) *Soit un automate temporisé* $\mathcal{A} =< \Sigma_{\mathcal{A}}, S_{\mathcal{A}}, s^0_{\mathcal{A}}, C_{\mathcal{A}}, E_{\mathcal{A}} >$.

L'ensemble d'intervalles $I_t = \{[T_1^1 T_2^2], ..., [T_1^i T_2^i], ..., [T_1^n T_2^n]\}$ *d'une transition* $t = S \xrightarrow{a} S' \in E_{\mathcal{A}}$ *est dit accessible ssi :*

$\exists Seq = S_0 \xrightarrow{a_0} S_1 \ldots S_i \xrightarrow{a_i} S, \exists e \in Exec(Seq), \exists v \in V(C_{\mathcal{A}}) \forall [T_1^i T_2^i] \in I_t, \exists T \in [T_1^i T_2^i]$, $e \Rightarrow v(x_i) = T$, *avec* $x_i \in C_{\mathcal{A}}$.

I_t *est complètement accessible ssi pour chaque séquence de transitions* Seq *permettant d'atteindre* t, *pour chaque exécution de* $Exec(Seq)$, I_t *est accessible.*

Or, aucune hypothèse ne permet d'affirmer, au niveau du modèle et de la construction des intervalles de temps, que ceux-ci sont, d'une part accessibles, puis complètement accessibles. Cependant, pour que le test d'une transition puisse être effectué, il est nécessaire que les contraintes d'horloges soient satisfaites donc que les horloges prennent une valeur

de l'intervalle de temps correspondant, quelquesoit l'exécution de l'implantation. Ce qui revient à dire que chaque intervalle de temps doit être complètement accessible.

Proposition 4.2.1 *Soit un automate temporisé* $\mathcal{A} =< \Sigma_{\mathcal{A}}, S_{\mathcal{A}}, s^0_{\mathcal{A}}, C_{\mathcal{A}}, E_{\mathcal{A}} >$.

Une transition $t \in E_{\mathcal{A}}$ *peut être testée, ssi* $\forall [T_1^i T_2^i](1 \leq i \leq card(C_{\mathcal{A}}))$, $[T_1^i T_2^i]$ *est complètement accessible.*

Preuve:

Supposons qu'il existe un intervalle $[T_1^i T_2^i]$ qui ne soit pas complètement accessible. Alors, il existe au moins une exécution $e \in Exec(Seq)$ d'une séquence d'actions Seq, telle que x_i ne prenne pas de valeur de $[T_1^i T_2^i]$. Alors, la valeur de x_i ne satisfait pas les contraintes d'horloges de la transition. Donc, la transition ne peut être testée.

■

Ainsi, si un ou plusieurs intervalles de temps ne peuvent pas être atteints, alors l'action ne pourra être testée. A nouveau, nous nous trouvons dans un cas où le système ne peut être que partiellement testé et où la couverture de fautes est réduite.

Soit $S_1 \xrightarrow[I'_t]{a} S_2 \xrightarrow[I_t]{b} S_3$ deux transitions consécutives, avec $I_t = \{[T_1^i T_2^i](1 \leq i \leq n = card(C_{\mathcal{A}}))\}$ l'ensemble des intervalles de temps, respectifs à chaque horloge, satisfaisant les contraintes d'horloges de la deuxième transition, et $I'_t = \{[T_1'^i T_2'^i](1 \leq i \leq n = card(C_{\mathcal{A}}))\}$ l'ensemble des intervalles de temps satisfaisant les contraintes d'horloges de la première.

Nous considérons que l'ensemble I_t d'intervalles de temps $[T_1^1 T_2^1], ..., [T_1^i T_2^i], ..., [T_1^n T_2^n]$ peut être complètement accessible par les horloges si les propositions suivantes sont satisfaites.

Considérons deux intervalles de temps [1 4] et [2 3], et considérons qu'une horloge x prend, dans le premier intervalle de temps, la valeur 4. Il est, dans ce cas, facile de voir que le deuxième intervalle de temps n'est pas accessible, car celui-ci est inclus et non égal dans le premier. En effet, les horloges croissent d'une façon strictement monotone, ainsi sans être réinitialisée, une horloge ne peut être égale à une valeur de temps déjà écoulée. La Proposition suivante vérifie donc que pour chaque horloge x_i l'intervalle de temps $[T_1^i T_2^i]$ qui doit être atteint par les horloges depuis $[T_1'^i T_2'^i]$ doit contenir des valeurs de temps supérieures à toute valeur de temps de $[T_1^i T_2^i]$. Soit donc la Proposition suivante :

Proposition 4.2.2 *Soit un automate temporisé* $\mathcal{A} =< \Sigma_{\mathcal{A}}, S_{\mathcal{A}}, s^0_{\mathcal{A}}, C_{\mathcal{A}}, E_{\mathcal{A}} >$, *et* $S_1 \xrightarrow[I'_t]{a} S_2 \xrightarrow[I_t]{b} S_3 \in E^*_{\mathcal{A}}$, *avec* $I'_t = \{[T_1'^i T_2'^i](1 \leq i \leq n = card(C_{\mathcal{A}}))\}$ *et* $I_t = \{[T_1^i T_2^i](1 \leq i \leq n = card(C_{\mathcal{A}}))\}$.

Les intervalles de temps $[T_1^1 T_2^1], ..., [T_1^i T_2^i], ..., [T_1^n T_2^n]$ *sont complètement accessibles ssi : A est un symbole de sortie, et* $\forall x_i \in C_{\mathcal{A}}, T_2^i > T_2'^i$

Preuve: Supposons que $T_2^i \leq T_2'^i$. $[T_1'^i T_2'^i]$ est alors composé des intervalles suivants $[T_1'^i T_1^i[+[T_1^i T_2^i]+]T_2^i T_2'^i]$. Si l'action est exécutée par le système à toute valeur d'horloges de $[T_2^i T_2'^i]$ (l'action représentant une réception de symbole est incontrôlable), alors l'intervalle de temps $[T_1^i T_2^i]$ ne peut pas être atteint par les horloges, sachant que celles-ci, par hypothèse, croissent de la même manière et sont strictement croissantes. Par conséquent,

la contrainte $T_2^i \geq T_2^{'i}$ doit être satisfaite. ■

Les horloges croissent de la même manière. De plus, pour que la deuxième transition $S_2 \xrightarrow[I_t]{b} S_3$, avec $I_t = \{[T_1^i T_2^i](1 \leq i \leq n = card(C_\mathcal{A}))\}$, soit passée, chaque horloge x_i doit prendre une valeur de l'intervalle $[T_1^i T_2^i]$. Ainsi, si $v = (X_1, ..., X_n)$ est un tuple quelconque de valeurs de temps prises par les horloges, lors du passage de la première transition, il doit exister un réel positif d tel que pour toute horloge x_i, $X_i + d$ appartient à l'intervalle $[T_1^i T_2^i]$. Si, pour une horloge x_j, $X_j + d$ n'appartient pas à $[T_1^j T_2^j]$, alors $[T_1^j T_2^j]$ n'est pas accessible. Ainsi, si aucune horloge est réinitialisée, nous obtenons la proposition suivante :

Proposition 4.2.3 *Soit un automate temporisé* $\mathcal{A} =< \Sigma_\mathcal{A}, S_\mathcal{A}, s_\mathcal{A}^0, C_\mathcal{A}, E_\mathcal{A} >$, *et* $S_1 \xrightarrow[I'_t]{a} S_2 \xrightarrow[I_t]{b} S_3 \in E_\mathcal{A}^*$, *avec* $I'_t = \{[T_1^{'i} T_2^{'i}](1 \leq i \leq n = card(C_\mathcal{A})\}$ *et* $I_t = \{[T_1^i T_2^i](1 \leq i \leq n = card(C_\mathcal{A})\}$.

Les intervalles de temps $[T_1^1 T_2^1], ..., [T_1^i T_2^i], ..., [T_1^n T_2^n]$ *sont complètement accessibles ssi :* $\forall x_i \in C_\mathcal{A}$ *et* $\forall x_j \in C_\mathcal{A}$, $i \neq j$, *avec* x_i *et* x_j *des horloges non réinitialisées avec la première transition, pour chaque valeur* x *de* $[T_1^{'i} T_2^{'i}]$ *et* y *de* $[T_1^{'j} T_2^{'j}]$, *il existe deux valeurs* X *de* $[T_1^i T_2^i]$ *et* Y *de* $[T_1^j T_2^j]$ *accessibles depuis* x *et* y. *Ce qui revient à :*
(1) $\exists d_1 \in \mathbb{R}^+, T_1^{'i} + d_1 \in [T_1^i T_2^i], T_1^{'j} + d_1 \in [T_1^j T_2^j]$,
(2) $\exists d_2 \in \mathbb{R}^+, T_1^{'i} + d_2 \in [T_1^i T_2^i], T_2^{'j} + d_2 \in [T_1^j T_2^j]$,
(3) $\exists d_3 \in \mathbb{R}^+, T_2^{'i} + d_3 \in [T_1^i T_2^i], T_1^{'j} + d_3 \in [T_1^j T_2^j]$,
(4) $\exists d_4 \in \mathbb{R}^+, T_2^{'i} + d_4 \in [T_1^i T_2^i], T_2^{'j} + d_4 \in [T_1^j T_2^j]$

Preuve: Soit $T_1^{'i} \leq x \leq T_2^{'i}$ et $T_1^{'j} \leq y \leq T_2^{'j}$ deux valeurs d'horloges de $x_i \in C_\mathcal{A}$ et $x_j \in C_\mathcal{A}$, obtenues après l'exécution de la première action.
Supposons les hypothèses (1), (2), (3) et (4) vraies.
Depuis (1), nous avons $x \geq T_1^{'i}$, $y \geq T_1^{'j}$ et $\exists d_1 \in \mathbb{R}^+, x + d_1 \geq T_1^i, y + d_1 \geq T_1^j$
Depuis (2), nous avons $x \geq T_1^{'i}$, $y \leq T_2^{'j}$ et $\exists d_2 \in \mathbb{R}^+, x + d_2 \geq T_1^i, y + d_2 \leq T_2^j$
Depuis (3), nous avons $x \leq T_2^{'i}$, $y \geq T_1^{'j}$ et $\exists d_3 \in \mathbb{R}^+, x + d_3 \leq T_2^i, y + d_3 \geq T_1^j$
Depuis (4), nous avons $x \leq T_2^{'i}$, $y \leq T_2^{'j}$ et $\exists d_4 \in \mathbb{R}^+, x + d_4 \leq T_2^i, y + d_4 \leq T_2^j$

Donc depuis (1) et (3), nous avons pour $T_1^{'i} \leq x \leq T_2^{'i}$, et $y \geq T_1^{'i}$, il existe $d \in \mathbb{R}^+$ tel que $T_1^i \leq x + d \leq T_2^i$, et $y + d \geq T_1^j$.
Depuis (2) et (4), nous avons pour $T_1^{'i} \leq x \leq T_2^{'i}$, et $y \leq T_2^{'j}$, il existe $d' \in \mathbb{R}^+$ tel que $T_1^i \leq x + d \leq T_2^i$, et $y + d \leq T_2^j$.
Donc pour tout $T_1^{'i} \leq x \leq T_2^{'i}$ et pour tout $T_1^{'j} \leq y \leq T_2^{'j}$ les intervalles de temps $[T_1^i T_2^i]$ et $[T_1^j T_2^j]$ sont accessibles. Ainsi quelquesoit l'exécution donnant les valeurs x et y, $[T_1^i T_2^i]$ et $[T_1^j T_2^j]$ sont complètement accessibles. ■

La Proposition suivante décrit le même cas que la Proposition précédente avec une ou plusieurs réinitialisations d'horloges.

Proposition 4.2.4 *Soit un automate temporisé* $\mathcal{A} =< \Sigma_\mathcal{A}, S_\mathcal{A}, s_\mathcal{A}^0, C_\mathcal{A}, E_\mathcal{A} >$, *et* $S_1 \xrightarrow[I'_t]{a} S_2 \xrightarrow[I_t]{b} S_3 \in E_\mathcal{A}^*$, *avec* $I'_t = \{[T_1^{'i} T_2^{'i}](1 \leq i \leq n = card(C_\mathcal{A})\}$ *et* $I_t = \{[T_1^i T_2^i](1 \leq i \leq n = card(C_\mathcal{A})\}$.

Les intervalles de temps $[T_1^1 T_2^1], ..., [T_1^i T_2^i], ..., [T_1^n T_2^n]$ *sont complètement accessibles ssi :* $\forall x_i \in C_{\mathcal{A}}$ *et* $\forall x_j \in C_{\mathcal{A}}$, $i \neq j$, *avec* x_j *réinitialisée avec la première action, l'écoulement de temps nécessaire pour atteindre* $[T_1^i T_2^i]$ *depuis* $[T_1'^i T_2'^i]$ *doit être inclus dans l'intervalle* $[T_1^j T_2^j]$, *ce qui revient à :*
$T_1^i \in [T_1^j + T_1'^i T_2^j + T_1'^i]$ *et* $T_2^i \in [T_1^j + T_2'^i T_2^j + T_2'^i]$

Preuve:

Soit x la valeur d'horloge de x_i après l'exécution de la première action. Pour x_j, cette valeur est égale à 0. Par conséquent, l'écoulement de temps d, nécessaire pour atteindre l'ensemble des intervalles de temps I_t, est tel que $T_1^j \leq d \leq T_2^j$. Soit $x' = x + d$ la valeur d'horloge atteinte par x_i dans $[T_1^i T_2^i]$. Nous obtenons $T_1'^i + d \leq x' = x + d \leq T_2'^i + d$, et $d = T_1^i - T_1'^i = T_2^i - T_2'^i$. Ainsi, nous obtenons $T_1^j \leq T_1^i - T_1'^i \leq T_2^j$, et finalement $T_1^i \in [T_1^j + T_1'^i T_2^j + T_1'^i]$. De la même manière, nous obtenons $T_2^i \in [T_1^j + T_2'^j T_2^j + T_2'^i]$. ■

Notons que le cas où toutes les horloges sont réinitialisées donne des intervalles de temps complètement accessibles. En effet, l'ensemble des horloges réinitialisées permet obligatoirement d'atteindre $\{0, ..., 0\}$.

Ainsi, pour une transition t consécutive de t' tel que $S_1 \xrightarrow[I'_t]{\{?!\}x} S_2 \xrightarrow[I_t]{\{?!\}y} S_3$, soit N_t le nombre d'intervalles de temps de $I_t = \{[T_1^1 T_2^1], ..., [T_1^n T_2^n]\}$ complètement accessibles depuis les intervalles $I'_t = \{[T_1'^1 T_2'^1], ..., [T_1'^n T_2'^n]\}$, en accord avec les propositions précédentes.

Définissons, dans un premier temps un facteur ρ mesurant la difficulté à atteindre I_t depuis l'ensemble d'intervalles de temps $I_{t'}$, obtenus après l'exécution de la première action. Le lecteur pourra noter que si t est une transition partant de l'état initial alors $I_{t'} = \{0, ..., 0\}$.
ρ est évalué par : $\rho(I_t) = \frac{N_t}{card(C_{\mathcal{A}})}$, et $0 \leq \rho(I_t) \leq 1$

Si $\rho(I_t)$ est différent de 1, un ou plusieurs intervalles de temps de I_t ne sont pas complètement accessibles par les horloges, par conséquent la deuxième action ne pourra pas toujours être testée après l'exécution de la première.

Une transition étant rarement suffisante pour atteindre une transition quelconque d'un automate, soit donc SEQ_t l'ensemble des séquences de $Q_{\mathcal{A}}$, permettant d'atteindre t. L'accessibilité de I_t avec $\sigma \in SEQ_t$ est notée $Reach(I_t, \sigma)$ et est évaluée par :
$Reach(I_t, \sigma) = \prod_{t \in \sigma} \rho(I_t)$

Nous pouvons finalement définir un facteur pour évaluer la difficulté à atteindre l'ensemble d'intervalles de temps I_t qui est appelé le *Degré d'accessibilité d'Intervalles de Temps* et noté RT. Celui-ci est obtenu par :

$$RT(I_t) = \frac{\sum_{\sigma_i \in SEQ_t} Reach(I_t, \sigma_i)}{Card(SEQ_t)} \text{ et } 0 \leq RT(I_t) \leq 1$$

Si, pour une transition $t \in E_{\mathcal{A}}$, $RT(I_t)$ est égal à 1, chaque horloge $x_i \in C_{\mathcal{A}}$ peut toujours atteindre l'intervalle de temps $[T_1^i T_2^i]$ correspondant. Dans ce cas, I_t est complètement accessible. A l'opposé, si $RT(I_t)$ est égal à 0, seules quelques exécutions permettront d'atteindre I_t donc il sera très difficile de tester t . Par conséquent, seule une partie du comportement du système pourra être testée.

Finalement, l'évaluation de l'accessibilité de tout intervalle de temps du système peut être calculée par un facteur, appelé le **Degré d'Accessibilité Temporelle des Automates Temporisés** et noté $\mathcal{R}_t$. Ce dernier est obtenu pour un automate temporisé $\mathcal{A}$ par :

$$\mathcal{R}_t(\mathcal{A}) = \frac{\sum\limits_{\xrightarrow[I_t]{\{!?\}x} \in E_{\mathcal{A}}} RT(I_t)}{Card(E_{\mathcal{A}})}, \text{ et } 0 < \mathcal{R}_t(\mathcal{A}) \leq 1$$

Ainsi, plus $\mathcal{R}_t(\mathcal{A})$ est grand et proche de 1, plus les intervalles de temps du système sont complètement accessibles et donc, plus le système est testable. Si $\mathcal{R}_t(\mathcal{A})$ est égal à 1, tout intervalle de temps est complètement accessible, quelquesoit l'exécution. Si $\mathcal{R}_t(\mathcal{A})$ est proche de 0, seule une faible partie du système pourra être testée. $\mathcal{R}_t$ ne peut être égal à 0, car $\{0, ...0\}$ est complètement accessible.

Ce facteur influence aussi le choix des préambules, faisant partie des séquences de test. En effet, pour atteindre une transition t à tester, toute transition d'un préambule p de t doit être étiquetée par des intervalles de temps complètement accessibles par les horloges. Dans le cas contraire, p ne peut être exécuté et t ne peut être testée.

Les résultats suivants illustrent l'accessibilité temporelle de chaque ensemble d'intervalles de temps de l'automate temporisé décrit en Figure 4.17.

Transition de $E_{\mathcal{A}}$	Intervalles de temps	$Card(SEQ)$	RT
$S_1 \xrightarrow{?b} S_2$	X[0 2[Y[0 2[	0	1
$S_2 \xrightarrow{!a} S_3$	X[2 2] Y[0 2[	3	$\frac{2}{3}$
$S_3 \xrightarrow{?c} S_2$	X[0 2[Y[0 2[	3	$\frac{2}{3}$
$S_2 \xrightarrow{!e} S_4$	X[0 2[Y[0 2[	2	1
$S_4 \xrightarrow{?a} S_2$	[0 2[Y[0 2[	3	$\frac{2}{3}$
$S_4 \xrightarrow{?d} S_5$	X[2 $+\infty$[Y[0 $+\infty$[	2	1
$S_5 \xrightarrow{!e} S_4$	X[2 $+\infty$[Y[0 $+\infty$[	2	1

TAB. 4.2 – Calcul du Degré d'Accessibilité Temporelle d'un automate temporisé

Pour le système, en sa globalité, nous obtenons $\mathcal{R}_t(\mathcal{A}) = \frac{6}{7}$. Nous pouvons conclure que la plupart des intervalles de temps sont complètement accessibles par les horloges, mais que le système sera partiellement testé. En effet, les intervalles de temps des transitions consécutives $S_5 \xrightarrow{!e} S_4 \xrightarrow{?a} S_2$ ne respectent pas la première proposition. Dans cet exemple,

le test successif des deux transitions ne pourra pas être fait, de part les contraintes d'horloges proposées.

4.2.4 Degré de Contrôle des Automates Temporisés

Le Degré de Contrôle, en premier lieu défini dans [KGD96], a pour but d'évaluer la difficulté à atteindre chaque action de l'implantation, pendant la phase de test. L'ensemble des séquences, permettant d'atteindre ces actions, est appelé l'ensemble des préambules (section 2.2.3), noté P. Chaque préambule $p \in P$ constitue un squelette des séquences de test, et donc, P garantit l'accès de chaque action. Par conséquent, P affecte le coût du test, et influence la qualité du test de systèmes temporisés. Comme l'évaluation de la qualité de test se fait avant la génération des séquences de test, ce facteur ne peut mesurer avec précision le coût du test. En fait, il donne une idée du coût possible de l'exécution des séquences de test. P n'est pas unique, mais nous considérons que $P \subseteq Q_{\mathcal{A}}$. Dans ce cas, d'une part, la longueur maximale d'un préambule $p \in P$ n'excède pas $n-1$, si n est le nombre d'états de la spécification, et d'autre part, p ne contient pas deux fois la même transition, car $p \in Q_{\mathcal{A}}$.

Comme il est rappelé en section 4.1.3, pour les systèmes non temporisés, la longueur des préambules est suffisante pour se faire une idée sur le coût du test et évaluer le Degré de Contrôle. Pour les systèmes temporisés, le coût du test ne va pas dépendre de la longueur des préambules mais du laps de temps nécessaire à les exécuter. Comme ce temps n'est pas constant mais dépend du comportement du système, lors de ses exécutions, nous considérerons les deux cas suivants :

Temps minimal d'exécution de préambule

Soit, pour un automate temporisé $\mathcal{A}$, $s \xrightarrow{\{?,!\}x} s' \in E_{\mathcal{A}}$, une transition, et soit $I_t = [T_1^1 T_2^1], ..., [T_1^n T_2^n]$ un ensemble d'intervalles de temps relatifs à chaque horloge de $C_{\mathcal{A}}$. Une telle action peut s'exécuter pour tout tuple de valeurs d'horloges satisfaisant I_t. Dans ce cas, nous considérerons que l'action est exécutée dès que chaque horloge x_i prendra une valeur de l'intervalle $[T_1^i T_2^i]$. Un laps de temps d'attente peut être nécessaire pour que chaque intervalle $[T_1^i T_2^i](1 \leq i \leq n)$ soit atteint par les horloges, donc pour que celles-ci satisfassent les contraintes d'horloges de la transition. Donc, comme nous considérons que les actions du préambules sont exécutées immédiatement, le temps d'exécution de tout le préambule, va être égal à la somme des laps de temps d'attente.

L'algorithme suivant permet de calculer ce temps minimal d'exécution d'un préambule p, noté E_{min}.

Algorithme

Calcul de E_{min}
Entrée : Préambule p
Sortie : temps minimal d'exécution de p, E_{min}
DEBUT :
$V = \{X_1, ..., X_N\}$ est un ensemble d'horloges initialisées à 0

$E_{min} = 0$
Pour chaque $t \in p$ étiquetée par les intervalles de temps $\{[T_1^1 T_2^1]...[T_1^n T_2^n]\}$

SI $X_1 \notin [T_1^1 T_2^1]$ OU ... OU $X_n \notin [T_1^n T_2^n]$
Alors $\begin{cases} \% \text{ calcul du laps de temps d'attente} \\ \% \text{ nécessaire pour atteindre chaque intervalle de temps} \\ d = MAX\{T_1^1 - X_1, ..., T_1^n - X_n\} \\ V = \{X_1 + d, ..., X_n + d\} \\ E_{min} = E_{min} + d \end{cases}$
Réinitialiser les horloges de λ (ensemble des horloges réinitialisées lors du passage de t)
FINPour
FIN

Temps maximal d'exécution de préambule

Soit, pour un automate temporisé $\mathcal{A}$, $s \xrightarrow{\{?,!\}x} s' \in E_{\mathcal{A}}$, une transition, et soit $I_t = [T_1^1 T_2^1], ..., [T_1^n T_2^n]$ un ensemble d'intervalles de temps relatifs à chaque horloge de $C_{\mathcal{A}}$. Cette fois nous considérons que l'action va être exécutée le plus tard possible, c'est-à-dire pour un tuple $(X_1, ..., X_n)$ tel que $\forall X_i (1 \leq i \leq n), X_i \in [T_1^i T_2^i]$ et tel que $\forall k \in \mathbb{R}^+, \exists j (1 \leq j \leq n) X_j + k \notin [T_1^j T_2^j]$. Étant donné que nous considérons que les actions du préambules sont exécutées le plus tard possible, un laps de temps d'attente maximal, qui peut s'écouler avant chaque exécution, doit être calculé et ajouté au temps maximal d'exécution.

A ces laps de temps, il faut encore ajouter les laps de temps d'attente nécessaires pour que toutes les horloges satisfassent chaque intervalle de temps d'une transition. Pour un préambule p, l'algorithme suivant, donne la somme de tous ces laps de temps, que nous notons E_{max}.

Algorithme

Calcul de E_{max}
Entrée : Préambule p
Sortie : temps maximal d'exécution de p, E_{max}
DEBUT :
$V = \{X_1, ..., X_N\}$ est un ensemble d'horloges initialisées à 0

$E_{max} = 0$
Pour chaque $t \in p$ étiquetée par $[T_1^1 T_2^1]...[T_1^n T_2^n]$

SI $X_1 \notin [T_1^1 T_2^1]$ OU ... OU $X_n \notin [T_1^n T^n 2]$
Alors
- % calcul du laps de temps d'attente
- % nécessaire pour atteindre chaque intervalle de temps
- $d = MAX\{T_1^1 - X_1, ..., T_1^n - X_n\}$
- $V = \{X_1 + d, ..., X_n + d\}$
- $E_{max} = E_{max} + d$

% calcul du laps de temps maximal d'attente avant exécution de l'action

$d = MIN\{[T_2^1 - X_1], ..., [T_2^n - X_n]\}$
$E_{max} = E_{max} + d$
$V = \{X_1 + d, ..., X_n + d\}$
Réinitialiser les horloges de λ (ensemble des horloges réinitialisées lors du passage de t)
FINPour
FIN

Ensuite, pour évaluer le coût nécessaire pour atteindre une transition t, avec un préambule p, nous définissons le *Degré de Contrôle de Transition*, noté CT_t et obtenu par :

$$CT(p,t) = 1 - \frac{E_{min}(p) + E_{max}(p)}{max_{p_i \in Q_{\mathcal{A}}}\{E_{min}(p_i) + E_{max}(p_i)\}}$$

$$\text{et } 0 \leq CT_t(p,t) \leq 1$$

Ce degré évalue la difficulté à atteindre une transition t, en comparant les deux exécutions de p avec le préambule $p_i \in P$ ayant le plus grand temps minimal et le plus grand temps maximal d'exécutions. Si $CT_t(p,t)$ est égal à 1, p ne demande que le temps minimal pour émettre et recevoir les symboles de p. Si $CT_t(p,t)$ est égal à 0, p sera le préambule de $Q_{\mathcal{A}}$ qui demande le plus de temps pour atteindre t.

Maintenant, nous pouvons mesurer la difficulté nécessaire pour atteindre toutes les transitions du système $\mathcal{A}$, en calculant la moyenne de CT_t pour chaque préambule $p_{\xrightarrow{t}}$, permettant d'atteindre une transition $t \in E_{\mathcal{A}}$. Ainsi, le **Degré de Contrôle des Automates Temporisés**, noté $\mathcal{C}_t$ est obtenu par :

$$\mathcal{C}_t(\mathcal{A}) = \frac{\sum_{t \in E_{\mathcal{A}}} CT(p_{\xrightarrow{t}}, t)}{card(E_{\mathcal{A}})} \text{ et } 0 \leq \mathcal{C}_t(\mathcal{A}) \leq 1$$

Par conséquent, plus le temps moyen d'exécution de l'ensemble des préambules est grand, plus le coût de test sera grand, et donc moins le système est testable. Si $\mathcal{C}_t(\mathcal{A})$ est égal à 1, nous nous trouvons avec le cas idéal où, soit aucun préambule n'est nécessaire pour atteindre chaque transition, soit les préambules demandent un temps d'exécution nul. Dans ce dernier cas les actions des préambules sont exécutées à la suite et immédiatement. Si, par contre, $\mathcal{C}_t(\mathcal{A})$ est égal à 0, le coût du test sera très élevé à cause de l'ensemble des préambules choisi. En effet, chacun des préambules demande le temps d'exécution le plus long de tous les préambules de $Q_{\mathcal{A}}$. L'ensemble P pourrait, si possible, être modifié pour obtenir des préambules dont le temps d'exécution serait plus court.

La mesure du Degré de Contrôle sur l'automate temporisé de la Figure 4.17 donne les résultats suivants. Nous avons choisi un ensemble de préambules décrit dans le tableau ci-dessous. La séquence de $Q_{\mathcal{A}}$, qui demande le plus grand temps d'exécution, est $?b!a?c!e?d$ et donne $E_{min} = 4$ et $E_{max} = 6$. De plus, nous bornons les différents intervalles de temps infini par $X = 4$ et $Y = 4$.

Preambule	Nombre de transitions atteintes	E_{min}	E_{max}	CT
ϵ	1	0	0	1
$?b!a$	1	2	2	$\frac{3}{5}$
$?b!e$	2	0	2	$\frac{4}{5}$
$?b!e?d$	1	2	4	$\frac{2}{5}$
$?b!a?c$	2	2	4	$\frac{3}{5}$

TAB. 4.3 – Calcul du Degré de Contrôle d'un automate temporisé

Comme $Card(E_{\mathcal{A}}) = 7$, nous obtenons $CT(\mathcal{A}) = \frac{22}{35}$. Ceci montre que l'ensemble P, que nous avons choisi, demande un temps moyen d'exécution par rapport aux temps minimal et maximal de toutes les exécutions du système. Le préambule $?b!a?c$ pourrait être remplacé par $?b$ car cette séquence est exécutée plus rapidement ($E_{min} = 0$, $E_{max} = 2$) et offre une meilleure qualité de test au système, tout en atteignant les mêmes transitions.

4.2.5 Degré d'Indépendance Temporelle des Automates Temporisés

La notion de temps dense est modélisée dans les automates temporisés par un ensemble d'horloges qui est rarement minimal (voir [DY96]). En effet, la plupart des spécifications sont décrites dans un haut niveau d'abstraction (LOTOS, ESTELLE, LDS, ...) décrivant, entre autre, des échéances des temporisateurs. Puis, Celles-ci sont plus tard compilées en automate temporisé, ayant un nombre d'horloges proportionnel à ces échéances de temporisateurs. Par conséquent, ces horloges sont rarement actives au même moment. Or, ce nombre, non minimal, d'horloges influence aussi le coût du test.

Premièrement, la complexité de la construction des intervalles de temps dépend principalement du nombre de ces horloges car, pour chaque transition, un intervalle de temps est généré pour chaque horloge du système. La complexité de la génération du graphe

des régions dépend aussi essentiellement du nombre d'horloges, en effet, pour un automate temporisé $\mathcal{A} =< \Sigma_{\mathcal{A}}, S_{\mathcal{A}}, s^0_{\mathcal{A}}, C_{\mathcal{A}}, E_{\mathcal{A}} >$, ce coût est proportionnel à $(card(S_{\mathcal{A}}) + card(E_{\mathcal{A}})).2^{\delta(\mathcal{A})}$, avec $\delta(\mathcal{A})$ la longueur de l'encodage binaire des contraintes des horloges [AD94]. Donc, pour un automate $\mathcal{A}$, le coût de la génération des séquences de test est influencé par $Card(C_{\mathcal{A}})$,

Le coût de l'exécution du test est aussi influencé par ce nombre, car la plupart des méthodes d'exécutions considèrent que le testeur doit contenir le même nombre d'horloges que la spécification, et que, pour chaque action, il doit vérifier l'état de chacune de ces horloges (architecture de test [ENDE97, PF99b]).

Comme le nombre d'horloges est fini mais peut prendre une valeur quelconque, leur influence sur le test semble difficile à évaluer. Mais si nous considérons une échéance de temporisateur par action dans le langage de haut-niveau, nous obtenons, dans l'automate temporisé, une horloge nouvellement créée par transition. Ainsi, pour un automate $\mathcal{A}$, si $Card(E_{\mathcal{A}})$ est le nombre de transitions, nous considérons, qu'au pire, $Card(E_{\mathcal{A}})$ horloges peuvent être utilisées. De ce fait, nous pouvons définir un facteur, appelé le **Degré d'Indépendance du temps des Automates Temporisés**, noté $\mathcal{IT}_t$, qui a pour but de mesurer le coût du test lié au nombre d'horloges. Celui-ci est obtenu par :

$$\mathcal{IT}_t(\mathcal{A}) = 1 - \frac{Card(C_{\mathcal{A}}) - 1}{Card(E_{\mathcal{A}}) - 1} \text{ et } 0 \leq \mathcal{IT}_t(\mathcal{A}) \leq 1$$

Donc, plus $\mathcal{IT}_t(\mathcal{A})$ est grand et proche de 1, plus le coût du test est faible et plus le système est testable. Si $\mathcal{IT}_t(\mathcal{A})$ est égal à 1, seule une horloge est utilisée pour spécifier le système, et le coût total du test dépend essentiellement du de degré de contrôle de $\mathcal{A}$. Si $\mathcal{IT}_t(\mathcal{A})$ est égal à 0, la spécification comporte autant de transitions que d'horloges, ce qui alourdit la génération des séquences de test et leurs exécutions.

En appliquant ce facteur sur l'automate temporisé de la Figure 4.17, nous obtenons le résultat suivant : $\mathcal{IT}_t(\mathcal{A}) = \frac{5}{6}$. Nous en déduisons que le coût du test sera peu influencé par le nombre d'horloges, sachant que deux horloges pour tout un système temporisé semble un nombre raisonnable et que ce nombre pourra difficilement être réduit grâce aux algorithmes décrits dans [DY96].

4.2.6 Complexité du calcul des Degrés

Cette partie est dédiée au calcul des complexités des degrés précédemment proposés. Ces complexités dépendent uniquement du nombre de transitions, du nombre d'états et du nombre d'horloges du système. Nous donnons des complexités sans recherche d'optimisation d'algorithme ou d'heuristique et en considérant les pires cas.

- Un automate temporisé $\mathcal{A} = (\Sigma_{\mathcal{A}}, S_{\mathcal{A}}, S^0_{\mathcal{A}}, C_{\mathcal{A}}, E_{\mathcal{A}})$ contient $K * C$ intervalles de temps, avec $K = card(E_{\mathcal{A}})$ et $C = card(C_{\mathcal{A}})$. Ainsi, la complexité du calcul du Degré de Forme Temporelle des Automates Temporisés est proportionnelle à $K * C$.

- La complexité du Degré d'Accessibilité Temporelle des Automates Temporisés est proportionnelle à $K * card(Q_{\mathcal{A}}) + M * K$, avec $M = card(S_{\mathcal{A}})$. $Q_{\mathcal{A}}$ est obtenu,

en générant l'arbre couvrant de $\mathcal{A}$, puis en énumérant les chemins partiels de l'arbre. Comme chaque transition apparaît une fois dans l'arbre, la complexité de la construction de $Q_{\mathcal{A}}$ est proportionnelle à $M * K$. De plus, pour chaque transition $t \in E_{\mathcal{A}}$, nous calculons la difficulté à atteindre les intervalles de temps I_t pour tout chemin permettant d'atteindre t. Le nombre de ces chemins étant au pire égal à $card(Q_{\mathcal{A}})$, la complexité totale du Degré d'Accessibilité Temporelle est donc proportionnelle à $K * card(Q_{\mathcal{A}}) + M * K$.

- Si K est le nombre de transitions de $\mathcal{A}$, alors l'ensemble des préambules contient au plus K séquences. Au pire, la longueur d'un préambule est de $M - 1$. Donc, comme le calcul du Degré de Contrôle des Automates Temporisés demande deux exécutions de chaque préambule, nous obtenons une complexité proportionnelle à $2 * M * K$.

- La complexité du Degré d'Indépendance Temporelle des Automates Temporisés est négligeable car elle consiste à faire une division entre deux valeurs, immédiatement utilisables, qui sont $card(E_{\mathcal{A}})$ et $card(C_{\mathcal{A}})$.

4.3 Qualité de test des graphes des régions

De même que pour les automates temporisés, nous proposons quatre degrés permettant d'évaluer la qualité de test des graphes des régions. Ceux-ci sont toujours axés sur les contraintes d'horloges du système, qui sont, dans ce cas, modélisées par des régions d'horloges. Le passage en graphe des régions pourrait faire penser que nous obtenons les mêmes degrés. Or, les propriétés des graphes de régions sont sensiblement différentes de celles des automates temporisés et donc la qualité de test de ce modèle va aussi demander une évaluation différente. En effet, une région est une sorte d'intervalle de temps dépendant de plusieurs horloges et chaque région d'horloges est successeur temporel à la précédente (voir section 3.1.2). Par conséquent, nous obtiendrons des facteurs différents et parfois même plus précis. Cependant, nous avons souhaité garder les mêmes noms de facteur pour ensuite s'interroger et commenter l'intérêt du passage en graphe des régions, qui demande un coût très important.

Les degrés que nous proposons sont le **Degré de Forme Temporelle des Graphes des Régions**, noté $\mathcal{S}_{rg}$, le **Degré d'Accessibilité Temporelle des Graphes des Régions**, noté $\mathcal{R}_{rg}$, le **Degré de Contrôle des Graphes des Régions**, noté $\mathcal{C}_{rg}$ et le **Degré d'Indépendance Temporelle des Graphes des Régions**, noté $\mathcal{IT}_{rg}$. De même que précédemment, ceux-ci sont réunis en un vecteur, appelé le **Vecteur de Testabilité Temporelle**, $TTV_{rg} = \mathcal{S}_{rg}, \mathcal{R}_{rg}, \mathcal{C}_{rg}, \mathcal{IT}_{rg}$. Ce Vecteur réuni au Vecteur de Testabilité $TV =< Def, Inf, D, D_{-1}, AB >$, défini à la section 4.1.3, permet d'évaluer la qualité de test des graphes des régions en étudiant leurs propriétés comportementales et temporelles.

Nous illustrerons le calcul de ces degrés sur le graphe des régions minimisé de la Figure 4.18 et les régions d'horloges de la Figure 4.19, obtenus depuis l'automate temporisé de la Figure 4.16.

Mais avant d'étudier ces degrés, voici quelques définitions préliminaires.

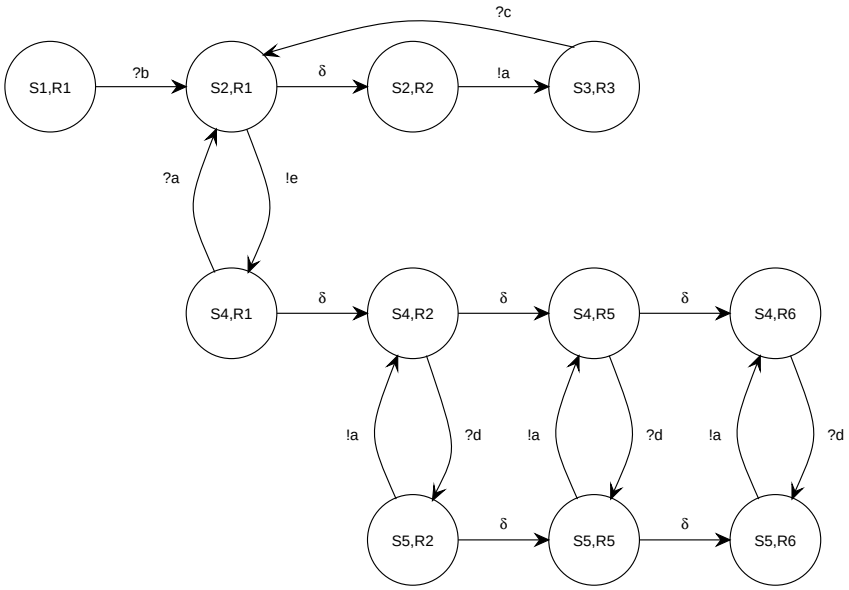


FIG. 4.18 – Un graphe des régions minimisé

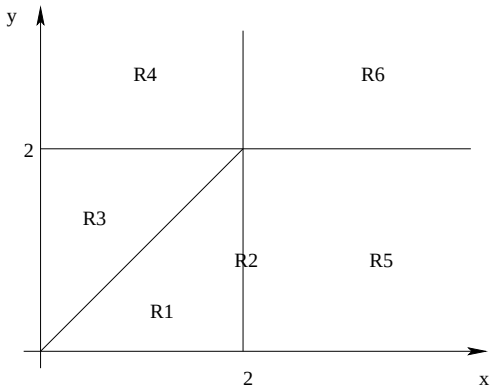


FIG. 4.19 – Régions d'Horloges

4.3.1 Définitions Préliminaires

Soit $\mathcal{A} =< \Sigma_{\mathcal{A}}, S_{\mathcal{A}}, s^0_{\mathcal{A}}, C_{\mathcal{A}}, E_{\mathcal{A}} >$ un automate temporisé, et $\mathcal{RA} = (\Sigma_{\mathcal{RA}}, S_{\mathcal{RA}}, s^0_{\mathcal{RA}}, E_{\mathcal{RA}})$ le graphe des régions de $\mathcal{A}$. Nous définissons pour $\mathcal{RA}$ les ensembles suivants :

- $R_{\mathcal{RA}} = \{[v] \mid \exists (s, [v]) \in S_{\mathcal{RA}}\}$, est l'ensemble des régions d'horloges de $\mathcal{RA}$
- $Q_{\mathcal{RA}} = \{$séquence $\sigma \in E^*_{\mathcal{RA}} \mid \forall$ transition $t \in \sigma, \sigma$ ne contient qu'une occurrence de $t\}$ est l'ensemble de couverture des séquences de $\mathcal{RA}$.
- $Exec(Seq)$ est l'ensemble infini d'exécutions d'une séquence d'actions de $E^*_{\mathcal{RA}}$. Une exécution est formée d'une suite de stimuli à des moments déterminés, d'observations à des moments déterminés et de laps de temps entre stimuli et observations.
- $U_R = \{(s, R) \xrightarrow{a} (s', R') \in E_{\mathcal{A}} \mid a \neq \delta, \}$, est l'ensemble des transitions qui modélisent une action pouvant s'exécuter à toute valuation d'horloges de R. Cet ensemble permet de référencer ces transitions pour chaque région d'horloges, dans le but de connaître et quantifier l'influence de chaque région d'horloges dans la qualité de test.
- $AT = \{(s, a, \lambda, s') \in E_{\mathcal{RA}} \mid a \neq \delta\}$, est l'ensemble des transitions étiquetées par un symbole différent de δ. Chacune de ces transitions modélise une action du système. $AT \subseteq E_{\mathcal{RA}}$

4.3.2 Degré de Forme Temporelle des Graphes des Régions

De même que précédemment, les graphes des régions peuvent être construits avec des régions d'horloges non bornées pour une ou plusieurs horloges. Ces régions d'horloges sont obtenues, lors du processus de transformation par les mêmes contraintes d'horloges décrites en section 4.2.2. Nous définissons une telle région d'horloges comme une région d'horloges infinie.

Définition 4.3.1 (Région d'Horloges Infinie) *Soit $\mathcal{RA} = (\Sigma_{\mathcal{RA}}, S_{\mathcal{RA}}, s^0_{\mathcal{RA}}, E_{\mathcal{RA}})$ un graphe des régions.*

Soit $[v]$ une région d'horloges de $R_{\mathcal{RA}}$. $[v]$ est une région d'horloges infinie ssi : $\forall d \in \mathbb{R}^+, \exists v \in [v], \exists x_i \in C_{\mathcal{A}}, v(x_i) + d \in [v]$

Tester une action, pouvant s'exécuter pour toute valuation d'horloges d'une région d'horloges infinie est impossible. En effet, chaque phase de test doit se terminer. Par conséquent, les régions d'horloges R_i doivent être bornées pour toute horloge donnant des valeurs infinies. Dans ce cas, seule une partie du système est vérifiée, car une partie de son comportement est testée avec une région d'horloges $R'_i \subset R_i$. Par conséquent, les régions d'horloges infinies influencent la qualité de test, car le système est partiellement testé et la couverture des fautes du système est réduite.

Cette influence des régions d'horloges incite à définir un nouveau facteur, permettant de mesurer quel intervalle de temps du système, modélisé dans la spécification, sera couvert par le test, mais aussi de quantifier par région d'horloges le nombre d'horloges à borner qu'il faut appliquer au système pour le tester.

Soit donc $I_{R_i} = \{x_j \in C_{\mathcal{A}} \mid \forall M \in \mathbb{R}^+, \exists v \in R_i \mid v(x_j) + M \in R_i\}$, l'ensemble des horloges du système non bornées, dans la région R_i. $Card(I_{R_i})$ affecte la qualité de test, car en effet, plus $Card(I_{R_i})$ est grand, moins le système sera couvert par les tests d'actions pouvant s'exécuter dans R_i. Ainsi, nous proposons un premier facteur, appelé le *Degré de*

Forme Temporelle de Région d'horloges et noté ST, mesurant cette évaluation. Ce dernier est obtenu par :

$$ST(R_i) = 1 - \frac{card(I_{R_i})}{card(C_{\mathcal{A}})}, \text{ et } 0 \leq ST(R_i) \leq 1$$

Si, $ST(R_i)$ est égal à 1, nous dirons que R_i est complètement formée, et toute action, pouvant s'exécuter à des valuations d'horloges de R_i, sera complètement testée. Si $ST(R_i)$ est égal à 0, R_i doit être bornée pour toute horloge, avant le test. Le système est de plus partiellement testé dans R_i.

Pour mesurer l'influence des régions d'horloges infinies sur l'ensemble du système, nous définissons un nouveau degré, appelé le **Degré de Forme Temporelle des graphes des Régions** et noté $\mathcal{S}_{rg}$. Celui-ci est évalué par :

$$\mathcal{S}_{rg}(\mathcal{RA}) = \frac{\sum_{R_i \in R_{\mathcal{RA}}} ST(R_i) * card(U_{R_i})}{card(AT)} \text{ et } 0 \leq \mathcal{S}_{rg}(\mathcal{RA}) \leq 1$$

Par conséquent, plus $\mathcal{S}_{rg}(\mathcal{RA})$ est grand et proche de 1, plus le système est couvert par le test et testable. Si $\mathcal{S}_{rg}(\mathcal{RA})$ est égal à 1, toutes les régions d'horloges du système sont bornées, donc $\forall R_i \in R_{\mathcal{RA}}, I_{R_i} = \emptyset$. Dans ce cas, le système peut être complètement testé, à la condition que toutes les régions d'horloges soient complètement accessibles lors de la phase de test. Par contre, si $\mathcal{S}_{rg}(\mathcal{RA})$ est égal à 0, chaque région d'horloges est infinie sur chaque horloge du système $\mathcal{RA}$. Dans ce cas, $\mathcal{RA}$ ne peut en fait que posséder une unique région d'horloge infinie R, avec $I_R = C_{\mathcal{A}}$. $\mathcal{RA}$ est alors le plus partiellement testé.

Si nous considérons le graphe des régions, illustré en Figure 4.18 et ses six régions d'horloges, illustrées en Figure 4.19, nous obtenons les résultats suivants :

Region d'horloges	Nombre d'horloges non bornées	$card(U_R)$	ST_{rg}
R_1	0	3	1
R_2	0	3	1
R_3	0	1	1
R_5	1	2	$\frac{1}{2}$
R_6	2	2	0

TAB. 4.4 – Calcul du Degré de Forme Temporelle d'un graphe des Régions

De plus, $Card(AT) = 11$, donc nous obtenons un facteur $\mathcal{S}_{rg} = (\mathcal{RA}) = \frac{8}{11}$. Celui-ci implique que le graphe des régions $\mathcal{RA}$ possède des régions d'horloges qui ne sont pas complètement formées temporellement. En effet, les régions d'horloges R_5 et R_6 ne sont pas finies, ce qui implique que le test d'actions, pouvant s'exécuter dans R_5 ou R_6, sera partiel. Ainsi des erreurs potentielles risquent de ne pas être trouvées.

4.3.3 Degré d'Accessibilité Temporelle du Graphe des Régions

La transformation de l'automate temporisé en graphe des régions génère des régions d'horloges qui sont une autre représentation des contraintes faites sur les horloges. Cette

transformation intègre une propriété fondamentale au test, qui est la propriété de successeur temporel. Celle-ci, donnée en Définition 3.1.2 implique que, pour un graphe des régions $\mathcal{RA}$, pour toute transition de $E_{\mathcal{RA}}$ $(S,R) \xrightarrow{\{?,!\}x} (S',R')$ toute valuation d'horloges de R permet d'atteindre une valuation d'horloges de R'.

Nous allons montrer que certaines régions d'horloges des graphes des régions ne peuvent être toujours accessibles, et que ceci est dû à certaines transitions du système. Voici la définition de l'accessibilité des régions d'horloges :

Définition 4.3.2 (Accessibilité des Régions d'horloges) *Soit un automate temporisé $A=<\Sigma_{\mathcal{A}}, S_{\mathcal{A}}, s^0_{\mathcal{A}}, C_{\mathcal{A}}, E_{\mathcal{A}}>$, $\mathcal{RA}=(\Sigma_{\mathcal{RA}}, S_{\mathcal{RA}}, s^0_{\mathcal{RA}}, E_{\mathcal{RA}})$ le graphe des régions obtenu, $R_{\mathcal{RA}}$ l'ensemble des régions d'horloges de $\mathcal{RA}$.*

Une région d'horloges $R \in R_{\mathcal{RA}}$ est dite accessible ssi :

$\exists Seq = (S_0,R_0) \xrightarrow{a_0} (S_1,R_1)...(S_i,R_i) \xrightarrow{a_i} (S,R) \in E^*_{\mathcal{RA}}, \exists e \in Exec_{Seq}, \exists (X_1,...,X_n) \in R, \exists v \in R$, $e \Rightarrow v(x_1,...,x_n) = (X_1,...,X_n)$ avec $\{x_1,...,x_n\} = C_{\mathcal{A}}$

R est dite complètement accessible ssi pour chaque séquence de transitions Seq permettant d'atteindre un état étiqueté par R, pour chaque exécution $e \in Exec_{Seq}$, R est accessible.

La Définition du graphe des régions ne permet pas d'affirmer que les régions d'horloges soient complètement accessibles. En fait, la Proposition suivante montre que chaque région d'horloges d'un graphe des régions est accessible mais pas nécessairement accessible.

Proposition 4.3.1 *Soit $\mathcal{RA}=(\Sigma_{\mathcal{RA}}, S_{\mathcal{RA}}, s^0_{\mathcal{RA}}, E_{\mathcal{RA}})$ un graphe des régions, $R_{\mathcal{RA}}$ l'ensemble des régions d'horloges de $\mathcal{RA}$.*

$\forall R \in R_{\mathcal{RA}}$, R est accessible, mais R n'est pas nécessairement complètement accessible.

Preuve:

Soit une transition $t=(s,R) \xrightarrow{\{?!\}x} (s',R')$.

D'après la Définition du graphe des régions, R' est successeur temporel à $R[\lambda \rightarrow 0]$, c'est à dire que R' est successeur temporel à R en réinitialisant les horloges de λ qui doit l'être sur la transition.

Donc $\forall v \in R[\lambda \rightarrow 0], \exists d \in \mathbb{R}, v+d \in R'$. Par conséquent, il existe une exécution telle que les horloges prennent comme valeur celles de la valuation d'horloges v et tel que R' est accessible.

Supposons que la transition soit étiquetée par un symbole de sortie " !a". Soit une autre transition $t'=(s,R) \xrightarrow{!b} (s'',R'')$. Le comportement du système permet de passer soit la première transition soit la deuxième. Donc $\exists e \in Exec_t, R'$ n'est pas accessible. Donc R' n'est pas complètement accessible.

■

Or, pour tester une transition $(s,R) \xrightarrow{\{?,!\}x} (s',R')$ d'un graphe des régions $\mathcal{RA}$, il est nécessaire que R soit complètement accessible.

Proposition 4.3.2 *Soit $\mathcal{RA}=<\Sigma_{\mathcal{RA}}, S_{\mathcal{RA}}, s^0_{\mathcal{RA}}, E_{\mathcal{RA}}>$ un graphe des régions, $R_{\mathcal{RA}}$ l'ensemble des régions d'horloges de $\mathcal{RA}$ et $Exec_{\mathcal{RA}}$, l'ensemble des exécutions de $\mathcal{RA}$.*

Une transition $t = (s, R) \xrightarrow{\{?,!\}x} (s', R') \in E_{\mathcal{RA}}$ *peut être testée ssi* R *est complètement accessible.*

Preuve: Voir Proposition 4.2.1 ∎

Ainsi, pour un graphe des régions $\mathcal{RA}$, soit $\rho(R_i, (S, R_j) \xrightarrow{a} (S', R_i))$ un premier facteur permettant d'évaluer la difficulté à atteindre la région d'horloges R_i de $\mathcal{RA}$ avec la transition $(S, R_j) \xrightarrow{\{?,!\}A} (S', R_i) \in E_{\mathcal{RA}}$, sachant que le tuple d'horloges du système est égal à une valuation d'horloges de R_j. L'évaluation de ce facteur dépend des trois propositions suivantes :

Proposition 4.3.3 *Soit un graphe des régions* $\mathcal{RA} = (\Sigma_{\mathcal{RA}}, S_{\mathcal{RA}}, s^0_{\mathcal{RA}}, E_{\mathcal{RA}})$*, et* $R_{\mathcal{RA}}$ *l'ensemble des régions d'horloges de* $\mathcal{RA}$*.*

$\forall (s, R_i) \xrightarrow{\{?,!\}x} (s', R_j) \in E_{\mathcal{RA}}$*, si* $(s, R_i) \in s^0_{\mathcal{RA}}$*, alors* R_i *est complètement accessible.*

Preuve:

Les horloges prennent comme valeur, à l'initialisation, $(0, ..., 0)$. R_i est étiquetée à un état initial du graphe des régions $\mathcal{RA}$. D'après la Définition du graphe des régions, les horloges sont initialisées à chacun des états initiaux. Donc R_i est au moins composée de la valuation d'horloges $(0, ..., 0)$. De plus aucune exécution est nécessaire pour atteindre R_i, donc R_i est complètement accessible.

∎

Dans ce cas, la difficulté d'atteindre R_i est notée $\rho(R_i, \epsilon)$, et est égale à 1.

Proposition 4.3.4 *Soit un graphe des régions* $\mathcal{RA} = (\Sigma_{\mathcal{RA}}, S_{\mathcal{RA}}, s^0_{\mathcal{RA}}, E_{\mathcal{RA}})$*,* $R_{\mathcal{RA}}$ *l'ensemble des régions d'horloges de* $\mathcal{RA}$ *et* $Exec_{\mathcal{RA}}$ *l'ensemble infini des exécutions de* $\mathcal{RA}$*.*

$\forall (s, R_j) \xrightarrow{?A} (s', R_i) \in E_{\mathcal{RA}}$*,* R_i*, si* $\exists (s, R_j) \xrightarrow{!x} (s'', R_k) \in E_{\mathcal{RA}}$ *alors* R_i *n'est pas complètement accessible.*

Preuve:

Un symbole d'entrée modélise un stimulus et plus précisément l'envoi d'un symbole. Ce symbole peut être émis par le testeur à tout moment, satisfaisant son émission. Le symbole de sortie $!x$ peut aussi être émis à tout moment par l'implantation. Ainsi, l'action permettant d'atteindre R_k peut être exécutée avant l'action permettant d'atteindre R_i. Dans ce cas, R_i n'est pas accessible. Donc R_i n'est pas complètement accessible. ∎

La difficulté à atteindre R_i avec $(S, R_j) \xrightarrow{?A} (S', R_i)$ dépend du nombre de transitions partant de (s, R_i) étiquetées par un symbole de sortie. Donc, cette difficulté, notée $\rho(R_i, (S, R_j) \xrightarrow{?A} (S', R_i))$ est obtenue par :
$\rho(R_i, (S, R_j) \xrightarrow{?A} (S', R_i)) = \frac{1}{Card(\{(s,R_j) \xrightarrow{!x} (S,R) \in E_{\mathcal{RA}}\}) + 1}$.

Proposition 4.3.5 *Soit un graphe des régions* $\mathcal{RA} = (\Sigma_{\mathcal{RA}}, S_{\mathcal{RA}}, s^0_{\mathcal{RA}}, E_{\mathcal{RA}})$*,* $R_{\mathcal{RA}}$ *l'ensemble des régions d'horloges de* $\mathcal{RA}$ *et* $Exec_{\mathcal{RA}}$ *l'ensemble infini des exécutions de* $\mathcal{RA}$*.*

$\forall (s, R_j) \xrightarrow{!x} (s', R_i) \in E_{\mathcal{RA}}$*, si* $\exists (s, R_j) \xrightarrow{\delta} (s'', R_k) \in E_{\mathcal{RA}}$ *ou* $\exists (s, R_j) \xrightarrow{!x} (s''', R_l) \in E_{\mathcal{RA}}$ *alors* R_i *n'est pas complètement accessible.*

Preuve:

Un symbole de sortie modélise l'attente d'un symbole. D'après la modélisation, le comportement du système peut être, soit d'exécuter l'action et d'atteindre R_i, soit d'atteindre une autre région d'horloges. L'exécution de cette action ne peut pas être forcée. Ainsi R_i n'est pas complètement accessible. ∎

Dans ce cas, la difficulté à atteindre R_i dépend du nombre de transitions partant de (s, R_j), étiquetée par un symbole de sortie ou δ. Ainsi, la difficulté à atteindre R_i, avec $(S, R_j) \xrightarrow{!A} (S', R_i)$, noté $\rho(R_i, (S, R_j) \xrightarrow{!A} (S', R_i))$, est évaluée par :
$\rho(R_i, (S, R_j) \xrightarrow{!A} (S', R_i)) = \frac{1}{Card(\{(s,R_j) \xrightarrow{\{\delta \cup !\}x} (S,R) \in E_{\mathcal{RA}}\})+1}$

Proposition 4.3.6 *Soit un graphe des régions $\mathcal{RA} = (\Sigma_{\mathcal{RA}}, S_{\mathcal{RA}}, s^0_{\mathcal{RA}}, E_{\mathcal{RA}})$, $R_{\mathcal{RA}}$ l'ensemble des régions d'horloges de $\mathcal{RA}$ et $Exec_{\mathcal{RA}}$ l'ensemble infini des exécutions de $\mathcal{RA}$.*

$\forall (s, R_j) \xrightarrow{\delta} (s', R_i) \in E_{\mathcal{RA}}$, si $\exists (s, R_j) \xrightarrow{!x} (s'', R_k) \in E_{\mathcal{RA}}$ alors R_i n'est pas complètement accessible.

Preuve:

δ modélise un écoulement de temps. Ce n'est pas une action et ce ne peut être forcé. Donc, de même, la modélisation implique que le comportement du système permet soit d'atteindre R_i soit d'atteindre une autre région d'horloges. Donc, R_i n'est pas complètement accessible. ∎

De même que précédemment, la difficulté à atteindre R_i avec $(S, R_j) \xrightarrow{\delta} (S', R_i)$ dépend du nombre de transitions partant de (s, R_i) étiquetées par un symbole de sortie. Donc, cette difficulté, notée $\rho(R_i, (S, R_j) \xrightarrow{\delta} (S', R_i))$ est obtenue par :
$\rho(R_i, (S, R_j) \xrightarrow{\delta} (S', R_i)) = \frac{1}{Card(\{(s,R_j) \xrightarrow{!x} (S,R) \in E_{\mathcal{RA}}\})+1}$.

Nous avons montré, jusqu'à présent, qu'une région d'horloges n'est pas complètement accessible et nous avons évalué cette accessibilité pour le passage d'une transition. Or, dans la plupart des cas, il est nécessaire d'exécuter une séquence d'actions pour atteindre une région d'horloges. Soit, donc, pour un graphe des régions $\mathcal{RA}$, $SEQ_{R_i} \in Q_{\mathcal{RA}}$ l'ensemble des séquences de transitions permettant d'atteindre R_i. L'évaluation de l'accessibilité de R_i avec une séquence $\sigma \in SEQ_{R_i}$, notée $Reach(R_i, \sigma)$ est obtenue par :
$Reach(R_i, \sigma) = \prod_{((s,r) \xrightarrow{a} (s',r')) \in \sigma} \rho(r', ((s, r) \xrightarrow{a} (s', r')))$.

Illustrons l'accessibilité d'une région d'horloges par l'exemple suivant. Soit le graphe des régions illustré en Figure 4.20.

La difficulté à atteindre R_k avec $\sigma = (S, R_j) \xrightarrow{\delta} (S, R_{j+1})...(S, R_{l-1}) \xrightarrow{\delta} (S, R_l) \xrightarrow{!A} (S, R_k)$ est exprimée par $Reach(R_k, \sigma) = \frac{1}{2}(k - i)$ avec $i < k$. En effet, chaque région d'horloges $R_{j+1}, ..., R_l$ a une chance sur deux d'être atteinte depuis $R_j, ..., R_{l-1}$.

Finalement, l'accessibilité de R_i est mesurée en prenant en compte toutes les séquences σ_i de SEQ_{R_i}. Nous définissons un nouveau facteur, appelé le *Degré d'Accessibilité des*

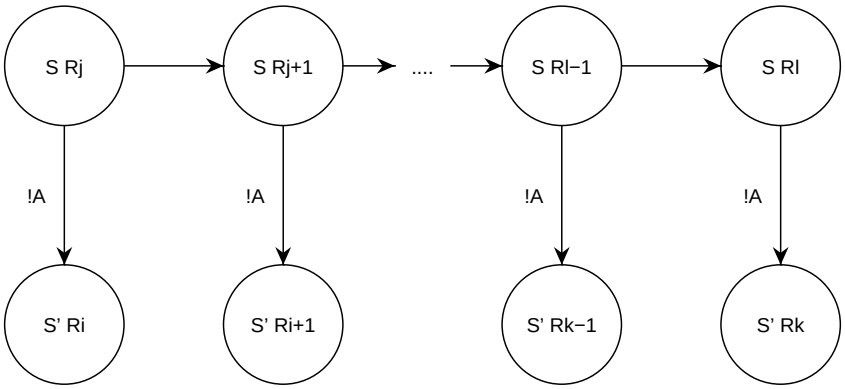


FIG. 4.20 – Un exemple d'Accessibilité de région d'horloges

Régions d'horloges et noté RT, qui est obtenu par :

$$RT(R_i) = \frac{\sum_{\sigma_i \in SEQ_{R_i}} Reach(R_i, \sigma_i)}{Card(SEQ_{R_i})} \text{ et } 0 < RT(R_i) \leq 1$$

Ainsi, si $RT(R_i)$ est égal à 1, R_i est complètement accessible, lors de la phase de test, par les horloges du système, avec n'importe quelle séquence de SEQ_{R_i}. De ce fait, toutes les actions pouvant s'exécuter à des valuations d'horloges de R_i sont aussi complètement accessibles. D'autre part, si $RT(R_i)$ est proche de 0, R_i est difficilement accessible car les chemins de SEQ_{R_i} permettent d'atteindre d'autres régions d'horloges. Le lecteur peut noter que RT ne peut être égal à 0, car d'après la Proposition 4.3.1, chaque région d'horloges R_i est accesssible au moins par une exécution.

Maintenant, nous pouvons définir le **Degré d'Accessibilité Temporelle des Graphes des Régions**. Ce facteur, noté $\mathcal{R}_{rg}$, évalue l'accessibilité de toutes les régions d'horloges d'un graphe des régions et est obtenu par :

$$\mathcal{R}_{rg}(A) = \frac{\sum_{R_i \in R_{\mathcal{RA}}} RT(R_i) * card(U_{R_i})}{card(AT)} \text{ et } 0 < \mathcal{R}_{rg}(A) \leq 1$$

Ainsi, plus $\mathcal{R}_{rg}$ est grand et proche de 1, plus les régions d'horloges du système sont accessibles et donc plus le système est testable. Si $\mathcal{R}_{rg}(A)$ est égal à 1, nous dirons que le système est complètement accessible temporellement. Si $\mathcal{R}_{rg}(A)$ est proche de 0, plusieurs actions du graphe des régions ne pourront être testées, car de nombreuses régions d'horloges ne pourront être atteintes par les horloges du système.

Les résultats suivants illustrent l'application de ce facteur sur le graphe des régions représenté en Figure 4.18.

Région d'horloges	$card(SEQ_{R_i})$	$Card(U_{R_i})$	RT_{rg}
R_1	5	3	1
R_2	5	3	$\frac{9}{10}$
R_3	2	1	1
R_5	18	2	$\frac{19}{36}$
R_6	54	2	$\frac{29}{72}$

TAB. 4.5 – Calcul du Degré d'accessibilité Temporelle d'un graphe des Régions

Le Degré obtenu est tel que $\mathcal{R}_t(\mathcal{A}) \simeq \frac{3}{4}$. Nous pouvons conclure que le système n'est pas complètement accessible temporellement, car les régions d'horloges R_2, R_5 et R_6 ne peuvent toujours être atteintes quelquesoit l'exécution du système. En effet, les transitions sortantes des états (S_5, R_2) et (S_5, R_5) ne peuvent pas toujours être passées. Si nous considérons toutes les exécutions du système, elles ont chacune une chance sur deux d'y être. Ainsi la couverture du test sera partielle et quelques transitions devront être testées plusieurs fois pour obtenir un verdict.

4.3.4 Degré de Contrôle des Graphes des Régions

L'évaluation du Degré de Contrôle pour le modèle du graphe des régions est presque similaire à celle obtenue sur le modèle des automates temporisés. En effet, les cas de test, générés à partir d'un graphe des régions, possèdent toujours des actions qui sont exécutées dans des intervalles de temps, ici modélisés par des régions d'horloges.

La principale différence provient du calcul des temps d'exécution minimal et maximal des préambules choisis pour le test (voir section 4.2.4). En effet, le laps de temps, nécessaire pour que les horloges puissent atteindre une région d'horloges depuis une autre et le laps de temps nécessaire pour qu'une action s'exécute le plus tard possible dans une région d'horloges doivent être calculés différemment que dans la section 4.2.4.

Calcul du laps de temps nécessaire pour atteindre une région d'horloges

Soit un graphe des régions $\mathcal{RA} = (\Sigma_{\mathcal{RA}}, S_{\mathcal{RA}}, s^0_{\mathcal{RA}}, E_{\mathcal{RA}})$.

Considérons la séquence $(s_1, R_1) \xrightarrow{a} (s_2, R_2) \xrightarrow{\delta} (s_3, R_3) \xrightarrow{b} (s_4, R_4) \in E^3_{\mathcal{RA}}$. Nous avons besoin de calculer le laps de temps, représenté par δ permettant d'atteindre R_3 depuis n'importe quelle valuation d'horloges $v_{init} \in R_2$, atteinte par l'ensemble des horloges après l'exécution de la première action. C'est-à-dire la plus petite valeur $d \in R^+$ telle que $v_{init} + d \in R_3$.

Soit $v_{init} = (v_1, ..., v_n)$. Par hypothèse, les horloges du système sont strictement croissantes et croissent de la même manière. Par conséquent, ces hypothèses nous permettent d'affirmer que les horloges prennent comme valeurs celles de l'équation :

$$f(x_1, ...x_n) = \begin{cases} x_1 - x_2 = v_1 - v_2 \\ ... \\ x_{n-1} - x_n = v_{n-1} - v_n \end{cases}$$

Ainsi, la première valuation d'horloges $v_{final} \in R_3$ atteinte par les horloges est obtenue en résolvant le système d'inéquations suivant :

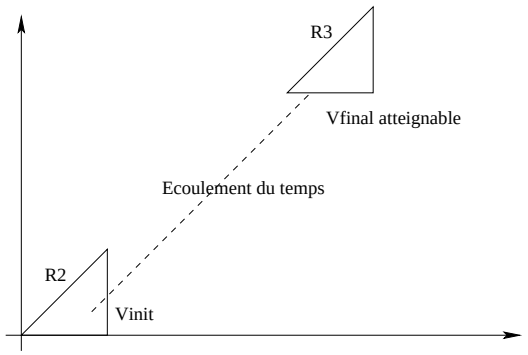


FIG. 4.21 – Atteindre une région d'horloges à la valuation d'horloges v_{final} depuis v_{init}

$$\Delta = \begin{cases} \text{Inéquations de } R_3 \\ x_1 - x_2 = v_1 - v_2 \\ ... \\ x_{n-1} - x_n = v_{n-1} - v_n \end{cases}$$

Finalement, le laps de temps δ est égal à la différence entre v_{init} et la plus petite solution v_{final} de Δ. Un exemple, illustrant ce laps de temps avec deux horloges, est donné en Figure 4.21.

Laps de temps nécessaire pour qu'une action s'exécute le plus tard possible dans une région d'horloges

Soit un graphe des régions $\mathcal{RA} =< \Sigma_{\mathcal{RA}}, S_{\mathcal{RA}}, s^0_{\mathcal{RA}}, E_{\mathcal{RA}} >$.

Une transition $(s, R) \xrightarrow{a} (s', R')$ modélise une action, pouvant s'exécuter entre la première valuation d'horloges v_{init} de R et la dernière v_{final} atteinte par le tuple d'horloges, lors d'une exécution. Le calcul du temps maximal d'exécution d'un préambule E_{max} nécessite de calculer la différence de temps entre v_{init} et v_{final}, qui n'est pas explicitement modélisée dans les graphes de régions. Un exemple de ce laps de temps est illustré en Figure 4.22.

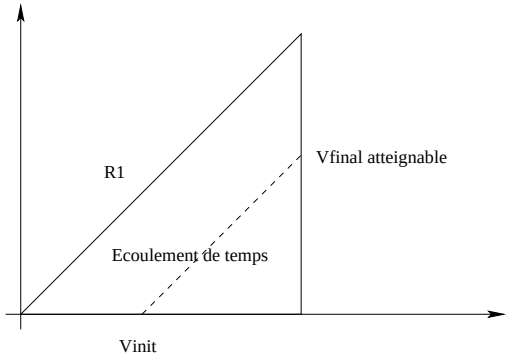


FIG. 4.22 – Atteindre une valuation d'horloges v_{final} depuis une valuation d'horloges v_{init} dans une même région d'horloges

Ce laps de temps est calculé comme suit : les horloges du système prennent toujours comme valeurs, depuis $v_{init} = (v1, ..., v_n)$ celles de l'équation suivante :

$$f(x_1, ...x_n) = \begin{cases} x_1 - x_2 = v_1 - v_2 \\ ... \\ x_{n-1} - x_n = v_{n-1} - v_n \end{cases}$$

Et v_{final} est la plus grande solution du système d'inéquations

$$\Delta = \begin{cases} \text{Inéquations de } R \\ x_1 - x_2 = v_1 - v_2 \\ ... \\ x_{n-1} - x_n = v_{n-1} - v_n \end{cases}$$

Finalement, le laps de temps est obtenu par la différence $v_{final} - v_{init}$.

De ce fait les temps minimal et maximal d'exécution d'un préambule p, $E_{min}(p)$ et $E_{max}(p)$ peuvent être obtenus par les algorithmes décrits en section 4.2.4, en remplaçant les calculs des laps de temps par ceux précédemment donnés.

L'évaluation de la difficulté à atteindre une transition $t_i nE_{\mathcal{RA}}$ d'un graphe des régions $\mathcal{RA}$, par un préambule p, peut être obtenue par la définition d'un nouveau facteur, appelé le *Degré de Contrôle de Transition*, noté CT_{rg} et obtenu par :

$$CT(p,t) = 1 - \frac{E_{min}(p) + E_{max}(p)}{max_{p_i \in Q_{\mathcal{RA}}}\{E_{min}(p_i) + E_{max}(p_i)\}}$$

$$\text{et } 0 \leq CT(p,t) \leq 1$$

Ainsi, nous pouvons définir un autre facteur, appelé le **Degré de Contrôle des Graphes des Régions** et noté $\mathcal{C}_{rg}$ permettant d'évaluer le coût d'exécution des préambules de P choisis pour le test, et donc évaluer le coût d'exécution du test. Celui-ci est obtenu par :

$$\mathcal{C}_{rg}(\mathcal{RA}) = \frac{\sum_{t \in AT} CT(p_{\xrightarrow{t}}, t)}{card(AT)} \text{ et } 0 \leq \mathcal{C}_{rg}(\mathcal{RA}) \leq 1$$

Par conséquent, plus $\mathcal{C}_{rg}$ est grand et proche de 1, plus le temps d'exécution des préambules est court, et donc plus le système est testable. Si $\mathcal{C}_{rg}(\mathcal{RA})$ est égal à 1, les préambules ne demandent comme temps d'exécution que celui nécessaire pour envoyer leurs symboles d'entrée et observer leurs symboles de sortie. Cette situation est obtenue si les régions d'horloges du graphe des régions $\mathcal{RA}$ sont représentées par le point $(0, ..., 0)$. A l'opposé, si $\mathcal{C}_{rg}(\mathcal{RA})$ est égal à 0, les préambules de P sont ceux du graphe des régions demandant le plus de temps pour être exécutés. Dans ce cas, le coût du test est grand.

L'application du degré de contrôle sur le graphe des régions, illustré en Figure 4.18 donne les résultats suivants, sachant que la séquence de $Q_{\mathcal{RA}}$ qui demande le plus grand temps d'exécution est $?b\delta!a?c!e\delta\delta\delta?d$ et donne $E_{min} = 2$ et $E_{max} = 4$. De plus, les régions d'horloges infinies sont bornées par $X = 4$ et $Y = 4$.

Préambule	Nb de transitions atteintes	E_{min}	E_{max}	CT_{rg}
ϵ	1	0	0	1
$?b$	1	0	2	$\frac{4}{5}$
$?b\delta$	1	2	2	$\frac{3}{5}$
$?b\delta!a$	1	2	2	$\frac{3}{5}$
$?b!e$	1	0	2	$\frac{4}{5}$
$?b!e\delta$	1	2	2	$\frac{3}{5}$
$?b!e\delta?d$	1	2	2	$\frac{3}{5}$
$?b!e\delta\delta$	1	2	4	$\frac{2}{5}$
$?b!e\delta\delta?d$	1	2	4	$\frac{2}{5}$
$?b!e\delta\delta\delta$	1	2	4	$\frac{2}{5}$
$?b\delta!a?c!e\delta\delta\delta$	1	4	6	0

TAB. 4.6 – Calcul du Degré de Contrôle d'un graphe des Régions

Comme $Card(AT) = 11$, nous obtenons le facteur final $\mathcal{C}_{rg}(\mathcal{RA}) = \frac{26}{55}$. Ces résultats montrent que l'exécution de l'ensemble P des préambules choisis, demande un coût moyen par rapport à l'exécution de tous les chemins du système. Cependant, le préambule $?b\delta!a?c!e\delta\delta\delta$ pourrait être remplacé par $?b!e\delta?d$ qui est un préambule s'exécutant avec un coût plus faible ($E_{min} = 2$, $E_{max} = 4$), et qui améliore donc la qualité de test de $\mathcal{RA}$.

4.3.5 Degré d'Indépendance temporelle des Graphes des Régions

Lors de la phase de transformation d'un automate temporisé en graphe des régions, de nouveaux états et de nouvelles transitions sont générés. Ceci est principalement dû au fait que pour chaque transition t, depuis l'ensemble des contraintes d'horloges de t, peuvent être générées plusieurs régions d'horloges. Pour une transition $t \in E_{\mathcal{A}}$, d'un automate temporisé $\mathcal{A}$, si n régions d'horloges satisfont les contraintes d'horloges de t alors nous obtiendrons n transitions équivalentes dans le graphe des régions, une transition par région d'horloges créée. Ainsi, l'action, modélisée par t sera au moins n fois testée, (au moins une fois dans chaque région d'horloges). Par conséquent, le coût du test d'une action du système pourra être plus important avec le graphe des régions que le coût du test de cette même action en gardant le modèle de l'automate temporisé. En fait, l'augmentation du coût est proportionnelle au nombre de régions d'horloges générées, satisfaisant l'exécution de l'action. Définissons donc un nouveau facteur mesurant la proportion de coût supplémentaire, obtenue en testant des actions à partir du graphe des régions.

Le nombre de régions d'horloges qui satisfont l'exécution d'une action de l'automate temporisé influençant le coût du test, soit donc N_R le nombre de région d'horloges avec lesquelles doit être testée une action source de l'automate temporisé.
$N_R(s \xrightarrow{a} s') = Card(\{t \in E_{\mathcal{RA}} \mid t = (S, R) \xrightarrow{b} (S', R'), b = a, S = s, S' = s'\}$.
Ce nombre de régions d'horloges implique donc qu'une action, modélisée par le concepteur, sera testée N_R fois, et plus N_R est grand, plus le coût du test sera élevé. Ceci motive la définition d'un premier facteur, appelé le Degré d'Influence Temporelle des Régions d'horloges et noté IND permettant d'évaluer le coût du test d'une action supplémentaire engendré par la transformation du graphe des régions. Ce facteur est obtenu par :

$$IND(s \xrightarrow{a} s', \mathcal{RA}) = 1 - \frac{N_R(s \xrightarrow{a} s') - 1}{card(R_{\mathcal{RA}} - 1)} \text{ et } 0 \leq IND(s \xrightarrow{a} s') \leq 1$$

Finalement, pour calculer le coût du test des actions de l'automate temporisé, elles-même transformées dans le graphe des régions, nous proposons le **Degré d'Influence Temporelle de Transformation en graphe des régions**, noté $\mathcal{IT}_{rg}$ et obtenu par :

$$\mathcal{IT}_{rg}(\mathcal{RA}) = \frac{\sum_{t \in E_{\mathcal{A}}} IND(t, \mathcal{RA})}{Card(AT)} \text{ et } 0 \leq \mathcal{IT}_{rg}(\mathcal{RA}) \leq 1$$

Ainsi, plus $\mathcal{IT}_{rg}(\mathcal{RA})$ est grand et proche de 1 plus le coût du test du graphe des régions est proche du coût du test de l'automate temporisé, et donc plus $\mathcal{RA}$ est testable. Si $\mathcal{IT}_{rg}(\mathcal{RA})$ est égal à 1, le coût du test de $\mathcal{RA}$ est le même que celui du test de $\mathcal{A}$, donc la transformation en graphe des régions n'a pas engendré d'efforts supplémentaires pour le test, autres que la transformation en elle-même. Si $\mathcal{IT}_{rg}(\mathcal{RA})$ est égal à 0, toute action de l'automate temporisé sera testée dans toutes les régions de $\mathcal{RA}$. Dans ce cas, le coût du test du graphe des régions est $card(R_{\mathcal{RA}})$ fois plus important que le coût du test de l'automate temporisé.

Transition de $E_{\mathcal{A}}$	N_R	$\mathcal{IT}_{rg}$
$S_1 \xrightarrow{?b} S_2$	1	1
$S_2 \xrightarrow{!a} S_2$	1	1
$S_2 \xrightarrow{?c} S_2$	1	1
$S_2 \xrightarrow{!e} S_4$	1	1
$S_4 \xrightarrow{?a} S_2$	1	1
$S_4 \xrightarrow{?d} S_5$	3	$\frac{1}{2}$
$S_5 \xrightarrow{!a} S_4$	3	$\frac{1}{2}$

TAB. 4.7 – Calcul du Degré d'Indépendance Temporelle d'un graphe des Régions

Comme $Card(AT) = 7$, nous obtenons un facteur tel que $I_{rg}(\mathcal{RA}) = \frac{6}{7}$ Nous pouvons conclure que le coût du test du graphe des régions demande plus d'efforts que celui du test de l'automate temporisé. Ceci est dû aux transitions $S_4 \xrightarrow{?d} S_5$ et $S_5 \xrightarrow{!a} S_4$ qui sont testées dans plusieurs régions d'horloges, donc pour lesquelles N_R est plus grand que 1.

4.3.6 Complexité des Degrés de Qualité de Test des graphes des régions

Le calcul de la complexité des degrés de qualité de test, définis pour le modèle de graphe des régions, est sensiblement le même que celui de la complexité des degrés de qualité de test des automates temporisés. Mais le lecteur peut noter que le nombre d'états, le nombre de transitions et le nombre d'intervalles de temps, modélisés par des régions d'horloges,

sont différents et plus grands que ceux obtenus depuis les automates temporisés. Ce qui implique, dans tous les cas, une complexité plus importante. A noter encore, que nous donnons des complexités sans rechercher d'optimisation d'algorithme, comme par exemple, le calcul de plusieurs degrés en même temps.

- Le calcul du Degré de Forme Temporelle des Graphes des Régions revient à chercher le nombre d'horloges qui donnent des valeurs infinies dans chaque région d'horloges. Par conséquent, la complexité du calcul du degré est proportionnelle à $C * R$, avec $R = card(R_{\mathcal{RA}})$ et $C = card(C_{\mathcal{A}})$.

- La complexité de la construction de $Q_{\mathcal{RA}}$ est proportionnelle à $M * K$ avec $M = card(S_{\mathcal{RA}})$ et $K = card(E_{\mathcal{RA}})$. Au pire, $card(Q_{\mathcal{RA}})$ séquences de $\mathcal{RA}$ permettent d'atteindre chaque région d'horloges de $R_{\mathcal{RA}}$. Ainsi, la complexité du Degré d'Accessibilité Temporelle des Graphes des Régions est proportionnelle à $R*card(Q_{\mathcal{RA}})+ M * K$.

- Le calcul du Degré de Contrôle des Graphes des Régions est le même que celui du Degré de Contrôlabilité des Automates Temporisés. Seuls le nombre de transitions et le nombre d'états sont plus importants dans le graphe des régions. Ainsi, la complexité du degré est proportionnelle à $2 * M * K$.

- Le but du Degré d'Influence Temporelle des Graphes des Régions est de rechercher pour chaque action d'un automate temporisé, combien d'actions ont été générées dans le graphe des régions. Donc, pour une transition de A, il faut au pire parcourir chaque transition de $\mathcal{RA}$. Donc la complexité est proportionnelle à $K * k$, avec $k = card(E_{\mathcal{A}})$.

4.4 Qualité de test liée à une méthodologie de test

Comme nous l'avons dit précédemment, nous proposons des degrés d'évaluation de la qualité de test qui peuvent être calculés avant la génération des séquences de test et indépendamment de la méthodologie choisie par le concepteur. Or, si ce dernier, connaît la méthodologie qui sera employée, il est possible d'affiner la mesure de la qualité de test.

Par exemple, de nombreuses méthodes de validation de systèmes temporisés considèrent que les symboles d'entrée sont émis une première fois le plus tôt possible, et une deuxième fois le plus tard possible. Ainsi la méthode, décrite dans [PF99a], a pour but de générer des séquences de test, depuis des objectifs de test, sur une spécification modélisée en graphe des régions. Chaque transition $(S_i, R_i) \xrightarrow{\{?,!\}x} (S_j, R_j)$ d'une séquence de test quelconque est appliquée à l'implantation comme suit :

- Si la transition est étiquetée par un symbole d'entrée, ce symbole est émis une fois à l'implantation le plus tôt possible et le plus tard possible, en respect avec les contraintes d'horloges. C'est-à-dire que le symbole est émis à la première valuation d'horloges de R_i atteinte par les horloges, puis à la dernière valuation d'horloges atteinte dans R_i.
- Si la transition est étiquetée par un symbole de sortie, ce symbole est attendu par

le testeur pour n'importe quelle valuation d'horloges de R_i.

Ainsi, l'exécution d'une séquence de test demande plusieurs applications de cette même séquence sur l'implantation. Connaissant la méthode d'exécution, un affinement du calcul du Degré de Contrôle peut être donné pour obtenir une meilleure précision sur le coût de l'exécution du test, en se rapprochant de la véritable phase d'exécution du test. Souhaitant garder la précocité de l'évaluation du degré, nous mesurerons ce dernier sur un ensemble de préambules P déjà choisis.

Degré de Contrôle des Graphes des Régions lié à la méthodologie de test [PF99a]

L'intérêt de la re-définition de ce facteur est de se rapprocher au plus près de l'exécution des préambules pris en compte lors de la construction des séquences de test de la méthodologie. D'une manière identique à l'exécution des séquences de test, les actions des préambules seront exécutées plusieurs fois pour évaluer leurs coûts d'exécution. Nous considérons donc , non pas un seul cas d'exécution, mais un arbre d'exécution, illustrant l'envoi des symboles d'entrée le plus tôt possible et le plus tard possible. Ne pouvant contrôler la réception des symboles de sortie, nous considérerons qu'ils sont reçus le plus tard possible, pour ne pas évaluer un coût plus faible que celui que nous pourrions obtenir lors de la phase de test.

Un arbre d'exécution est composé de branches étiquetées soit par un symbole du graphe des régions, soit par $wait(X)$, modélisant un moment d'attente égal à X. Pour un préambule p, cet arbre, noté T_p, est construit grâce à l'algorithme suivant :

Algorithme

Construction d'Arbre d'exécution
Entrée : Préambule p
Sortie : Arbre d'exécution de p
DEBUT :
$V = \{X_1, ..., X_n\}$ est un ensemble de variables modélisant l'ensemble d'horloges $x_1, ...x_n \in C_{\mathcal{A}}$

Const-arbre(p,$(s,[v]) \rightarrow (s',[v'])$,V)
%cas [v] pas atteinte par les horloges
SI $\exists X_i \in V \mid X_i \notin [v]$

ALORS
- Ajouter $\xrightarrow{wait(d)}$ avec $d \in \mathbb{R}^+$ tel que $\forall X_i \in V, X_i + d \in (v)$ et $\forall d' \in \mathbb{R}^+ < d, \exists X_i \in V \mid X_i + d' \notin [v]$
- $V = \{X_1 + d, ..., X_n + d\}$
- Cons-tree(p,next action of p,V)

%Cas des symboles d'entrée
SI A= ?I
SI $\exists d \in \mathbb{R}^+ \mid \forall X_i \in V, X_i + d \in [v]$

ALORS
- Ajouter $\xrightarrow{wait(d)}\xrightarrow{?I}$ avec $d \in \mathbb{R}^+ \mid \forall X_i \in V, X_i + d \in [v]$ et $\forall d' \in \mathbb{R}^+ > d, \exists X_i \in V \mid X_i + d' \notin [v]$
- $V_2 = \{X_1 + d, ..., X_n + d\}$
- Réinitialiser $X_i \in V_2$ si l'horloge x_i est réinitialisée avec la transition
- Const-arbre(p, prochaine transition de p,V_2)
- Ajouter une seconde branche $\xrightarrow{?I}$
- Réinitialiser $X_i \in V$ si l'horloge x_i est réinitialisée avec la transition
- Const-arbre(p, prochaine transition de p,V)

SINON
- Ajouter $\xrightarrow{?I}$
- Réinitialiser $X_i \in V$ si l'horloge x_i est réinitialisée avec la transition
- Const-arbre(p, prochaine transition de p,V)

%Cas des symboles de sortie
SI A= !O
SI $\exists d \in \mathbb{R}^+ \mid \forall X_i \in V, X_i + d \in [v]$

ALORS
- Ajouter $\xrightarrow{wait(d)}\xrightarrow{!O}$ avec $d \in \mathbb{R}^+ \mid \forall X_i \in V, X_i + d \in [v]$ et $\forall d' \in \mathbb{R}^+ > d, \exists X_i \in V \mid X_i + d' \notin [v]$
- $V_2 = \{X_1 + d, ..., X_n + d\}$
- Réinitialiser $X_i \in V_2$ si l'horloge x_i est réinitialisée avec la transition
- Const-arbre(p, prochaine action de p,V_2)

SINON
- Ajouter $\xrightarrow{!O}$
- Réinitialiser $X_i \in V$ si l'horloge x_i est réinitialisée avec la transition
- Const-arbre(p, prochaine action de p,V)

FIN

Par exemple, pour atteindre la transition $(S_4, R_1) \xrightarrow{!a} (S_2R_1)$ du graphe des régions de la Figure 4.18, nous utilisons le préambule "?b!e" qui donne l'arbre d'exécution de la

Figure 4.23. En parcourant chaque branche de l'arbre, nous pouvons en déduire que le temps d'exécution maximal est de 2.

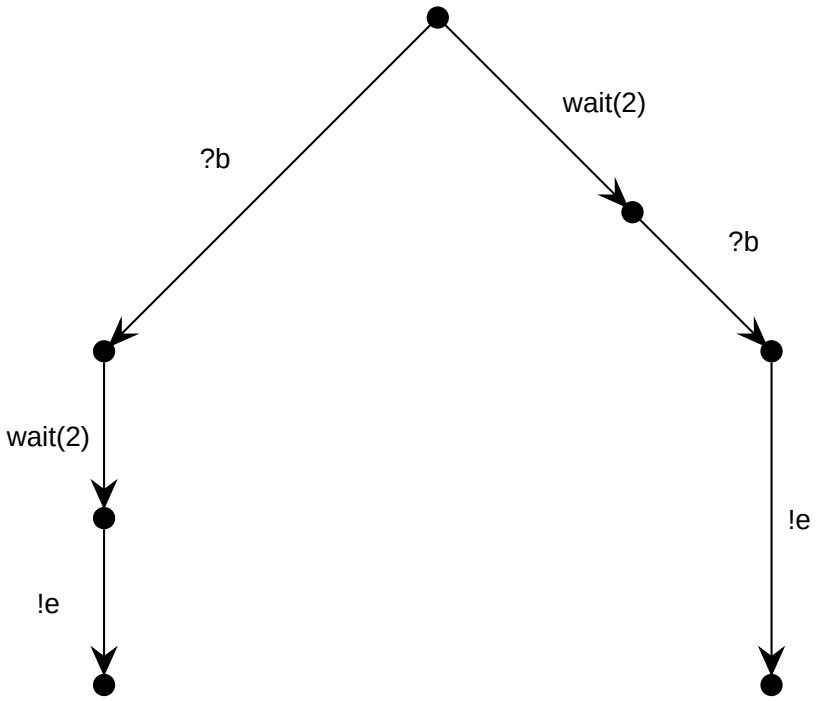


FIG. 4.23 – Arbre d'exécution de ?b !e

Par conséquent, pour un graphe des régions $\mathcal{RA} = (\Sigma_{\mathcal{RA}}, S_{\mathcal{RA}}, s^0_{\mathcal{RA}}, E_{\mathcal{RA}})$, nous définissons le *Degré de Contrôle de Transition*, noté CT qui mesure la difficulté à atteindre une transition $t \in E_{\mathcal{RA}}$ avec un préambule p. Soit T_p l'arbre d'exécution de p et $\delta(T_p)$, le temps total nécessaire à exécuter toutes les branches de T_p. CT est obtenu par :

$$CT(p,t) = 1 - \frac{\delta(T_p)}{max\{\delta(T_{p_i}) \mid p_i \in Q_{\mathcal{RA}}\}}$$

$$\text{et } 0 \leq CT(p,t) \leq 1$$

Ce degré évalue le coût du test, induit par p en comparant le temps d'exécution de Tp avec le temps d'exécution des arbres d'exécution des préambules de $Q_{\mathcal{RA}}$.

Nous pouvons maintenant, mesurer la difficulté à atteindre toutes les transitions du système, en définissant le **Degré de Contrôle du Graphe des Régions** lié à la méthodologie de [PF99a]. Notons encore $p_{\xrightarrow{t}}$, le préambule permettant d'atteindre la transition $t \in E_{\mathcal{RA}}$. Ce degré, noté $\mathcal{C}_{rg}$ est obtenu par :

$$\mathcal{C}_{rg}(\mathcal{RA}) = \frac{\sum_{t \in AT} CT(p_{\xrightarrow{t}}, t)}{card(AT)} \text{ et } 0 < \mathcal{C}_{rg}(\mathcal{RA}) \leq 1$$

Si $\mathcal{C}_{rg}(\mathcal{RA})$ est égal à 1, les préambules du système ne nécessitent aucun coût pour être exécutés. Si $\mathcal{C}_{rg}(\mathcal{RA})$ est égal à 0, les préambules choisis sont ceux qui demandent le plus grand temps d'exécution du système.

4.5 Mesure complète de la qualité de test

La mesure de la qualité de test des systèmes temporisés, modélisés par des automates temporisés ou des graphes des régions, est obtenue en calculant les vecteurs de testabilité TV, définis en section 4.1.3. Une valeur unique de qualité de test peut être donnée en combinant chaque composante des vecteurs, avec une prédominance pour certaines composantes selon l'ordre d'intérêt des critères que le concepteur a choisi. Si nous considérons que l'importance des degrés est équivalente, la qualité de test complète d'un système temporisé $\mathcal{A}$ est définie comme étant égale à la somme de la norme des deux vecteurs.

Définition 4.5.1 *La qualité de Test complète d'un système temporisé $\mathcal{A}$, notée QT est donnée par*
- $QT = ||TV|| + ||TTV_t||$ *si $\mathcal{A}$ est un automate temporisé,*
- $QT = ||TV|| + ||TTV_{rg}||$ *si $\mathcal{A}$ est un graphe des régions*

La qualité de test de $\mathcal{A}$ est donc optimale si nous obtenons comme vecteurs $TTV_t(\mathcal{A}) =< 1,1,1,1 >$ et $TV(\mathcal{A}) =< 1,1,1,1,1 >$, si $\mathcal{A}$ est un automate temporisé ou $TTV_{rg}(\mathcal{A}) =< 1,1,1,1 >$ et $TV(\mathcal{A}) =< 1,1,1,1,1 >$, si $\mathcal{A}$ est un graphe des régions. Cependant, obtenir un vecteur $TTV =< 1,1,1,1 >$ semble difficile en raison de l'opposition entre les performances du système, la couverture du test du système et le coût du test. En fait, la meilleure qualité de test d'un système réside encore une fois dans les critères de qualité du concepteur.

Par exemple, si le concepteur souhaite tester un logiciel ou un protocole, de la manière la plus rapide qu'il soit et sans se préoccuper de la couverture du test, le calcul du vecteur $< \mathcal{C}_t, \mathcal{IT}_t >$ ou $< \mathcal{C}_{rg}, \mathcal{IT}_{rg} >$ est suffisant pour évaluer la qualité du test. Au contraire, s'il veut obtenir une couverture complète de test du système, indépendamment du coût, seul le vecteur $< \mathcal{S}_t, \mathcal{R}_t >$, ou $< \mathcal{S}_{rg}, \mathcal{R}_{rg} >$ a besoin d'être évalué.

L'intérêt de mesurer la qualité de test avant génération des séquences de test couvre deux aspects, qui ont été présentés dans [VM95] :
- la sélection d'un ensemble de tests ayant une grande capacité de révéler les fautes.
- la modélisation de système dont l'exécution révèle automatiquement des dysfonctionnements au niveau de l'implantation si celle-ci contient des erreurs.

Suivant la méthodologie de test utilisée, la qualité de test et les degrés que nous avons proposés contribuent à améliorer l'un ou l'autre des aspects. En effet, en ce qui concerne les méthodologies à base d'objectifs de test [FPS00, PLC98], la qualité de test permet de choisir des séquences permettant d'aboutir à un verdict et de couvrir le test de l'objectif de test, mais aussi d'obtenir un coût de test le plus faible en choisissant par exemple l'ensemble des préambules à l'aide du degré de contrôle.

Les méthodologies, générant des séquences de test sur l'ensemble du système [AKA00, ENDKE98, PF99b, LC97, RNHW98, SVD01], demanderont à ce que l'implantation révèle

automatiquement ses dysfonctionnements. Ainsi, les degrés permettent de connaître les modifications à apporter et tendent à améliorer la qualité du système en vue d'un test.

4.6 Amélioration de la Qualité de Test des Systèmes Temporisés

Amélioration du Degré de Forme Temporelle

L'unique solution à apporter pour améliorer ce degré est d'obtenir des intervalles de temps ou des régions d'horloges bornées. La transformation d'un comportement infini du système en un comportement fini semble difficile si le système doit garder les mêmes propriétés comportementales. La solution que nous proposons est de borner chaque horloge x_i par un réel B_i. Pour garder le comportement infini, nous ajoutons des contraintes d'horloges de telle sorte que si x_i atteint la borne B_i alors x_i reprend la valeur minimale satisfaisant le franchissement de la transition.

Ainsi, pour une transition d'un automate temporisé, $s \xrightarrow[xopA]{t} s'$, avec $op \in \{>, \geq\}$, nous ajoutons une transition $s \xrightarrow[xopA, x<B, x:=A]{} s$, avec $A \in \mathbb{R}$, $B \in \mathbb{R}$ et $B > A$. Ces contraintes permettent d'obtenir un intervalle de temps fini, tout en gardant un comportement infini.

L'inconvénient majeur de cette transformation est qu'elle apporte des réinitialisations sur les horloges qui ne sont pas définies dans le modèle de l'automate temporisé. Ces réinitialisations obligent de modifier le modèle de l'automate temporisé et peuvent rendre très difficile le passage en graphe des régions, car elles pourraient impliquer des sauts d'une région d'horloges à une autre. Ainsi cette solution n'est envisageable que si la méthodologie de test employée ne demande pas de génération de graphe des régions.

Un autre faible remède à ce mal, consiste à borner d'une manière cohérente les intervalles de temps ou régions d'horloges par rapport aux autres intervalles ou régions d'horloges bornées du système. Prenons l'exemple des régions d'horloges. Soit $INF_{x_i}(R)$, la plus petite valeur que peut prendre l'horloge x_i dans R, et $SUP_{x_i}(R)$, la plus grande. Une région d'horloges Reg d'un graphe des régions $\mathcal{RA}$, infinie peut être bornée, pour chaque horloge x_i donnant des valeurs infinies, par une valeur B_i telle que : $B_i = INF_{x_i}(R) + max\{(SUP_{x_i}(R_i) - INF_{x_i}(R_i)) \mid R_i$ est une région d'horloges de $\mathcal{RA}\}$. Ceci permet d'obtenir des régions d'horloges bornées homogènes, c'est-à-dire de taille identique. Les intervalles de temps peuvent être facilement bornés de la même manière.

Amélioration du Degré d'Accessibilité Temporelle

Pour les automates temporisés, l'amélioration du degré peut être obtenue, soit en transformant les intervalles de temps pour qu'ils satisfassent les propositions d'accessibilité, soit en élaborant un autre algorithme de construction de ces intervalles. L'algorithme décrit au chapitre 6, permet de construire des régions d'horloges, qui sont étiquetées aux transitions et qui satisfont les contraintes d'horloges des transitions auxquelles elles sont étiquetées, de l'automate temporisé. Cette modélisation, appelée automate des régions, permet d'obtenir des ensembles de régions d'horloges qui sont complètement accessibles depuis un autre ensemble de régions d'horloges. Et ces régions d'horloges ne sont autre

que des intervalles dépendant de plusieurs horloges.

Pour augmenter l'accessibilité des régions d'horloges, il faudrait agréger les régions d'horloges qui satisfont la même exécution d'une transition étiquetée par un symbole de sortie. En effet, même après minimisation, des régions d'horloges peuvent encore être réunies localement dans le graphe, car d'après l'algorithme de minimisation décrit dans [ACH+92], si deux régions d'horloges satisfont localement l'exécution d'une action mais ne satisfont pas ensemble l'exécution d'une autre action alors ces deux régions d'horloges sont considérées séparément sur l'ensemble du graphe des régions.

Or, réunir de telles régions d'horloges localement, c'est-à-dire dans certains états permettrait d'améliorer l'accessibilité temporelle. Par exemple, la Figure 4.24 montre un cas où cette réunion de régions d'horloges est possible. La Figure 4.25 illustre un cas où, par contre, ça ne l'est pas. Dans ce dernier cas, une réunion de régions d'horloges est impossible, car le comportement du système est différent dans R_2, R'_{n-1} et les autres régions d'horloges.

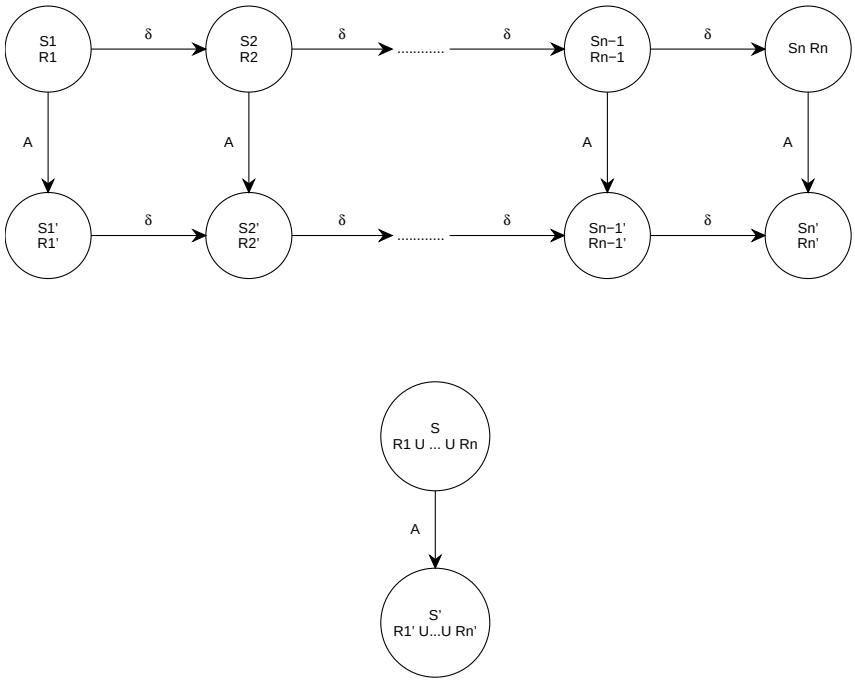


FIG. 4.24 – Cas d'union de régions d'horloges

Proposition 4.6.1 *Soit un graphe des régions* $\mathcal{RA} =< \Sigma_{\mathcal{RA}}, S_{\mathcal{RA}}, s^0_{\mathcal{RA}}, E_{\mathcal{RA}} >$ *et :*

- $S \subseteq S_{\mathcal{RA}}$
- $(S_i, R_i) \xrightarrow{\delta} (S_{i+1}, R_{i+1}) \in E_{\mathcal{RA}}$, $1 \leq i < n$, *avec* $n \in \mathbb{N}$, $(S_i, R_i) \in S, (S_{i+1}, R_{i+1}) \in S$,
- $(S'_i, R'_i) \xrightarrow{\delta} (S'_{i+1}, R'_{i+1}) \in E_{\mathcal{RA}}$, $1 \leq i < n$, *avec* $(S'_i, R'_i) \in S, (S'_{i+1}, R'_{i+1}) \in S$,
- $(S_j, R_j) \xrightarrow{a} (S'_j, R'_j) \in E_{\mathcal{RA}}$, $1 \leq j \leq n$, *avec* $A \in \Sigma_{\mathcal{RA}}$

(graphe illustré en Figure 4.24)

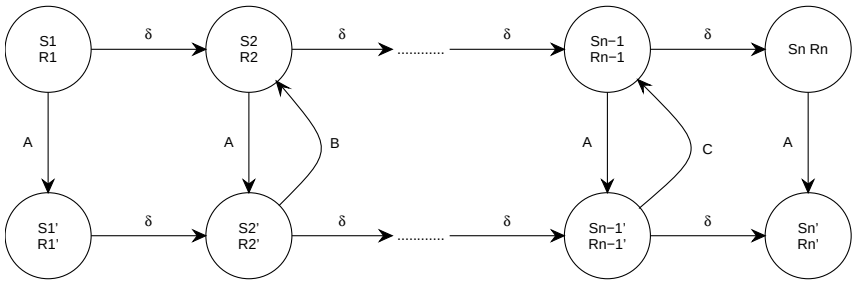


FIG. 4.25 – Cas où l'union de régions d'horloges est impossible

Les régions d'horloges R_1,...,R_n, puis R'_1,...,R'_n peuvent être réunies et donner la transition $t = (S, R_1 \cup ... \cup R_n) \xrightarrow{a} (S', R'_1 \cup ... \cup R'_n)$, sans pour autant modifier le comportement du système, ssi :

$\forall (S_i, R_i) \in S, \forall (S_j, R_j) \in S$, *i* $\neq j$, *tel que* $\exists (S_i, R_i \xrightarrow{\delta} (S_j, R_j) \in E_{\mathcal{RA}}$,

- *si* $\exists (S_i, R_i) \xrightarrow{b} (S_k, R_k)$, *alors* $\exists (S_j, R_j) \xrightarrow{b} (S_l, R_l)$,
- *si* $\exists (S_k, R_k) \xrightarrow{b} (S_i, R_i)$, *alors* $\exists (S_l, R_l) \xrightarrow{b} (S_j, R_j)$,

Cette proposition signifie plus simplement, que des régions d'horloges peuvent être réunies si et seulement si tout état, étiqueté par ces régions d'horloges, a les mêmes transitions entrantes et sortantes.

Preuve:

Supposons que $\forall (S_i, R_i) \in S, \exists (S_i, R_i) \xrightarrow{a} (Si', R'_i)$. Ceci modélise que l'action peut être exécutée pour toute valuation d'horloges de $R_1 \cup ... \cup R_n$ et que les horloges atteignent une valuation d'horloges des régions d'horloges R'_1 ou ... ou R'_n.

Si nous réunissons les régions d'horloges, nous obtenons $t = (S, R_1 \cup ... \cup R_n) \xrightarrow{a} (S', R'_1 \cup ... \cup R'_n)$. Ceci modélise toujours que l'action peut être exécutée pour toute valuation d'horloges de $R_1 \cup ... \cup R_n$ et que les horloges atteignent une valuation d'horloges de $R'_1 \cup ... \cup R'_n$. Par conséquent, le comportement du système reste le même.

Supposons maintenant qu'il existe une transition $\exists (S_i, R_i) \xrightarrow{b} (S_k, R_k)$ telle qu'il existe un état $(Sj, Rj) \in S$ n'ayant pas cette transition sortante.

Si les régions d'horloges $R_i (1 \leq i \leq n)$ et $R'_j (1 \leq j \leq n)$ sont réunies, nous obtenons $(S, R_1 \cup ... \cup R_n) \xrightarrow{a} (S', R'_1 \cup ... \cup R'_n)$ et $(S, R_1 \cup ... \cup R_n) \xrightarrow{b} (S', R'_1 \cup ... \cup R'_n)$. En effet, les régions d'horloges $R1$, ..., R_n ont été remplacées par $R_1 \cup ... \cup R_n$, et pour respecter le comportement du système, nous devons ajouter $(S, R_1 \cup ... \cup R_n) \xrightarrow{b} (S', R'_1 \cup ... \cup R'_n)$.

Or, dans ce cas, la contrainte d'horloges sur la transition $(S_i, R_i) \xrightarrow{b} (S_k, R_k)$ n'est pas respectée car, après réunion, la transition peut être passée pour au moins une valuation d'horloges différente de R_i. Donc le comportement du système est modifié.

Le même raisonnement peut être appliqué pour une transition entrante à (S_i, Ri) ■

Ainsi, en réunissant les régions d'horloges qui peuvent l'être, le degré d'accessibilité temporelle peut être amélioré mais rien ne montre que nous pouvons toujours obtenir un

système complètement accessible temporellement.

Prenons l'exemple du graphe des régions de la Figure 4.18. Celui-ci peut être rendu complètement accessible en agrégeant les régions R_2, R_5 et R_6. Le graphe des régions obtenu est illustré en Figure 4.26.

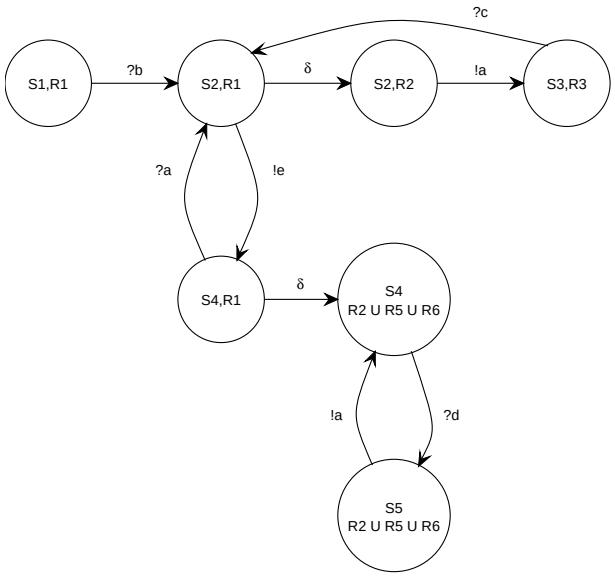


FIG. 4.26 – Graphe des régions modifié et complètement accessible

L'inconvénient de cette modification est qu'elle demande de considérer localement les régions d'horloges dans la spécification, c'est-à-dire de considérer l'ensemble des régions d'horloges non pas sur l'ensemble du système mais sur chacun de ces états. Ainsi, la définition du graphe des régions n'est pas respectée, ce qui implique la définition d'un autre modèle.

Amélioration du Degré de Contrôle

Que le système $\mathcal{A}$ soit un automate temporisé ou un graphe des régions, ce degré peut être optimal si l'ensemble des préambules choisi demande le coût le plus faible d'exécution, ou plus précisément, si ces préambules sont ceux qui peuvent être exécutés le plus vite possible. Ceci peut être fait, en calculant le degré sur chaque séquence de $Q_{\mathcal{A}}$ et en prenant l'ensemble minimal des séquences, permettant d'atteindre toutes les transitions de $\mathcal{A}$ et donnant des degrés de contrôle les plus proches de 1.

Amélioration du Degré d'Indépendance Temporelle

Pour les automates temporisés, ce degré dépend du nombre d'horloges du système. Comme nous l'avons décrit en section 4.2.3, ce nombre est rarement minimal. L'algorithme, décrit dans [DY96], permet de réduire et d'obtenir l'ensemble minimal d'horloges que requiert le système pour garder le même comportement. Et cette réduction permet d'améliorer le degré d'indépendance temporelle et donc la qualité de test du système. Mais, l'algorithme ne permet pas d'obtenir une seule horloge pour le système. Ainsi, il

n'est donc pas toujours possible d'obtenir un degré égal à 1.

Pour les graphes des régions, améliorer ce facteur reviendrait à agréger les régions d'horloges dans lesquelles une action de l'automate temporisé peut être exécutée, de la même manière que dans la section 4.6. Ainsi, de même que pour l'accessibilité, ce degré peut être amélioré, mais la transformation de la spécification demande une modification de la définition du graphe des régions. De plus, il n'est pas toujours possible d'obtenir un degré égal à 1.

Comme nous venons de le voir, améliorer la qualité du test des systèmes temporisés est loin d'être trivial, car l'amélioration peut demander une refonte du comportement du système. De plus, les transformations de la spécification ne peuvent toujours conduire à l'obtention d'une qualité de test optimale. En effet, les intervalles de temps infinis sont difficiles à transformer, l'accessibilité des régions d'horloges demande une agrégation qui n'est pas toujours possible, et l'ensemble d'horloges d'un système ne peut pas toujours être réduit à une horloge. Dans ce cas, la qualité de test aura pour rôle d'informer le concepteur des parties du système qui ne pourront pas toujours être testées.

4.7 Quelques conclusions sur l'étude de la qualité des systèmes temporisés

Nous avons défini différents facteurs permettant d'évaluer la qualité de test des automates temporisés et des graphes des régions. Ceux-ci considèrent les contraintes d'horloges de ces modèles. Nous avons montré comment ils influencent la phase de test et comment ils peuvent être calculés. Nous avons décrit que des implantations temporisées peuvent être seulement partiellement testées à cause de la propriété d'infinité et de non accessibilité que peuvent avoir ses intervalles de temps. Ainsi, la qualité de test de tels systèmes ne dépend pas uniquement de leur observabilité et de leur contrôlabilité. Ceci motive la définition de la qualité de test des systèmes temporisés modélisés par des automates temporisés ou des graphes de régions.

Définition 4.7.1 *Un système temporisé, modélisé soit par un automate temporisé soit par un graphe des régions, est testable ssi il est observable, contrôlable et si toute valeur de temps peut toujours être atteinte par les horloges : c'est-à-dire si tout intervalle de temps ou région d'horloges est borné et peut être complètement accessible par le tuple d'horloges.*

Discussion

Ce travail nous a permis d'étudier les propriétés temporelles des automates temporisés et des graphes des régions. Pour finir, il serait donc intéressant de commenter l'apport de la transformation du graphe des régions, transformation qui demande un coût très important et qui peut engendrer une explosion combinatoire du nombre d'états créés ([AD94]).

Un premier avantage de cette transformation en graphe des régions est que la notion de temps, représentée dans l'automate temporisé uniquement par des inégalités sur horloges,

est exprimée par des régions d'horloges, pouvant être obtenues même si la spécification contient des contraintes du type x_i op $x_j + A$. Ces régions d'horloges simplifient le test de conformité, et plus précisément, la génération des séquences de test. En effet, d'une part, nous avons une vue plus représentative du comportement du système, car le graphe modélise le comportement du système à chaque moment, et les phases d'attentes (laps de temps) .

Le second avantage de cette transformation est que nous obtenons ainsi des régions d'horloges qui sont successeurs temporels et donc accessibles (section 4.3.3). Ceci évite le test d'actions à des valeurs de temps qui ne pourront jamais être atteintes par les horloges.

Par conséquent, la transformation en graphe des régions est très intéressante et permet un test plus précis. Cependant, ce modèle n'est pas exempt de défauts. Mis à part le nombre d'états exponentiel qui peut être généré, le graphe des régions ne garantit pas l'accessibilité complète des régions d'horloges et implique que certaines actions de l'automate temporisé seront testées plusieurs fois (voir section 4.3.3 et section 4.3.5).

Ce problème d'accessibilité est dû au fait qu'un intervalle de temps de l'automate temporisé est "trop" découpé en régions d'horloges. En effet, si un intervalle de temps est découpé en plusieurs régions d'horloges et si le testeur ne peut forcer l'exécution de l'action alors cette exécution se produira dans une seule région d'horloges. Selon le découpage, il peut être très difficile d'atteindre toutes les régions d'horloges. Cette accessibilité ne dépend que de l'exécution de l'implantation. De plus, si l'intervalle est découpé, cela implique que le test, d'une action de l'automate temporisé, se fasse plusieurs fois dans le graphe des régions. Cela implique donc un coût parfois beaucoup plus élevé du test, mais, à contre-sens, un raffinement du test.

Ainsi, nous déduisons que la représentation des régions d'horloges est nécessaire, mais que le graphe en lui-même engendre des problèmes nuisant à la qualité du test. Or, il est difficile d'appliquer une méthode de test directement sur l'automate temporisé car les contraintes temporelles sont difficilement exploitables directement. Une autre solution serait de conserver la structure de l'automate temporisé en y ajoutant la représentation des régions d'horloges. Cette nouvelle modélisation, que nous appelons automate des régions, sera présentée au chapitre 6.

Chapitre 5

Méthodologies de Test Temporisées

Pratiquer des tests sur des systèmes ou des applications constitue une étape fondamentale dans le cycle de vie du logiciel. Cependant, peu d'outils rendent possible le test de systèmes temporisés. Bien que les chercheurs s'intéressent de plus en plus aux systèmes temporisés, les tester reste difficile. En effet, le temps reste une notion difficilement observable. Pourtant, qu'il soit vu comme une notion continue [AD94, AFH99] ou discrète [Ost92, Ost94], celui-ci est généralement représenté par de simples variables, appelées des horloges. Mais, au fil des exécutions des systèmes, ces variables semblent bien capricieuses : elles peuvent donner des valeurs infinies, des valeurs non accessibles, peuvent être réinitialisées de temps en temps. Et c'est cet amalgame qui rend le test temporisé complexe et qui augmente son coût. Plusieurs méthodologies de test ont cependant été proposées (dont certaines ont été décrites au chapitre 2.5) et peuvent se répertorier, en deux classes :

– les méthodes de test basées sur une approche par objectif de test. Celles-ci consistent à tester que l'implantation satisfasse certaines propriétés comportementales et/ou

temporelles réunies dans un objectif de test. Ce concept d'objectif de test (normalisé ISO [ISO91b]), fort apprécié par les industriels, de par le coût faible des tests engendrés, a donné naissance à plusieurs méthodologies temporisées dans [CCKS95, PLC98, FPS00].

– les méthodes de test à génération de séquences de test automatique, vérifiant l'ensemble de l'implantation par rapport à la spécification et qui garantissent une équivalence plus ou moins forte entre ces deux modèles. Plusieurs méthodes ont été présentées pour divers modèles de systèmes temporisés comme les automates modélisés avec temps discret [Ost92], les automates modélisés avec temps dense [LC97, RNHW98, CL97, COG98, SVD01, AKA00, NS01, AS96], les graphes des régions [ENDKE98, ENFDE97].

Parmi tous ces travaux, nous avons noté que certains points pouvaient être améliorés, tant du point de vu du coût de test, que du point de vu du choix des tests à effectuer. Ayant aussi montré, dans le chapitre précédent, que des problèmes d'accessibilité et d'infinité du temps peuvent subvenir, notre contribution dans ce chapitre vise à présenter, pour chaque classe précédemment citée, une nouvelle méthodologie de test basée sur le modèle du graphe des régions. Plus précisément, nous décrivons les points suivants :

- La première partie présente une méthodologie de test basée sur objectif de test [SPF01, SF01c]. Deux méthodologies de test ont déjà été proposées pour vérifier la satisfaction d'objectifs de test sur une implantation. Ce que nous apportons par rapport à celles-ci est premièrement la prise en compte d'objectifs de test acceptants et refusants, c'est-à-dire d'objectifs de test qui doivent être respectivement satisfaits dans l'implantation et d'objectifs contenant des propriétés qui doivent être satisfaites et insatisfaites. Comme la méthodologie décrite dans [FPS00], nous transformons l'automate temporisé en graphe des régions : ceci permet de pouvoir utiliser des contraintes d'horloges du type $x_i = x_j + A$ qui permettent de comparer les horloges entre elles. Sans la transformation en graphe des régions, ces contraintes ne peuvent être prises en compte. Par rapport à la méthodologie de [FPS00], nous ajoutons le fait que les propriétés temporelles de l'objectif de test puissent être différentes de celles de la spécification. Ceci, implique qu'une synchronisation entre l'objectif de test et l'implantation doit être effectuée pour pouvoir vérifier que l'implantation satisfasse l'objectif de test. Par la suite, nous décrivons avec précision la phase d'exécution des séquences de test obtenues. Et finalement, nous décrivons la complexité complète de la méthodologie.
- La deuxième partie présente une méthodologie testant l'ensemble de l'implantation, grâce à la notion de caractérisation d'états. Plusieurs méthodologies ont été proposées pour caractériser les états temporisés de graphes des régions et ainsi générer des séquences de test. Or, plutôt que d'adapter la caractérisation d'états directement sur les graphes des régions, leurs auteurs ont préféré modifier le graphe, par une succession de transformations, pour appliquer une méthode "classique" de caractérisation, la méthode Wp ([FBK+91]). Ces transformations apportant un coût supplémentaire important, nous proposons donc une méthode originale de caractérisation d'états de graphe des régions, qui ne nécessite pas d'apporter de modifications au modèle. Nous définissons donc, dans un premier temps, la caractérisation d'états temporisés

et proposons un nouvel algorithme de génération de l'ensemble de caractérisation. Puis nous décrivons les étapes permettant de générer des séquences de test. Cette caractérisation étant une nouvelle approche, nous présentons de plus une nouvelle relation de conformité entre spécification et implantation. De même que précédemment, nous montrons comment exécuter les séquences de test obtenues. Puis, nous décrivons la complexité complète de la méthodologie.

- La dernière partie consiste à transformer les séquences de test, générées depuis les méthodologies précédentes en TTCN (Tree and Tabular Combined Notation), qui est un langage normalisé ISO [ISO91a], modélisant des séquences de test et qui est reconnu par la plupart des testeurs. Cette partie contribue premièrement à montrer l'exécution de nos séquences de test au plus bas niveau, car les expressions TTCN illustrent quelles doivent être les émissions et les réceptions de symboles, lors de la phase de test, les moments de ces émissions et réceptions et les laps de temps entre celles-ci. Cette phase de transformation de séquences de test en TTCN prouve aussi que nos travaux peuvent directement s'appliquer dans le monde industriel.

5.1 Méthodologie basée sur les Objectifs de Test Temporisés

Les industriels sont généralement soucieux du coût et du temps nécessaire au processus de développement d'un produit. C'est pourquoi, ceux-ci préfèrent utiliser des méthodes de test de conformité, ne prenant pas en compte la spécification complète mais vérifiant des propriétés pertinentes de celle-ci. Et l'ensemble de ces propriétés sont appelées objectif de test (section 2.2.2). Des méthodologies de test, basées sur objectif de test, ont été proposées pour les systèmes non temporisés dans [Bri87, Tre96], pour les automates temporisés dans [PLC98]. Nous avons, nous-même, décrit une méthodologie axée sur le modèle du graphe des régions dans [FPS00], qui permettait de tester de propriétés comportementales mais qui gardait les mêmes propriétés temporelles de la spécification. Le travail suivant, propose de montrer comment générer des séquences de test avec objectif de test temporisé contenant des contraintes d'horloges quelconques.
Nous considérons aussi qu'un objectif de test temporisé peut être acceptant ou refusant, ce que ne permettent pas les méthodes de test temporisé précédentes (elles ne considèrent que des objectifs acceptants). Un tel objectif de test ne peut être directement appliqué à l'implantation car il peut représenter un comportement différent de celui de la spécification. Par conséquent, il est injecté dans des chemins de la spécification pour être ensuite indirectement appliqué, Un produit, appelé produit synchronisé, doit donc être effectué entre l'objectif de test et la spécification.

Par conséquent, la partie principale de cette étude consiste à définir ce produit synchronisé entre la spécification et l'objectif de test temporisé, permettant de générer des séquences de test. Par la suite, nous proposons une méthode d'exécution de ces séquences prenant en compte l'accessibilité des régions d'horloges.

5.1.1 Objectif de Test Temporisé

Un objectif de test, pour les méthodologies classiques (non temporisées) [Bri87, Tre96] représente une suite de propriétés comportementales, qui sont intégrées dans les séquences de test, puis qui doivent ou ne doivent pas être vérifiées par l'implantation sous test.

Cette technique permet au concepteur de choisir le comportement à tester, et réduit l'exploration de la spécification, donc réduit le coût du test. Un objectif de test est modélisé par un graphe acyclique, contenant des propriétés comportementales et/ou temporelles. Celui-ci peut être soit :

- acceptant, ce qui indique que l'objectif de test doit être satisfait lors de la phase de test. Un objectif de test acceptant doit inclure des propriétés comportementales et temporelles appartenant à la spécification. Il permet de tester une partie de la spécification.

- refusant, ce qui indique que l'objectif de test contient des propriétés qui doivent être refusées, lors de la phase de test, par l'implantation. Dans ce cas l'objectif de test contient des propriétés comportementales et/ou temporelles appartenant ou non à la spécification.

Un objectif de test, sur les systèmes temporisés, doit donc modéliser des propriétés comportementales, mais aussi contenir des propriétés temporelles, qui peuvent être différentes de celles décrites dans la spécification. Nous proposons qu'un objectif de test temporisé soit modélisé par un automate temporisé.

Un objectif de test acceptant, ne doit contenir que des propriétés qui peuvent se vérifier dans la spécification. Comme celui-ci doit être complètement vérifié lors de la phase de test, pour montrer qu'il est acceptant nous étiquetons les états finaux par "accept". Par contre, les objectifs de test refusant sont aussi composés de propriétés refusantes n'appartenant pas à la spécification. Pour savoir quelles propriétés doivent être refusées, pour chaque transition de l'objectif de test, l'état d'arrivée est soit étiqueté par "accept", ce qui montre que le symbole de la transition et ses contraintes d'horloges peuvent être vérifiés dans l'implantation, soit étiqueté par "refuse", ce qui montre le contraire. La définition d'un objectif de test est donc la suivante :

Définition 5.1.1 (Objectif de test temporisé) *Un Objectif de Test Temporisé Acceptant, OTT est un automate temporisé acyclique* $OTT =< \Sigma_{OTT}, S_{OTT}, s^0_{OTT}, C_{OTT}, E_{OTT} >$ *tel que les états finaux sont étiquetés par "Accept".*

Un Objectif de Test Temporisé Refusant, OTT' *est aussi un automate temporisé acyclique* $OTT' =< \Sigma_{OTT'}, S_{OTT'}, s^0_{OTT'}, C_{OTT'}, E_{OTT'} >$ *tel que chaque état est étiqueté soit par "Accept", soit par "Refuse".*

Définition 5.1.2 (Transition acceptante et refusante d'un objectif de test) *Soit OT un objectif acceptant. Chaque transition de OT est appelée une transition acceptante.*

Supposons maintenant que OT soit un objectif refusant. Les transitions de OT dont l'état d'arrivée est étiqueté par "Accept" sont appelées des transitions acceptantes. Les transitions de OT dont l'état d'arrivée est étiqueté par "Refuse" sont appelées des transitions refusantes.

Un exemple d'objectif de test temporisé Acceptant, composé d'un état final Accept, sur l'automate temporisé de la Figure 4.16 est illustré en Figure 5.27.

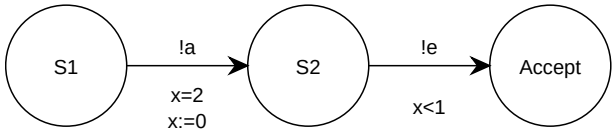


FIG. 5.27 – Objectif de test temporisé acceptant

Un objectif refusant est illustré en Figure 5.28.

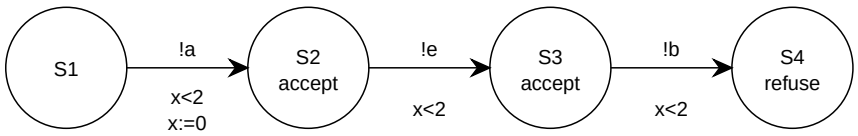


FIG. 5.28 – Objectif de test temporisé refusant

Les objectifs de test peuvent être créés manuellement, par le concepteur, ou peuvent être générés automatiquement. Au niveau du test non temporisé, des travaux sur cette génération automatique ont été décrits dans [CCKS95, CGPT95].

Une génération automatique d'objectifs de test temporisés semble plus difficile par le fait qu'une spécification temporisée contient non seulement des propriétés comportementales mais aussi des propriétés temporelles complexes. Toutefois, la qualité de test et les degrés définis en section 4.2, peuvent aider à cette génération automatique en les calculant sur des chemins de la spécification. En effet, les objectifs de test générés pourraient être ceux contenant des intervalles de temps bornés et accessibles de la spécification, ou inclus dans cette dernière (utilisation des Degrés de Forme Temporelle et d'Accessibilité Temporelle). Dans ce cas, les propriétés de l'objectif de test seraient vérifiables.

Puis, parmi les objectifs de test possibles, nous pourrions retenir ceux qui nécessitent le plus faible coût d'exécution (Utilisation du Degré de Contrôle).

5.1.2 Produit Synchronisé Temporisé

Comme nous l'avons vu précédemment, un objectif de test temporisé contient des actions et des contraintes d'horloges sur ces actions, qui appartiennent ou pas à la spécification. Or, les séquences de test qui seront générées, doivent contenir l'objectif de test mais doivent aussi refléter le comportement de la spécification pour être appliquées à l'implantation.

De ce fait, une sorte d'intersection entre la spécification et l'objectif de test est nécessaire pour générer ces séquences de test. Et nous appelons cette intersection un **Produit Synchronisé Temporisé**. Cette opération génère un graphe contenant l'objectif de test, et respectant les propriétés comportementales et temporelles de la spécification.

Le but d'un produit synchronisé est de faire correspondre chaque transition acceptante de l'objectif de test vers une transition de la spécification, pour fusionner leur comportement respectif. Dans le cas des graphes des régions, non seulement les symboles doivent être fusionnés mais aussi les régions d'horloges.

Plus précisément, pour deux graphes des régions $S =< \Sigma_S, S_S, s_S^0, E_S >$ et $OTT =< \Sigma_{OTT}, S_{OTT}, s_{OTT}^0, E_{OTT} >$, le produit synchronisé temporisé entre deux transitions $(s_1, R_S) \xrightarrow{A} (s_2, R_2) \in E_S$ et $(s'_1, R_{OTT}) \xrightarrow{A} (s'_2, R'_2) \in E_{OTT}$ étiquetées par le même symbole A, implique une synchronisation du comportement, modélisé par le symbole A, et de la région d'horloges pendant laquelle ce comportement peut s'exécuter. Ainsi, une première phase consiste à connaître les valuations d'horloges qui satisfont les contraintes d'horloges des deux transitions, puis celles qui satisfont soit la transition de la spécification, soit la transition de l'objectif de test. La connaissance de ces différentes valuations d'horloges est suffisante pour effectuer le test. En effet, ces valuations d'horloges permettent au testeur de savoir quand émettre ou recevoir les symboles des séquences de test.

Régions d'horloges synchronisées

Dans la partie qui suit, nous utilisons deux opérateurs qui sont, l'intersection de régions d'horloges, noté $\cap$ et la restriction de régions d'horloges, noté $/$. Ces opérateurs permettent de définir et d'engendrer les régions d'horloges synchronisées. Ceux-ci sont définis par :

Définition 5.1.3 (Intersection de régions d'horloges, $\cap$) *Soit R et R' deux régions d'horloges et Δ_R, Δ'_R leurs inéquations respectives.*
L'intersection de R et R' est obtenue par :
$$R \cap R' = \{v \mid v \text{ est solution de } \left\{ \begin{array}{l} \text{inéquations de } \Delta_R \cup \{x_i \in \mathbb{R}^+ \mid x_i \in \Delta'_R, x_i \notin \Delta_R\} \\ \text{inéquations de } \Delta'_R \cup \{x_i \in \mathbb{R}^+ \mid x_i \in \Delta_R, x_i \notin \Delta'_R\} \end{array} \right. \}$$

Définition 5.1.4 (Soustraction de régions d'horloges, $/$) *Soit R et R' deux régions d'horloges et Δ_R, Δ'_R leurs inéquations respectives.*
La soustraction de R' à R est obtenue par :
$R/R' = \{v \mid v \text{ est solution de } \Delta_R \text{ et } v \text{ ne satisfait pas } \Delta'_R\}$

Considérons donc $S =< \Sigma_S, S_S, s_S^0, E_S >$ et $OTT =< \Sigma_{OTT}, S_{OTT}, s_{OTT}^0, E_{OTT} >$. Le produit synchronisé temporisé entre deux transitions $(s_1, R_S) \xrightarrow{A} (s_2, R_2) \in E_S$ et $(s'_1, R_{OTT}) \xrightarrow{A} (s'_2, R'_2) \in E_{OTT}$ produit différentes régions d'horloges générées par une synchronisation des deux régions d'horloges R_S et R_{OTT}. Celles-ci sont :

- **Région PASS :**
  La région d'horloges R_{pass}, appelée **Région PASS**, rassemble les valuations d'horloges satisfaisant l'exécution de l'action de S et les contraintes de OTT, c'est à dire les valuations d'horloges qui appartiennent à la fois à R_S et à R_{OTT}, donc $R_{pass} = R_S \cap R_{OTT}$. Et si l'action du produit synchronisé est exécutée à une valuation d'horloges de R_{pass}, alors cela signifie que le test de l'action sur l'implantation est concluant et que l'objectif de test est satisfait.

- **Région INCONCLUANT :**
  La région d'horloges $R_{inconcluant}$, appelée **Région Inconcluant**, représente les valuations d'horloges qui satisfont l'exécution de l'action de S mais pas celle de l'action de l'objectif de test OTT. Par conséquent, si l'action synchronisée est exécutée à une valuation d'horloges de $R_{inconclusive}$, l'implantation respecte la spécification, mais pas l'objectif de test. $R_{inconclusive}$ contient les valuations d'horloges de R_S privées de celles de R_{OTT}, ce que nous notons par R_S/R_{OTT}.

- **Région FAIL :**
  Cette région d'horloges, notée R_{fail} représente l'ensemble des valuations d'horloges qui ne satisfont pas l'exécution de l'action de S. Si lors de la phase de test, une action est exécutée à toute valuation d'horloges de R_{fail}, alors l'implantation est erronée.

- **Région INCOHERENT :**
  La région d'horloges $R_{incohrent}$ est une partie de la région FAIL, et représente les valuations d'horloges satisfaisant la transition de l'objectif de test OTT, mais ne satisfaisant pas la transition de S. Nous qualifions cette région d'horloges comme étant incohérente car elle contribue à tester un comportement qui n'existe pas dans l'implantation. Un exemple simple de ce type de régions d'horloges est obtenu lorsque deux régions d'horloges n'ayant pas de valuations d'horloges communes sont synchronisées. Cette région d'horloge est obtenue par R_{OTT}/R_S.

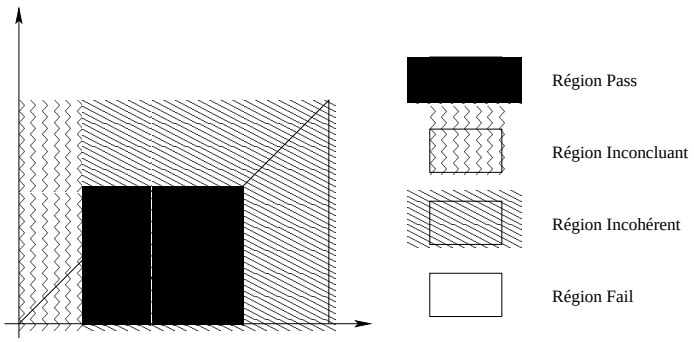


FIG. 5.29 – Les différentes régions d'horloges obtenues par le produit synchronisé temporisé

La Figure 5.29 illustre un exemple de régions d'horloges qui peuvent être générées par un produit synchronisé temporisé de deux régions d'horloges, avec un système à deux horloges.

Produit synchronisé entre une spécification et un objectif de test acceptant

Le produit synchronisé temporisé, que nous proposons, entre une spécification S et un objectif de test OTT modélisé par des graphes des régions, est inspiré de la définition du produit synchronisé de deux automates temporisés étendus (extension de l'automate

temporisé avec ajout de variables non temporelles) décrit dans [Lau99, PLC98]. Il a donc pour but, d'une part de synchroniser les transitions de la spécification et de l'objectif de test qui sont étiquetées par le même symbole, et d'autre part de générer les différentes régions d'horloges synchronisées qui ont été définies précédemment.

Sachant que l'objectif de test est acceptant, nous pouvons déduire l'hypothèse que chaque propriété comportementale et temporelle de OTT est incluse dans la spécification. Plus précisément, chaque symbole de OTT doit exister dans S et chaque région d'horloges de OTT doit être incluse dans une région d'horloges de S. Ainsi, pour obtenir le produit synchronisé, il suffit d'appliquer chaque propriété de OTT sur S, c'est-à-dire, de générer et d'étiqueter les régions d'horloges de synchronisation sur les transitions du graphe des régions S ayant les mêmes symboles que le graphe des régions OTT dans l'ordre.

La définition et la construction du graphe résultant du produit synchronisé temporisé entre un objectif de test acceptant et une spécification est donc la suivante :

Définition 5.1.5 (Produit synchronisé temporisé avec un objectif de test Acceptant) *Soit $S = (\Sigma_S, S_S, s^0_S, E_S)$ et $OTT = (\Sigma_{OTT}, S_{OTT}, s^0_{OTT}, E_{OTT})$ deux graphes des régions. Le Produit Synchronisé Temporisé entre S et OTT est un graphe $SP = (\Sigma_{SP}, S_{SP}, s^0_{SP}, E_{SP})$ défini par :*

- $\Sigma_{SP} \subseteq \Sigma_S$,
- $S_{SP} \subseteq S_S$,
- $s^0_{SP} = s^0_S$,
- E_{SP} *est l'ensemble des transitions* $s_i \xrightarrow{a, PASS(R), INCONCLUSIVE(R')} s_{i+1}$, *avec* $s_i \in S_{SP}$, $s_{i+1} \in S_{SP}$, $a \in \Sigma_{SP}$, R *une région PASS et* R' *une région Inconcluant.*

E_{SP} et S_{SP} sont construits grâce à l'algorithme suivant.

Pour simplifier la lecture, pour un état $e = (S, R)$ d'un graphe des régions, nous notons $Reg(e) = R$, la région d'horloges étiquetée dans e.

Algorithme

Entrée :
OTT(Objectif de test temporisé acceptant), S(Spécification), PTP (ensemble des chemins de OTT)
Sortie :
SP(Produit synchronisé)
Début :

POUR nbpathtp = 0 à Card(PTP)
Currenttp=PTP(nbpathtp)= $tp_1 \xrightarrow{b_1} tp_2 \dots tp_m \xrightarrow{b_m} tp_{m+1}$

PathS=RechercherComplet(Currenttp,S)
POUR countpaths = 0 à Card(PathS)
path=PathS(countpaths)=$p_1 \xrightarrow{a_1} p_2 \dots p_n \xrightarrow{a_n} p_{n+1}$
j=1
POUR i=1 à n

- TANTQUE Etiquette($tp_j \xrightarrow{b_j} tp_{j+1}$) = δ
- FAIRE $j = j + 1$
- Si Etiquette($p_i \xrightarrow{a_i} p_{i+1}$) = Etiquette($tp_j \xrightarrow{b_j} tp_{j+1}$) ET Etiquette($p_i \xrightarrow{a_i} p_{i+1}$) $\neq \delta$
- Alors
  - Ajouter S_i et S_{i+1} à S_{SP}
  - Ajouter $s_i \xrightarrow{a_i, PASS(Reg(p_i) \cap Reg(tp_j)), INCONCLUANT(Reg(p_i)/Reg(tp_j))} s_{i+1}$ à E_{SP}
  - SI $j < m$ ALORS $j = j + 1$
- SINON
  - Ajouter S_i et S_{i+1} à S_{SP}
  - Ajouter $s_i \xrightarrow{a_i, PASS(Reg(p_i)), INCONCLUSIVE(\emptyset)} s_{i+1}$ à E_{SP}

FINPOUR
Étiqueter $S_n \in S_{SP}$ par ACCEPT
FINPOUR
FINPOUR
FIN

Cet algorithme est composé des points suivants :

1. Chaque chemin de l'objectif de test est inclus dans un chemin partiel de la spécification, donc les chemins de la spécification, qui contiennent tous les symboles du chemin de l'objectif de test dans l'ordre et qui peuvent donc être synchronisés avec un chemin de l'objectif de test sont extraits ;
2. Chaque chemin p de la spécification est ensuite synchronisé avec un chemin de l'objectif de test tp. Si deux transitions ts de p et t de tp sont étiquetées par le même symbole, alors une transition de synchronisation est construite, avec ce symbole, la région PASS égale à l'intersection entre la région d'horloges de la spécification et celle de l'objectif de test. La région Inconcluant correspond à celle de la spécification privée de la région d'horloges de l'objectif de test. Dans le cas où la transition ts de p ne peut être synchronisée avec une transition de tp, la transition résultante est étiquetée avec le symbole de ts, une région PASS égale à la région d'horloges

étiquetée dans l'état de départ de ts, et une région Inconcluant vide. Cette dernière transition permet de faire avancer le comportement de l'implantation en vue de vérifier une propriété de l'objectif qui ne peut encore être exécutée.

3. Puis, le denier état du produit synchronisé est étiqueté par "Accept".

Le lecteur peut remarquer que le graphe du produit synchronisé ne peut s'appeler graphe des régions. Premièrement, les étiquettes de transition sont sensiblement différentes. D'autre part, le produit synchronisé peut générer des régions d'horloges synchronisées sans garantie de succession temporelle.

Produit synchronisé entre une spécification et un objectif de test refusant

La principale différence induite par l'utilisation d'objectifs de test refusant consiste à ce que des propriétés comportementales et temporelles, n'appartenant pas à la spécification S mais appartenant à l'objectif de test OTT, doivent figurer dans les séquences de test pour vérifier leurs refus par l'implantation.

Pour synchroniser une transition $(s'_1, R_{OTT}) \xrightarrow{A} (s'_2, R')$ et avec une transition de la spécification S, il faut donc en premier lieu considérer l'étiquette de l'état (s'_2, R') :

- si cette étiquette est "Accept", alors la synchronisation est obtenue comme précédemment, c'est-à-dire en cherchant une transition de la spécification ayant le même symbole et en générant une région PASS et INCONCLUANT.

- si cette étiquette est "Refuse", alors il faut ajouter des propriétés au produit synchronisé n'appartenant pas à la spécification. S'il s'agit d'un symbole, aucune transition de la spécification ne peut se synchroniser, donc la transition produit est, à peu de choses près, celle de l'objectif de test, étiquetée par le symbole A, par une région PASS égale à R_{OTT} et par une région Inconcluant Vide. Si la propriété de refus est une propriété temporelle, au moins une transition de la spécification peut se synchroniser avec celle de l'objectif, sur le symbole uniquement. La région PASS est toujours R_{OTT} mais la région Inconcluant est la région d'horloges R_S de la spécification, si $(s_1, R_S) \xrightarrow{A} (s_2, R)$ est une transition de S qui peut se synchroniser avec celle de l'objectif de test.

Ainsi, une nouvelle définition et surtout une nouvelle génération de produit synchronisé doivent être données pour considérer et ajouter au graphe produit les propriétés refusantes de l'objectif de test.

Définition 5.1.6 (Produit synchronisé temporisé avec un objectif de test Refusant) *Soit $S = (\Sigma_S, S_S, s^0_S, E_S)$ et $OTT = (\Sigma_{OTT}, S_{OTT}, s^0_{OTT}, E_{OTT})$ deux graphes des régions. Le Produit Synchronisé Temporisé entre S et OTT est un graphe $SP = (\Sigma_{SP}, S_{SP}, s^0_{SP}, E_{SP})$ défini par :*

- $\Sigma_{SP} \subseteq \Sigma_S \cup \Sigma_{OTT}$,
- S_{SP} *est un ensemble d'états étiquetés par "Accept" ou "refuse",*
- s^0_{SP} *est un état initial,*

– E_{SP} *est l'ensemble des transitions* $s_i \xrightarrow{a,PASS(R),INCONCLUSIVE(R')} s_{i+1}$, *avec* $s_i \in S_{SP}$, $s_{i+1} \in S_{SP}$, R *une région PASS et* R' *une région Inconcluant.*

E_{SP} *et* S_{SP} *sont construits grâce à l'algorithme suivant.*

Une fois de plus, pour simplifier la lecture, pour un état $e = (S, R)$ d'un graphe des régions, nous notons $Reg(e)$, la région d'horloges étiquetée dans e.

Algorithme

Entrée : OTT(Objectif de test refusant), S(Spécification), PTP (ensemble des chemins de OTT)
Sortie : SP(Produit synchronisé)
Début :
Ajouter S_0 à S_{SP}
POUR nbpathtp = 0 à Card(PTP)

Currenttp=PTP(nbpathtp)= $tp_1 \xrightarrow{b_1} tp_2 \dots tp_m \xrightarrow{b_m} tp_{m+1}$
PathS=RechercherPartie(Currenttp,S)
POUR countpaths = 0 à Card(PathS)

path=PathS(countpaths)=$p_1 \xrightarrow{a_1} p_2 \dots p_n \xrightarrow{a_n} p_{n+1}$
j=1, t=1
POUR i=1 à n

TANTQUE Etiquette($tp_j \xrightarrow{b_j} tp_{j+1}$) = δ
FAIRE $j = j + 1$
SI Etiquette($p_i \xrightarrow{a_i} p_{i+1}$) = Etiquette($tp_j \xrightarrow{b_j} tp_{j+1}$) ET Etiquette($p_i \xrightarrow{a_i} p_{i+1}$) $\neq \delta$

ALORS
- SI Etiquette(tp_{j+1}) = $refuse$
- ALORS Ajouter $S_{t+1} = (s_{t+1}, refuse)$ à S_{SP}
- Ajouter $S_t \xrightarrow{a_i, PASS(Reg(tp_j)), INCONCLUANT(Reg(p_i))} S_{t+1}$ à E_{SP}
- SINON Ajouter $S_{t+1} = (s_{t+1}, accept)$ à S_{SP}
- Ajouter $S_t \xrightarrow{a_i, PASS(Reg(p_i) \cap Reg(tp_j)), INCONCLUANT(Reg(tp_j) \cap Reg(p_i))} S_{t+1}$ à E_{SP}
- SI $j < m$ ALORS $j = j + 1$, $t = t + 1$

SINON
- SI Etiquette(tp_{j+1}) = $accept$
- ALORS
  - Ajouter $S_{t+1} = (s_{t+1}, accept)$ à S_{SP}
  - Ajouter $S_t \xrightarrow{a_i, PASS(Reg(p_i)), INCONCLUSIVE(\emptyset)} S_{t+1}$ à E_{SP}
  - SI $j < m$ ALORS $j = j + 1$, $t = t + 1$
- SINON
  - $k = i + 1, syn = faux$
  - TANTQUE $k < n$ FAIRE
  - SI Etiquette($tp_j \xrightarrow{b_j} tp_{j+1}$) = Etiquette($p_k \xrightarrow{a_k} p_{k+1}$)
  - ALORS $syn = vrai$
  - $FINTANTQUE$
  - SI $syn = faux$
  - Ajouter $S_{t+1} = (s_{t+1}, refuse)$ à S_{SP}
  - Ajouter $S_t \xrightarrow{b_j, PASS(Reg(tp_j)), INCONCLUSIVE(\emptyset)} S_{t+1}$ à E_{SP}
  - $i = i - 1$
  - SI $syn = vrai$
  - Ajouter $S_{t+1} = (s_{t+1}, accept)$ à S_{SP}
  - Ajouter $S_t \xrightarrow{a_i, PASS(Reg(p_i)), INCONCLUSIVE(\emptyset)} S_{t+1}$ à E_{SP}
  - SI $j < m$ ALORS $j = j + 1$, $t = t + 1$

TANTQUE $j \leq m$
- Ajouter $S_{t+1} = (s_{t+1}, refuse)$ à S_{SP}
- Ajouter $S_t \xrightarrow{b_j, PASS(Reg(tp_j)), INCONCLUSIVE(\emptyset)} S_{t+1}$ à E_{SP}
- $j = j + 1$, $t = t + 1$

FINPOUR
FINPOUR
FIN

Cet algorithme est composé des points suivants :

1. Pour chaque chemin de l'objectif de test, nous recherchons les chemins de la spécification, qui contiennent dans l'ordre un ou plusieurs symboles du chemin de l'objectif de test.
2. Chaque chemin p de la spécification est ensuite synchronisé avec un chemin de l'objectif de test tp.
   - Si deux transitions ts de p et t de tp sont étiquetées par le même symbole (ALORS de la 1^{re} condition), une transition de synchronisation est construite avec ce symbole. Si la transition se termine par un état de refus, la région PASS est égale à la région d'horloges de l'objectif de test. Dans le cas contraire, la région PASS est égale à l'intersection entre la région d'horloges de la spécification et celle de l'objectif de test et la région Inconcluant correspond à celle de la spécification privée de la région d'horloges de l'objectif de test.
   - Dans le cas où la transition ts de p ne peut être synchronisée avec une transition de t de tp (SINON de la 1^{re} condition), l'algorithme distingue si la transition de l'objectif de test se termine par un état accept ou refuse. S'il s'agit d'accept (1^{re} accolade du SINON), alors une synchronisation avec t sera possible avec une autre transition du chemin de la spécification. Donc, la transition produit est étiquetée par le symbole de ts et une région PASS égale à la région d'horloges étiquetée dans l'état de départ de ts. Cette dernière transition permet de faire avancer le comportement de l'implantation en vue de vérifier une propriété de l'objectif qui ne peut encore être exécutée.
     S'il s'agit de refuse (2^{me} accolade du SINON), nous recherchons si une synchronisation sur un symbole est possible avec une autre transition de p. Si une synchronisation est possible, nous nous retrouvons dans le cas précédent. Si une synchronisation est impossible, alors la transition qui est ajoutée est à peu de choses près, celle de l'objectif de test. Ceci permettra de vérifier que ces propriétés de l'objectif de test sont refusées par l'implantation.
3. Si des transitions de l'objectif de test n'ont pas été synchronisées avec des transitions du chemin de la spécification, alors celles-ci sont ajoutées.

5.1.3 Génération des Séquences de Test

Nous proposons que l'objectif de test et les chemins de la spécification, qui peuvent être synchronisés avec celui-ci, soient transformés en graphe des régions, puis minimisés par l'algorithme décrit dans [ACH+92]. Ces deux étapes ont pour but de pouvoir utiliser des contraintes permettant la comparaison d' horloges entre elles et d'obtenir les intervalles de temps minimaux, modélisés par des régions d'horloges, qui satisfont l'exécution de chaque action. La traduction en graphe des régions minimal permet aussi de ne pas considérer les valeurs de temps qui ne seront jamais atteintes par les horloges.

Ainsi, il est maintenant possible de décrire les étapes de la génération des séquences de test. Nous supposons, que la spécification S et l'objectif de test OTT sont décrits par le même formalisme, à savoir le modèle du graphe des régions.

Auparavant, les hypothèses suivantes sur la spécification et l'implantation, doivent être satisfaites pour appliquer la méthodologie.

Hypothèses de Test

- La spécification et l'implantation doivent posséder le même nombre d'horloges et le même alphabet.
- Depuis n'importe quel état de la spécification, nous ne pouvons obtenir une transition sortante, étiquetée par un symbole d'entrée et une transition sortante, étiquetée par un symbole de sortie, dont les contraintes d'horloges impliquent qu'elles soient satisfaites simultanément.
- L'implantation doit contenir un état initial unique avec ses horloges initialisées à 0. Depuis cet état, il n'existe pas de transitions sortantes étiquetées par un symbole de sortie. En effet, les séquences de test sont appliquées à l'implantation après l'avoir amenée à son état initial.

Étapes de la génération des cas de test

1. Les séquences de transitions partant de l'état initial de S contenant les symboles étiquetés sur les transitions acceptantes de OTT, dans le même ordre, sont premièrement extraites et nommées $TS_1(S), ..., TS_n(S)$.
2. Les séquences $TS_1(S), ..., TS_n(S)$ et OTT sont transformées en graphes des régions minimaux, grâce à l'algorithme décrit dans [ACH+92]. Les graphes obtenus sont nommés respectivement $RGMin(RG(TS_1(S))), ..., RGMin(RG(TS_n(S)))$ et $RGMin(RG(OTT))$. Chaque région d'horloges infinie est de plus bornée d'une façon homogène par rapport aux autres régions d'horloges (voir section 4.6)
3. Puis, chaque graphe des régions $RGMin(RG(TS_1(S))), ...,$ $RGMin(RG(TS_n(S)))$ est synchronisé avec l'objectif de test $RGMin(RG(OTT))$. La définition du produit synchronisé temporisé de deux graphes des régions est donnée en section 5.1.2. Selon le type de l'objectif de test, acceptant ou refusant, un produit synchronisé temporisé est proposé.
4. A partir des graphes obtenus, le chemins partant de l'état initial de chaque graphe, sont les séquences de test qui seront appliquées à l'implantation.

Exprimons la génération de séquences de test grâce aux exemples d'objectifs illustrés en Figures 5.36 et 5.39.

Prenons comme premier exemple, l'objectif de test acceptant de la Figure 5.27. Celui-ci est transformé en graphe des régions illustré en Figure 5.30, qui est composé des régions d'horloges présentées en Figure 5.31.

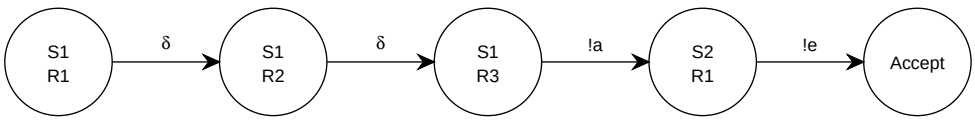


FIG. 5.30 – Graphe des Régions obtenu depuis l'objectif de test acceptant

De même, nous obtenons depuis la spécification illustrée en Figure 4.16, plusieurs chemins dont celui de la Figure 5.32. Ce dernier génère les régions d'horloges de la Figure 5.33, et le graphe des régions de la Figure 5.34.

FIG. 5.31 – Régions d'horloges obtenues depuis l'objectif de test acceptant

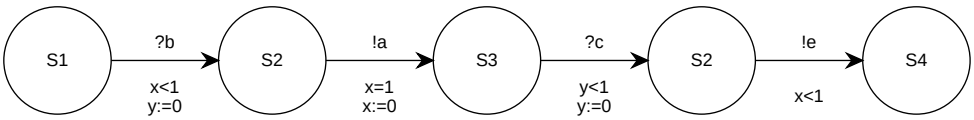


FIG. 5.32 – Un chemin de la spécification

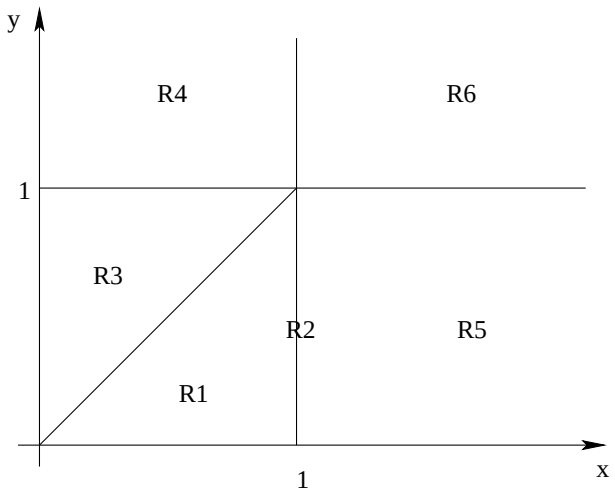


FIG. 5.33 – Régions d'horloges obtenues depuis le chemin de la spécification

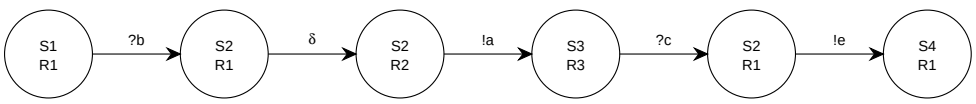


FIG. 5.34 – Graphe des régions obtenu depuis le chemin de la spécification

Finalement, le produit synchronisé entre les deux graphes des régions illustrés en Figures 5.34 et 5.30 donne les régions d'horloges de la Figure 5.35 et le cas de test de la Figure 5.36.

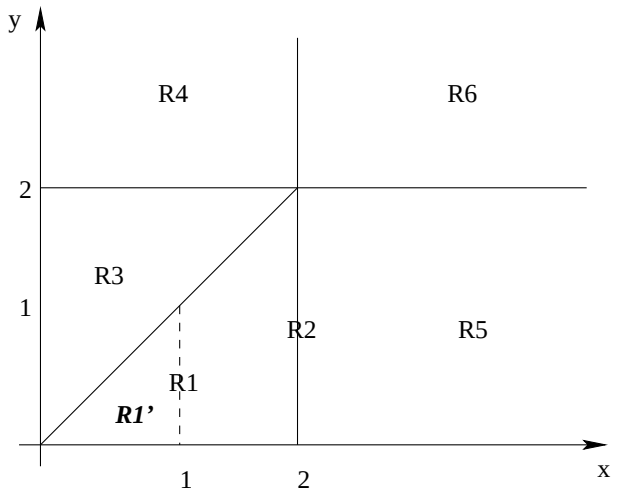


FIG. 5.35 – Régions d'horloges obtenues depuis le produit synchronisé

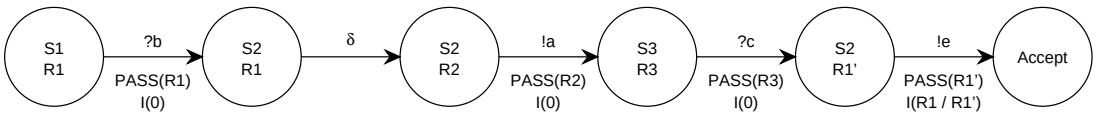


FIG. 5.36 – Cas de test obtenu depuis le produit synchronisé

Soit maintenant l'exemple d'objectif de test refusant de la Figure 5.28. Celui-ci est transformé en graphe des régions illustré en Figure 5.37, qui est composé des régions d'horloges illustrées en Figure 5.38

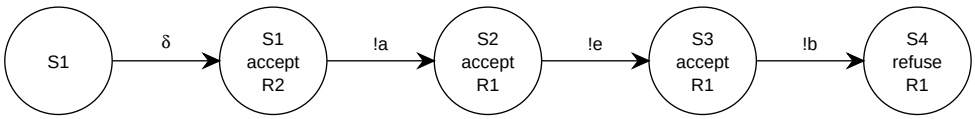


FIG. 5.37 – Graphe des Régions obtenu depuis l'objectif de test refusant

Depuis la spécification illustrée en Figure 4.16, nous obtenons plusieurs chemins dont celui de la Figure 5.32 qui donne les régions d'horloges de la Figure 5.33, et est transformé en graphe des régions de la Figure 5.34.

Le produit synchronisé entre les deux graphes des régions illustrés en Figures 5.34 et 5.37 donne les régions d'horloges de la Figure 5.33 et le cas de test de la Figure 5.39.

FIG. 5.38 – Régions d'horloges obtenues depuis l'objectif de test refusant

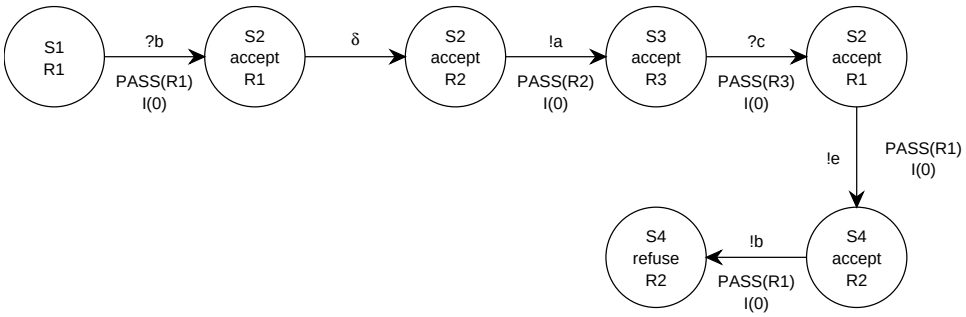


FIG. 5.39 – Cas de test obtenu depuis le produit synchronisé

5.1.4 Exécution des Séquences de Test

Lors de la phase de test, le testeur doit appliquer les séquences de test sur l'implantation, c'est-à-dire qu'il doit émettre ou recevoir des symboles dans des régions PASS. Or, les régions d'horloges sont des représentations denses du temps et contiennent donc un nombre infini de valuations d'horloges qui ne peuvent pas toutes être utilisées lors du test.

De plus, il est très difficile, voir impossible d'atteindre une valuation d'horloges précise d'une région d'horloges. En effet, une transition étiquetée par un symbole de sortie, modélise une action de l'implantation qui n'est pas contrôlable, et qui peut donc se produire à tout moment satisfaisant son exécution. Ainsi avec deux exécutions différentes, nous pouvons atteindre, sans toujours le contrôler, deux valuations différentes d'une région d'horloges R, qui sont inaccessibles l'une de l'autre. La Figure 5.40 illustre ce cas. La première exécution permet d'atteindre la première valuation d'horloges v_{init1} et la deuxième exécution v_{init2}.

Pour résoudre ces problèmes, une première hypothèse, appelée **hypothèse d'uniformité** a été proposée dans [BGM91]. Celle-ci affirme que "tester le logiciel avec une seule valeur dans un domaine uniforme, pour un cas donné, est suffisant". Les méthodologies de test proposent, d'une manière unanime, pour les symboles de sortie, d'attendre et d'observer ces symboles dans l'intervalle de temps satisfaisant leurs observations. Par contre, pour l'émission des symboles d'entrée vers l'implantation, deux méthodes sont proposées. Si v_{init} v_{final} sont les première et dernière valuations d'horloges satisfaisant l'émission d'un symbole d'entrée $?A$, alors les auteurs de [LC97, PLC98, ENDE97] proposent d'émettre A vers l'implantation au voisinage de $\frac{v_{final}-v_{init}}{2}$, et ceux de [PF99b, FPS00] proposent de l'émettre au voisinage de v_{init} et v_{final}. Nous proposons de garder le principe de la deuxième solution. Celle-ci donne un coût plus important mais garantit une meilleure couverture des régions d'horloges.

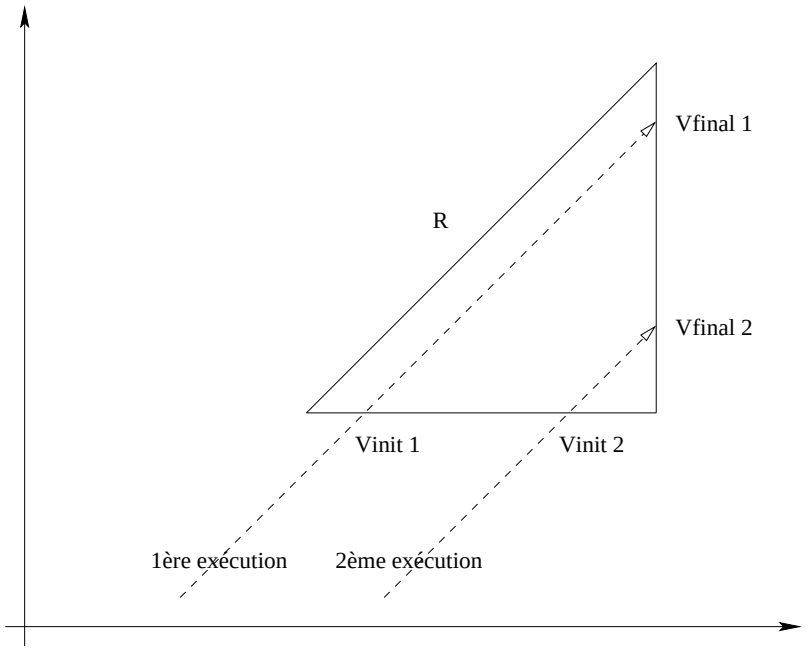


FIG. 5.40 – 2 exécutions permettant d'atteindre deux valuations d'horloges

Par contre, aucune de ces méthodes d'exécution ne considère qu'une région d'horloges d'une séquence de test ne soit accessible lors de son exécution. Or, il est primordial que le testeur puisse vérifier qu'une région d'horloges soit accessible ou non lors de l'exécution d'un test pour éviter des attentes inutiles. L'architecture de test que nous choisissons pour exécuter les tests sur l'implantation, est celle décrite dans [PF99b] (voir section 3.3). Celle-ci permet de tester des systèmes temporisés de la manière la plus réaliste qui soit, car elle considère que le testeur n'a aucune connaissance interne de l'implantation et qu'il simule la partie temporelle pour tester. Ainsi, en prenant en compte ces critères, nous proposons une autre méthode d'exécution du test :

1. Soit $s \xrightarrow{?a,PASS(R),INCONCLUANT(R')} s'$ une transition d'une séquence de test, étiquetée par un symbole d'entrée $?a$. Le symbole a peut être émis par le testeur vers l'implantation pour toute valuation d'horloges de R ou R'. Nous proposons que le testeur émette a "le plus tôt possible" et "le plus tard possible", c'est-à-dire au voisinage de la première valuation d'horloges $v_{init} \in R \cup R'$ atteinte par les horloges lors d'une première exécution, et au voisinage de la dernière valuation d'horloges $v_{final} \in R \cup R'$ atteinte par les horloges lors d'une seconde exécution. Pour que les horloges atteignent v_{final}, un laps de temps peut être nécessaire. Ce temps d'attente est calculé à la volée par le testeur. (voir section 5.1.4). Ce calcul permet au testeur de savoir s'il peut atteindre v_{final}.
2. Soit $s \xrightarrow{!a,PASS(R),INCONCLUANT(R')} s'$ une transition d'une séquence de test, étiquetée par un symbole d'entrée $!a$. Le testeur attend la réception du symbole $!a$ de l'implantation sous test. Si cette réception est obtenue à une valeur de la région PASS, alors, l'objectif de test est satisfait. Si cette réception est obtenue à une va-

leur de la région INCONCLUANT, seule la specification est respectée, l'objectif n'a pu être testé car les horloges n'ont pu atteindre toutes les régions PASS.

3. Une transition $s \xrightarrow{\delta} s'$ modélise un laps de temps permettant d'atteindre une région d'horloges. Par conséquent, aucun test n'est effectué. Cependant, le testeur calcule le temps d'attente à la volée et attend. Le fait de calculer ce laps de temps permet de savoir si la prochaine région d'horloges est accessible et de ne pas attendre inutilement (voir section 5.1.4 pour le calcul).

En résumé, nous proposons donc, pour chaque transition étiquetée par un symbole d'entrée, que le testeur applique le symbole deux fois sur l'implantation à des moments différents et, pour chaque transition étiquetée par un symbole de sortie, que le testeur attende le symbole dans la région d'horloges satisfaisant le passage de la transition.

Par conséquent, un cas de test, contenant n transitions étiquetées par des symboles d'entrée sera exécuté 2^n fois. Pour représenter tous ces cas, nous développons l'exécution des cas de test sous forme d'un arbre, appelé **Arbre d'Exécution**.

Arbre d'Exécution d'un cas de test

Considérons une séquence de test *stest* de l'ensemble des séquences de test SEQ obtenues par le produit synchronisé temporisé entre une spécification et un objectif de test. Avant que *Seq* soit exécutée par le testeur, celle-ci est transformée en Arbre d'Exécution, rassemblant toutes les exécutions de *Seq* qui seront faites par le testeur. Cet arbre est composé de noeuds et d'arcs étiquetés par :

- **send(A)** modélisant l'envoi d'un symbole d'entrée $?A$ par le testeur vers l'implantation sous test,

- **recv(A)** modélisant la réception d'un symbole de sortie $!A$ par le testeur depuis l'implantation,

- *wait* modélisant le laps de temps nécessaire pour que les horloges atteignent la valuation d'horloges v_{final} à partir de v_{init}, donc $wait = v_{final} - v_{init}$. Ce laps de temps doit être calculé à la volée car il dépend de l'exécution (voir Figure 5.40),

- δ modélisant le laps de temps nécessaire pour atteindre une région d'horloges depuis une valuation d'horloges v_{init}. De même que précédemment, ce laps de temps doit être calculé à la volée car le temps d'attente pour passer d'une région d'horloges à une autre varie suivant l'exécution.

L'algorithme suivant transforme une séquence de test $stest \in SEQ$ en un arbre d'exécution T_{stest}. Il consiste à ajouter, pour chaque transition de *stest* étiquetée par un symbole d'entrée, deux branches modélisant les deux émissions de symbole et, pour chaque transition de *stest* étiquetée par un symbole de sortie, une branche modélisant une réception de symbole. Chaque branche est commencée par δ, modélisant un laps de temps, pouvant être nul, qui permet d'atteindre les régions PASS et INCONCLUANT si elles ne sont pas directement accessibles temporellement. Le calcul à la volée de ce laps de temps peut

donner une valeur de temps inaccessible, ce qui impose l'arrêt du test. Ceci est particulièrement utile lorsque nous testons des contraintes temporelles qui n'appartiennent pas à la spécification.

Algorithme

Construction d'Arbre d'exécution
Entrée : Séquence de test Seq
Sortie : Arbre d'exécution de Seq
DEBUT :
Const-Arbre(Seq,$s \xrightarrow{A,PASS(R),INCONCLUSIVE(R')} s'$)

SI A=δ %Laps de temps δ
Alors
- Ajouter une branche $\xrightarrow{\delta}$
- Cons-Arbre(SEQ, prochaine transition de SEQ)

Si A est un Symbole d'entrée
Si $\exists v_1 \in R, v_2 \in R$, tel que $v_1 \neq v_2$
Alors
- Ajouter une branche $\xrightarrow{\delta}\xrightarrow{wait}\xrightarrow[Rt]{send(I)}$, avec
- Rt l'ensemble des horloges réinitialisées
- Cons-Arbre(Seq, prochaine transition de SEQ)
- Ajouter une seconde branche $\xrightarrow{\delta}\xrightarrow[Rt]{send(I)}$, avec
- Rt l'ensemble des horloges réinitialisées
- Cons-Arbre(Seq, prochaine transition de Seq)

Sinon
- Ajouter une branche $\xrightarrow{\delta}\xrightarrow[Rt]{send(I)}$, avec
- Rt l'ensemble des horloges réinitialisées
- Cons-Arbre(Seq, prochaine transition de SEQ)

Si A est un symbole de sortie
Alors
- Ajouter une branche $\xrightarrow[Rt]{recv(O),PASS(R),INCONCLUSIVE(R')}$, avec
- Rt l'ensemble des horloges réinitialisées
- Cons-Arbre(Seq, prochaine transition de Seq)

FIN

Finalement, chaque branche de cet arbre peut être donnée au testeur pour être exécutée. En effet, le testeur connaît les symboles attendus et à émettre, les instants durant lesquels ils doivent être émis et reçus, et les durées pendant lesquelles il devra attendre. Nous avons choisis de créer deux branches pour chaque emission de symbole vers l'implantation, représentant le fait de les émettre à deux valuations d'horloges différentes. Or, il est tout à fait possible que le concepteur souhaite que les symboles d'entrée soient émis à plusieurs valuations. Dans ce cas, si N est le nombre de valuations d'horloges différentes auxquelles seront émis les symboles d'entrée, il suffit de modifier l'algorithme pour qu'il crée N branches telles que le temps d'attente $wait$ soit égal à $\frac{i}{N-1}(v_{final} - v_{init}), (0 \leq i < N)$, si le concepteur souhaite des émissions de symboles régulières.

Les laps de temps *wait* et δ sont obtenus à la volée par les calculs suivants :

Laps de temps δ nécessaire pour atteindre une région d'horloges

Soit la séquence $s_1 \xrightarrow{A,PASS(R_1),INCONCLUANT(R'_1)} s_2 \xrightarrow{\delta} s_3 \xrightarrow{B,PASS(R_2),INCONCLUANT(R'_2)} s_4$. Lors d'une exécution d'un test, le testeur calcule le laps de temps d'attente pour atteindre R_2 ou R'_2 à partir d'une valuation d'horloges $v_{init} = (v_1, ..., v_n)$, atteinte par les horloges après avoir exécuté la première action. Plus précisément, le testeur calcule à la volée la plus petite valeur $d \in \mathbb{R}^+$ telle que $v_{init} + d \in R_2$.

Comme les horloges croissent strictement et de la même manière, elles prennent les valeurs de l'équation :

$$f(x_1, ..., x_n) = \begin{cases} x_1 - x_2 = v_1 - v_2 \\ ... \\ x_{n-1} - x_n = v_{n-1} - v_n \end{cases}$$

Par conséquent, la première valuation d'horloges v_{final}, atteinte par les horloges dans R_2, est unique et est obtenue en résolvant le système d'inéquations

$$\Delta = \begin{cases} \text{Inéquations de } R_2 \cup R'2 \\ x_1 - x_2 = v_1 - v_2 \\ ... \\ x_{n-1} - x_n = v_{n-1} - v_n \end{cases}$$

Finalement, le laps de temps δ est égal à la différence entre v_{init} et la plus petite solution v_{final} de Δ, moins la durée ϵ, nécessaire pour résoudre Δ. L'exemple donné en Figure 4.21 illustre ce cas.

Laps de temps *wait* nécessaire pour atteindre une valuation d'horloges dans une région d'horloges

Pour les transitions $\xrightarrow{?A,PASS(R),INCONCLUANT(R')}$, un laps de temps *wait* est nécessaire pour envoyer le symbole A le plus tard possible, c'est-à-dire à une valuation $v_{final} \in \mathbb{R}$ depuis la première valuation d'horloges $v_{init} = (v_1, ..., v_n) \in \mathbb{R}$ atteinte par les horloges lors de l'exécution. La Figure 4.22 illustre cet autre cas.

Sachant que le tuple des horloges du système est égal à $v_{init} = (v1, ..., v_n)$ et que les horloges croissent d'une manière identique, elles prennent comme valeurs celles de l'équation suivante :

$$f(x_1, ...x_n) = \begin{cases} x_1 - x_2 = v_1 - v_2 \\ ... \\ x_{n-1} - x_n = v_{n-1} - v_n \end{cases}$$

Et v_{final} est la plus grande solution du système d'inéquations

$$\Delta = \begin{cases} \text{Inéquations de } R \\ x_1 - x_2 = v_1 - v_2 \\ ... \\ x_{n-1} - x_n = v_{n-1} - v_n \end{cases}$$

Finalement, le laps de temps *wait* est obtenu par la différence $v_{final} - v_{init}$ moins la durée ϵ, nécessaire pour résoudre Δ.

L'exemple de la Figure 5.41 illustre l'arbre d'exécution obtenu depuis la séquence de test représentée en Figure 5.36.

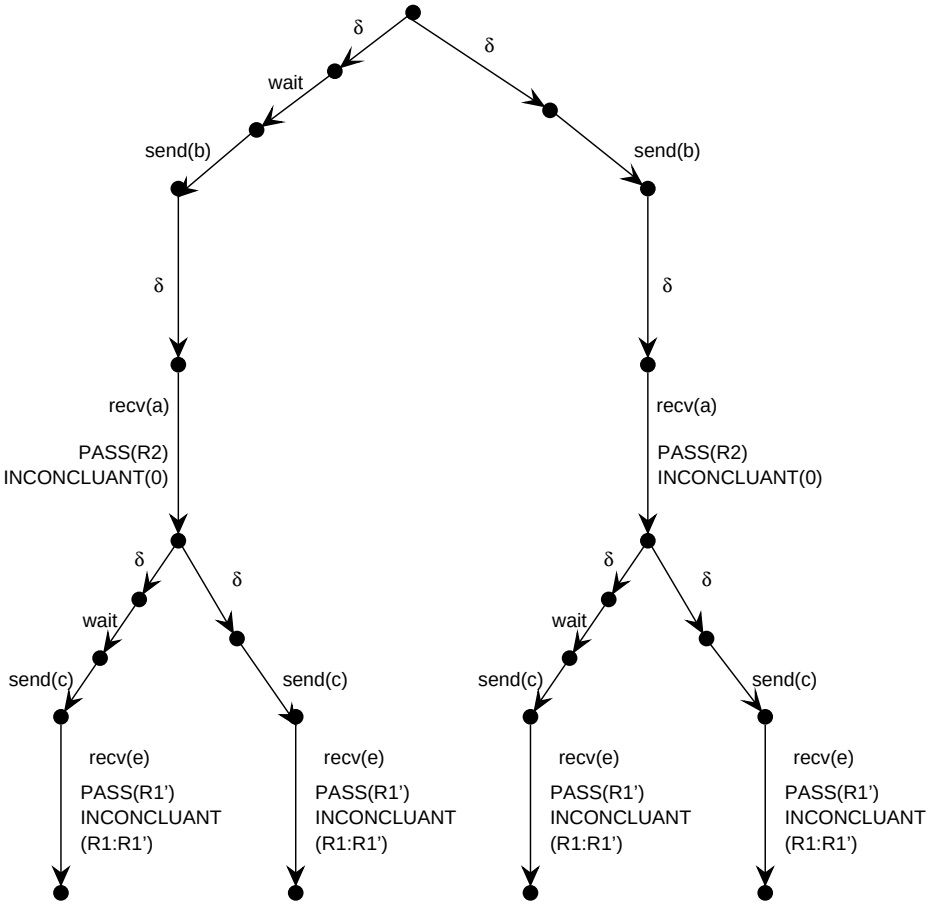


FIG. 5.41 – Arbre d'exécution

5.1.5 Verdict du test sur la satisfaction de l'objectif de test temporisé

Après avoir exécuté les séquences de test sur l'implantation, un verdict sur la satisfaction de l'objectif de test peut être fourni. Étant donné que l'implantation est vue comme une boîte noire, celui-ci dépend des observations faites sur l'implantation et sur le type de l'objectif de test temporisé, acceptant ou refusant.

Ainsi, la transition $t = s \xrightarrow{A,PASS(R),INCONCLUANT(R')} s'$ appartenant à une séquence de test, engendre plusieurs possibilités de verdict, notées $Obs(t)$, par le testeur et nous pouvons conclure par $Obs(t)=$:

- PASS ssi le symbole A est un symbole de sortie et A est reçu à une valuation d'horloges de R,
- PASS ssi le symbole A est un symbole d'entrée et A est émis à une valuation de R,
- INCONCLUANT ssi le symbole A est un symbole de sortie et A est reçu à une valuation d'horloges de R',
- INCONCLUANT ssi le symbole A est un symbole d'entrée et A est émis à une valuation d'horloges de R',
- FAIL ssi le symbole A est reçu à une valuation d'horloges $v \notin R \cup R'$,
- FAIL ssi le symbole B est reçu et $A \neq B$.

Ces verdicts, déduits par l'observation des réactions de l'implantation, sont locaux et ne prennent pas en compte le fait que le test doit être satisfait (objectif de test acceptant) ou doit être refusé (objectif de test refusant). Ainsi, en tenant compte du type de l'objectif et de ces observations, il est possible de définir un verdict sur la satisfaction de l'objectif de test :

Définition 5.1.7 (Verdict) *Soit $\mathcal{A} = (\Sigma_{\mathcal{A}}, S_{\mathcal{A}}, s^0_{\mathcal{A}}, E_{\mathcal{A}})$, $R_{\mathcal{A}}$ un automate temporisé, OTT, un objectif de test temporisé, I une implantation sous test, et SEQ l'ensemble des séquences test. Le verdict de l'application d'une séquence $stest \in SEQ$ sur I, noté $V(I, stest)$, est donné par :*
Si OTT est un objectif de test Acceptant, alors

$$V(I, stest) = \begin{cases} PASS \text{ ssi } \forall t \in stest, \\ Obs(t) = PASS \\ FAIL \text{ ssi } \exists t \in stest, \\ Obs(t) = FAIL \\ \\ INCONCLUANT \text{ autrement} \end{cases}$$

Si OTT est un objectif de test Refusant, alors

$$V(I, stest) = \begin{cases} PASS \text{ ssi } \forall t = s \xrightarrow{\{?,!\}A, PASS(R), INCONCLUANT(R')} s' \in stest, \\ \text{avec } s' \text{ étiquetée par accept, } Obs(t) = PASS \\ PASS \text{ ssi } \forall t = s \xrightarrow{\{?,!\}A, PASS(R), INCONCLUANT(R')} s' \in stest, \\ \text{avec } s' \text{ étiquetée par refuse, } Obs(t) = FAIL \\ FAIL \text{ ssi } \exists t = s \xrightarrow{\{?,!\}A, PASS(R), INCONCLUANT(R')} s' \in stest, \\ \text{avec } s' \text{ étiquetée par accept, } Obs(t) = FAIL \\ FAIL \text{ ssi } \exists t = s \xrightarrow{\{?,!\}A, PASS(R), INCONCLUANT(R')} s' \in stest, \\ \text{avec } s' \text{ étiquetée par refuse, } Obs(t) = PASS \\ INCONCLUANT \text{ autrement} \end{cases}$$

Finalement l'attribution du verdict sur l'ensemble de la phase de test, noté $V(I, OTT)$ *est donné par :*

$$V(I, OTT) = \begin{cases} PASS, \text{ ssi } \forall stest \in SEQ, V(I, stest) = PASS, \\ FAIL \text{ ssi } \exists stest \in SEQ, V(I, stest) = FAIL, \\ \grave{I}NCONCLUANT \text{ autrement} \end{cases}$$

5.1.6 Complexité de la Génération des cas de test

Les points suivants analysent la complexité de chaque étape de la méthodologie. Celle-ci est difficile à calculer avec précision car elle dépend du nombre d'horloges, du nombre d'états et du nombre de transitions de la spécification. Ainsi, nous considérerons le pire des cas, c'est-à-dire que l'objectif de test a autant de transitions et d'états que la spécification, et que le nombre de chemins extraits de la spécification est égal à son nombre de transitions.

- **Extraction des chemins de la spécification :**
  L'extraction demande à ce que depuis chaque état e de la spécification S, soient parcourues toutes les transitions accessibles depuis e pour trouver, dans le même ordre, les symboles des transitions acceptantes de l'objectif de test. Si K est le nombre de transitions de la spécification, dans le pire des cas, il faut parcourir K transitions depuis chacun de ses états. Ainsi, si M est le nombre d'états de la spécification, la complexité à extraire les chemins de la spécification contenant les symboles de l'objectif de test est proportionnelle à $M * K$.

- **Produit synchronisé temporisé :** La complexité du produit synchronisé d'une spécification et d'un objectif de test dépend du nombre de transitions et du nombre d'horloges de la spécification. Supposons qu'une séquence Ch obtenue à partir de la spécification, ait M états, K transitions et C horloges. Le nombre de transitions du graphe des régions est au pire égal au nombre de transitions du graphe des régions non minimisé de Ch. Donc ce nombre est proportionnel à $K * 2^{\delta(Ch)}$ [AD94]. Ainsi, la complexité de la génération du produit synchronisé avec un objectif acceptant est proportionnelle à $K * 2^{\delta(Ch)} * C$. Dans le pire des cas, nous obtenons K séquences depuis la spécification et K séquences depuis l'objectif de test. Au pire, nous obtenons $2 * K$ graphes des régions, donc la complexité de la génération de tous les produits synchronisés est proportionnelle à $2 * K^2 * 2^{\delta(Ch)} * C$. La complexité de la génération du produit synchronisé avec un objectif refusant est proportionnelle à

$2 * K * 2^{\delta(Ch)} * C$. Dans ce cas, la complexité de la génération de tous les produits synchronisés est proportionnelle à $4 * K^2 * 2^{\delta(Ch)} * C$.

- **Génération des Arbres d'exécutions**
  La complexité à générer les arbres d'exécutions dépend de la longueur et du nombre de cas de test. Les graphes des régions ont au plus $K * 2^{\delta(Ch)}$ transitions ([AD94]), ainsi la longueur d'un cas de test, après la synchronisation, est au pire égale à la somme des transitions d'un chemin Ch de la spécification et des transitions de OTT, donc au pire égale à $K*2^{\delta(Ch)}+K*2^{\delta(OTT)}$. Si le cas de test est composé de symboles d'entrée et d'un unique symbole de sortie, pour chaque symbole d'entrée, deux branches sont ajoutées à l'arbre, par conséquent, la complexité à générer les arbres d'exécutions est proportionnelle à $2^{2*K*2^{\delta(Ch)}} * N$, avec N le nombre de séquences de test.

5.1.7 Une implantation erronée

Illustrons la détection de fautes potentielles sur une implantation erronée, et considérons par conséquent l'implantation de la Figure 5.42. Celle-ci contient deux erreurs : la première est une faute comportementale et concerne la transition $S_2 \xrightarrow{!B} S_3$, étiquetée par " !B" et qui devrait être étiquetée par " !A". La seconde erreur est une violation des contraintes d'horloges de la transition $S_1 \xrightarrow{?B} S_2$. En effet, la contrainte devrait être $x < 2$. Considérons la séquence de test illustrée en Figure 5.36 et son arbre d'exécution illustré en Figure 5.41. Cette séquence peut-elle détecter les erreurs précédentes ?

Lors de la phase de test, le testeur émet les symboles d'entrée et reçoit les symboles de sortie de l'arbre d'exécutions .

Selon la première branche de l'arbre d'exécution, le testeur émet "B", à une valuation d'horloges égale à $(2-\epsilon, 2-\epsilon)$. L'implantation n'accepte pas le symbole et reste donc dans l'état initial, car ses contraintes d'horloges ne sont pas respectées. Mais, pour l'instant le testeur n'a pas détecté l'erreur. Le testeur attendant ensuite le symbole "A" dans la région R_2. L'implantation n'offre aucune réaction, donc aucun symbole est n'émis. Le testeur n'ayant pas reçu le symbole "A" conclut que l'implantation est donc erronée.

Avec la troisième branche de l'arbre d'exécution, le testeur émet "B" à une valuation d'horloges égale à $(0, 0)$. Puis, il attend une réaction de la part de l'implantation sous test, qui devrait être la réception du symbole "A" pour une valuation de la région PASS ou INCONCLUANT. L'unique réaction qu'il peut observer est la réception de "B" qui est un symbole inattendu. Le testeur en déduit que l'implantation est erronée.

Par conséquent, les deux fautes ont été détectées par l'exécution de la séquence de test à des valeurs de temps différentes. Cependant, il est impossible de détecter où se situe l'erreur. Un diagnostic est donc nécessaire.

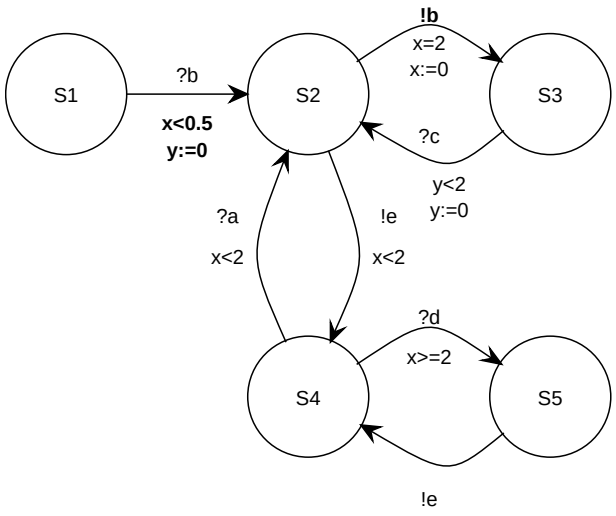


FIG. 5.42 – Une implantation erronée

5.2 Méthodologie de test basée sur la Caractérisation Temporisée d'États

Actuellement, plusieurs méthodologies de test de systèmes temporisés proposent de générer ces séquences de test sur un modèle inspiré du graphe des régions, grâce à la caractérisation des états du système [ENDKE98, SVD01]. Cette caractérisation est une signature unique qui distingue chaque état de la spécification ([Gil62]) et qui doit être ensuite retrouvée dans l'implantation sous test. Ces méthodologies appliquent, successivement, différentes transformations sur le graphe des régions pour que les méthodes de caractérisation d'états de systèmes non temporisés [Cho78, FBK+91, LPB93], puissent être appliquées. Ces transformations apportent un coût supplémentaire qui n'a pas lieu d'être. En effet, au lieu d'adapter le modèle à la méthode de caractérisation d'états, il est plus intéressant de caractériser directement les états du modèle.

Notre méthodologie assure donc la génération de séquences de test en adaptant les étapes de la méthodologie de test W_p [FBK+91] (voir section 2.2.3). Nous présentons une nouvelle définition de la caractérisation d'états de graphe des régions et un algorithme capable de générer l'ensemble des séquences de caractérisation. Nous présentons ensuite une méthode d'exécution des séquences de test sur l'implantation et la complexité complète de la méthodologie. Nous émettons toutefois l'hypothèse qu'à partir de chaque état de la spécification, nous ne pouvons avoir plusieurs transitions étiquetées par des symboles de sortie. Celle-ci est nécessaire pour assurer lors de la phase de test, que les séquences d'actions qui caractérisent les états, peuvent être exécutées.

5.2.1 Relation de Conformité

Le test de conformité entre la spécification, décrite sous forme de notation formelle et son implantation, donnée sous la forme d'une boîte noire, nécessite une relation d'équivalence, appelée relation de conformité. De nombreuses relations de conformité ont été

définies pour les systèmes non temporisés, comme par exemple dans [Pha94, Tre96] (voir dans le chapitre 2).

Pour prendre en compte la notion de temps, nous proposons une nouvelle relation, basée sur les traces, qui prend en compte l'exécution des actions du système et leurs contraintes d'horloges. Cette relation de conformité considère un couple contenant un symbole et l'intervalle de temps pendant lequel il sera émis ou reçu. Et nous appelons ce couple une **Action Temporisée**.

Définition 5.2.1 (Action Temporisée et Séquence d'Actions Temporisées) *Soit un graphe des régions* $\mathcal{RA} = (\Sigma_{\mathcal{RA}}, S_{\mathcal{RA}}, s^0_{\mathcal{RA}}, E_{\mathcal{RA}})$ *et* $R_{\mathcal{RA}}$*, l'ensemble des régions d'horloges de* $\mathcal{RA}$*.*

Une Action Temporisée de $\mathcal{RA}$ *est un couple* $\langle A, R\rangle$*, avec* $A \in \Sigma_{\mathcal{RA}}$ *et* $R \in R_{\mathcal{RA}}$*.*

Une séquence temporisée $\sigma = \langle A_1, R_1\rangle \dots \langle A_n, R_n\rangle, n \in \mathbb{R}^{+*}$ *est une séquence d'Actions temporisées* $\langle A_i, R_i\rangle, (1 \leq i \leq n)$*. Nous notons aussi* $TIMSEQ_{\mathcal{RA}}$*, l'ensemble fini des séquences temporisées de* $\mathcal{RA}$*.*

De ce fait, $\langle ?A, R\rangle$ implique que le symbole $?A$ peut être donné au système à toute valuation d'horloges de R. De même, $\langle !A, R\rangle$ implique que le symbole $!A$ peut être reçu du système à une valuation d'horloges de R. $\langle \delta, R\rangle$ exprime un laps de temps s'écoulant dans la région d'horloges R.

La spécification et l'implantation étant supposées décrites dans le même formalisme, c'est-à-dire le modèle du graphe des régions. Leur équivalence est mise en forme par la relation de conformité suivante, basée sur leurs traces temporisées, et nous dirons que l'implantation I est conforme à la spécification S, ssi I est équivalent en trace temporisée à S.

Définition 5.2.2 (Relation de Conformité) *Deux séquences d'actions temporisées* $\sigma =< A_1, R_1 > ... < A_i, R_i > ... < A_n, R_n >, n \in \mathbb{R}^{+*}$ *et* $\sigma' =< B_1, R'_1 > ... < B_i, R'_i > ... < B_n, R'_n >, n' \in \mathbb{R}^{+*}$ *sont deux traces temporisées équivalentes ssi* $\forall i \mid 1 \leq i \leq n$ *:*

- $A_i = B_i$,
- *si* A_i *et* B_i *sont des symboles d'entrée, alors :*
  1. $\exists (v_{init}, v_{final}) \in R'_i \times R'_i, (v_{init}, v_{final}) \in R_i \times R_i$,
  2. $\exists (v_{init}, v_{final}) \in R_i \times R_i, (v_{init}, v_{final}) \in R'_i \times R'_i$
- *Si* A_i *et* B_i *sont des symboles de sortie,* A_i *peut être observé à au moins une valuation d'horloges de* R'_i

Soit $Trace((S_1, R_1)) = \{\sigma =< A_1, R_1 > ... < A_n, R_m > \mid \exists (S_i, R_j)$ *(avec* $1 \leq i \leq n+1, 1 \leq j \leq m+1), (S_1, R_1) \xrightarrow{A_1} ...(S_n, R_m) \xrightarrow{A_n} (S_{n+1}, R_{m+1})\}$*, l'ensemble des traces temporisées de l'état* (S_1, R_1)*. Deux états* s *et* s' *sont équivalents en traces temporisées, noté* $s \sim_{TimedTrace} s'$ *ssi* $\forall \sigma \in Trace(s) \exists \sigma' \in Trace(s')$ *tel que* σ *et* σ' *soient deux traces temporisées équivalentes.*

Finalement, deux graphes des régions I *et* S *sont équivalents en trace temporisées, noté* $I \sim_{TimedTrace} S$*, ssi* $i_0 \sim_{TimedTrace} s_0$*, avec* s_0 *l'état initial de* S*, et* i_0 *l'état initial de* I*.*

Pour conclure sur la conformité de l'implantation I par rapport à la spécification S, des séquences de test doivent être dérivées pour expérimenter I et vérifier la satisfaction de la relation de conformité. Ces séquences de test sont des séquences de tuples définies par :

Définition 5.2.3 (Séquences de Test Temporisées) *Soit $\mathcal{RA} = (\Sigma_{\mathcal{RA}}, S_{\mathcal{RA}}, s^0_{\mathcal{RA}}, E_{\mathcal{RA}})$ un graphe des régions et $R_{\mathcal{RA}}$ l'ensemble des régions d'horloges de $\mathcal{RA}$.*

Une séquence de test temporisée T dérivée de $\mathcal{RA}$ est une séquence de tuples $\langle A, R, l\rangle$ où :
- *$A \in \Sigma_{\mathcal{RA}}$,*
- *$R \in R_{\mathcal{RA}}$,*
- *$l : T \rightarrow \{pass, fail, inconclusive\}$ est une fonction d'étiquetage.*

Par l'application de ces séquences de test, plusieurs erreurs peuvent être trouvées sur l'implantation sous test. Le modèle de fautes, réunissant les types de fautes que nous pouvons obtenir, sur le modèle du graphe des régions est défini en section 3.2.1.

5.2.2 Caractérisation d'États Temporisés

La première définition de la caractérisation d'états est décrite dans [Gil62], et utilisée une première fois dans une méthodologie de test dans [Cho78]. Elle peut être résumée par "Un ensemble de caractérisation W est composé de séquences d'entrées qui permettent la distinction du comportement de chaque paire d'états d'un automate minimal, en observant leurs réactions"

Le modèle du graphe des régions, tel que nous l'avons défini dans la section 3.1.2, contient des propriétés temporelles et ne demande pas à ce que chacun de ses états soit obligatoirement stimulé par un symbole d'entrée pour réagir avec un symbole de sortie. Par conséquent, pour distinguer deux états d'un graphe des régions, nous devons considérer de nouvelles propriétés, qui n'existent pas dans les systèmes non temporisés. La distinction entre deux états d'un graphe des régions, devant être observable, va dépendre essentiellement des symboles de sortie, qui sont émis par le système lors de ses exécutions, et de ses propriétés temporelles. Plus précisément, cette distinction va toujours dépendre du symbole observé (comme dans les systèmes non temporisés), mais aussi du moment de cette observation.

Ainsi, pour distinguer deux états d'un graphe des régions, et pour générer l'ensemble de caractérisation d'états temporisés, nous considérerons les propriétés comportementales observables et les propriétés temporelles caractérisant un état. Plus précisément, seront prises en compte, les séquences d'actions temporisées qui peuvent être exclusivement exécutées à partir d'un état et celles qui ne peuvent jamais être exécutées, quelquesoit l'exécution à partir de cet état. Pour un état S, nous appelons ces séquences d'actions temporisées, respectivement, les **séquences acceptantes** de S et les **séquences refusantes** de S.

Ces concepts de caractérisation incitent à présenter une première définition d'un ensemble de caractérisation d'états temporisés.

Définition 5.2.4 (Ensemble de Caractérisation d'états Temporisés, TW) *Un ensemble de Caractérisation Temporisé TW, est composé de séquences d'actions temporisées*

permettant de distinguer le comportement de chaque paire d'états dans un graphe des régions minimal, en observant et comparant leurs réactions et le moment de l'observation de ces réactions.

Séquence acceptante et refusante d'un état de graphe des régions

Définition 5.2.5 (Séquence Acceptante et Refusante d'un système temporisé) *Soit S un système temporise où la modélisation du temps est obtenue par un ensemble d'horloges prenant des valeurs réelles.*

Une séquence d'actions σ temporisées est dite acceptante d'un état e quelconque de S ssi toutes les actions temporisées de σ peuvent être exécutées séquentiellement à partir de e.
Une séquence d'actions σ temporisées est dite refusante d'un état quelconque e ssi, σ ne peut jamais être complètement exécutée depuis e. C'est-à-dire, si pour chaque action temporisée $\langle ?A, R\rangle$, "$?A$" est donné à S pour n'importe quelle valuation d'horloges v in$R$, alors il existe $\langle !B, R'\rangle \in \sigma$ telle que soit "$!B$" n'est pas observé, soit "$!B$" est reçu à une valuation d'horloges $v_2 \notin R'$.

Ces séquences d'actions temporisées vont permettre de distinguer deux états d'un graphe des régions entre eux, en observant le comportement dans le temps qui est accepté et refusé par ces deux états. Les séquences acceptantes d'un état temporisé sont composées d'éléments de base qui sont les séquences d'actions temporisées présentées dans les propositions suivantes.

Une action temporisée $\langle !A, R\rangle$ contenant un symbole de sortie $!A$ est "forcément" acceptante d'un état quelconque (S, R) si (S, R) possède une transition sortante étiquetée par le symbole $!A$. Cet aspect est décrit par la proposition suivante :

Proposition 5.2.1 *Soit $\mathcal{RA} = (\Sigma_{\mathcal{RA}}, S_{\mathcal{RA}}, s^0_{\mathcal{RA}}, E_{\mathcal{RA}})$ un graphe des régions et (S, R) un état de $\mathcal{RA}$.*

Une action temporisée $\langle A, R\rangle \in TIMSEQ_{\mathcal{RA}}$, avec $A \in \{! \times \Sigma_{\mathcal{RA}}\}$, est une séquence acceptante de $(S, R) \in S_{\mathcal{RA}}$ et permet d'atteindre (S', R'), noté $(S, R) \overset{\langle A,R\rangle}{\Rightarrow} (S', R')$, ssi la transition $(S, R) \xrightarrow{!A} (S', R')$ appartient à $E_{\mathcal{RA}}$.

Preuve: Supposons que l'état $(S, R) \in S_{\mathcal{RA}}$ a une transition sortante $(S, R) \xrightarrow{!A} (S', R') \in E_{\mathcal{RA}}$. Cette transition modélise la réception observable du symbole $!A$ à une valuation d'horloges de R. $\langle !A, R\rangle$ peut donc être exécutée depuis (S, R). $\langle !A, R\rangle$ est donc une séquence acceptante de (S, R). ∎

Le fait qu'une action temporisée $\langle !A, R\rangle$ soit acceptée ou refusée par l'implantation peut être garanti en observant ses réactions. En effet, si elle est acceptée alors le symbole "!A" pourra être observé à une valuation d'horloges appartenant à R. Il n'en est rien pour une action temporisée $\langle ?A, R\rangle$ contenant un symbole d'entrée. Donc nous dirons que de telles actions temporisées sont toujours acceptées par tout état. C'est ce que décrit la définition suivante. Cette dernière va permettre de distinguer deux états uniquement par rapport aux réactions observables du système dans le temps.

Définition 5.2.6 *Soit* $\mathcal{RA} = (\Sigma_{\mathcal{RA}}, S_{\mathcal{RA}}, s^0_{\mathcal{RA}}, E_{\mathcal{RA}})$ *un graphe des régions et* (S, R') *un état de* $\mathcal{RA}$.

Une action temporisée $\langle A, R\rangle \in TIMSEQ_{\mathcal{RA}}$, *avec* $A \in \{? \times \Sigma_{\mathcal{RA}}\} \cup \{\delta\}$ *est définie comme étant une action acceptante de* (S, R'), *quelquesoit* (S, R'). *Dans ce cas, nous notons que* $\langle A, R\rangle$ *est une action acceptante de* (S, R'), *par* $(S, R') \overset{\langle A,R\rangle}{\Rightarrow} (S'', R'')$, *ssi* $(S, R') \overset{A}{\rightarrow} (S'', R'') \in E_{\mathcal{RA}}$ *et* $R = R'$. *Sinon, nous notons* $(S, R') \overset{\langle A,R\rangle}{\Rightarrow} (S, R')$

Supposons maintenant le cas d'un état d'un graphe des régions tel qu'un laps de temps doive s'écouler depuis celui-ci pour pouvoir exécuter une action temporisée $\langle !A, R\rangle$ observable. La Proposition suivante montre que $\langle !A, R\rangle$ est une action acceptante de cet état.

Proposition 5.2.2 *Soit* $\mathcal{RA} = (\Sigma_{\mathcal{RA}}, S_{\mathcal{RA}}, s^0_{\mathcal{RA}}, E_{\mathcal{RA}})$ *un graphe des régions et* (S_1, R_1) *un état de* $\mathcal{RA}$.

Une action temporisée $\langle A, R\rangle \in TIMSEQ_{\mathcal{RA}}$, *avec* $A \in \{! \times \Sigma_{\mathcal{RA}}\}$, *est une action acceptante de* (S_1, R_1) *et permet d'atteindre* $(S', R') \in S_{\mathcal{RA}}$, *noté* $(S_1, R_1) \overset{\langle !A,R\rangle}{\Rightarrow} (S', R')$, *ssi* (S_1, R_1) *a une séquence de transitions sortante* $(S_1, R_1) \overset{\delta}{\rightarrow} (S_2, R_2)...(S_n, R_n) \overset{\delta}{\rightarrow} (S, R) \overset{!A}{\rightarrow} (S', R')$, *avec* $(S_i, R_i) \overset{\delta}{\rightarrow} (S_{i+1}, R_{i+1}) \in E_{\mathcal{RA}} (1 \leq i < n)$, $(S_n, R_n) \overset{\delta}{\rightarrow} (S, R) \in E_{\mathcal{RA}}$ *et* $(S, R) \overset{!A}{\rightarrow} (S', R') \in E_{\mathcal{RA}}$.

Preuve:

Une action temporisée $\langle !A, R\rangle$ modélise un symbole de sortie $!A$ qui doit être observé à une valuation d'horloges de R. Supposons qu'un état $(S_1, R_1) \in S_{\mathcal{RA}}$ a une séquence de transitions sortante $(S_1, R_1) \overset{\delta}{\rightarrow} (S_2, R_2)...(S_n, R_n) \overset{\delta}{\rightarrow} (S, R) \overset{!A}{\rightarrow} (S', R')$. Cette séquence modélise un laps de temps suivi de la réception d'un symbole de sortie $!A$ à une valuation d'horloges de R. Un laps de temps n'étant pas observable, $\langle !A, R\rangle$ est exécutable depuis (S_1, R_1), c'est-à-dire que le symbole "!A" peut au moins une fois être observé à une valuation d'horloges de R. Donc $\langle !A, R\rangle$ est une séquence acceptante de (S_1, R_1).

■

Grâce à ces propositions, nous avons la possibilité de redéfinir une séquence acceptante et refusante d'un état temporisé d'un graphe des régions. En effet, une séquence acceptante peut être construite en concaténant les séquences acceptantes des propositions précédentes.

Définition 5.2.7 (Séquence Acceptante et Refusante d'un graphe des régions) *Soit* $\mathcal{RA} = (\Sigma_{\mathcal{RA}}, S_{\mathcal{RA}}, s^0_{\mathcal{RA}}, E_{\mathcal{RA}})$ *un graphe des régions et* $S, R)$ *un état de* $\mathcal{RA}$.

Une séquence d'actions temporisées $\sigma = \langle A_i, R_i\rangle$ *avec* $1 \leq i \leq n$ *est une séquence acceptante de* (S, R) *permettant d'atteindre un état* (S', R'), *notée* $(S, R) \overset{\sigma}{\Rightarrow} (S', R')$ *ssi :*

- $(S, R) \overset{\langle A_1,R_1\rangle}{\Rightarrow} (S_1, R_1)$, *avec* $(S_1, R_1) \in S_{\mathcal{RA}}$,
- $\forall i (1 < i < n), (S_i, R_i) \overset{\langle A_{i+1},R_{i+1}\rangle}{\Rightarrow} (S_{i+1}, R_{i+1})$, *avec* $(S_i, R_i) \in S_{\mathcal{RA}}$, $(S_{i+1}, R_{i+1}) \in S_{\mathcal{RA}}$,
- $(S_{n-1}, R_{n-1}) \overset{\langle A_n,R_n\rangle}{\Rightarrow} (S', R')$, *avec* $(S_{n-1}, R_{n-1}) \in S_{\mathcal{RA}}$

Un séquence refusante $\sigma \in TIMSEQ_{\mathcal{RA}}$ *de* (S, R), *est une séquence d'actions temporisées tel qu'il existe* $\langle !A, R\rangle \in TIMSEQ_{\mathcal{RA}}$, *et* $\sigma = \sigma' \langle !A, R\rangle \sigma''$, σ' *est une séquence*

acceptante $(S,R) \stackrel{\sigma'}{\Rightarrow} (S',R')$, *et* $\langle !A,R\rangle$ *n'est pas acceptante de* (S',R') *et donc ne vérifie pas au moins l'une des Propositions 5.2.1 ou 5.2.2, ou la définition 5.2.6*

Ensemble de Caractérisation d'États de graphe des régions

Ayant défini les séquences acceptantes et refusantes d'un état d'un graphe des régions, il est maintenant possible d'aboutir à notre but principal, à savoir la distinction de deux états d'un graphe des régions entre eux. En effet, si un état (S,R) d'un graphe des régions, a une séquence acceptante σ qui est refusante d'un état S',R' alors nous pouvons en déduire que le comportement de S est différent de celui de S'.

Proposition 5.2.3 *Soit* $\mathcal{RA} = (\Sigma_{\mathcal{RA}}, S_{\mathcal{RA}}, s^0_{\mathcal{RA}}, E_{\mathcal{RA}})$ *un graphe des régions et deux états* (S,R) *et* (S',R') *de* $\mathcal{RA}$.

Supposons que $\sigma \in TIMSEQ_{\mathcal{RA}}$ *soit une séquence acceptante de* (S,R) *et une séquence refusante de* (S',R'). *Alors les réactions observables dans le temps du système, obtenues à partir de* (S,R), *sont différentes de celles obtenues à partir de* (S',R').

Preuve:

D'après la proposition 5.2.6, les action temporisées $\langle A,R\rangle$, avec $A \in \{? \times \Sigma\} \cup \delta$, sont des séquences acceptantes pour tout état de $\mathcal{RA}$.

Donc à partir de (S',R'), il existe un état (S_1,R_1), une séquence temporisée $\sigma' \in TIMSEQ_{\mathcal{RA}}$ et une action temporisée $\langle !A,R\rangle \in TIMSEQ_{\mathcal{RA}}$, tel que $(S',R') \stackrel{\sigma}{\Rightarrow} (S_1,R_1)$ et $(S_1,R_1) \stackrel{\langle !A,R\rangle}{\not\Rightarrow}$. Ceci implique, à partir de (S_1,R_1), que soit le symbole $!A$ ne peut être observé, soit il est observable à une valuation d'horloges n'appartenant pas à R.

Depuis (S,R), il existe aussi deux états (S_2,R_2) et (S_3,R_3) tels que $(S,R) \stackrel{\sigma'}{\Rightarrow} (S_2,R_2)$ et $(S_2,R_2) \stackrel{\langle !A,R\rangle}{\Rightarrow} (S_3,R3)$. Ceci implique que le symbole $!A$ est observable à une valuation d'horloges de R.

Par conséquent, les réactions observables dans le temps à partir de (S_1,R_1) et de (S_2,R_2) sont différentes. (S,R) et (S',R') n'ont pas les mêmes réactions observables dans le temps. ■

Nous en déduisons qu'en comparant les séquences acceptantes et refusantes de deux états d'un graphe des régions, nous pouvons les distinguer. En effet, si ces séquences sont appliquées à deux états et donnent des réactions observables dans le temps différentes alors la Définition 5.2.4 nous permet d'affirmer qu'ils sont distincts et donc caractérisés entre eux. En appliquant le même raisonnement sur chaque paire d'états du système, nous obtenons un ensemble de séquences d'actions temporisées qui permettent de caractériser le comportement d'un état par rapport aux autres.

Nous pouvons maintenant définir l'ensemble de caractérisation d'états temporisés d'un graphe des régions $\mathcal{RA}$ en comparant les séquences acceptantes et refusantes de chaque paire d'états de $\mathcal{RA}$. Plus précisément, si un état s possède un ensemble de séquences acceptantes et refusantes différent de celui de chaque état d'un graphe des régions, alors s est caractérisé des autres états. Si pour chaque état d'un graphe des régions, nous suivons le même raisonnement, nous obtenons l'ensemble de caractérisation d'états temporisé TW.

Une condition nécessaire à l'existence d'un ensemble de caractérisation d'états sur un graphe des régions $\mathcal{RA}$, est que $\mathcal{RA}$ doit être minimal. En effet, dans le cas contraire, au moins deux états de $\mathcal{RA}$ ne sont pas distinguables.

Proposition 5.2.4 *Soit* $\mathcal{RA} = (\Sigma_{\mathcal{RA}}, S_{\mathcal{RA}}, s^0_{\mathcal{RA}}, E_{\mathcal{RA}})$ *un graphe des régions. Un ensemble de caractérisation TW existe sur* $\mathcal{RA}$ *si et seulement si* $\mathcal{RA}$ *est minimal.*

Preuve:

Supposons que $\mathcal{RA}$ ne soit pas minimal. Alors il existe deux états $(S_1, R_1) \in S_{\mathcal{RA}}$ et $(S_2, R_2) \in S_{\mathcal{RA}}$ tels que $\exists A \in \Sigma, \exists (S, R) \in S_{\mathcal{RA}}, (S_1, R_1) \xrightarrow{A} (S, R) \in E_{\mathcal{RA}},$ et $(S_2, R_2) \xrightarrow{A} (S, R) \in E_{\mathcal{RA}}$.

Supposons maintenant que (S_1, R_1) et (S_2, R_2) soient distinguables. Alors il existe $\sigma \in TIMSEQ_{\mathcal{RA}}$ tel que soit $(1)\sigma$ est une séquence acceptante de (S_1, R_1) et refusante de (S_2, R_2), soit $(2)\sigma$ est une séquence acceptante de (S_2, R_2) et refusante de (S_1, R_1).
(1) il existe $\langle B, Reg\rangle \in TIMSEQ_{\mathcal{RA}}$, $(S', R') \in S_{\mathcal{RA}}$ et $\sigma' \in TIMSEQ_{\mathcal{RA}}$ tel que $(S_2, R_2) \overset{\sigma'}{\Rightarrow} (S', R')$ et $(S', R') \overset{\langle B,Reg\rangle}{\not\Rightarrow}$.

- si $\sigma' = \epsilon$, alors $(S_1, R_1) \overset{\langle B,Reg\rangle}{\Rightarrow} (S, R)$ et $(S_2, R_2) \overset{\langle B,Reg\rangle}{\not\Rightarrow}$. Or (S_1, R_1) et (S_2, R_2) modélisent le même comportement, donc contradiction.
- si $\sigma' \neq \epsilon$ et $\sigma' = \langle A, R\rangle\sigma''$, alors $(S_2, R_2) \overset{\langle A,R\rangle}{\Rightarrow} (S, R)$, $(S, R) \overset{\sigma''}{\Rightarrow} (S', R')$ et $(S', R') \overset{\langle B,Reg\rangle}{\not\Rightarrow}$. Donc, il existe un chemin tel que $(S_1, R_1) \overset{\langle A,R\rangle}{\Rightarrow} (S, R)$, $(S, R) \overset{\sigma''}{\Rightarrow} (S', R')$ et $(S', R') \overset{\langle B,Reg\rangle}{\not\Rightarrow}$. Donc, pour une exécution utilisant ce chemin, σ est une séquence refusante de (S_1, R_1). Donc contradiction.

(2) de même que (1).
De ce fait, (S_1, R_1) et (S_2, R_2) ne sont pas distinguables et TW n'existe pas si $\mathcal{RA}$ n'est pas minimal.

■

Nous définissons ensuite l'ensemble de caractérisation d'états temporisés, non pas sur l'ensemble des états d'un graphe des régions $\mathcal{RA}$ mais sur un sous-ensemble $S_{reduit} = \{(s, R) \in S_{\mathcal{RA}} \mid \exists (s, R) \xrightarrow{A} (s', R') \in E_{\mathcal{RA}}, A \neq \delta\}$. La motivation de ce choix est due au fait que les états n'appartenant pas à S_{reduit}, représentent uniquement un passage d'une région d'horloges vers une autre région d'horloges. Or ces laps de temps peuvent selon les exécutions être très proches de 0 et indécelables par le testeur. De plus ces états sont ajoutés lors de la transformation de l'automate temporisé en graphe des régions et n'existent donc pas explicitement dans l'implantation.

Finalement, l'ensemble de caractérisation d'états d'un graphe des régions $\mathcal{RA}$ est obtenu par la proposition suivante.

Proposition 5.2.5 (Ensemble de Caractérisation d'États Temporisés) *Soit* $\mathcal{RA} = (\Sigma_{\mathcal{RA}}, S_{\mathcal{RA}}, s^0_{\mathcal{RA}}, E_{\mathcal{RA}})$ *Un graphe des régions minimal, et* $S_{reduit} = \{(s, R) \in S_{\mathcal{RA}} \mid \exists (s, R) \xrightarrow{A} (s', R') \in E_{\mathcal{RA}}, A \neq \delta\}$.

Deux états (S_1, R_1) *et* (S_2, R_2) *de* S_{reduit} *sont distinguables par une séquence d'actions temporisées* σ, *noté* $(S_1, R_1) \overset{\neq}{\sigma} (S_2, R_2)$ *ssi :*

- *soit* $\exists (S', R') \in S_{\mathcal{RA}}, (S_1, R_1) \stackrel{\sigma}{\Rightarrow} (S', R')$ *et* $(S_2, R_2) \stackrel{\sigma}{\not\Rightarrow}$
- *soit* $\exists (S', R') \in S_{\mathcal{RA}}, (S_2, R_2) \stackrel{\sigma}{\Rightarrow} (S', R')$ *et* $(S_1, R_1) \stackrel{\sigma}{\not\Rightarrow}$

Notons $TW_{(S_1,R_1)}$*, l'ensemble des séquences d'actions temporisées distinguant* (S_1, R_1) *des autres états de* S_{reduit}*.* $TW_{(S_1,R_1)}$ *est composé de séquences tel que :*

- $\forall (S, R) \in S_{reduit} \neq (S_1, R_1), \exists \sigma \in TW_{(S_1,R_1)}, (S_1, R_1) \stackrel{\neq}{\sigma} (S, R)$.
- $\exists \sigma \in TW_{(S_1,R_1)}$, $\sigma \in TIMSEQ_{\mathcal{RA}}$ *est une séquence acceptante finie par une action temporisée* $\langle A, R \rangle$*, avec* $A \in \{! \times \Sigma_{\mathcal{RA}}\}$

Finalement, un Ensemble de Caractérisation d'États Temporisés de $\mathcal{RA}$*, noté* $TW_{\mathcal{RA}}$ *est égal à* $\{TW_{(S_1,R_1)}, ...TW_{(S_n,R_n)}\}$*, avec* $\{(S_1, R_1), ..., (S_n, R_n)\} = S_{\mathcal{RA}}$.

Preuve:

Soit une séquence d'actions temporisées $\sigma \in TIMSEQ_{\mathcal{RA}}$ telle que $\exists (S', R') \in S_{\mathcal{RA}}$, $(S_1, R_1) \stackrel{\sigma}{\Rightarrow} (S', R')$ et $(S_2, R_2) \stackrel{\sigma}{\not\Rightarrow}$ ou $\exists (S', R') \in S_{\mathcal{RA}}, (S_2, R_2) \stackrel{\sigma}{\Rightarrow} (S', R')$ et $(S_1, R_1) \stackrel{\sigma}{\not\Rightarrow}$. D'après la Proposition 5.2.3, les réactions observables dans le temps de (S_1, R_1) et (S_2, R_2) sont différentes.
Soit maintenant un état (S_1, R_1) et son ensemble de caractérisation TW_{S_1,R_1}. Pour tout état $(S, R) \in S_{reduit}$ différent de (S_1, R_1), il existe $\sigma \in TIMSEQ_{\mathcal{RA}}$, tel que σ donne des réactions observables différentes dans le temps entre (S_1, R_1) et les autres états de S_{reduit}. L'ensemble TW est tel que pour tout état $(S, R) \in S_{reduit}$ et pour tout état $(S', R') \in S_{reduit}$ différent de (S, R), il existe une séquence $\sigma \in TW$ qui donne des réactions observables dans le temps différentes entre (S, R) et (S', R').
D'après la Définition 5.2.5, TW est donc un ensemble de caractérisation d'états de $\mathcal{RA}$. ■

Soit l'exemple de graphe des régions de la Figure 5.43 et les régions d'horloges illustrées en Figure 5.44. Pour chaque état, nous obtenons les ensembles de caractérisations suivants : $TW_{S_1,R_1} = \{\langle ?b, R_1\rangle\langle !e, R_1\rangle, \langle !e, R_1\rangle\}$, $TW_{S_2,R_1} = \{\langle !e, R_1\rangle\}$, $TW_{S_4,R_1} = \{\langle ?a, R_1\rangle\langle !e, R_1\rangle, \langle !e, R_1\rangle\}$

Ainsi, l'état S_2 est caractérisé par $\langle !e, R_1\rangle$ car il est le seul à donner ce symbole de sortie. La séquence $\langle ?b, R_1\rangle\langle !e, R_1\rangle$ permet de distinguer S_1 de S_4 car elle est acceptante pour S_1 et refusante de S_4. C'est-à-dire qu'en donnant " ?b" à S_4, le symbole " !e" ne pourra être observé. Par contre $\langle ?b, R_1\rangle\langle !e, R_1\rangle$ n'est pas suffisante pour caractériser S_1 et S_2 car c'est une séquence acceptante des deux états. Ainsi, nous ajoutons $\langle !e, R_1\rangle$ à TW_{S_1,R_1} car c'est une séquence refusante de S_1 et acceptante de S_2. Ces séquences suffisent à caractériser les trois états. Cependant, elles ne permettent aucune observation à partir de l'état S_4. Pour montrer qu'il existe, nous ajoutons à son ensemble de caractérisation la séquence acceptante $\langle ?a, R_1\rangle\langle !e, R_1\rangle$.

Algorithmes de génération de l'ensemble de caractérisation TW

Pour générer l'ensemble TW, nous adaptons l'algorithme de génération de l'ensemble de caractérisation d'états décrit dans [Gil62].

Cet algorithme consiste à générer des tables, appelées des "tables P_k", permettant de distinguer deux à deux des groupes d'états. En construisant toutes les tables P_k, (au plus n-1 tables si n est le nombre d'états de la spécification) et en parcourant ces tables, chaque état peut être distingué des autres. Notre but est de redéfinir les tables P_k et de montrer

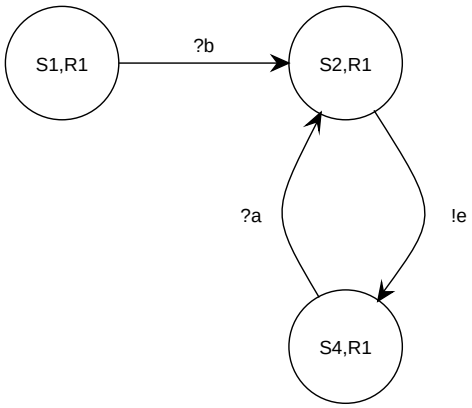


FIG. 5.43 – Un graphe des régions

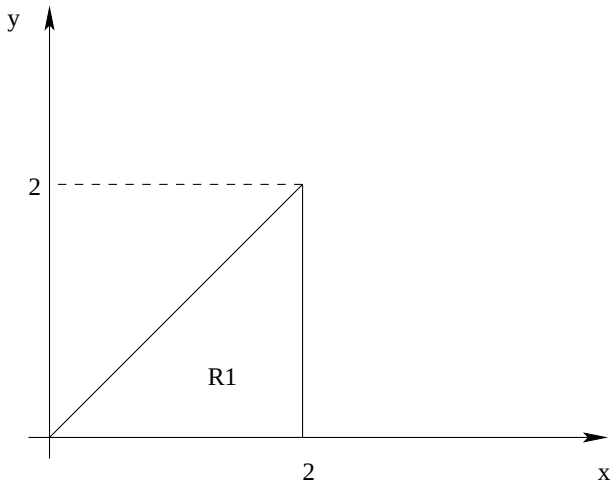


FIG. 5.44 – Régions d'horloges du graphe des régions de la Figure 5.43

comment construire la première table P_1, pour qu'à partir de celle-ci, l'algorithme décrit dans [Gil62], construise les tables suivantes.

Pour les automates non temporisés, les tables P_k sont construites sur l'ensemble des états à caractériser et sur l'ensemble des symboles de l'automate. Le modèle utilisé dans notre méthodologie étant le graphe des régions, les tables P_k seront donc construites sur l'ensemble des états temporisés du graphe des régions à caractériser et sur son ensemble d'actions temporisées. Dans une table P_k, les états, non distinguables par un ensemble de séquences d'actions temporisées de longueur allant de 1 à k, sont regroupés. Chaque groupe est identifié par un indice unique qui est ajouté aux états du groupe. Ainsi, deux groupes a et b sont séparés si chaque état de a a un ensemble de séquences acceptantes tel que chaque état du groupe b, refuse au moins une séquence de cet ensemble. La lecture de la dernière table ne permet pas de trouver les ensembles de caractérisation. Il est nécessaire pour cela de parcourir plusieurs tables par un algorithme donné ci-après.

Ainsi, la table P_1 montre les groupes d'états qui sont distingués par un ensemble d'actions temporisées (séquences de longueur 1), selon la Proposition 5.2.5. Nous construisons la table P_1 grâce aux points suivants :

1. nous construisons une première table sur $S_{\mathcal{RA}}$ et $TIMESEQ_{\mathcal{RA}}$, appelée la **table de refus**. Celle-ci a une entrée "A" (acceptée) pour un état $S \in S_{\mathcal{RA}}$ et une action temporisée $\langle A, R\rangle \in TIMESEQ_{\mathcal{RA}}$ ssi $S \overset{\langle A,R\rangle}{\Rightarrow} S'$, avec S' un état quelconque de $\mathcal{RA}$. Autrement, l'entrée de la table est "R" exprimant l'action temporisée est refusante pour S. Ainsi, chaque état S a une entrée pour chaque action temporisée, illustrant si celle-ci est une action acceptante ou refusante de S. Cette table permet donc de caractériser des états par rapport aux lignes de la table, grâce à un ensemble d'actions temporisées. Elle permet aussi de réunir en groupe les états qui ne sont pas distinguables deux à deux.
2. Nous rassemblons les états, ayant les mêmes lignes dans la table de refus, dans un groupe étiqueté par un indice quelconque mais unique.
3. Pour un état S et une action temporisée $\langle A, R\rangle$, nous avons une entrée dans la table P_1 qui est l'état S' tel que $S \overset{\langle A,R\rangle}{\Rightarrow} S'$ ou $S' = S$ si $S \overset{\langle A,R\rangle}{\not\Rightarrow}$.
4. un indice est attaché à chaque entrée de la table tel que l'indice est celui du groupe auquel l'entrée appartient.

Les autres tables P_k sont construites d'une façon récursive. La table P_{k+1} est obtenue à partir de la table P_k comme suit : une paire de lignes du même groupe dans la table P_k qui, pour chaque colonne, donne un indice identique sont groupées dans la table $Pk+1$. Une paire de lignes du même groupe dans la table P_k qui, pour chaque colonne, donne des indices différents sont disjointes dans la table $Pk+1$. Les groupes de la table P_k sont donc éventuellement séparés en plusieurs groupes dans la table P_{k+1}. Si la table P_{k+1} est la même que la table P_k, alors la table P_k est la dernière.

Par exemple, depuis les états illustrés en Figure 5.43 et des régions illustrées en Figure 5.44, nous obtenons la table de refus décrite dans le Tableau 5.8, puis les tables P_1 et P_2 décrites grâce aux Tableaux 5.9 et 5.10.

$S_{\mathcal{RA}}$ / $TIMSEQ_{\mathcal{RA}}$	$\langle ?bR_1 \rangle$	$\langle ?aR_1 \rangle$	$\langle !eR_1 \rangle$
(S_1, R_1)	A	A	R
(S_2, R_1)	A	A	A
(S_4, R_1)	A	A	R

TAB. 5.8 – Table de Refus

	$S_{\mathcal{RA}}$ / $TIMSEQ_{\mathcal{RA}}$	$\langle ?bR_1 \rangle$	$\langle ?aR_1 \rangle$	$\langle !eR_1 \rangle$
Groupe A	$(S_1, R_1)_A$	$(S_2, R_1)_B$	$(S_1, R_1)_A$	$(S_1, R_1)_A$
	$(S_4, R_1)_A$	$(S_4, R_1)_A$	$(S_2, R_1)_B$	$(S_4, R_1)_A$
Groupe B	$(S_2, R_1)_B$	$(S_2, R_1)_B$	$(S_2, R_1)_B$	$(S_4, R_1)_A$

TAB. 5.9 – Table P_1

Grâce à ces tables, il est maintenant possible de distinguer une paire d'états (S, S') ([Gil62]). L'algorithme se résume par les étapes suivantes :

Soit deux états S et S'.

1. Trouver l tel que S et S' sont dans le même groupe dans P_{l-1} et dans deux groupes différents dans P_l. Mettre un compteur k à 1
2. Si $l-k > 0$ alors aller à l'étape 3. Si $l-k = 0$, l'action temporisée A_k est donnée dans la table de refus tel que A_k distingue S et S'. $A_1...A_k$ est la séquence de distinction minimale de S et S'.
3. Dans P_{l-k}, l'action temporisée A_k est l'action temporisée d'une colonne tel que les lignes S et S' donne des entrées E_i et E'_j avec des indices différents. $S = E$ et $S' = E'$. Incrémenter k de 1 et retourner à l'étape 2.

Par exemple, selon les tables des Figures 5.8, 5.9 et 5.10, les états (S_1, R_1) et (S_4, R_1) sont distinguables par $\langle ?b, R_1 \rangle \langle !e, R_1 \rangle$. En effet, nous cherchons dans la dernière table où (S_1, R_1) et (S_4, R_1) sont dans le même groupe. $\langle ?b, R_1 \rangle$ donne des entrées avec indice différent pour (S_1, R_1) et (S_4, R_1), qui sont $(S_2, R_1)_B$ et $(S_4, R_1)_A$. Puis dans la table de refus, nous cherchons quelle est l'action qui distingue (S_2, R_1) et (S_4, R_1), et nous obtenons $\langle !e, R_1 \rangle$.

5.2.3 Génération des séquences de Test

Les étapes de la méthodologie vont rester, à quelques modifications près, celles de la méthodologie de caractérisation d'états de systèmes non temporisés Wp décrite dans [FBK+91]. C'est en fait l'ensemble de caractérisation d'états temporisés, noté TW, défini en section 5.2.2 qui apportera l'adaptation de la méthodologie au modèle du graphe des régions. Les hypothèses de test sur la spécification et l'implantation, puis les étapes de la méthodologie sont décrites ci-dessous.

Hypothèses de Test

Les hypothèses suivantes sur la spécification décrite par un automate temporisé et l'implantation doivent être satisfaites pour appliquer la méthodologie de test.

	$S_{\mathcal{RA}}$ / $TIMSEQ_{\mathcal{RA}}$	$\langle ?bR_1 \rangle$	$\langle ?aR_1 \rangle$	$\langle !eR_1 \rangle$
Groupe A	$(S_1, R_1)_A$	$(S_2, R_1)_C$	$(S_1, R_1)_A$	$(S_1, R_1)_A$
Groupe B	$(S_4, R_1)_B$	$(S_4, R_1)_B$	$(S_2, R_1)_C$	$(S_4, R_1)_B$
Groupe C	$(S_2, R_1)_C$	$(S_2, R_1)_C$	$(S_2, R_1)_C$	$(S_4, R_1)_B$

TAB. 5.10 – Table P_2

- La spécification et l'implantation doivent posséder le même alphabet,
- La spécification doit être déterministe sur l'ensemble de l'alphabet. C'est-à-dire, depuis n'importe quel état, nous ne pouvons obtenir deux transitions sortantes étiquetées par le même symbole, dont les mêmes contraintes d'horloges impliquent qu'elles soient satisfaites simultanément.
- Depuis n'importe quel état de la spécification, nous ne pouvons obtenir une transition sortante, étiquetée par un symbole d'entrée et une transition sortante, étiquetée par un symbole de sortie, dont les contraintes d'horloges sont satisfaites simultanément.
- Depuis n'importe quel état de la spécification, nous ne pouvons obtenir deux transitions sortantes chacune étiquetée par un symbole de sortie différent.
- L'implantation doit contenir un état initial unique avec ses horloges initialisées à 0. Depuis cet état, il n'existe pas de transitions sortantes étiquetées par un symbole de sortie. En effet, les séquences de test sont appliquées à l'implantation après l'avoir amenée à son état initial.

Étapes de la Méthodologie

La méthodologie de test est composée de deux phases :

- La première vérifie si tous les états de la spécification peuvent être identifiés dans l'implantation sous test. Nous vérifions tous les états sauf ceux qui ont une unique transition sortante étiquetée par δ. Nous appelons $S_{reduit} \subseteq S_{\mathcal{RA}}$ l'ensemble des états vérifiés par la première phase et $S_{reduit} = \{(s, R) \in S_{\mathcal{RA}} \mid \exists (s, R) \xrightarrow{A} (s', R') \in E_{\mathcal{RA}}, A \neq \delta\}$.
- La seconde phase vérifie si toutes les transitions de la spécification sont correctement implantées. Étant donné qu'avec la première phase, nous considérons les états de S_{reduit}, cette phase vérifie toutes les transitions sortantes des états de S_{reduit}. Cet ensemble de transitions, noté $E_{reduit} \subseteq E_{\mathcal{RA}}$ est tel que $E_{reduit} = \{(s, R) \xrightarrow{A} (s', R') \in E_{\mathcal{RA}} \mid (s, R) \in S_{reduit}\}$.

Comme pour la méthode Wp décrite dans [FBK+91], nous utilisons quelques ensembles pour générer les séquences de test. Ceux-ci sont l'ensemble de Couverture d'États (ensemble des préambules) et l'ensemble de Couverture de Transitions.

Définition 5.2.8 (Ensemble de Couverture d'États Q) *Un ensemble de Couverture d'états Q pour un graphe des régions $\mathcal{RA} =< \Sigma_{\mathcal{RA}}, S_{\mathcal{RA}}, s^0_{\mathcal{RA}}, E_{\mathcal{RA}} >$ est composé, pour chaque état $(S_i, R_i) \in S_{\mathcal{RA}}$, d'une séquence d'actions temporisées $\sigma \in TIMSEQ_{\mathcal{RA}}$ qui permet d'atteindre (S_i, R_i) depuis l'état initial (S_0, R_0).*

Définition 5.2.9 (Ensemble de Couverture de Transitions P) *Un ensemble de Couverture de Transition P pour un graphe des régions $\mathcal{RA} =< \Sigma_{\mathcal{RA}}, S_{\mathcal{RA}}, s^0_{\mathcal{RA}}, E_{\mathcal{RA}} >$ est un ensemble de séquences d'actions temporisées tel que pour chaque transition $(S_i, R_i) \xrightarrow{A} (S_j, R_j) \in E_{\mathcal{RA}}$, P est composé de deux séquences temporisées $\sigma \in TIMSEQ_{\mathcal{RA}}$ et $\sigma\langle A, R_i\rangle$ tel que σ permet d'atteindre (S_i, R_i) depuis l'état initial (S_0, R_0) et $\sigma\langle A, R_i\rangle$ permet d'atteindre (S_j, R_j) aussi depuis (S_0, R_0).*

Les étapes de la méthodologie de test sont présentées ci-dessous :

1. L'automate temporisé, modélisant une spécification et satisfaisant les hypothèses précédentes, est premièrement traduit en graphe des régions minimal sur son ensemble d'états, en utilisant l'algorithme décrit dans [ACH+92]. Les régions d'horloges infinies sont bornées d'une façon homogène par rapport aux autres régions d'horloges (voir section 4.6)
2. L'ensemble de Couverture d'États Q est construit sur S_{reduit}. L'ensemble de Couverture de Transitions P est construit sur l'ensemble des transitions E_{reduit}.
3. L'ensemble $R = P/Q$ de séquences temporisées est ensuite obtenu. Le fait de priver P de Q permet, dans le second ensemble de séquences de test, de ne pas ajouter de séquences redondantes déjà construites dans le premier ensemble de séquences de test.
4. Chaque état $(S_i, R_i) \in S_{reduit}$ est caractérisé en utilisant l'algorithme de la section 5.2.2. Nous obtenons un ensemble de caractérisation d'états temporisé $TW = \{TW_{(S_1,R_1)}, ..., TW_{(S_n,R_n)}\}$
5. Un premier ensemble de séquences de test, noté $\prod_1$ est généré tel que $\prod_1 = P.TW = \{p.\sigma \mid p \in P, \sigma \in TW\}$. "." représente l'opérateur de concaténation.
6. Un second ensemble de séquences de test $\prod_2$ est généré tel que $\prod_2 = R \otimes TW =$ $\{r.\sigma_i \mid r \in R, (S_0, R_0) \xrightarrow{r} (S_i, R_i)$, et $\sigma_i \in TW_{(S_i,R_i)}$ si $(S_i, R_i) \in S_{reduit}\} \cup$ $\{r.\langle\delta, R_i\rangle...\langle\delta, R_{n-1}\rangle\sigma_n, (n > i) \mid r \in R, (S_0, R_0) \xrightarrow{r} (S_i, R_i)$, et $(S_i, R_i) \xrightarrow{\delta} (S_{i+1}, R_{i+1})$ $...(S_{n-1}, R_{n-1}) \xrightarrow{\delta} (S_n, R_n)$, et $\sigma_n \in TW_{(S_n,R_n)}$ si $\forall j, (i \leq j \leq n), (S_j, R_j) \notin S_{reduit}\}$. "$\otimes$" représente l'opérateur de concaténation particulier qui ajoute à la suite d'un état S_i l'ensemble de caractérisation $TW(S_i)$.
7. Finalement, une étiquette est ajoutée à chaque action temporisée de chaque sequence $\sigma \in \prod_1 \cup \prod_2$. Nous obtenons un tuple $\langle A, R, l\rangle$, où l est :
   - *inconcluant* si A est un symbole d'entrée ou δ. l modélise que l'action temporisée ne permet pas de conclure sur le déroulement du test car aucune réaction n'est observable depuis l'implantation.
   - *pass* si A est un symbole de sortie et si $\sigma = \sigma'\langle A, R, l\rangle\sigma''$ et $(S_0, R_0) \overset{\sigma'}{\Rightarrow} (S, R), (S, R) \overset{\langle A,R\rangle}{\Rightarrow}$. *pass* modélise que l'action temporisée est acceptée par l'état auquel le symbole est appliqué.
   - *fail* si A est un symbole de sortie, si $\sigma = \sigma'\langle A, R, l\rangle\sigma''$ et si $(S_0, R_0) \overset{\sigma'}{\Rightarrow} (S, R), (S, R) \overset{\langle A,R\rangle}{\not\Rightarrow}$. $fail$ modélise que l'action temporisée est refusée par l'état auquel le symbole est appliqué.

D'après [FBK+91], $\prod_1$ permet de vérifier la première phase de la méthodologie et $\prod_2$ la seconde.

En appliquant la méthode sur l'exemple de graphe des régions de la Figure 5.43, nous obtenons les séquences de test suivantes (les séquences redondantes ont été supprimées) :
$\prod_1 = \{\langle ?b, R_1, inconcluant\rangle\langle !e, R_1, pass\rangle, \langle ?a, R_1, inconcluant\rangle\langle !e, R_1, fail\rangle, \langle !e, R_1, fail\rangle,$
$\langle ?b, R_1, inconcluant\rangle\langle ?b, R_1, inconcluant\rangle\langle !e, R_1, pass\rangle, \langle ?b, R_1, inconcluant\rangle$
$\langle ?a, R_1, inconcluant\rangle\langle !e, R_1, pass\rangle, \langle ?b, R_1, inconcluant\rangle\langle !e, R_1, pass\rangle, \langle ?b, R_1, inconcluant\rangle$
$\langle !e, R_1, pass\rangle\langle ?b, R_1, inconcluant\rangle\langle !e, R_1, fail\rangle, \langle ?b, R_1, inconcluant\rangle\langle !e, R_1, pass\rangle$
$\langle ?a, R_1, inconcluant\rangle\langle !e, R_1, pass\rangle, \langle ?b, R_1, inconcluant\rangle\langle !e, R_1, pass\rangle\langle !e, R_1, fail\rangle\}$

$\prod_2 = \{\langle ?b, R_1, inconcluant\rangle\langle !e, R_1, pass\rangle\langle ?a, R_1, inconcluant\rangle\langle ?a, R_1, inconcluant\rangle$
$\langle !e, R_1, pass\rangle, \langle ?b, R_1, inconcluant\rangle\langle !e, R_1, pass\rangle\langle ?a, R_1, inconcluant\rangle\langle ?b, R_1, inconcluant\rangle$
$\langle !e, R_1, pass\rangle, \langle ?b, R_1, inconcluant\rangle\langle !e, R_1, pass\rangle\langle ?a, R_1, inconcluant\rangle\langle !e, R_1, pass\rangle,$
$\}$

5.2.4 Exécution des Séquences de Test

La méthode d'exécution est sensiblement la même que celle décrite en section 5.2.4. La principale différence entre les méthodes d'exécutions des deux méthodologies est due au fait que cette fois les régions ne sont pas découpées en plusieurs intervalles. Ainsi il ne sera plus nécessaire d'ajouter une branche étiquetée par δ avant chaque branche nouvellement créée pour montrer qu'un laps de temps peut être nécessaire pour passer d'une région d'horloges à une autre.

L'architecture de test est toujours celle décrite dans [PF99b]. La méthode d'exécution des tests est la suivante :

- Soit $\langle ?A, R, v\rangle$ un tuple d'une séquence de test, contenant un symbole d'entrée $?A$. Le symbole "A" est émis par le testeur vers l'implantation *le plut tôt* et *le plus tard possible*.

- Soit $\langle !A, R, v\rangle$ un autre tuple d'une séquence de test, contenant un symbole de sortie $!a$. Le testeur attend la réception du symbole A depuis l'implantation pour une valeur de R.

- Autrement, pour un tuple $\langle \delta, R, v\rangle$, modélisant un laps de temps permettant une région d'horloges depuis une autre, le testeur calcule à la volée la valeur de ce laps de temps et attend. Ce calcul permet aussi de vérifier que la région d'horloges à atteindre est accessible.

De même que précédemment, une même séquence de test est traduite en **Arbre d'Exécution**.

Arbre d'Exécution d'une séquence de test

Chaque séquence de test *stest* est donc traduite en Arbre d'exécution tel que chaque branche partant du noeud racine représente une exécution de *stest*. Cet arbre est composé de noeuds et de branches étiquetées par :

- **send(A)** représentant l'envoi du symbole "A" vers l'implantation par le testeur,

- **recv(A,R,v)** représentant l'attente de la réception du symbole "A" par le testeur à une valuation d'horloges de R. Si l'étiquette de verdict v est égale à $pass$ alors "A" doit être reçu pour que l'implantation soit conforme à la spécification. Si celle-ci est égale à $fail$ alors le symbole ne doit pas être reçu à une valuation d'horloges de R. Cette distinction est due aux séquences d'actions de l'ensemble TW qui peuvent être acceptantes ou refusantes,

- $wait$ modélisant le laps de temps nécessaire pour que les horloges atteignent la valuation d'horloges v_{final} depuis v_{init}, donc $wait = v_{final} - v_{init}$. Ce laps de temps doit être calculé à la volée car il dépend de l'exécution (voir Figure 5.40),

- δ modélisant le laps de temps nécessaire pour atteindre une région d'horloges depuis une valuation d'horloges v_{init}.

L'algorithme suivant transforme une séquence de test en un arbre d'exécution. Pour chaque tuple de la séquence de test contenant un symbole de sortie, nous avons une branche créée contenant le symbole, la région d'horloges rassemblant toutes les valuations d'horloges pour lesquelles ce symbole doit être reçu et un verdict indiquant si ce symbole doit ou ne doit pas être reçu. Comme pour la méthodologie précédente, deux branches sont créées si le symbole est un symbole d'entrée.

Algorithme

Construction d'Arbre d'exécution
Entrée : Séquence de test Seq
Sortie : Arbre d'exécution de Seq
DEBUT :
Const-Arbre(Seq,$\langle A, R, v\rangle$)

SI A=δ %Laps de temps δ
Alors { Ajouter une branche $\xrightarrow{\delta}$
Cons-Arbre(SEQ, prochain tuple de SEQ)

Si A est un Symbole d'entrée
Si $\exists v_1 \in R, v_2 \in R$, tel que $v_1 \neq v_2$
Alors {
Ajouter une branche $\xrightarrow{wait}\xrightarrow[Rt]{send(I)}$, avec
Rt l'ensemble des horloges réinitialisées
Cons-Arbre(Seq, prochain tuple de SEQ)
Ajouter une seconde branche $\xrightarrow[Rt]{send(I)}$, avec
Rt l'ensemble des horloges réinitialisées
Cons-Arbre(Seq, prochain tuple de Seq)
Sinon {
Ajouter une branche $\xrightarrow[Rt]{send(I)}$, avec
Rt l'ensemble des horloges réinitialisées
Cons-Arbre(Seq, prochain tuple de SEQ)

Si A est un symbole de sortie
Alors {
Ajouter une branche $\xrightarrow[Rt]{recv(O,R,v)}$, avec
Rt l'ensemble des horloges réinitialisées
Cons-Arbre(Seq, prochain tuple de Seq)
FIN

Le testeur connaît les symboles attendus et les symboles à émettre, les moments durant lesquels ils doivent être émis et reçus, une étiquette de verdict montrant si les symboles de sortie doivent être reçus ou non (séquence acceptante ou refusante), et les moments pendant lesquels il devra attendre. Les laps de temps $wait$ et δ sont obtenus à la volée par les calculs décrits en section 4.3.4 et 4.3.4 respectivement.

Prenons l'exemple suivant de séquence de test : $\langle ?b, R_1, inconcluant\rangle\langle !e, R_1, pass\rangle$ $\langle ?a, R_1, inconcluant\rangle\langle !e, R_1, pass\rangle$. Celle-ci donne l'arbre d'exécution de la Figure 5.45

5.2.5 Verdict du test sur la conformité de l'implantation

En appliquant l'ensemble des séquences de test et en observant les réactions dans le temps de l'implantation I, un verdict sur la conformité de celle-ci par rapport à la spécification S peut être donné. Pour une action temporisée $\langle A, R\rangle$, nous pouvons obtenir plusieurs possibilités de vervict, notées $Obs(\langle A, R\rangle, I)$, et conclure par $Obs(\langle A, R\rangle, I) =$

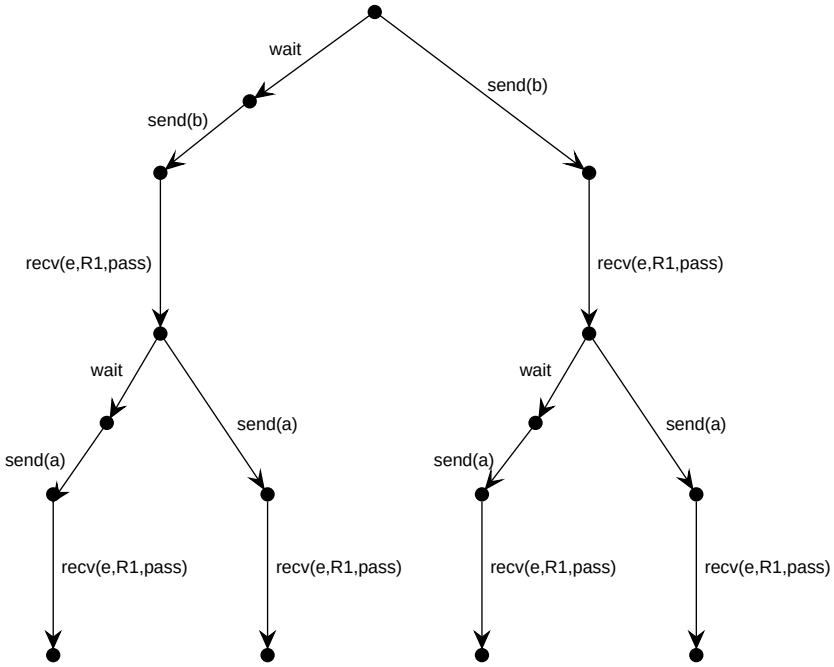


FIG. 5.45 – Arbre d'exécution

- Pass si A est un symbole de sortie et A est reçu par le testeur à une valuation d'horloges $v \in R$,
- FAIL si A est un symbole de sortie et A n'est pas reçu ou si A est reçu à une valuation d'horloges $v \notin R$,
- FAIL si le symbole de sortie $B \neq A$ est reçu par le testeur,
- Inconcluant si A est un symbole d'entrée ou δ.

En observant ces réactions et en comparant celles-ci avec l'étiquette de verdict de chaque tuple des séquences de test, nous pouvons conclure sur le succès ou l'échec du test, c'est à dire sur la conformité de l'implantation.

Définition 5.2.10 (Verdict) *Soit SEQ l'ensemble des séquences de test générées par les étapes de la section 5.2.3. $SEQ = \prod_1 \cup \prod_2$.*

Soit aussi I une implantation sous test et $stest = \langle A_1, R_1, l_1\rangle ... \langle A_n, R_n, l_n\rangle$ une séquence de test de SEQ. L'attribution du verdict de l'application de $stest$ sur I, noté $V(I, stest)$, est donné par :

$$V(I, stest) = \begin{cases} PASS \text{ si } \forall \langle A_i, R_i, l_i\rangle \in stest, Obs(\langle A_i, R_i\rangle, I) = l_i \\ FAIL \text{ autrement} \end{cases}$$

Finalement, le Verdict sur la conformité de I par rapport à la spécification S, noté $V(I, S)$, est donnée par :

$$V(I,S) = \begin{cases} PASS \text{ si } \forall stest \in Seq, V(I, stest) = PASS \\ FAIL \text{ autrement} \end{cases}$$

5.2.6 Complexité de la Méthodologie

Les points suivants analysent la complexité de chaque étape de la méthodologie.

- **Génération des ensembles P et Q :** P est généré en construisant premièrement un arbre de test et en énumérant les chemins partiels de l'arbre (voir [Cho78]). La complexité à générer P est proportionnelle au produit du nombre d'états $S_{\mathcal{RA}}$ et du nombre de transitions $K_{\mathcal{RA}}$ du graphe des régions $\mathcal{RA}$. Si nous considérons le pire cas, c'est-à-dire un graphe des régions minimal $\mathcal{RA}$ donc le nombre d'états est égal à celui du graphe des régions non minimal équivalent à $\mathcal{RA}$, alors $S_{\mathcal{RA}} = S * 2^{\delta(\mathcal{A})}$ et $K_{\mathcal{RA}} = S * K * 2^{\delta(\mathcal{A})}$, avec S le nombre d'états de la spécification et K le nombre de ses transitions ([AD94]). La complexité de la génération de Q est proportionnelle au nombre d'états de $\mathcal{RA}$, soit $S * 2^{\delta(\mathcal{A})}$.

- **Génération de l'ensemble TW :** Nous n'avons pas la complexité exacte de la génération de TW car elle dépend essentiellement du nombre de P_k tables construites. Si $N = card(S_{reduit})$ est le nombre d'états de $\mathcal{RA}$ à caractériser et $E = card(E_{reduit})$ est le nombre de transitions de $\mathcal{RA}$ prises en compte par la méthodologie, alors la complexité à générer TW est approximativement proportionnelle à $N^2 * E$.

  En effet, pour un état et une action temporisée de $TIMSEQ_{\mathcal{RA}}$, nous obtenons une entrée dans une P_k table. Par conséquent, la complexité à générer une P_k table est proportionnelle à $N * E$. Nous ne pouvons déterminer à l'avance le nombre de P_k tables qui seront construites, mais dans le pire des cas, nous pouvons obtenir $N - 1$ tables (voir [Gil62]) sur un automate minimal, ainsi la complexité à générer l'ensemble des P_k tables est , dans le pire des cas, proportionnelle à $N * (N-1) * E$, ou approximativement $N^2 * E$.

- **Génération des Arbres d'exécutions :** idem que dans la méthodologie précédente.

5.2.7 Une implantation erronée

Illustrons la détection de fautes potentielles sur une implantation erronée, et considérons par conséquent l'implantation de la Figure 5.46. Celle-ci contient trois erreurs : la première est une faute de transfert et concerne la transition $S_4 \xrightarrow{?A} S_1$, qui au lieu d'arriver sur l'état S_1, devrait arriver sur l'état S_2 . La deuxième erreur est une violation des contraintes de temps de la transition $S_1 \xrightarrow{?B} S_2$. En effet, la contrainte devrait être $x < 2$. La dernière erreur est l'ajout de la transition $S_4 \xrightarrow{!e} S_1$.

Prenons l'exemple de la séquence de test $\langle ?b, R_1, inconcluant\rangle\langle !e, R_1, pass\rangle$ $\langle ?a, R_1, inconcluant\rangle\langle !e, R_1, pass\rangle$. Son arbre d'exécution est illustré en Figure 5.45.

Selon la première branche de l'arbre d'exécution, le testeur émet "b", à une valuation

d'horloges égale à $(2-\epsilon, 2-\epsilon)$. L'implantation n'accepte pas le symbole et reste donc dans l'état initial, car ses contraintes d'horloges ne sont pas respectées. Le testeur attendant le symbole "e", n'observe donc aucune réaction. L'implantation est donc erronée.

Lors de l'exécution de la troisième branche, le testeur émet "b" à une valuation d'horloges égale à $(0, 0)$. Puis, il attend une réaction de la part de l'implantation sous test, qui est la réception du symbole "e" pour une valuation de la région R_1. La réaction étant observée à une valuation d'horloges V, le testeur émet aussitôt le symbole "a". Puis il attend le symbole "E" à une valuation d'horloges de R_1. Or l'implantation se trouve dans l'état S_1 donc il n'y a aucune réaction de sa part. Le testeur ne reçoit aucun symbole donc l'implantation est erronée.

Pour détecter la transition supplémentaire, prenons la séquence de test $\langle ?b, R_1,$ $inconcluant\rangle\langle !e, R_1, pass\rangle\langle !e, R_1, fail\rangle$. Le testeur émet "b" à une valuation d'horloges de R_1, puis reçoit le symbole "e" à une valuation d'horloges de R_1. Le testeur attend pour vérifier qu'il ne reçoit pas un autre symbole "e". Or, l'implantation émet ce symbole. Par conséquent, S_2 et S_4 ne peuvent être caractérisés et l'implantation est erronée.

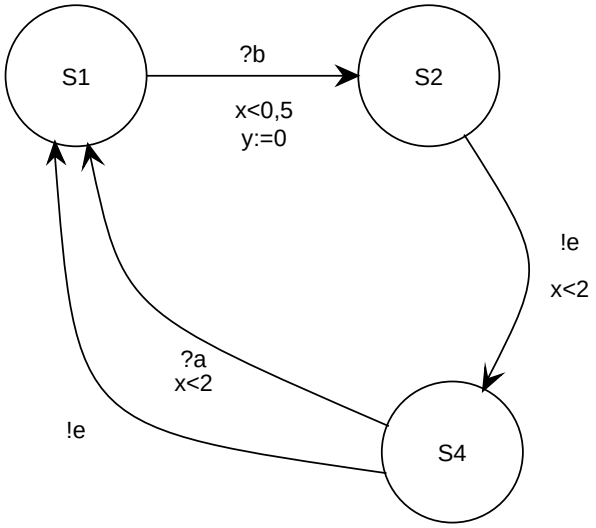


FIG. 5.46 – Une implantation erronée

5.2.8 Validité de la relation de conformité

Cette section a pour but de démontrer que la relation de conformité est satisfaite en appliquant sur l'implantation, les séquences de test dont la génération est décrite en section 5.2.3. A proprement parler, nous démontrons l'équivalence de traces entre l'implantation et le graphe des régions R_{reduit}.

R_{reduit} est le graphe des régions inclu dans le graphe des régions $\mathcal{RA}$, obtenu depuis la spécification, en ne prenant en compte que les états de S_{reduit}. R_{reduit} est le graphe des régions $R_{reduit} =< \Sigma, S, s^0, E >$ obtenu depuis $\mathcal{RA} =< \Sigma_{\mathcal{RA}}, S_{\mathcal{RA}}, s^0_{\mathcal{RA}}, E_{\mathcal{RA}} >$ tel que :

- $\Sigma = \Sigma_{\mathcal{RA}}$,
- $S = \{s \in S_{\mathcal{RA}} \mid \exists s \xrightarrow{A} s' \in E_{\mathcal{RA}}, A \neq \delta\}$.
- s^0 est l'état initial de S_{reduit}.

– $E = \{s \xrightarrow{A} s' \in E_{\mathcal{RA}} \mid s \in S\}$.

Pour prouver l'équivalence de trace, nous nous inspirons de la preuve de la validité des méthodes W et Wp [Cho78, FBK+91]. Le but de la démonstration est d'abord de prouver que S_{reduit} et I sont V-équivalents, avec V un ensemble de séquences d'actions temporisées.

Définition 5.2.11 (V-équivalence) *Soit S_i et I_j deux états respectifs de R_{reduit} et I et soit V un ensemble de séquences temporisées. S_i est dit V-équivalent à I_j, si les séquences acceptantes de S_i, appartenant à V, sont des séquences acceptantes de I_j et si les séquences refusantes de S_i, appartenant à V, sont des séquences refusantes de I_j. Autrement, S_i et I_j sont dit V-distinguables.*

Cette V-équivalence est elle-même démontrée en prouvant qu'il existe un isomorphisme de R_{reduit} vers I. Ainsi les six Propositions suivantes prouvent cette V-équivalence. Elles amèneront elles-mêmes à prouver l'équivalence de traces par une dernière Proposition.

Définition 5.2.12 *Soit S_i et I_j deux états respectifs de R_{reduit} et I. S_i est dit équivalent à I_j ssi S_i est V-équivalent à I_j pour tout ensemble V.*

Définition 5.2.13 *Deux graphes des régions R_{reduit} et I sont dit V-équivalents ssi S_0 et I_0 sont V-équivalents, avec S_0 et I_0 les états initiaux de S_{reduit} et I.*

Cette V-équivalence va être prouvée grâce à la prise en compte d'un isomorphisme de R_{reduit} vers I. Auparavant, donnons la définition de cet isomorphisme.

Définition 5.2.14 *Un isomorphisme de R_{reduit} vers I est une fonction f des états de R_{reduit} vers les états de I tel que : a) f est bijective b) si $S_i \xrightarrow{x} S_j \in E$ alors $f(S_i) \xrightarrow{x} f(S_j)$ à I. R_{reduit} est dit isomorphe à I.*

Définition 5.2.15 *Soit R_{reduit} le graphe des régions minimal et TW un ensemble de séquences temporisées. TW est dit ensemble de caractérisation d'états ssi TW peut distinguer toute paire d'états de R_{reduit}.*

Les propositions suivantes prouvent que R_{reduit} et I sont isomorphes. Ceci permettra de prouver la TW-équivalence entre R_{reduit} et I.

Proposition 5.2.6 *R_{reduit} est isomorphe à $I \Leftrightarrow$ V-équivalence est un isomorphisme de R_{reduit} vers I pour un V quelconque.*

Preuve:

$\Leftarrow$: Si V-équivalence est un isomorphisme alors R_{reduit} est isomorphe à I.
$\Rightarrow$: l'isomorphisme implique l'équivalence (résultat mathématique), ce qui implique la V-équivalence. ∎

Proposition 5.2.7 *TW-équivalence est un isomorphisme de R_{reduit} vers I ssi :*

1. *Pour tout S_i dans R_{reduit}, il existe I_j dans I tel que I_j est TW-équivalent à S_i.*

2. *Pour tout* $S_i \xrightarrow{x} S_j$ *dans* R_{reduit}*, il existe* I_k *et* I_l *dans* I *tel que* I_k *et* I_l *sont* TW*-équivalents à* S_i *et* S_j *respectivement et* $I_k \xrightarrow{x} I_l$ *dans* I*.*

Preuve:

D'après 1), pour chaque S_i de R_{reduit}, il existe au moins un état I_j de I tel que I_j est TW-équivalent à S_i. Comme R_{reduit} et I ont le même nombre d'états, et que TW distingue chaque paire d'états de R_{reduit}, il doit exister au plus un état dans I qui est TW-équivalent à un état de R_{reduit}. Donc TW-équivalence est injective.

TW-équivalence est aussi bijective car, d'après les hypothèses I a le même ensemble de symboles, et d'après 2) chaque état de I est TW-équivalent à au moins un état de R_{reduit}.

D'après 2), nous pouvons dire que $I_k = f(S_i)$ et $I_l = f(S_j)$ et nous obtenons donc la condition 2) de la définition de l'isomorphisme (Définition 5.2.14). Ainsi, TW-équivalence est une fonction f, qui est un isomorphisme de S_{reduit} vers I.

■

Proposition 5.2.8 *Supposons que pour tout* S_i *de* R_{reduit}*, il existe* I_j *de* I *tel que* I_j *est* TW*-équivalent à* S_i*. Alors* I_j *est* TW*-équivalent à* S_i *ssi* I_j *est* $TW_{(S_i)}$*-équivalent à* S_i*, ce qui peut encore se noter par :* I_j *est* TW*-équivalent à* $S_i \Leftrightarrow I_j$ *est* $TW_{(S_i)}$*-équivalent à* S_i*.*

Preuve: ($\Rightarrow$) D'après la proposition 5.2.5 $TW_{(S_i)} \subseteq TW$, donc $\Rightarrow$ est vérifié.

($\Leftarrow$) Supposons que la proposition soit fausse. Alors il existe S'_i de R_{reduit} tel que S'_i TW-équivalent à I_j (hypothèse). Nous avons aussi I_j est $TW_{(S_i)}$-équivalent à S_i et, d'après ($\Rightarrow$), S'_i est TW-équivalent à $I_j \Rightarrow S'_i$ est $TW_{(S_i)}$-équivalent à I_j, donc S_i est $TW_{(S_i)}$-équivalent à S'_i. Ce qui est en contradiction avec la définition de $TW_{(S_i)}$. Donc ($\Leftarrow$) est vérifié. ■

En utilisant la Proposition 5.2.8, la Proposition 5.2.7 peut être réécrite par :

Proposition 5.2.9 TW*-équivalence est un isomorphisme de* R_{reduit} *vers* $I \Leftrightarrow$

1. *Pour tout état* S_i *de* R_{reduit}*, il existe un état* I_j *de* I *tel que* I_j *est* TW*-équivalent à* S_i*, en particulier* I_0 *est* TW*-équivalent à* S_0*.*
2. *Si* $S_i \xrightarrow{x} S_j$ *appartient à* R_{reduit}*, alors il existe les états* I_k *et* I_l *de* I *tel que* I_k *est* $TW_{(S_i)}$*-équivalent à* S_i *et* I_l *est* $TW_{(S_j)}$*-équivalent à* S_j*, et* $I_k \xrightarrow{x} I_l$ *est dans* I*.*

Proposition 5.2.10 *Les conditions 1 et 2 de la Proposition 5.2.9 sont vraies si et seulement si* R_{reduit} *et* I *sont* $Q.TW$*-équivalents (premier ensemble de séquences de test) et* $R \otimes TW$*-équivalents (deuxième ensemble de séquences de test).*

Preuve:

R_{reduit} et I sont $Q.TW$-équivalents implique la condition 1) : nous déduisons de la définition de l'ensemble Q que pour tout $S_i \in R_{reduit}$ il existe $p_i \in Q$ tel que l'exécution de p_i permet d'atteindre S_i, depuis S_0. Comme I a le même alphabet que S, il existe I_k dans I tel que l'exécution de p_i permet d'atteindre I_k depuis I_0. Comme I_0 est $Q.TW$-équivalent à S_0, I_k est TW-équivalent à S_i. En prenant $\epsilon \in Q$, nous avons en particulier

S_0 TW-équivalent à I_0.

R_{reduit} et I sont $Q.TW$-équivalents et $R \otimes TW$-équivalents implique la condition 2) : la définition de l'ensemble de couverture de transition $P = Q \cup R$ implique que pour toute transition $S_i \xrightarrow{x} S_j$, il existe p_i et $p_i.x$ tel que l'exécution de p_i permet d'atteindre S_i depuis S_0 et tel que l'exécution de $p_i.x$ permet d'atteindre S_j depuis S_0. Comme I a le même alphabet que R_{reduit}, il existe I_k et I_l dans I atteints en appliquant respectivement p_i et $p_i.x$ à I_0.

Si $p_i \in Q$, comme par hypothèse I_0 est $\{p_i\}.TW$-équivalent à S_0, I_k est TW-équivalent à S_i et I_k est $TW_{(S_i)}$-équivalent à S_i, d'après la Proposition 5.2.8. De même, si $p_i.x \in Q$, nous avons I_l est TW_{S_j}-équivalent à S_j

Si $p_i \in R$, comme par hypothèse I_0 est $\{p_i\}.TW_{(S_i)}$-équivalent à S_0, I_k est $TW_{(S_i)}$-équivalent à S_i. De même, si $p_i.x \in R$, I_l est $TW_{(S_j)}$-équivalent à S_j.

Donc, I_k est $TW_{(S_i)}$-équivalent à S_i dans les deux cas, et I_l est $TW_{(S_j)}$-équivalent à S_j. Comme l'exécution de p_i permet d'atteindre I_k depuis I_0 et l'exécution de $p_i.x$ permet d'atteindre I_l depuis I_0 et I est déterministe, nous obtenons $I_k \xrightarrow{x} I_l$. Donc la deuxième condition est satisfaite. ∎

Finalement, depuis les Propositions 5.2.6, 5.2.9 et 5.2.10, nous déduisons directement le théorème suivant :

Théorème 5.2.1
R_{reduit} est équivalent à $I \Leftrightarrow R_{reduit}$ et I sont $Q.TW$-équivalents et $R \otimes TW$-équivalents.

De cette équivalence, nous pouvons enfin déduire l'équivalence de traces entre R_{reduit} et I.

Proposition 5.2.11 *Si R_{reduit} et I sont équivalents alors R_{reduit} et I sont équivalents en traces, noté $I \sim_{TimedTrace} R_{reduit}$.*

Preuve:
Supposons que R_{reduit} et I ne soient pas équivalents en traces. Alors il existe une séquence temporisée $\langle A, R\rangle$ tel qu'il existe S_i de R_{reduit} et I_j de I tel que S_i est $\{ \langle A, R\rangle \}$-distinguable à I_j. Soit $p_i \in TIMSEQ_{R_{reduit}}$ tel que S_i est atteint en exécutant p_i depuis S_0. Comme I est déterministe et a le même alphabet que S_{reduit}, p_i permet d'atteindre I_j depuis I_0.

Par hypothèse, R_{reduit} et I sont équivalents donc S_0 est $\{p_i.\langle A, R\rangle\}$-équivalent à I_0, d'après la Définition 5.2.13 et le théorème 5.2.1. Donc S_i est $\{\langle A, R\rangle\}$-équivalent à I_j, ce qui est en contradiction avec l'hypothèse. ∎

5.3 Transformation des Séquences de Test en TTCN

TTCN (Tree and Tabular Combined Notation) est un langage normalisé [ISO91a], qui représente les séquences de test en illustrant les symboles que le testeur doit émettre et recevoir, et qui est reconnu par la plupart des entreprises de test. La dernière version

TTCN-3 [Ga00, Gra00] prend en compte le langage ASN.1 et permet diverses représentations des séquences de test. De plus, la transformation en TTCN des séquences de test permet leurs utilisations directes par les industriels.

Rappel sur le langage TTCN

TTCN permet de décrire des séquences de test dites abstraites, c'est-à-dire exprimant principalement des interactions ou des primitives de services, sans détailler le codage de ces interactions ou primitives.

Deux formats pour TTCN ont été proposés : TTCN-GR (Graphic) qui modélise les séquences de test par tableaux et TTCN-MP (Machine Processable) qui traduit ces tableaux en un format accepté par les testeurs. Une expression TTCN-GR de séquence de test est composée de deux parties :

1. une partie déclarative contenant la déclaration des PCO (Point de Contrôle et d'Observation) à utiliser par le test, la déclaration des horloges et les Unités de Service et de Protocole.
2. une partie dynamique qui décrit la séquence de test, c'est-à-dire les envois et les réceptions de symboles, l'interrogation et la réinitialisation des horloges et les verdicts liés à chaque observation.

Prenons l'exemple de séquence de test de système non temporisé, $?a?b!e!c$ et le PCO P permettant l'émission et la réception des symboles depuis le testeur vers l'implantation. La séquence de test a été obtenue à partir de la spécification, or nous exécutons cette dernière depuis le testeur. Ainsi, l'attente de réception d'un symbole depuis l'implantation, doit être transformée en une émission et inversement. Donc les symboles commençant par " !" doivent maintenant commencer par " ?" et inversement. Nous obtenons donc l'expression TTCN suivante :

Description	Verdict	Commentaire
P !a	Inconcluant	
P !b		
P ?e	Pass	
P ? Otherwise	Fail	
P ?c	Pass	
P ? Otherwise	Fail	

TAB. 5.11 – Expression TTCN d'une séquence de test

TTCN est bien connu pour représenter des séquences de test générées depuis des systèmes non temporisés. Mais, TTCN regroupe aussi plusieurs mots-clés utiles pour générer des expressions TTCN contenant des propriétés temporelles. Parmi celles-ci, nous citerons :

- START <x> permet d'initialiser l'horloge x.
- READTIMER<x><n> lit la valeur de l'horloge x et la sauvegarde dans la variable n

- ?ELAPSE<nb> permet au testeur d'attendre pendant une période égale à nb
- CANCEL <x> désactive l'horloge x

D'autres commandes permettant d'exprimer des expressions temporelles ont été proposées dans [WG97] et dans [WG99]. Ces dernières n'ont cependant pas été normalisées, donc nous nous contenterons des mots-clés cités. Cependant, nous aurons besoin de trois macros permettant de calculer les laps de temps *wait* (section 5.1.4) et δ (section 5.1.4) et de vérifier qu'une valuation d'horloges appartient à une région d'horloges.

Transformation des séquences de test en TTCN

A partir des deux méthodologies de test présentées précédemment, nous obtenons des formats de séquences de test différents. Ainsi, nous proposons premièrement des expressions générales de transformation de séquences de test en TTCN, puis nous ajusterons ces expressions pour que nos deux formats de séquences de test puissent être transformés. Donc, dans un premier temps, considérons les séquences de test comme des séquences d'actions temporisées.
Dans la partie déclarative, nous supposons que les horloges du testeur sont correctement décrites. Celles-ci doivent aussi être initialisées, avant d'exécuter une séquence de test. Si les horloges du testeur sont $X_1, ..., X_N$, nous les initialisons par : $STARTX_1...STARTX_N$.
Nous utilisons, de plus, trois macros qui, comme pour les arbres d'exécutions, permettent de calculer les laps de temps rencontrés lors de la phase d'exécution du test :

1. la première, appelée $C_1(v, R)$ retourne la durée du laps de temps δ, nécessaire pour atteindre la prochaine région d'horloges R depuis une valuation d'horloges v (Calcul en section 4.3.4).

2. La deuxième, appelée $C_2(v, R)$ retourne les deux valuations d'horloges $V_{reduit} = (V_1, ..., V_N)$ et $V_{max} = (V'_1, ..., V'_N)$, qui sont la première et la dernière valuation d'horloges de la région d'horloges R, atteintes par les horloges lors de l'exécution, sachant que le tuple d'horloges est égal à la valuation d'horloges v (Calcul en section 4.3.4).

3. La troisième, appelée $C_3(v, R)$ retourne true si v est une valuation d'horloges de R et false sinon. Pour savoir si v appartient à R, il suffit de vérifier que v satisfait les inéquations de R.

Nous pouvons maintenant présenter les différentes expressions qui traduisent une séquence d'actions temporisées en TTCN :

Expression TTCN du laps de temps δ

Soit les deux actions temporisées consécutives $\langle \delta, R' \rangle \langle A, R \rangle$, avec $A \neq \delta$. La première action temporisée modélise un laps de temps permettant d'atteindre la région d'horloges R. Ce laps de temps est traduit en TTCN par :

Description	Verdict	Commentaire
READTIMER X_1 T_1 $\vdots$ READTIMER X_N T_N $C_1(< T_1, ...T_N >, R) \rightarrow duree$ $[duree > 0]$?ELAPSE *dure*		Calcul du laps de temps à la volée Atteindre R

TAB. 5.12 – Expression TTCN du laps de temps δ

Dans le cas où plusieurs laps de temps sont successifs (plusieurs actions temporisées contenant δ consécutives), nous considérons un seul laps de temps global. Ceci est possible car les horloges croissent d'une manière strictement croissante et de la même manière. De plus, aucune réinitialisation d'horloge n'est permise pendant un laps de temps.

Expression TTCN d'une action temporisée contenant un symbole d'entrée

Soit une action temporisée $\langle ?A, R\rangle$ contenant un symbole d'entrée $?A$. Tout comme lors de la construction des arbres d'exécutions, nous considérons que les symboles d'entrée sont émis, lors d'une première exécution, au voisinage de la première valuation d'horloges v_{init} de R atteinte par les horloges, puis qu'ils sont réémis, lors d'une deuxième exécution, au voisinage de la dernière valuation d'horloges v_{final} de R atteinte par les horloges. Ainsi, deux expressions TTCN différentes sont présentées, l'une pour la première exécution et l'autre pour la deuxième.

Voici l'expression TTCN remplaçant l'action temporisée $\langle ?A, R\rangle$ pour la première exécution :

Description	Verdict	Commentaire
!A	INCONCLUANT	émission de "A"
CANCEL X_1		réinitialiser x_1 si nécessaire
START X_1		
$\vdots$		
CANCEL X_n		réinitialiser x_n si nécessaire
START X_n		

TAB. 5.13 – Expression TTCN d'une action temporisée contenant un symbole d'entrée :1^{re} exécution

Pour la seconde exécution, un laps de temps *wait* peut être nécessaire pour atteindre la valuation d'horloges v_{final}. Celui-ci est obtenu grâce à la macro C_2.

Description	Verdict	Commentaire
READTIMER X_1 T_1 $\vdots$ READTIMER X_N T_N $C_2(< T_1, ..., T_N >, R) \rightarrow$ $(V1_{min}, ..., VN_{min})(V1_{max}, ..., VN_{max})$?ELAPSE $V1_{max} - T_1$!A CANCEL X_1 START X_1 $\vdots$ CANCEL X_n START X_n	INCONCLUANT	Atteindre v_{final} émission de "A" réinitialiser x_1 si nécessaire réinitialiser x_n si nécessaire

TAB. 5.14 – Expression TTCN d'une action temporisée contenant un symbole d'entrée :2^{me} exécution

Expression TTCN d'une action temporisée contenant un symbole de sortie

Soit une action temporisée $\langle !A, R\rangle$ contenant un symbole de sortie $!A$. Cette action temporisée modélise l'attente du symbole $!A$ par le testeur à une valuation d'horloges de R. La macro C_3 est utilisée pour vérifier que le symbole $!A$ est reçu dans la région d'horloges R.

L'expression TTCN de cette action temporisée est la suivante :

Description	Verdict	Commentaire
?A READTIMER X_1 T_1 $\vdots$ READTIMER X_N T_N $C_3(< T_1, ..., T_N >, R) \rightarrow Estdans$ $[Estdans == true]$ $[Estdans == false]$?otherwise CANCEL X_1 START X_1 $\vdots$ CANCEL X_n START X_n	 PASS FAIL Fail	réception de "A" "A" reçu dans R "A" pas reçu dans R Réception d'un symbole incorrect réinitialiser x_1 si nécessaire réinitialiser x_n si nécessaire

TAB. 5.15 – Expression TTCN d'une action temporisée contenant un symbole de sortie

Expression TTCN d'une séquence de test générée par la première méthodologie de test (section 5.1)

Par rapport aux séquences d'actions temporisées, dont nous venons de présenter la transformation en TTCN, les séquences de test obtenues grâce à la méthodologie de test basée sur les objectifs de test, ont une particularité qui est l'ajout d'une région Inconcluant. Ce dernier verdict informe que l'objectif de test n'a pas été satisfait, mais que pour autant, l'implantation est conforme à la spécification.

Ainsi, pour une transition $s \xrightarrow{?A,PASS(R),INCONCLUANT(R')} s'$, étiquetée par un symbole d'entrée $?A$, le testeur peut émettre $?A$ à des valuations de $R \cup R'$. Si $?A$ est émis à une valuation d'horloges de R, le verdict est PASS et le test peut continuer. Si $?A$ est émis à une valuation de R', alors le test ne peut respecter les propriétés temporelles de l'objectif de test, le verdict est donc inconcluant. Les deux expressions TTCN, illustrant les deux émissions de symboles, doivent donc prendre en compte ce verdict. Elles deviennent donc :

Description	Verdict	Commentaire
!A		émission de "A"
READTIMER X_1 T_1		
$\vdots$		
READTIMER X_N T_N		
$C3(< X_1, ..., X_N >, R) \rightarrow Estdans$		
$[Estdans == true]$	PASS	émission dans Pass
$[Estdans == false]$	INCONCLUANT	émission dans inconcluant
CANCEL X_1		réinitialiser x_1 si necéssaire
START X_1		
$\vdots$		
CANCEL X_n		réinitialiser x_n si necéssaire
START X_n		

TAB. 5.16 – Expression TTCN d'une action temporisée contenant un symbole d'entrée :1^{me} exécution

Description	Verdict	Commentaire
READTIMER X_1 T_1		
$\vdots$		
READTIMER X_N T_N		
$C_2(< T_1, ..., T_N >, R) \rightarrow$		
$(V1_{min}, ..., VN_{min})(V1_{max}, ..., VN_{max})$		
?ELAPSE $V1_{max} - T_1$		Atteindre v_{final}
!A		émisssion de "A"
$C3(< X_1, ..., X_N >, R) \rightarrow Estdans$		
$[Estdans == true]$	PASS	émission dans Pass
$[Estdans == false]$	INCONCLUANT	émission dans inconcluant
CANCEL X_1		réinitialiser x_1 si nécessaire
START X_1		
$\vdots$		
CANCEL X_n		réinitialiser x_n si nécessaire
START X_n		

TAB. 5.17 – Expression TTCN d'une action temporisée contenant un symbole d'entrée :2^{me} exécution

Les transitions $s \xrightarrow{!A, PASS(R), INCONCLUANT(R')} s'$, étiquetée par un symbole de sortie $!A$, peuvent être reçues soit dans la région PASS soit dans la région INCONCLUANT. Et à un type de région, un type de de verdict ! L'expression TTCN est donc la suivante :

Description	Verdict	Commentaire
?A		réception de "A"
READTIMER X_1 T_1		
$\vdots$		
READTIMER X_N T_N		
$C_3(< T_1, ..., T_N >, R) \rightarrow Estdans$		
$[Estdans == true]$	PASS	"A" reçu dans R
$C_3(< T_1, ..., T_N >, R') \rightarrow Estdans2$		
$[Estdans2 == true]$	INCONCLUANT	"A" reçu dans R'
$[Estdans == false$ AND		
$Estdans2 == false]$	FAIL	"A" pas reçu dans $R \cup R'$
?otherwise	FAIL	Réception d'un symbole incorrect
CANCEL X_1		réinitialiser x_1 si nécessaire
START X_1		
$\vdots$		
CANCEL X_n		réinitialiser x_n si nécessaire
START X_n		

TAB. 5.18 – Expression TTCN d'une action temporisée contenant un symbole de sortie

L'expression des laps de temps n'est, quant à elle, pas modifiée.

Expression TTCN d'une séquence de test générée par la deuxième méthodologie de test (section 5.2)

Cette fois, les séquences de test, obtenues grâce à la méthodologie de test basée sur la caractérisation d'états, sont très proches des séquences d'actions temporisées. En effet, pour un tuple $\langle A, R, l\rangle$ d'une séquence de test, la seule différence est la prise en compte de l'étiquette de verdict l. Celle-ci est d'ailleurs uniquement modifiée et donc importante lors de la réception de symboles de sortie par le testeur. Ainsi, en prenant en compte cette étiquette l, nous obtenons les verdicts suivants : si $l = pass$, alors le verdict que doit rendre le testeur est le verdict classique, à savoir : pass si le symbole est reçu à une valuation d'horloges de R et fail autrement. Si $l = fail$, l'action temporisée $\langle A, R\rangle$ ne doit pas pouvoir se vérifiée sur l'implantation : donc, le verdict est l'inverse de ce qui précède, à savoir : fail si le symbole est reçu à une valuation d'horloges de R et pass autrement.

l'expression TTCN d'une réception de symbole devient :

Description	Verdict	Commentaire
?A		réception de "A"
READTIMER X_1 T_1		
$\vdots$		
READTIMER X_N T_N		
$C_3(< T_1, ..., T_N >, R) \rightarrow Estdans$		
$[Estdans == true$ AND $l == pass]$	PASS	"A" reçu et attendu dans R
$[Estdans == false$ AND $l == false]$	PASS	"A" pas reçu et pas attendu dans R
$[Estdans == false$ AND $l == true]$	FAIL	"A" pas reçu et attendu dans R
$[Estdans == true$ AND $l == false]$	FAIL	"A" reçu et pas attendu dans R
?otherwise		
$[l == false]$	PASS	"A" pas attendu pas reçu
$[l == true]$	FAIL	"A" attendu et pas reçu
CANCEL X_1		réinitialiser x_1 si nécessaire
START X_1		
$\vdots$		
CANCEL X_n		réinitialiser x_n si nécessaire
START X_n		

TAB. 5.19 – Expression TTCN d'une action temporisée contenant un symbole de sortie

Les expressions TTCN de laps de temps et d'émission de symboles ne sont pas modifiées.

Chapitre 6

Une approche sur la minimisation des automates temporisés en graphes des régions

L'une des étapes, souvent indispensable avant la phase de validation, est la minimisation du modèle représentant le système, car en effet la plupart des méthodes de test des systèmes temporisés ou non, posent l'hypothèse de minimalité sur le système qu'elles sont censées traiter.

Le passage direct en graphe des régions peut engendrer un nombre d'états exponentiel. Ce nombre d'états augmente considérablement le coût total du test, et d'autre part réduit l'engouement de ces méthodes. Cette minimisation peut être faite par quelques algorithmes, dont les plus utilisés sont ceux décrits dans [ACH+92, YL93]. Dans ce qui suit, nous montrerons que ces algorithmes apportent quelques inconvénients pour la phase de test.

Nous présentons ainsi un nouvel algorithme, permettant de générer un graphe des régions depuis un automate temporisé, tel que chaque région d'horloges soit accessible et

tel que chaque valuation d'horloges de ces régions d'horloges soit aussi accessible.

Dans ce qui suit, nous rappelons, dans une première section, le fonctionnement des deux algorithmes décrits dans [ACH+92] et [YL93]. Puis, nous décrivons une nouvelle approche de minimisation et les phases qui composent notre algorithme. Nous détaillons dans la section suivante chacune de ces phases, c'est-à-dire les algorithmes détaillés et les fonctions utilisées. Enfin, nous prouvons que le graphe obtenu est bien un graphe des régions, qu'il est sémantiquement équivalent à l'automate temporisé d'origine, et que son nombre d'états est réduit par rapport à celui du graphe des régions défini dans [AD94].

6.1 Quelques algorithmes de minimisation

Cette section a pour but de présenter les algorithmes de minimisation décrits dans [ACH+92, YL93]. Leur étude permet d'illustrer leurs inconvénients et de proposer un nouvel algorithme de minimisation.

6.1.1 Algorithme décrit dans [ACH+92]

Le but de cet algorithme consiste à générer un graphe des régions minimal sur les états en réunissant les valeurs de temps en zones. Une zone est un système d'inéquations sur les horloges pouvant être représenté par un polyhèdre. A partir de la zone $Z = \mathbb{R}^n$ (avec n le nombre d'horloges) , l'algorithme découpe Z d'une manière successive dans le but d'obtenir plusieurs zones stables.

Plus précisément, une zone est dite stable si les deux conditions suivantes sont satisfaites :

- respect de l'écoulement de temps : s'il existe une valuation d'horloges d'une zone Z permettant d'atteindre une zone Z' par un écoulement de temps, alors toute valuation d'horloges de Z peut atteindre Z'.
- respect des transitions explicites : si une valuation d'horloges de Z peut atteindre une zone Z' par le franchissement d'une transition t alors toute valuation de Z peut atteindre Z' par le franchissement d'une transition t'.

Ainsi, l'algorithme procède comme suit : il parcours une à une les transitions de l'automate temporisé. Pour une transition t, il découpe la zone Z courante (zone dans laquelle les horloges prennent une valeur avant le franchissement de t) suivant les contraintes temporelles de t pour obtenir deux zones stables. Ce découpage peut entraîner l'instabilité de zones qui étaient stables auparavant. Dans ce cas, l'algorithme retourne sur ces zones et les découpe. L'algorithme se termine lorsque toutes les zones sont stables et que toutes les transitions sont franchies.

6.1.2 Algorithme décrit dans [YL93]

L'algorithme de [YL93] permet minimiser les systèmes temporisés modélisés par des systèmes de transitions. Ce modèle est proche de la représentation en graphe des régions à la différence que les états sont des couples composés par un état de l'automate temporisé initial et d'une valuation d'horloges. L'ensemble des états étant infini, ce modèle est de plus constitué par une partition initiale π réunissant certains de ces états en des blocs.

Ainsi, à partir d'un système de transition Q et d'une partition π, cet algorithme génère le système minimal Q' contenant une partition π' minimale, en rendant stable chaque bloc de π. Un bloc B est dit stable ssi chaque état de B peut atteindre un état d'un même bloc C par un transition ou un écoulement de temps.

Pour ce faire, l'algorithme commence par marquer l'état d'horloges $(s0, v0)$, avec $v0 = (0, ..., 0)$, et le bloc initial $B0$. Au lieu de stabiliser $B0$, il recherche et marque les blocs accessibles à partir de $(s0, v0)$ soit par une transition soit par un écoulement de temps. Une fois tous les blocs marqués, s'ils sont stables le système de transition obtenu est minimisé. Dans le cas contraire, chaque bloc B instable est découpé en B' et B'', avec B' stable. Après ce découpage, un bloc C atteignant B' peut être à son tour instable. L'algorithme se termine lorsque tout bloc est stable.

6.1.3 Discussion

Le premier algorithme de la section précédente génère, à partir d'un automate temporisé, un graphe des régions minimal sur l'ensemble de ses états. Celui-ci peut engendrer plusieurs retours successifs sur des zones pour les rendre stable. Il utilise de plus des opérateurs coûteux tel que la recherche une zone prédécesseur à une autre. Ainsi, le coût de cet algorithme est important.

Le deuxième algorithme permet d'obtenir une partition minimale contenant un ensemble de valuations d'horloges minimal, dans un temps polynômial (sur l'ensemble des symboles et des contraintes de l'automate temporisé). Cet algorithme permet ainsi de solutionner le problème soulevé ci-dessus. Cependant, cet algorithme ne donne pas comme résultat un graphe des régions mais une partition des états du système de transition initial. Ainsi, aucun graphe n'est construit.

Ainsi, il nous semble intéressant de proposer un nouvel algorithme de généreration de graphe des régions minimal, demandant un coût plus faible que celui de [ACH+92]. Ceci peut être fait en évitant des retours successifs sur les zones pour les stabiliser et en utilisant uniquement les opérateurs d'intersection et d'union sur les zones.

L'algorithme suivant propose une approche à cette minimisation. Une thèse étant limitée dans le temps, nous ne proposons pas de comparaison avec [ACH+92] et [YL93]. Une évaluation se fera dans un futur proche.

6.2 Une nouvelle approche de minimisation des automates temporisés en graphe des régions

L'algorithme consiste, dans un premier temps, à générer les régions d'horloges réunissant les valuations d'horloges qui satisfont le franchissement des transitions de l'automate temporisé initial. La construction d'une région s'effectue en simulant l'évolution des horloges à partir d'une valuation d'horloges ou d'une région d'horloges. Ceci donne une région temporaire composée de valeurs accessibles par les horloges, qui peut être représentée par un système d'inéquations constitué de contraintes temporelles de AT et de nouvelles inéquations inhérente à l'évolution des horloges. Cette région temporaire est ensuite découpée suivant toutes les contraintes d'horloges de l'automate. Ceci évite qu'une région d'horloges soit par la suite découpée par une autre contrainte de AT.

Après avoir générer ces régions d'horloges, nous construisons le graphe des régions. Dans cette phase, nous considérons de plus les écoulements de temps permettant de passer d'une région d'horloges à une autre. Ainsi, nous ajoutons des régions d'horloges supplémentaires modélisant ces écoulements de temps.

En détaillant, les phases 1 et 2 sont consacrées à construire des régions d'horloges. Un modèle intermédiaire, appelé l'automate des régions, stockera les régions d'horloges générées. La phase 3 construit un premier graphe des régions en accord avec la définition d'Alur et Dill. Celui-ci sera déjà réduit par rapport au graphe des régions initial des mêmes auteurs. Certaines régions d'horloges précédemment obtenues, ayant été découpées par toutes les contraintes de l'automate temporisé, peuvent être réunies sans changer la sémantique du système. Donc, nous avons introduit une phase 4, qui permettra de réunir ces régions d'horloges qui satisfont l'exécution d'une même action, en accord avec la Proposition 4.6.1 décrites dans la section 4.6 (Proposition visant à améliorer l'accessibilité temporelle des régions d'horloges).

L'algorithme de génération de graphe des régions se résume donc par les phases suivantes :

- **Phase 1 :** Recherche des contraintes d'horloges qui seront utilisées lors de la création et du découpage des régions d'horloges dans la phase suivante,

- **Phase 2 :** Construction des régions d'horloges satisfaisant toutes les contraintes d'horloges de l'automate temporisé. Génération d'un modèle intermédiaire, décrit en section 6.4.2 appelé automate des régions, stockant ces régions d'horloges,

- **Phase 3 :** Génération d'un premier graphe des régions réduit,

- **Phase 4 :** Réduction de ce graphe des régions grâce à la Proposition 4.6.1 décrite en section 4.6.

6.3 Découpage de régions d'horloges

Le fait de découper une région d'horloges R consiste à ajouter une contrainte d'horloges Ct au système d'inéquations de R tel qu'il existe une ou plusieurs valuations d'horloges de R qui satisfont Ct et tel que les autre valuations d'horloges ne la satisfont pas. Ainsi, au moins deux régions d'horloges $R_1 \subset R$ et $R_2 \subset R$, sont construites : toute valuation d'horloges de l'une satisfaisant Ct, toute valuation d'horloges de l'autre ne la satisfaisant pas.

Par exemple, la Figure 6.47 montre comment une région d'horloges R est découpée, en deux régions d'horloges, par la contrainte d'horloges $X > 3$. En effet, cette contrainte n'est pas toujours satisfaite : une région d'horloges $R_1 \subset R$ satisfait la contrainte, tandis que $R_2 \subset R$ ne la satisfait pas.

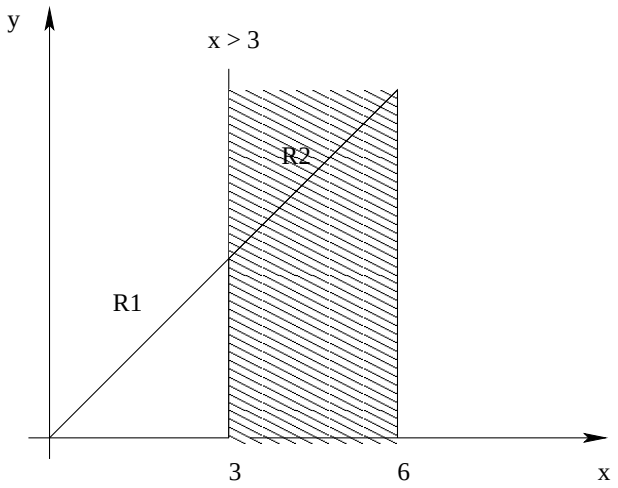


FIG. 6.47 – Découpage d'une région d'horloges

6.4 Algorithme de construction d'un graphe des régions réduit à partir d'un automate temporisé

Dans cette section, nous présentons les quatres phases de l'algorithme de minimisation d'un automate temporisé AT en un graphe des régions GR. Chaque phase est spécifique à une tâche précise et est indispensable à la génération du graphe des régions final.

6.4.1 Phase 1 : Ajout de contraintes d'horloges à l'automate temporisé

Cette phase est consacrée à la création et l'ajout de contraintes d'horloges à l'ensemble des contraintes d'horloges de l'automate temporisé. Ces contraintes, qui n'existent pas dans l'automate initial, sont pourtant susceptibles de découper certaines régions d'horloges lors de leurs constructions.

Prenons l'exemple de la Figure 6.48, illustrant des régions d'horloges. Supposons que les régions d'horloges R et $R1' \cup R2'$ soient obtenues par la transition $S_i \xrightarrow[x\leq 2\ y:=0]{A} S_j$, contenant un ensemble de réinitialisations d'horloges $\lambda = \{y\}$.

Supposons aussi que $R1' \cup R2'$ soit découpée par la contrainte $Ct = y \leq x - 1$. D'après la Figure, nous pouvons de suite observer que $R1'$ et $R2'$ ne peuvent être en même temps, successeur temporel à $R[\lambda \rightarrow 0]$, qui est la région d'horloges obtenue à partir de R en appliquant les réinitialisations d'horloges de λ. Donc R doit être découpée. En effet,

- pour $R1'$, soit par exemple les valuations d'horloges $(1,1)$ et $(0,0)$, appartenant à R. Après réinitialisation de y, ces valuations d'horloges donnent $(1,0)$ et $(0,0)$. Or $(1,0)$ appartient à $R1'$ et $(0,0)$ appartient à $R2'$. Donc $R1'$ ne peut être successeur à $R[\lambda \rightarrow 0]$.

- pour $R2'$, les valuations d'horloges $(1,1)$ et $(0,0)$, appartenant à R, donnent toujours, après réinitialisation de y, les valuations d'horloges $(1,0)$ et $(0,0)$. Or $(1,0)$

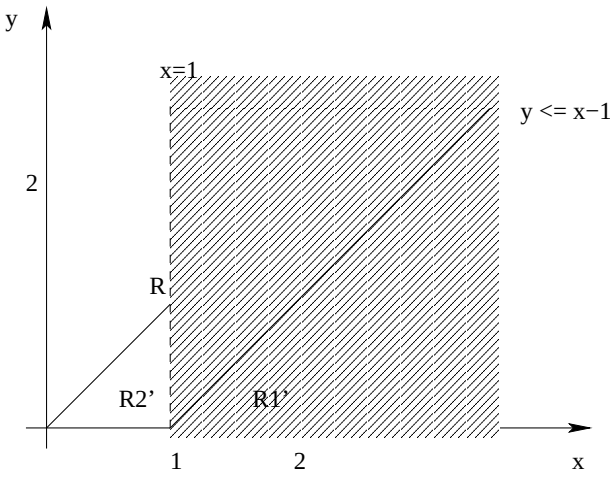


FIG. 6.48 –

appartient à $R1'$ et $(0,0)$ appartient à $R2'$. Donc $R2'$ ne peut être successeur à $R[\lambda \rightarrow 0]$.

En fait, R doit être découpée par la contrainte d'horloges Ct', construite depuis Ct en remplaçant les horloges de λ par 0. Dans notre exemple, Ct' est donc égale à $x \geq 1$, et découpe R en deux régions d'horloges R_1 et R_2, tel que $R1'$ est successeur temporel à l'une des deux régions d'horloges et $R2'$ est successeur temporel à l'autre. Ceci se généralise grâce à la Proposition suivante.

Proposition 6.4.1 *Soit $\mathcal{A} =< \Sigma_{\mathcal{A}}, S_{\mathcal{A}}, s^0_{\mathcal{A}}, C_{\mathcal{A}}, E_{\mathcal{A}} >$ un automate temporisé. Soient R et R' deux régions d'horloges telles que $R' = R[\lambda \rightarrow 0]$, avec λ un ensemble d'horloges réinitialisées.*
Supposons que la contrainte d'horloges $Ct = x_i \ op \ x_j + A(A \in \mathbb{R}, (x_i, x_j) \in C^2_{\mathcal{A}}, op \in \{\leq, \geq, >, <, =\})$ découpe R' en R'_1 et R'_2. Alors, R doit être découpée par
$Ct' = \begin{cases} Ct \\ \text{remplacer les horloges de } \lambda \text{ par } 0 \end{cases}$
et être remplacée par R_1 et R_2, pour que R'_1 soit successeur temporel de la région d'horloges R_1 ou R_2 et R'_2 soit successeur temporel de l'autre région d'horloges.

Preuve:

R' est découpée par Ct et donne les régions d'horloges R'_1 et R'_2. Supposons que la région d'horloges qui satisfait Ct est R'_1.
Ainsi, $\exists v'_1 \in R'_1$ tel que v'_1 satisfait Ct et $\exists v'_2 \in R'_2$ tel que v'_2 ne satisfait pas Ct. De plus, $R' = R[\lambda \rightarrow 0]$ Donc, $\exists v_1 \in R_1, v_2 \in R_2$ tel que $v_1[\lambda \rightarrow 0]$ satisfait Ct et $v_2[\lambda \rightarrow 0]$ ne satisfait pas Ct.
La contrainte Ct' est telle que chaque horloge $x_i \in \lambda$ peut prendre n'importe quelle valeur pour satisfaire Ct'. Par conséquent, nous pouvons affirmer que v_1 satisfait Ct et que v_2 ne satisfait pas Ct.

Premièrement, nous pouvons conclure que la région d'horloges R'_1 ou R'_2 ne peut être successeur temporel à R, car pour chacune de ces régions d'horloges il existe une valuation $v \in R$, telle que $v[\lambda \rightarrow 0]$ n'appartient pas soit à R'_1 soit à R'_2.

Par contre, si R est découpée par Ct' en R_1 et R_2, alors R'_1 est successeur temporel à R_1 et R'_2 est successeur temporel à R_2, car $\forall v \in R_1, v[\lambda \rightarrow 0] \in R'_1$ et $\forall v \in R_2, v[\lambda \rightarrow 0] \in R'_2$ ■

Pour obtenir un graphe des régions, les régions d'horloges, que nous construirons dans la phase suivante, doivent être successeurs temporels. De plus, nous découpons à l'avance les régions d'horloges suivant toutes les contraintes susceptibles de les découper. Donc, pour chaque contrainte d'horloges $Ct = x_i$ op $x_j + A(A \in \mathbb{R}, (x_i, x_j) \in C^2_{\mathcal{A}}, op \in \{\leq, \geq, >, <, =\})$ nous ajoutons aux transitions $S_i \xrightarrow{A} S_j$ de l'automate temporisé, ayant un ensemble de réinitialisation d'horloges $\lambda \neq \emptyset$, la contrainte Ct' obtenue en remplaçant dans Ct les horloges de λ par 0. Ceci est fait par l'algorithme suivant :

Algorithme

Génération de nouvelles contraintes d'horloges susceptibles de créer de nouvelles régions d'horloges
Entrée : Automate temporisé $\mathcal{A}$
Sortie : Automate temporisé $\mathcal{A}$
DEBUT :
Pour chaque transition $t = S_i \xrightarrow{A} S_j \in E_{\mathcal{A}}$, avec $\lambda \neq \emptyset$
  Pour chaque contrainte d'horloges de $\mathcal{A}$, $Ct = x_i op x_j + A$
  avec $A \in \mathbb{R}, (x_i, x_j) \in C^2_{\mathcal{A}}$, op $\in \{\leq, \geq, >, <, =\}$
  $Ct' = Ct$
  Pour chaque $x_i \in \lambda$
  $Ct' = Ct'$ en remplaçant x_i par 0
  Fin Pour
  Étiqueter Ct' sur t
  Fin Pour
Fin Pour
FIN

6.4.2 Phase 2 : Génération de l'automate des régions

Le but de cette phase de l'algorithme consiste, pour chaque transition d'un automate temporisé, à générer toutes les régions d'horloges qui satisfont les contraintes d'horloges de cette transition, c'est à dire à générer les régions d'horloges contenant les valeurs d'horloges qui mettent à vrai chaque contrainte d'horloges de la transition. Ces régions d'horloges seront ensuite étiquetées sur la transition. Le nouvel automate obtenu, appelé automate des régions, sera une étape intermédiaire de la transformation de l'automate temporisé en un graphe des régions réduit correspondant. Cet automate des régions est défini par :

Définition 6.4.1 *Soit un automate temporisé* $\mathcal{A} =< \Sigma_{\mathcal{A}}, S_{\mathcal{A}}, s^0_{\mathcal{A}}, C_{\mathcal{A}}, E_{\mathcal{A}} >$. *Un automate des régions AR de* $\mathcal{A}$ *est un tuple* $< \Sigma_{AR}, S_{AR}, s^0_{AR}, C_{AR}, E_{AR} >$ *tel que :*

- $\Sigma_{AR} = \Sigma_{\mathcal{A}}$,
- $S_{AR} = S_{\mathcal{A}}$,
- $s^0_{AR} = s^0_{\mathcal{A}}$,

- $C_{AR} = C_{\mathcal{A}}$
- E_{AR} *est l'ensemble des transitions tel que pour chaque transition* $< S_i, S_j, a, \lambda, Ct > \in E_{\mathcal{A}}$, *il existe une transition* $< (S_i, S_j, a, ENS - R, \lambda, Ct > \in E_{AR}$, *telle que pour chaque région d'horloges* R_k *de* $ENS - R_i$, R_k *satisfait les contraintes d'horloges de* Ct.

Non seulement cet automate nous est utile pour générer le graphe des régions, mais il comporte aussi une propriété intéressante sur l'accessibilité temporelle définie en section 4.2.3 qui influence la qualité de test. En effet, chaque ensemble de régions d'horloges, étiquetés sur une transition d'un automate des régions, est complètement accessible. Cette propriété montre que ce modèle peut être une bonne opportunité au test, car il permet d'obtenir le meilleur degré d'accessibilité temporelle (voir section 4.6) et permet donc d'obtenir une meilleure qualité de test, tout en gardant la même sémantique que l'automate temporisé et le même nombre d'états (Preuves de l'accessibilité et de l'équivalence sémantique donnée ci-après). Il est donc tout à fait possible, dans un proche avenir, que nous nous intéressions vivement à ce modèle pour proposer de nouvelles méthodologies de test.

L'algorithme de génération d'un automate temporisé en un automate des régions produit les étapes suivantes :

- La première étape de l'algorithme consiste à générer les régions d'horloges qui satisfont les contraintes d'horloges de chaque transition t sortante aux états initiaux. Ces régions d'horloges sont obtenues par la fonction *Calculer-R-init*. Chacune de ces régions d'horloges R est ensuite découpée, grâce à la fonction *Découper* par les contraintes d'horloges qui ne sont pas toujours satisfaites quelquesoit le tuple de valeurs de R. Les régions d'horloges obtenues sont ensuite étiquetées sur la transition t. Puis, t est marquée.

- L'étape suivante consiste, pour chaque transition $t = S_i \xrightarrow{A} S_j$ de l'automate temporisé précédée d'une transition marquée $S_k \xrightarrow[ENS-R]{A} S_i$, à générer les régions d'horloges qui satisfont les contraintes d'horloges de t. Pour ce faire, la fonction *Calculer-R-suivante* génère pour chaque région d'horloges Rp de $ENS - R$, la région d'horloges Rt successeur temporel de Rp contenant toutes les valuations d'horloges accessibles depuis celles de Rp. Puis elle ajoute au système d'inéquations de Rt les contraintes de la transition $S_i \xrightarrow{A} S_j$. Si la région d'horloges obtenue est non vide, alors celle-ci est encore découpée, grâce à la fonction *Découper*, par les contraintes d'horloges, qui ne sont pas toujours satisfaites quelquesoit le tuple de valeurs de Rt, et donne un ensemble de région d'horloges. Finalement, cet ensemble est étiqueté à la transition $S_i \xrightarrow{A} S_j$.

L'algorithme qui correspond à ces étapes est le suivant. Les fonctions et leurs descriptions plus détaillées sont décrites par la suite.

Algorithme

Génération de l'automate des régions
Entrée : Automate temporisé
Sortie : Automate des régions
DEBUT :
Pour chaque transition $S_i \xrightarrow{A_i} S_j$ non marquée telle que $S_i = S_0$ ou $\exists S_k \xrightarrow[ENS-R]{A_k} S_i \in E_{\mathcal{A}}$ marquée

Si $S_i = S_0$

%Première étape
R_{temp} = Calculer-R-init(contrainte $S_i \xrightarrow{A_i} S_j$)
$Ens - de - regions$ = Découper(R)

Sinon

%Deuxième étape
Pour chaque région d'horloges$R \in ENS - R$
R_{temp} = Calculer-R-suivante(contrainte $S_i \xrightarrow{A_i} S_j, R, \lambda$)
avec λ l'ensemble des horloges réinitialisées de $S_k \xrightarrow[ENS-R]{A_k} S_i$
Si $R_{temp} \neq \emptyset$ Alors
$Ens - de - regions = Ens - de - regions \cup$ Découper(R_{temp})
$FINPOUR$

Étiqueter $S_i \xrightarrow{A_i} S_j$ avec Ens-de-regions

Marquer $S_i \xrightarrow{A_i} S_j$

FINPOUR
FIN

Découpage des régions d'horloges par toutes les contraintes d'horloges de l'automate temporisé : Fonction Découper

Le but de la fonction *Découper* est de découper (qui ne l'aurait deviné) une région d'horloges R par toutes les contraintes d'horloges qui ne sont pas toujours satisfaites quelquesoit les valuations d'horloges de R. Ceci évite que la région d'horloges R soit découpée par la suite.

Cette fonction recherche, dans un premier temps, les contraintes d'horloges de l'automate temporisé susceptibles de découper R. Puis, pour chaque contrainte Ct, elle construit une région d'horloges temporaire Rt obtenue par l'union du système d'inéquations de R et de Ct. Elle vérifie ensuite si Rt découpe une des régions d'horloges, déjà générée, de l'ensemble $Ens - de - regions$. Si c'est le cas, elle ampute cette région d'horloges. Par la suite, elle découpe Rt avec toutes les régions d'horloges de $Ens - de - regions$. Au final, si Rt est non vide, Rt est une régions d'horloges de R qui ne se trouve pas encore dans $Ens - de - regions$. Donc, elle y est ajoutée.

Algorithme

Découper
Entrée : Région d'horloges R
Sortie : Ensemble de régions d'horloges Ens-de-regions
DEBUT :
Pour chaque horloge $x_i \in C_{\mathcal{A}}$

$$\begin{cases} \text{recherche des contraintes d'horloges qui découpent } R \\ MIN = min\{X_i \mid (X_1, ..., X_n) \text{ est un sommet de } R\} \\ MAX = max\{X_i \mid (X_1, ..., X_n) \text{ est un sommet de } R\} \\ C_{x_i} = \{\text{contraintes } x_i \text{ op } A \mid A \in \mathbb{R}, MIN \leq A \leq MAX\} \\ \cup\{\text{contraintes } x_i \text{ op } x_j + A \mid A \in \mathbb{R}, (x_i x_j) \in C_{\mathcal{A}}^2\} \end{cases}$$

FINPOUR
Ens-de-régions= $\{R\}$
% découpage de R suivant les contraintes d'horloges précédentes
Pour chaque contrainte d'horloges $Ct \in \{C_{x_1}, ..., C_{x_n}\}$
$\Delta = \Delta_R \cup Ct$
R_{temp} est la région d'horloges obtenue depuis Δ
Pour chaque $R_i \in Ens - de - regions$
Si $R_i / R_{temp} \neq \emptyset$

$$\text{Alors} \begin{cases} Ens - de - regions = Ens - de - regions / R_i \\ ENS - R_i = R_i / R_{temp} \\ Ens - de - regions = Ens - de - regions \cup ENS - R_i \end{cases}$$

Pour chaque $R_i \in Ens - de - regions$
Si $R_{temp} / R_i \neq \emptyset$
Alors $\{ \; R_{temp} = R_{temp} / R_i$
Si $R_{temp} \neq \emptyset$
Alors $Ens - de - regions = Ens - de - regions \cup R_{temp}$
FINPOUR
FIN

Génération des régions d'horloges satisfaisant les contraintes d'horloges des transitions partant des états initiaux : Fonction Calculer-R-init

La fonction *Calculer-R-init* génère la région d'horloges obtenue par les contraintes d'horloges d'une transition $S_0 \xrightarrow{A} S_j$ avec S_0 un état initial de l'automate temporisé. Cette région d'horloges ne peut être illustrée que par un segment de droite comme le montre la Proposition suivante :

Proposition 6.4.2 *Soit R_{init} la région d'horloges obtenue par les contraintes d'horloges C_t d'une transition $S_0 \xrightarrow{A} S_j$ avec S_0 un état initial, d'un automate temporisé.*

- *Si les contraintes d'horloges sont toutes de la forme $x_i \leq A$ ou $x_i < A_k$ ($A_k \in \mathbb{R}^+$), alors R_{init} est la région d'horloges obtenue par $0 \leq x_1 = x_2 = ... = x_n$ op A et $A = min\{A_j \mid x_i \text{ op } A_j \in C_t\}$,*

- *Si les contraintes d'horloges sont toutes de la forme* $x_i \geq A_k$ *ou* $x_i > A_k$ *(*$A_k \in \mathbb{R}^+$*), alors* R_{init} *est la région d'horloges obtenue par* $x_1 = x_2 = ... = x_n$ *op* A *et* $A = max\{A_j \mid x_i \ op \ A_j \in C_t\}$,

- *Si il existe une contrainte d'horloges* $x_i = A_k$ *(*$A_k \in \mathbb{R}^+$*), alors* R_{init} *est la région d'horloges* $(A_k, A_k, ..., A_k)$.
- *Sinon* R_{init} *est la région d'horloges obtenue par les inéquations :*
  $x_1 = x_2 = ... = x_n$ *op1* A *et* $x_1 = x_2 = ... = x_n$ *op2* B*, avec* $A = min\{A_j \mid x_i \ op1 \ A_j \in C_t, \ op1 \ \in \{\leq, <\}\}$*, puis avec* $B = max\{B_j \mid x_i \ op2 \ B_j \in C_t, \ op2 \ \in \{\geq, >\}\}$

Preuve:

Les horloges sont initialisées dans les états initiaux d'un automate temporisé. Elles sont strictement croissantes et croissent d'une manière identique, donc elles prennent comme valeur celles de l'équation $x_1 = x_2 =, ..., = x_n$.

- Si les contraintes d'horloges sont toutes de la forme $x_i \leq A_k$ ou $x_i < A_k$, alors R_{init} rassemble les valuations d'horloges qui satisfont l'équation de droite $x_1 = x_2 =, ..., = x_n$ et qui satisfont toutes les contraintes de la transition. Pour satisfaire toutes ces contraintes, pour toute horloge x_j, R_{init} doit satisfaire les contraintes $x_j op A_k$ avec $op \in \{\leq, <\}$,et notamment xj op A avec $A = min\{A_j \mid x_i \text{ op } A_j \in C_t\}$. Comme R_{init} doit aussi satisfaire $x_1 = x_2 =, ..., = x_n$, nous obtenons que R_{init} est la région d'horloges telle que $0 leq x_1 = x_2 = ... = x_n$ op A et $A = min\{A_j \mid x_i \text{ op } A_j \in C_t\}$ quelquesoit l'horloge x_i.
- Si les contraintes d'horloges sont toutes de la forme $x_i \geq A_i$ ou $x_i > A_i$, alors R_{init} rassemble aussi les valuations d'horloges qui satisfont l'équation de droite $x_1 = x_2 =, ..., = x_n$ et toutes les contraintes de la transition. Pour satisfaire toutes ces contraintes, pour toute horloge x_j, R_{init} doit satisfaire les contraintes x_j op A_k avec $op \in \{\geq, >\}$,et notamment x_j op A avec $A = max\{A_j \mid x_i \text{ op } A_j \in C_t\}$. Comme R_{init} doit aussi satisfaire $x_1 = x_2 =, ..., = x_n$, nous obtenons que R_{init} est la région d'horloges telle que $x_1 = x_2 = ... = x_n$ op A et $A = max\{A_j \mid x_i \text{ op } A_j \in C_t\}$, quelquesoit l'horloge x_i.
- S'il existe une contrainte d'horloges $x_i = A_k$, alors R_{init} rassemble les valuations d'horloges qui satisfont l'équation de droite $x_1 = x_2 =, ..., = x_n$ et $x_i = A_k$. Donc, R_{init} est la valuation d'horloges $(A_k, A_k, ..., A_k)$.
- Ce cas est la combinaison des deux premiers cas précédents.

■

Le lecteur peut s'interroger, à juste titre, sur le fait que nous ne traitons pas les contraintes d'horloges du type x_i op $x_j + A$. Étiqueter une telle contrainte d'horloges sur une transition partant de l'état initial serait inutile. En effet, d'après la définition des automates temporisés, initialiser les horloges dans les états initiaux est une hypothèse forte. Donc, avant le passage de la première transition, les horloges ont toutes la même valeur. Ce type de contrainte d'horloges est donc toujours satisfaite.

Finalement, nous obtenons donc l'algorithme suivant :

Algorithme

Calculer-R-init

Entrée : Ensemble Ct de contraintes d'horloges d'une transition $S_0 \xrightarrow{A} S_i \in E_{\mathcal{A}}$
Sortie : Région d'horloges R
DEBUT :
SI pour chaque x_i op $A_k \in Ct$, $op \in \{\leq, <\}$, $A_j \in \mathbb{R}^+$
ALORS
$A = min\{A_j \mid x_i \text{ op } A_j \in C_t\}$
OP=op
R est construite par les inéquations $0 \leq x_1 = x_2 = ... = x_n$ OP A
SI pour chaque x_i op $A_k \in Ct$, $op \in \{\geq, >\}$, $A_j \in \mathbb{R}^+$
ALORS
$B = max\{A_j \mid x_i \text{ op } A_j \in C_t\}$
OP=op
R est construite par les inéquations $x_1 = x_2 = ... = x_n$ OP B
Si il existe x_i op1 $A_k \in Ct$, $op1 \in \{\leq, <\}$, $A_k \in \mathbb{R}^+$
et il existe x_j op2 $A_l \in Ct$, $op2 \in \{\geq, >\}$, $A_l \in \mathbb{R}^+$
ALORS
$B = max\{A_j \mid x_i \text{ op1 } A_j \in C_t, \text{ op1 } \in \{\geq, >\}\}$
OP1=op1
$A = min\{A_j \mid x_i \text{ op2 } A_j \in C_t, \text{ op2 } \in \{\leq, <\}\}$
R est construite par les inéquations $x_1 = x_2 = ... = x_n$ OP1 A et $x_1 = x_2 = ... = x_n$ OP2 B
FIN

Génération des régions d'horloges successeur temporel à une région d'horloges déjà existante : Fonction Calculer-R-suivante

La fonction *Calculer-R-suivante* génère la région d'horloges R qui est successeur temporelle à une autre région d'horloges Rp en prenant en compte les contraintes d'horloges d'une transition $S_i \xrightarrow{A} S_j$ de l'automate temporisé. Dans un premier temps est générée la région d'horloges Rt successeur temporelle à Rp sans prendre en compte ces contraintes d'horloges. Celle-ci réunit toutes les valeurs de temps qui peuvent être prises par les horloges si ces dernières ont au préalable été égales à une valeur de Rp. Puis les contraintes de la transition sont appliquées à Rt pour obtenir une région d'horloges successeur temporelle à Rp satisfaisant les contraintes de la transition. La génération de Rt, qui est l'idée principale de la fonction est obtenue en faisant l'intersection de plusieurs plans. Ceci est détaillé dans la Proposition suivante :

Proposition 6.4.3 *Soit une région d'horloges Rp et son ensemble de sommets Som. La région d'horloges successeur temporelle à Rp, sans la prise en compte de contraintes d'horloges, est une région d'horloges Rt construite par l'intersection de plans tels que :*
$\forall S = (S_1, ..., S_n) \in Som, \forall (x_{i-1}, x_i) \in C^2_{\mathcal{A}} (2 \leq i \leq n)$

- *S'il existe une valuation $v = (v_1, ..., v_n) \in Rp$ telle que $v(x_i) = S_i$ et $v(x_{i-1}) > S_{i-1}$ alors est construit le plan $x_{i+1} - x_i \leq S_{i+1} - S_i$, qui prend en compte les valuations d'horloges v' accessibles à partir de celles ayant une coordonnée $v(x_{i-1})'$ supérieure à celle du sommet S.*

- *S'il existe une valuation* $v = (v_1, ..., v_n) \in Rp$ *telle que* $v(x_i) = S_i$ *et* $v(x_{i-1}) < S_{i-1}$ *alors est construit le plan* $x_{i+1} - x_i \geq S_{i+1} - S_i$*, qui prend en compte les valuations d'horloges* v' *accessibles à partir de celles ayant une coordonnée* $v(x_{i-1})'$ *inférieure à celle du sommet* S*.*
- *S'il existe une valuation* $v = (v_1, ..., v_n) \in Rp$ *telle que* $v(x_{i-1}) = S_{i-1}$ *et* $v(x_i) > S_i$ *alors est construit le plan* $x_{i+1} - x_i \geq S_{i+1} - S_i$*, qui prend en compte les valuations d'horloges* v' *accessibles à partir de celles ayant une coordonnée* $v(x_i)'$ *supérieure à celle du sommet* S*.*
- *S'il existe une valuation* $v = (v_1, ..., v_n) \in Rp$ *telle que* $v(x_{i-1}) = S_{i-1}$ *et* $v(x_i) < S_i$ *alors est construit le plan* $x_{i+1} - x_i S_{i+1} - S_i$*, qui prend en compte les valuations d'horloges* v' *accessibles à partir de celles ayant une coordonnée* $v(x_i)'$ *inférieure à celle du sommet* S*.*
- *sinon, nous obtenons la droite* $x_{i+1} - x_i = S_{i+1} - S_i$

Preuve: Soit un sommet $S = (S_1, ..., S_n) \in Som$. Les horloges à partir de ce sommet, prennent comme valeurs celles de l'équation $x_{i-1} - x_i = S_{i-1} - S_i, (2 \leq i \leq n)$, car les horloges sont strictement croissantes et croissent d'une manière identique.
Soit maintenant un couple d'horloges $(x_{i-1}, x_i) \in C^2_{\mathcal{A}}$.

- 1^{er} cas : S'il existe une valuation $v = (v_1, ..., v_n) \in Rp$ telle que $v(x_i) = S_i$ et $v(x_{i-1}) > S_{i-1}$, cela signifie qu'il existe des valuations d'horloges v dont la coordonnée $v(x_{i-1})$ est plus grande que la coordonnée S_{i-1} du sommet S. Depuis ces valuations, les horloges prennent comme valeurs celles d'une équation de droite parallèle à celle partant du sommet S. Si nous considérons toutes ces valuations d'horloges v, nous obtenons donc le plan, $x_{i+1} - x_i \leq S_{i+1} - S_i$, supérieur à la droite partant du sommet S, qui prend en compte les valeurs accessibles depuis les valuations d'horloges v.

- 2^{me} cas : S'il existe une valuation $v = (v_1, ..., v_n) \in Rp$ telle que $v(x_i) = S_i$ et $v(x_{i-1}) < S_{i-1}$, cela signifie qu'il existe des valuations d'horloges v dont la coordonnée $v(x_{i-1})$ est plus petite que celle de la coordonnée S_{i-1} du sommet S. Depuis ces valuations, les horloges prennent comme valeurs celles d'une équation de droite parallèle à celle partant du sommet S. Si l'on considère toutes ces valuations d'horloges, nous obtenons donc le plan, $x_{i+1} - x_i \geq S_{i+1} - S_i$, inférieur à la droite partant du sommet S, qui prend en compte les valeurs accessibles depuis les valuations d'horloges v.

- Il en est de même pour les 3^{me} et 4^{me} cas.

- sinon, il n'existe pas de valuations d'horloges ayant des coordonnées supérieure ou inférieure à celles du sommet, donc nous obtenons la droite $x_{i+1} - x_i = S_{i+1} - S_i$.

L'intersection de tous les plans obtenus depuis les sommets de Som permet donc de prendre uniquement en compte les valuations d'horloges ayant des coordonnées comprises entre celles des sommets, donc de Rp, et d'obtenir uniquement des valuations d'horloges accessibles depuis chaque valuation d'horloges de Rp.

Donc, pour chaque valuation d'horloges $v \in Rp$, il existe $d \in \mathbb{R}^+$ tel que $v + d = v'$ et v' appartenant à l'intersection des plans, qui est la région d'horloges Rt. Cette dernière phrase est la définition de successeur temporel, et montre que Rt est successeur temporel à Rp. ■

L'algorithme correspondant à la proposition précédente est donc le suivant :

Algorithme

Calculer-R-suivante
Entrée : Ensemble Ct de contraintes d'horloges d'une transition $S_i \xrightarrow{A} S_j \in E_{\mathcal{A}}$,
région d'horloges de départ Rp, ensemble λ d'horloges réinitialisées avec la transition précédente
Sortie : Région d'horloges R
DEBUT :
Ens-Sommet=Sommet(Rp)
Pour chaque horloge $x_i \in \lambda$
  Pour chaque sommet $S = (X_1, ..., X_n) \in Ens - Sommet$
    $X_i = 0$
  FINPOUR
FINPOUR
$\Delta = Ct$
Pour chaque sommet $S = (X_1, ..., X_n) \in Ens - Sommet$
  Pour i=2 à n
  Choix :
  cas 1 : Si $\exists S' = (X'_1, ..., X'_n) \in Ens - Sommet, X_i = X'_i$ et $X'_{i-1} > X_{i-1}$ Alors $OP = \leq$
  cas 2 : Si $\exists S' = (X'_1, ..., X'_n) \in Ens - Sommet, X_i = X'_i$ et $X'_{i-1} < X_{i-1}$ Alors $OP = \geq$
  cas 3 : Si $\exists S' = (X'_1, ..., X'_n) \in Ens - Sommet, X_{i-1} = X'_{i-1}$ et $X'_i > X_i$ Alors $OP = \geq$
  cas 4 : Si $\exists S' = (X'_1, ..., X'_n) \in Ens - Sommet, X_{i-1} = X'_{i-1}$ et $X'_i < X_i$ Alors $OP = \leq$
  $Defaut : OP = " = "$
  Fin Choix
  $\Delta = \Delta \cup X_i - X_{i-1}$ OP $S_i - S_{i-1}$
Fin Pour
R est la région d'horloges composée par le système d'inéquations Δ
FIN

Notons que dans l'algorithme précédent, nous avons en premier lieu construit une région d'horloges Rp' telle que $Rp' = Rp[\lambda \rightarrow 0]$, c'est-à-dire que Rp' est la région d'horloges atteinte par les horloges en prenant en compte celles qui sont réinitialisées lors du passage de la transition $S_i \xrightarrow{A} S_j$. La région d'horloges Rt est donc successeur temporel à Rp' et non à Rp. Nous pouvons dire aussi que Rt est successeur temporel à $Rp[\lambda \rightarrow 0]$, ce qui signifie que Rt est successeur temporel à Rp en appliquant les réinitialisations d'horloges de λ.

Depuis l'algorithme, nous pouvons de plus déduire que la région d'horloges R résultat est successeur temporel à $Rp[\lambda \rightarrow 0]$, c'est-à-dire que R est successeur temporel à Rp en appliquant les réinitialisations d'horloges de λ. Soit donc la Proposition suivante :

Proposition 6.4.4 *La région d'horloges R obtenue à partir de l'algorithme précédent est successeur temporel à $Rp[\lambda \rightarrow 0]$*

Preuve:

Une première région d'horloges Rt est obtenue par l'algorithme. D'après la Proposition 6.4.3, celle-ci est successeur temporel à $Rp' = Rp[\lambda \rightarrow 0]$. Soit une quelconque contrainte d'horloges Ct.

1. Si la contrainte d'horloges Ct ne découpe pas Rt, $R = Rt$, et R est successeur temporel à Rp'.
2. Supposons maintenant que Ct découpe Rt et donne R. Supposons que R ne soit pas successeur temporel de Rp'. Alors, $\exists v \in Rp', \forall d \in \mathbb{R}^+, v + d \notin R$. Rt est successeur temporel à Rp', donc $\forall v' \in Rp, \exists d' \in \mathbb{R}^+, v' + d' \in Rt$. Comme $R \subset Rt$, nous en déduisons que $v + d$ ne satisfait pas Ct. Montrons qu'il y a contradiction.
   Dans la suite, nous considérons que Rp', soit satisfait une contrainte Ct, soit ne la satisfait pas. $Rp' = Rp[\lambda \rightarrow 0]$ et Rp ayant été découpée selon l'algorithme de génération d'automate des régions, par toutes les contraintes de l'automate temporisé, il ne peut exister des valuations d'horloges de Rp' qui satisfont Ct et d'autres qui ne satisfont pas Ct
   - Soit $Ct = x_i$ op A_i, avec $x_i \in C_{\mathcal{A}}$ et $op \in \{\leq, <\}$. Si Rp' satisfait Ct, $\forall v \in Rp'$, v satisfait Ct, donc $\exists d \in \mathbb{R}^+$ tel que $v + d$ satisfait Ct. Donc contradiction. Si Rp' ne satisfait pas Ct, alors $\forall v \in Rp'$, $v(x_i) > A_i$, donc $\forall d \in \mathbb{R}^+$, $v + d > A_i$, donc $v + d$ ne satisfait pas Ct. Cela signifie que toute valuation d'horloges accessibles depuis celles de Rp' ne satisfont pas Ct. Par conséquent, Rt n'est pas découpée, et contradiction.
   - Soit $Ct = x_i$ op A_i, avec $x_i \in C_{\mathcal{A}}$ et $op \in \{\geq, >\}$. Si Rp' satisfait Ct, $\forall v \in Rp', \forall d \in \mathbb{R}^+, v(x_i)+d$ op A_i est vraie , donc $v+d$ satisfait Ct. Donc contradiction. Si Rp' ne satisfait Ct, alors $\forall v \in Rp', \exists d \in \mathbb{R}^+, v(x_i)+d$ op A_i, donc $v+d$ satisfait Ct, et contradiction.
   - Soit $Ct = x_i$ op $x_j + A_i$, avec $(x_i, x_j) \in C^2_{\mathcal{A}}$,$op \in \{\geq, \leq, <, >\}$ et $A_i \in \mathbb{R}$. Si Rp' satisfait Ct, alors $\forall v \in Rp', v(x_i)$ op $v(x_j) + A_i$ est vraie. $\forall d \in \mathbb{R}^+$, $v(x_i + d)$ op $v(x_j + d) + A_i$ est vraie. Donc $v + d$ satisfait Ct, d'où une contradiction. Si Rp' ne satisfait pas Ct, alors $\forall v \in Rp', v(x_i)$ op $v(x_j) + A_i$ est fausse. $\forall d \in \mathbb{R}^+$, $v(x_i + d)$ op $v(x_j + d) + A_i$ est fausse. Cela signifie que toute valuation d'horloges accessibles depuis celles de Rp' ne satisfont pas Ct. Par conséquent, Rt n'est pas découpée, et contradiction.

   Donc, quelquesoit la contrainte d'horloges, nous avons une contradiction. Donc R est bien successeur temporel à $Rp' = Rp[\lambda \rightarrow 0]$.

■

Ainsi, quelquesoit les deux transitions successives $S_i \xrightarrow[ENS-R]{A_1} S_j \xrightarrow[ENS-R']{A_2} S_k$ d'un automate des régions, une région d'horloges R' de $ENS-R'$ est obligatoirement successeur temporel d'une région d'horloges $R_2 = R[\lambda \rightarrow 0]$, avec $R \in ENS - R$, R_2 la région d'horloges obtenue en réinitialisant les horloges de λ sur R, et λ, l'ensemble des horloges réinitialisées de la transition $S_i \xrightarrow[ENS-R]{A_1} S_j$.

D'où la Proposition suivante :

Proposition 6.4.5 *Soit un automate des régions* $AR =< \Sigma_{AR}, S_{AR}, s^0_{AR}, C_{AR}, E_{AR} >$, *et soit deux transitions successives* $S_i \xrightarrow[ENS-R]{A_1} S_j \xrightarrow[ENS-R']{A_2} S_k$. *Notons aussi* λ *les horloges réinitialisées lors du passage de la transition* $S_i \xrightarrow[ENS-R]{A_1} S_j$.
Pour chaque région d'horloges $R' \in ENS - R'$, *il existe au moins une région d'horloges* $R[\lambda \rightarrow 0]$ *telle que* R' *est successeur temporel à* $R[\lambda \rightarrow 0]$.

Preuve:

D'après l'algorithme, pour chaque transition $S_j \xrightarrow{A_2} S_k$, précédée d'une transition $S_i \xrightarrow[ENS-R]{A_1} S_j$, nous construisons grâce à la fonction *Calculer-R-suivante*, une région d'horloges R successeur temporel à une région d'horloges $Rp' = Rp[\lambda \rightarrow 0]$, avec $Rp \in ENS - R$. (Proposition 6.4.4). R est ensuite éventuellement découpée par une quelconque contrainte d'horloges Ct et donne Rs. La région Rs est aussi successeur temporel à R. Ceci se prouve toujours grâce à la Proposition 6.4.4, car dans cette Proposition, nous avons considéré qu'une région Rt successeur temporel à Rp', découpée par une contrainte d'horloges, donnait une région d'horloges R toujours successeur temporel à Rp'. La région d'horloges Rs est ensuite étiquetée dans $ENS - R'$. Donc, pour chaque région d'horloges $Rs \in ENS - R'$, il existe au moins une région $Rp' = Rp[\lambda \rightarrow 0]$ telle que Rs est successeur temporel à Rp'. ∎

Concluons cette introduction aux automates des régions, en montrant que chaque ensemble de régions d'horloges, étiquetées sur les transitions d'un automate des régions, est complètement accessible.

Proposition 6.4.6 *Soit un automate des régions déterministe*
$AR =< \Sigma_{AR}, S_{AR}, s^0_{AR}, C_{AR}, E_{AR} >$.

Pour chaque couple de transitions successives $S_i \xrightarrow[ENS-R]{A_1} S_j \xrightarrow[ENS-R']{A_2} S_k$, $ENS - R'$ *est complètement accessible depuis* $ENS - R[\lambda \rightarrow 0]$.

Preuve:

$ENS - R$ réunit les valeurs de temps qui satisfont l'exécution de l'action modélisée par la transition. Soit donc v une quelconque valuation d'horloges d'une région d'horloges $R \in ENS - R$ pendant laquelle l'action est exécutée. Depuis l'état S_i, AR étant déterministe, la transition t ne peut être passée que si l'action est exécutée. La Proposition 6.4.5 prouve qu'il existe une région d'horloges R' successeur temporel à $R[\lambda \rightarrow 0]$, avec λ l'ensemble des horloges réinitialisées lors du passage de la transition $S_i \xrightarrow[ENS-R]{A_1} S_j$. Donc après l'exécution de l'action, une valuation d'horloges $v' \in ENS - R'$ est atteinte. Par conséquent, quelquesoit l'exécution de l'action, une valuation de $ENS - R'$ est accessible, donc $ENS - R'$ est complètement accessible depuis $ENS - R$. ∎

6.4.3 Phase 3 : Génération du Graphe des Régions

Le but de cette phase de l'algorithme consiste à construire le graphe des régions GR à partir de l'automate des régions AR obtenu dans la phase précédente. Dans cette étape,

le graphe des régions est construit à partir des régions de AR, et pour l'instant aucune agrégation de régions d'horloges n'est faite.

L'algorithme de génération de GR est composé des étapes suivantes :

- La première étape génère, depuis une transition $S_i \xrightarrow[ENS-R]{A} S_j$, les états (S_i, R) du graphe des régions, avec $R \in ENS - R$, et les transitions partant de ces états étiquetée par le symbole A. La construction de ces transitions est obtenue par la fonction *Ajouter-transition*.

- La deuxième étape consiste à générer les transitions étiquetées par δ, modélisant un écoulement de temps, qui partent des états du graphe des régions construits dans la première phase. Pour ce faire, nous cherchons l'ensemble de toutes les régions d'horloges étiquetées avec S_i. Cet ensemble, noté $E - R$ est tel que $E - R = \{R \mid (S_i, R) \in S_{GR}\}$. Nous cherchons ensuite les régions d'horloges successeur temporel à la région R de $E - R$ ayant les plus petites valuations d'horloges, grâce à la fonction *Chercher-successeur*.

  Grâce à cette fonction, nous obtenons une liste $ListeR$ de régions d'horloges successeurs temporels à R et tel que pour un couple (R_k, R_{k+1}) de la liste, R_{k+1} est successeur temporel à R_k. *Chercher-successeur* ajoute aussi les régions d'horloges R_δ qui sont traversées pour passer d'une région d'horloges vers une autre, et qui n'ont pas été générées dans la Phase précédente. La liste obtenue est ordonnée de la région d'horloges ayant les plus petites valuations d'horloges à celle ayant les plus grandes. Puis pour un couple (R_k, R_{k+1}) de la liste, nous ajoutons finalement la transition $(S_i, R_k) \xrightarrow{\delta} (S_i, R_{k+1})$ qui montre que R_{k+1} est successeur temporel à R_k.

L'algorithme qui correspond à ces étapes est le suivant. Les fonctions et leurs descriptions plus détaillées sont décrites par la suite.

Algorithme

Génération du graphe des régions
Entrée : Automate des régions AR
Sortie : Graphe des régions GR
DEBUT :
Pour chaque transition $S_i \xrightarrow[ENS-R_i]{A_i} S_j$ non marquée telle que $S_i \in s^0_{AR}$
ou $\exists S_k \xrightarrow{A_k} S_i \in E_{AR}$ marquée
Pour chaque région d'horloges R étiquetée dans $ENS - R_i$
$\begin{cases} \text{Si } (S_i, R) \notin S_{GR} \text{ Alors Ajouter } (S_i, R) \text{ à } S_{GR} \\ \text{Ajouter-transition}((S_i, R), t) \end{cases}$
FinPour
$ENS - R = \{R \mid (S_i, R) \in S_{GR}\}$
TantQue $ENS - R \neq \emptyset$
$\begin{cases} R_{min} = min\{R \mid R \in ENS - R\} \\ etat = (S_i, R_{min}) \\ Liste - R = Chercher - successeur(R_{min}) \\ \text{Si } Liste - R = \{R_{min}\} \text{ Alors } ENS - R = ENS - R/R_{min} \\ \text{Pour chaque région } R_l \in Liste - R \\ \quad \begin{cases} \text{Si } R_l \in ENS - R \text{ Alors } ENS - R = ENS - R/R_l \\ \text{Si } (S_i, R_l) \notin S_{GR} \\ \text{Alors Ajouter}((S_i, R_l)) \text{ à } S_{GR} \\ \text{Ajouter } etat \xrightarrow{\delta} (S_i, R_l) \text{ à } E_{GR} \\ etat = (S_i, R_l) \end{cases} \end{cases}$
Fin TantQue
Marquer t
FinPour
FIN

Génération des transitions du graphe des régions, étiquetées par un symbole différent de δ : Fonction Ajouter-transition

Pour une transition $t = S_i \xrightarrow[ENS-R]{A} S_j$ de l'automate des régions AR, la fonction *Ajouter-transition* construit une transition $(S_i, R) \xrightarrow{A} (S_j, R')$, étiquetée par le symbole A et partant d'un état (S_i, R) du graphe des régions, donné à la fonction. D'après la définition du graphe des régions, si aucune réinitialisation n'est appliquée aux horloges lors du passage de la transition, alors $R' = R$. Dans le cas contraire, si l'ensemble des horloges réinitialisées λ est non vide, $R' = R[\lambda \rightarrow 0]$ est la région d'horloges accessible depuis R uniquement en réinitialisant les horloges de λ. Nous obtenons donc l'algorithme suivant :

Algorithme

Ajouter-transition
Entrée : transition $t = S_i \xrightarrow{A} S_j$, état (S_i, R)
Sortie : transition de GR
DEBUT :
Si t n'a pas de réinitialisation sur horloges ($\lambda = \emptyset$)

Alors
- Ajouter (S_j, R) à S_{GR}
- Ajouter $(S_i, R) \xrightarrow{A} (S_j, R)$ à E_{GR}

Sinon
- $ENS - S = \emptyset$
- Pour chaque Sommet S_k de R
- $ENS - S = ENS - S \cup S_k$ auquel les réinitialisations ont été appliquées
- *FinPour*
- Chercher la région R' ayant pour sommets ceux de $ENS - S$
- Ajouter (S_j, R') à S_{GR}
- Ajouter $(S_i, R) \xrightarrow{A} (S_j, R')$ à E_{GR}

FinSi
FIN

Génération des transitions étiquetées par δ symbolisant le passage d'un région d'horloges à une autre : Fonction Chercher-successeur

Lors de la génération du graphe des régions GR, pour construire les transitions étiquetées par δ, modélisant le passage d'une région d'horloges à une autre, nous utilisons la fonction *Chercher-successeur* qui donne la liste ordonnée des régions d'horloges successeur temporel à une région d'horloges R. Celle-ci est ordonnée de la région d'horloges ayant les plus petites valuations d'horloges à celle ayant les plus grandes valuations d'horloges.

Cette liste est construite en deux étapes. La première crée une liste ordonnée $ListeR2$ rassemblant les régions d'horloges existantes et successeur temporel à R. Pour tout couple (R_k, R_{k+1}) de $ListeR2$, la deuxième étape vérifie si un laps de temps est nécessaire pour passer de R_k à R_{k+1}. Dans le cas où il existe, R_k et R_{k+1} ne sont pas accolées, donc nous créons une région d'horloges R_δ successeur temporel à R_k qui est traversée par les horloges pour passer de R_k à R_{k+1}.

Prenons l'exemple illustré en Figure 6.49. La région d'horloges successeur temporel à R est R'. Or R et R' ne sont pas accolées. Ainsi, la région d'horloges R_δ est construite par la fonction *Calculer-R-suivante* et est ajoutée à la liste entre R et R'. Dans cet exemple, la liste des régions successeur temporel à R est donc $\{R_\delta, R'\}$.

L'algorithme correspondant à ces étapes est le suivant :

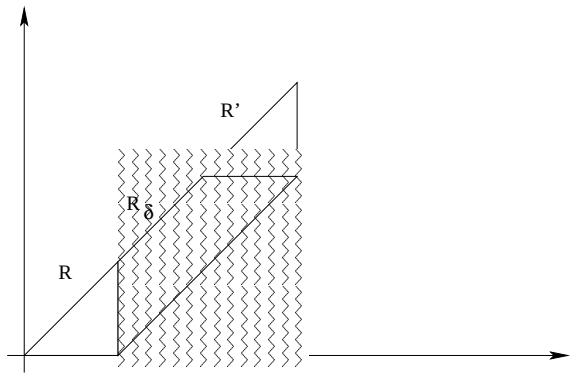


FIG. 6.49 – Construction de la liste de régions d'horloges successeur temporel à une région d'horloges R

Algorithme

Chercher-successeur
Entrée : Région d'horloges R
Sortie : Liste de régions d'horloges $ListeR$
DEBUT :
$ListeR2 = \{R_i \mid \forall v \in R_i, \exists d \in \mathbb{R}^+, v - d \in R$ et R_i successeur temporel à $R\}$
Ordonner les Régions d'horloges de $ListeR2$ de la région d'horloges ayant les plus petites valuations d'horloges à celles ayant les plus grandes
Region=R
Pour chaque $R_l \in ListeR2$
Si $\exists v$ tel que $\exists d \in \mathbb{R}^+, v - d \in Region, v \notin R_l$
Alors
- % Region et R_l ne sont pas voisines
- Pour chaque horloge x_i
- $M_{x_i} = min\{S_i \mid S = (S_1, ...S_i, ..., S_n)$ est un sommet de $R_l\}$
- $Ct = Ct \cup \{x_i < M_{x_i}\}$
- $FinPour$
- Calcul de la région d'horloges successeur à $Region$
- $R_{delta} = Calculer - R - suivante(Ct, Region, \emptyset\}$
- AjouterFin($ListeR, R_\delta$)
- AjouterFin($ListeR, R_l$)

Sinon AjouterFin($ListeR, R_l$)
$Region = R_l$
FinPour
FIN

A partir de la liste des régions d'horloges obtenue $ListeR$, l'algorithme de génération de graphe des régions construit une transition $R_k \xrightarrow{\delta} R_{k+1}$, pour tout couple $(R_k, R_{k+1}) \in ListeR^2$. Ceci modélise que R_{k+1} est successeur temporel à R_k. Prouvons alors, grâce à la Proposition suivante, que R_{k+1} est successeur temporel à R_k.

Proposition 6.4.7 *La liste de régions d'horloges ListeR obtenue grâce à Chercher-successeur est telle que* $\forall(R_k, R_{k+1}) \in ListeR^2$, R_{k+1} *est successeur temporel à* R_k.

Preuve:

Nous allons prouver que :
(1) pour tout couple (R_k, R_{k+1}) de $ListeR2$, R_{k+1} est successeur temporel à R_k.
(2) pour tout triplet (R_k, R_δ, R_{k+1}) de $ListeR$, avec R_δ une région d'horloges modélisant un laps de temps pour passer de R_k à R_{k+1}, R_{delta} est successeur temporel à R_k et R_{k+1} est successeur temporel à R_δ.

(1)Soit donc un couple $(R_k, R_{k+1}) \in ListeR2^2$. D'après les hypothèses, $\forall v \in R_k, \exists d_1 \in \mathbb{R}^+, v - d_1 \in R$. Toujours d'après les hypothèses, R_{k+1} est successeur temporel à R, donc $\forall v' \in R, \exists d_2 \in \mathbb{R}^+, v' + d_2 \in R_{k+1}$. Par conséquent, $\forall v \in R_k, \exists d = d_2 - d_1 \in \mathbb{R}^+, v + d \in R$. D'autre part, d'après les hypothèses, les valuations d'horloges de R_{k+1} sont supérieures à celles de R_k, donc $\forall v'' \in R_{k+1}, ||v'' - v'|| > ||v - v'||$, avec $||\ ||$ modélisant la norme. Donc $d_2 - d_1 \in R^+$. Ainsi, R_{k+1} est successeur temporel à R_k.

(2) R_{delta} est successeur temporel à R_k car R_{delta} est générée par la fonction *Calculer-R-suivante* (Proposition 6.4.4). D'après la Preuve de la Proposition 6.4.3, les valuations d'horloges de R_{delta} sont toutes accessibles de celles de R_k. Donc $\forall v \in R_{delta}, \exists d \in \mathbb{R}^+, v - d \in R_k$. De plus, R_{delta} et R_{k+1} sont successeur temporel à R_k. Nous nous retrouvons donc dans le cas (1).

■

6.4.4 Phase 4 : Agrégation des régions d'horloges du graphe des régions

Cette phase permet d'agréger les régions d'horloges qui peuvent l'être selon la Proposition 4.6.1. Cette Proposition agrège toutes les régions d'horloges du graphe des régions GR sans que le comportement de GR soit modifié. Elle permet non seulement la réduction du nombre d'horloges mais aussi une amélioration de l'accessibilité des régions d'horloges de GR.

6.5 Equivalence sémantique entre l'automate temporisé et le graphe des régions obtenu

Cette section démontre que l'automate temporisé et l'automate des régions construit en phase 2 sont équivalents, puis que l'automate des régions et le graphe des régions construit en phase 4 le sont aussi. Enfin, nous montrons que ce graphe des régions est réduit par rapport à celui-ci défini dans [AD94].

Equivalence entre un automate temporisé et l'automate des régions construit en Phase 2

L'équivalence entre un automate temporisé AT et un automate des régions AR est triviale car en fait, nous ne faisons qu'ajouter des informations sur les transitions. Cette équivalence sera, par contre, primordiale pour prouver l'équivalence entre AT et le graphe des régions généré dans la prochaine phase.

Proposition 6.5.1 *Soit un automate temporisé* $AT =< \Sigma_{AT}, S_{AT}, s^0_{AT}, C_{AT}, E_{AT} >$, *et* AR *l'automate des régions généré par l'algorithme précédent.*

AT *et* AR *sont sémantiquement équivalents*

Preuve:

Pour chaque transition $S_i \xrightarrow{A} S_j \in E_{AT}$, il existe $S_i \xrightarrow[ENS-R]{A} S_j \in E_{AR}$, avec $ENS-R$ les régions d'horloges satisfaisant les contraintes d'horloges de $S_i \xrightarrow{A} S_j$, et inversement. Donc AT et AR sont sémantiquement équivalents et acceptent le même langage.

■

Équivalence entre un automate des régions et le graphe des régions construit en Phase 4

Prouvons pour commencer que l'automate final est un graphe des régions, grâce à la définition d'Alur et Dill [AD94] (section 3.1.2).

Proposition 6.5.2 *Le graphe GR, obtenu par l'algorithme de la phase 4, est un graphe des régions.*

Preuve:

D'après la définition, le graphe des régions GR est un graphe des régions ssi :

1. $\forall t = (S_i, R) \xrightarrow{A} (S_j, R')$, R', $A \neq \delta$ est successeur temporel à R, en appliquant à R les réinitialisations d'horloges.
2. $\forall t = (S_i, R) \xrightarrow{\lambda} (S_i, R')$, R', R' est successeur temporel à R.

(1)D'après l'algorithme de génération d'un automate des régions en graphe des régions, si $A \neq \delta$, alors depuis une transition $S_i \xrightarrow[ENS-R]{A} S_j$, nous obtenons pour chaque région d'horloges de $R \in ENS-R$ la transition $(S_i, R) \xrightarrow{A} (S_j, R')$, avec soit $R' = R$ s'il n'y a aucune réinitialisation, ou soit $R' = R[\lambda \rightarrow 0]$ la région d'horloges obtenue à partir de R en réinitialisant les horloges de l'ensemble des horloges réinitialisées λ. Donc, $\forall v \in R, \exists d \in \mathbb{R}^+, v[\lambda \rightarrow 0] + d \in R'$, avec $v[\lambda \rightarrow 0]$ une valuation d'horloges à laquelle ont été appliquées les réinitialisations des horloges de λ.

(2) Les transitions étiquetées $(S_i, R) \xrightarrow{\lambda} (S_i, R')$ dans le graphe des régions sont obtenues dans l'algorithme de génération du graphe des régions de telle sorte que le couple (R, R') provient d'une liste de régions d'horloges générée par la fonction *Chercher-successeur*. D'après la Proposition 6.4.7, R' est successeur temporel à R.

■

L'équivalence entre un automate des régions et son graphe des régions est démontré grâce à la Proposition suivante :

Proposition 6.5.3 *Soit GR, le graphe des régions GR obtenu par l'algorithme de génération de graphe des régions à partir d'un automate des régions AR. GR et AR sont sémantiquement équivalents.*

Preuve:

Pour prouver que GR et AR acceptent le même langage, nous montrons le fait que toute transition de l'une des deux spécifications, étiquetée par un symbole différent de δ, possède un équivalent chez le deuxième. Puis nous montrons que chaque état d'un des deux graphes, pouvant être atteint par un chemin du graphe, possède un équivalent, pouvant aussi être atteint par un chemin du graphe. Plus précisément, il faut prouver les points suivants :

1. $\forall S_i \xrightarrow[ENS-R]{A} S_j \in E_{AR}$, avec $A \neq \delta$, $\forall R \in ENS-R, \exists (S_i, R) \xrightarrow{A} (S_j, R') \in E_{GR}$.
2. $\forall (S_i, R) \xrightarrow{A} (S_j, R) \in E_{GR}$, avec $A \neq \delta$, $\exists S_i \xrightarrow[ENS-R]{A} S_j \in E_{AR}$ avec $R \in ENS-R$.
3. $\forall S_i \in S_{AR}$, s'il existe un chemin permettant d'atteindre S_i, alors $\forall (S_i, R) \in S_{GR}$, il existe un chemin permettant d'atteindre (S_i, R).
4. $\forall (S_i, R_i) \in S_{GR}$, s'il existe un chemin permettant d'atteindre (S_i, R_i), alors $\forall S_i \in S_{AR}$, il existe un chemin permettant d'atteindre S_i.

(1) Soit $S_i \xrightarrow[ENS-R]{A} S_j \in E_{AR}$, avec $A \neq \delta$. D'après l'algorithme de génération de graphe des régions, cette transition génère les transitions du graphe des régions suivantes : $(S_i, R) \xrightarrow{A} (S_j, R')$ avec $R \in ENS-R$.

(2) Soit $(S_i, R) \xrightarrow{A} (S_j, R) \in E_{GR}$, avec $A \neq \delta$. D'après l'algorithme de génération de graphe des régions, $(S_i, R) \xrightarrow{A} (S_j, R)$ est générée depuis une transition $S_i \xrightarrow[ENS-R]{A} S_j \in E_{AR}$ (fonction *Ajouter-transition*), tel que $R \in ENS-R$.

(3) Pour prouver le troisième point, nous utilisons un raisonnement par récurrence. D'après les hypothèses, $S_i \in S_{AR}$ est atteint par un chemin de AR. Soit $S_k \xrightarrow[ENS-R]{A} S_i$ la transition permettant d'atteindre S_i. Supposons que $\forall R \in ENS-R$, $(S_k, R) \in S_{GR}$ soit atteint par un chemin de GR.
Soit (S_i, R') un état de GR. Si $R' = R[\lambda \rightarrow 0]$, c'est-à-dire si R' est obtenue depuis R en réinitialisant les horloges de λ qui doivent être réinitialisées sur la transition $S_k \xrightarrow[ENS-R]{A} S_i$, alors l'algorithme de génération de graphe des régions, a construit une transition $(S_k, R) \xrightarrow{A} (S_i, R')$. Donc (S_i, R') est atteint par un chemin du graphe des régions.
Sinon, d'après l'algorithme de génération de graphe de régions, l'unique possibilité pour construire (S_i, R'), avec $R' \neq R[\lambda \rightarrow 0]$, est qu'il existe une transition $S_i \xrightarrow[ENS-R']{B} S_j$ dans

AR telle que $R' \in ENS - R'$. D'après la Proposition 6.4.5, nous savons aussi que R' est successeur temporel à $R_1 = R[\lambda \rightarrow 0]$ avec $R \in ENS - R$. L'étape 1 de l'algorithme construit $\forall R \in ENS - R$, les transitions $(S_k, R) \xrightarrow{A} (S_i, R_1)$ avec $R_1 = R[\lambda \rightarrow 0]$. Donc (S_i, R') est-il accessible depuis (S_i, R_1) ? L'étape 2 de l'algorithme recherche toutes les régions d'horloges $\{R_2, ...R_l, R', R_m, ..., R_n\}$ de R_1 et construit, entre autre, les transitions $(S_i, R_1) \xrightarrow{\delta} (S_i, R_2)...(S_i, R_l) \xrightarrow{\delta} (S_i, R')$. Donc depuis (S_k, R) il existe un chemin permettant d'atteindre (S_i, R').
Nous avons le cas particulier $(S_0, R) \in S^0_{GR}$ de GR qui est atteint par "ϵ", modélisant un symbole vide. Donc en raisonnant par récurrence sur toutes les transitions de tous les chemins de l'automate des régions AR, s'il existe un chemin permettant d'atteindre S_i dans AR, il existe alors un chemin permettant d'atteindre (S_i, R) dans AR.

(4)Pour prouver le quatrième point, nous utilisons aussi un raisonnement par récurrence. D'après les hypothèses, $(S_i, R') \in S_{GR}$ est atteint par un chemin de AR. Soit $(S_k, R) \xrightarrow{A} (S_i, R')$ la transition permettant d'atteindre (S_i, R'). Supposons que $S_k \in S_{AR}$ soit atteint par un chemin de AR. S_i est-il atteint par un chemin de AR ?
Si $A \neq \delta$, alors la transition $(S_k, R) \xrightarrow{A} (S_i, R')$ est construite par la fonction *Ajouter-transition*, depuis une transition $S_k \xrightarrow[ENS-R]{A} S_i \in E_{AR}$, avec $R \in ENS - R$. Donc la transition $S_k \xrightarrow[ENS-R]{A} S_i$ appartient à l'automate des régions AR et, il existe un chemin de AR permettant d'atteindre S_i.
Si $A = \delta$, alors d'après la définition du graphe des régions, $S_i = S_k$. D'après les hypothèses, S_k est atteint par un chemin de AR.
Nous avons le cas particulier S_0 de AR qui est atteint par "ϵ", modélisant un symbole vide. Donc en raisonnant par récurrence sur toutes les transitions de tous les chemins du graphe des régions GR, s'il existe un chemin permettant d'atteindre (S_i, R) dans GR, il existe alors un chemin permettant d'atteindre S_i dans AR. ■

D'après les Propositions 6.5.1 et 6.5.3, qui prouvent que l'automate des régions AR généré par la phase 2 est équivalent à un automate AT et que le graphe des régions généré par la phase 4 est équivalent à AR, nous pouvons écrire le théorème suivant qui affirme l'équivalence entre AT et GR.

Théorème 6.5.1
Soit AT un automate temporisé et GR le graphe des régions obtenu par les phases 1, 2 et 3. AT et GR sont sémantiquement équivalents

Proposition 6.5.4 *Soit $\mathcal{RA}$ le graphe des régions obtenu grâce à l'algorithme décrit dans [AD94], et GR le graphe des régions obtenu grâce aux quatre phases précédentes. GR est un graphe des régions réduit par rapport à $\mathcal{RA}$.*

Preuve:

Soit R une région d'horloges de GR. Cette région d'horloges est un polyhèdre, contenant des sommets des segments ouverts et des plans. La construction du graphe des régions $\mathcal{RA}$ va au moins générer les régions d'horloges supplémentaires suivantes, inclues dans R : chaque sommet de R et chaque segment ouvert de R sont des régions d'horloges dans $\mathcal{RA}$. Par conséquent, $\mathcal{RA}$ a plus d'états que GR. D'après le théorème 6.5.1 et d'après les

auteurs de [AD94], $\mathcal{RA}$ est sémantiquement équivalent à un automate temporisé AT, qui est lui-même sémantiquement équivalent à GR. Donc GR est sémantiquement équivalent à $\mathcal{RA}$. GR est donc un graphe des régions réduit par rapport à $\mathcal{RA}$. ∎

Pour finir cette partie, il serait intéressant de faire le lien entre la notion d'équivalence sémantique (entre l'automate temporisé et le graphe des régions réduit, Théorème 6.5.1) et la notion d'équivalence de trace temporisée (entre une implantation et un graphe des régions, Définition 5.2.1). Ce lien peut être obtenu en définissant une équivalence directement entre la spécification et l'implantation et en vérifiant que cette équivalence est préservée en transformant la spécification en graphe des régions réduit.

Cette équivalence ne peut correspondre à la relation de conformité (définition 5.2.1) car celle-ci est donnée sur deux formalismes identiques. Ainsi, nous proposons une seconde relation, appelée C-équivalence (équivalence de comportement).

Définition 6.5.1 (C-équivalence) *Soit* $\mathcal{AT} =< \Sigma_{\mathcal{AT}}, S_{\mathcal{AT}}, s^0_{\mathcal{AT}}, C_{\mathcal{AT}}, E_{\mathcal{AT}} >$ *un automate temporise et* I *une implantation.* $\mathcal{AT}$ *et* I *sont C-equivalents ssi :*

Pour chaque séquence de transition $\sigma = s_0 \xrightarrow[ct_1]{A_1} s_2...s_{n-1} \xrightarrow[ct_n]{A_n} s_n$ $(s_{i-1} \xrightarrow[ct_i]{A_i} s_i \in E^n_{\mathcal{AT}}, 0 < i < n+1)$*, il existe une séquence d'actions temporisée* $\sigma' = \langle B_1, R_1\rangle...\langle B_m, R_m\rangle \in TIMSEQ_I, (m > n)$ *tel que :*
l'application ordonnée de chaque action temporisée $\langle B_i, R_i\rangle$*, avec* $B_i \neq \delta$*, permet le franchissement de toute les transitions de* σ*.*

Grâce à la Proposition suivante, vérifions que cette équivalence est toujours satisfaite si l'implantation est conforme au graphe des régions réduit de la spécification.

Proposition 6.5.5 *Soit* $\mathcal{AT}$ *un automate temporisé,* GR *son graphe des régions associé et* I *une implantation.*

Si I *est conforme a* GR*, selon la relation de conformité (Définition 5.2.1), alors* $\mathcal{AT}$ *et* I *sont C-équivalents.*

Preuve:

GR et $\mathcal{AT}$ sont sémantiquement équivalents donc $\forall\sigma'' = \langle A''_1, R''_1\rangle...\langle A''_m, R''_m\rangle \in TIMSEQ_{GR}, \exists\sigma = s_0 \xrightarrow[ct_1]{A_1} s_2...s_{n-1} \xrightarrow[ct_n]{A_n} s_n$ $(s_{j-1} \xrightarrow[ct_j]{A_j} s_j \in E^n_{\mathcal{AT}}(0 < j < n+1))$, tel que l'application ordonnée de chaque action temporisée $\langle A''_i, R''_i\rangle(0 < i < m+1)$ permet le franchissement de toute les transitions de σ.

Supposons que I soit conforme à GR. Alors, $\forall\sigma'' = \langle A''_1, R''_1\rangle...\langle A''_m, R''_m\rangle \in TIMSEQ_{GR}$, $\exists\sigma' = \langle B_1, R_1\rangle...\langle B_m, R_m\rangle \in TIMSEQ_I$ tel que $A''_k = B_k$ et $R_k \subseteq R''_k(0 < k < m+1)$. Ainsi, l'application des actions temporisées $\langle B_k, R_k\rangle(0 < k < m+1)$ permet toujours le franchissement des transitions de σ. Donc I est C-équivalent à $\mathcal{AT}$. ∎

6.6 Conclusion

Dans ce chapitre, nous avons décrit un algorithme de génération directe d'un graphe des régions réduit à partir d'un automate temporisé. Au fil des quatre phases de l'algorithme, nous avons prouvé que le graphe obtenu est bien un graphe des régions, que celui-ci est sémantiquement équivalent à l'automate temporisé de départ et que son nombre d'états est inférieur au nombre d'états du graphe des régions généré par l'algorithme décrit dans [AD94].

La phase 3 nous a permis de plus d'introduire une nouvelle modélisation des systèmes temporisés, l'automate des régions. En résumé, il correspond à une sorte de mixage des cotés positifs des automates temporisés et des graphes des régions. C'est-à-dire qu'il conserve le nombre d'états et de transitions de l'automate temporisé mais que chaque transition t est en plus étiquetée de l'ensemble des régions d'horloges satisfaisant t. Il est aussi à noter que ces ensembles sont complètement accessibles temporellement, ce qui garantit un degré d'accessibilité temporelle d'un quelconque automate des régions optimum (égal à 1) et donc une qualité de test améliorée.

Par la suite, nous avons l'intention de nous pencher, une fois de plus, sur ces différentes phases pour vérifier si elles peuvent être simplifiées et pour réduire le coût de la transformation. Par exemple, il est possible que le passage en automate des régions ne soit pas nécessaire, et que les phases 3 et 4 puissent être scindées.

Finalement, comme nous l'avons affirmé dans l'introduction, le but de ce travail est d'obtenir directement un graphe des régions minimisé. Pour arriver à ce stade, il nous faut encore prouver que pour chaque région d'horloges R obtenue dans la phase 3, il n'existe pas de région d'horloges $R' \supset R$ qui, si elle remplaçait R dans le graphe des régions, ne changerait pas son comportement. Dans un futur proche, une comparaison de coût entre cet algorithme et ceux de [ACH+92] et [YL93] sera ainsi évaluée.

Chapitre 7

Conclusion et Perspectives

Conclusion

La validation d'un système ou d'un logiciel est aujourd'hui très coûteuse par rapport au coût total de la génération du produit final. Il est par conséquent nécessaire de réduire ce coût tout en détectant un nombre maximum d'erreurs dans l'implantation. Cette thèse contribue à cet objectif en étudiant la qualité de test des systèmes temporisés, modélisés par des automates temporisés ou des graphes des régions. Nous avons mis en évidence que cette qualité pouvait être améliorée non seulement en étudiant et modifiant certaines propriétés de la spécification, mais aussi en appliquant d'autres méthodes de génération de séquences de test.

Ainsi, dans le chapitre 4, nous avons défini et évalué la qualité de test des automates

temporisés puis des graphes des régions. Pour chacun de ces modèles, nous avons analysé plusieurs propriétés influençant le coût du test et la détection d'erreurs. Nous avons montré l'impact de ces propriétés sur la qualité de test en définissant plusieurs facteurs et en montrant comment les calculer. Nous avons ainsi défini :

- le Degré de Forme Temporelle, qui évalue la difficulté à tester une spécification par rapport à ses valeurs de temps infinies,
- le Degré d'Accessibilité Temporelle, qui évalue la difficulté à atteindre, grâce aux horloges, des intervalles de temps ou des régions d'horloges,
- le Degré de Contrôlabilité, qui mesure le coût de l'exécution d'un ensemble des préambules sur la spécification,
- le Degré d'Indépendance Temporelle, qui évalue le coût du test, soit par rapport au nombre d'horloges (automate temporisé), soit par rapport au nombre de tests supplémentaires qu'apporte la transformation de l'automate temporisé en graphe des régions.

Ceux-ci mis sous la forme d'un vecteur appelé TTV et réunis à un autre vecteur appelé TV, définissent et évaluent la qualité de test des deux modèles précédents. Nous avons décrit que des implantations temporisées peuvent être seulement partiellement testées à cause des propriétés d'infinité et de non accessibilité que peuvent induire des intervalles de temps (ou régions d'horloges). Ainsi, la qualité de test de tels systèmes ne dépend plus uniquement de leur observabilité et de leur contrôlabilité.

Ensuite, nous avons présenté des modifications possibles à appliquer sur des spécifications dans le but d'améliorer leurs qualités de test. Nous avons aussi décrit que cette amélioration peut être difficile voire impossible dans certains cas. Elle peut demander une refonte du comportement du système et ne peut pas toujours conduire à l'obtention d'une qualité de test optimale. En effet, les intervalles de temps infinis sont difficiles à transformer, l'accessibilité des régions d'horloges demande une agrégation qui n'est pas toujours possible, et l'ensemble d'horloges d'un système ne peut pas toujours être réduit à une horloge. Ainsi, la qualité de test aura pour rôle d'informer le concepteur des parties du système qui ne pourront pas toujours être testées.

Le chapitre 5 a été consacré à la présentation de deux méthodologies de test. La première est orientée Objectif de test et permet de vérifier n'importe quelle suite de propriétés comportementales ou temporelles sur l'implantation. Suivant que l'objectif de test soit acceptant ou refusant, un produit synchronisé temporisé a été défini dans le but de réunir les propriétés de la spécification et de l'objectif de test, pour générer les séquences de test.

La seconde est une méthodologie basée sur la caractérisation d'états de graphe des régions. Elle permet de caractériser des états temporisés sans apporter aucune modification au système, en tirant partie des propriétés comportementales mais aussi temporelles de ce dernier. Son utilisation permet essentiellement de réduire les coûts du test, tout en obtenant un bon pouvoir de détection d'erreurs. Nous avons défini cette caractérisation et donné un algorithme générant les séquences d'actions distinguant chaque paire d'états.

Pour générer des séquences de test exécutables à partir des séquences abstraites, nous avons présenté un algorithme de traduction de ces séquences en arbres. Ces arbres permettent aussi de calculer à la volée si des régions d'horloges ne pourront pas être accessibles, ce qui permet l'arrêt précoce d'un test impossible. Nous décrivons aussi comment

transformer en TTCN, ces séquences de test, ce qui permet l'utilisation directe de ces méthodologies par des testeurs industriels.

Le chapitre 6 a présenté une solution, encore embryonnaire, au problème de la génération du graphe des régions pouvant engendrer un nombre d'états exponentiels. L'algorithme que nous décrivons, permet de transformer la spécification en graphe des régions, tout en évitant la génération d'un premier graphe pouvant contenir ce nombre exponentiel d'états. Le but de ce travail est d'obtenir un algorithme de minimisation offrant un coût plus faible que ceux déjà proposés dans la littérature.

Perspectives

Terminons cette thèse, en indiquant quelques extensions possibles et quelques travaux envisageables dans le domaine du test temporisé. Ces deniers étant relativement nombreux, nous nous contenterons d'indiquer des compléments naturels permettant d'achever certaines études, ainsi que quelques développements envisageables, proches de nos recherches.

- Nous n'avons développé l'étude de la qualité de test que sur les systèmes temporisés, modélisés par des automates temporisés ou des graphes des régions. Il serait intéressant de se pencher sur d'autres modélisations, éventuellement sur des techniques de description formelle de haut niveau, telles que RT-LOTOS et Time-ESTELLE. La qualité de test pourrait ainsi se mesurer tout au long du cycle de vie. Les systèmes distribués, non temporisés, puis temporisés pourraient aussi être étudiés, par exemple par le biais des automates parallèles définis dans le chapitre 2. Le fait d'être distribué rend la vue de ces systèmes plus difficile et abstraite. Par conséquent, l'étude de leurs qualités de test pourrait permettre de mesurer les parties de ces systèmes qui peuvent être testées et d'évaluer le coût de leurs validations. La motivation de cette étude est d'autant plus importante que d'autres propriétés, telles que les actions internes interagissant entre deux modules, sont présentes dans ces systèmes.
- Plusieurs travaux peuvent étendre la méthode de minimisation étudiée au chapitre 6. Premièrement, il nous faut prouver que nous pouvons obtenir un graphe des régions minimal sur le nombre de ses états et sur ses régions d'horloges. Puis nous devons définir une borne sur les régions d'horloges obtenues. Une comparaison du coût de cet algorithme avec ceux de [ACH+92, YL93] devra aussi être évaluée. Plusieurs améliorations de l'algorithme sont aussi envisageables. Par exemple, le fait de découper une région d'horloges, nouvellement créées, par toutes les contraintes temporelles de l'automate, pourrait être simplifié. Il peut être envisageable de suivre l'idée de l'algorithme de [YL93]. C'est-à-dire qu'il pourrait être efficace de rechercher les régions d'horloges accessibles et de les découper ensuite.
- La prise en compte d'environnements de test temporisés est aussi une notion à considérer. Leur utilisation permettrait de se rapprocher d'un contexte réel. Une étude de l'influence d'un environnement sur la qualité de test peut aussi être à prévoir, car nous n'avons plus un accès direct à l'implantation. En ce qui concerne les systèmes

non temporisés, une telle étude a été faite dans [DAdSS01]. L'ajout de la notion de temps peut induire d'autres propriétés, comme les temps de latence nécessaires pour traverser l'environnement, ce qui demande certainement une nouvelle analyse.

- Prévoir le coût du test et permettre une meilleure détection des erreurs sur une implantation est une première étape. En aval, c'est-à-dire après la phase de test, un diagnostic sur les erreurs temporelles détectées est impératif pour modifier et améliorer la fiabilité du système. Il est en effet difficile de cerner quelle erreur induit une mauvaise observation depuis l'implantation : il peut s'agir d'un mauvais symbole ou d'une erreur sur une ou plusieurs contraintes temporelles ou encore d'une erreur sur une ou plusieurs réinitialisations d'horloges. Le choix semble vaste.
- Enfin, après ces résultats théoriques, il est nécessaire d'illustrer leurs utilités par le biais d'un outil regroupant : l'évaluation de la qualité de test telle que celle décrite dans le chapitre 4, la génération de séquences de test selon les deux méthodes du chapitre 5 et l'algorithme de transformation des automates temporisés en graphe des régions décrit dans le chapitre 6. Dans cette optique, nous avons implanté la méthodologie orientée objectif de test. L'outil obtenu prend en entrée un automate temporisé et un objectif de test, les traduit en graphe des régions puis génère le produit synchronisé. Outre le fait que de nombreux algorithmes restent à implanter, il serait intéressant de les appliquer sur des protocoles ou applications réels tels que le MAP-DSM dans son intégralité. D'autres difficultés, d'ordre plus pratiques, seront alors certainement soulevées telles que les problèmes d'espace mémoire et de performances.

Annexe A

Etude de cas : Le protocole GSM_MAP_DSM

Dans cet annexe, nous appliquons les résultats des travaux précédents sur une partie d'un protocole réel, le GSM-MAP-DSM. C'est-à-dire que nous évaluons sa qualité de test, une première fois sur un automate temporisé le modélisant et une seconde fois sur un graphe des régions. Les résultats obtenus sont ensuite commentés pour montrer les améliorations à apporter aux deux spécifications. Puis, nous appliquons les deux méthodes présentées au chapitre 5 et décrivons comment sont générées les séquences de test.

A.1 Description du Protocole GSM_MAP_DSM

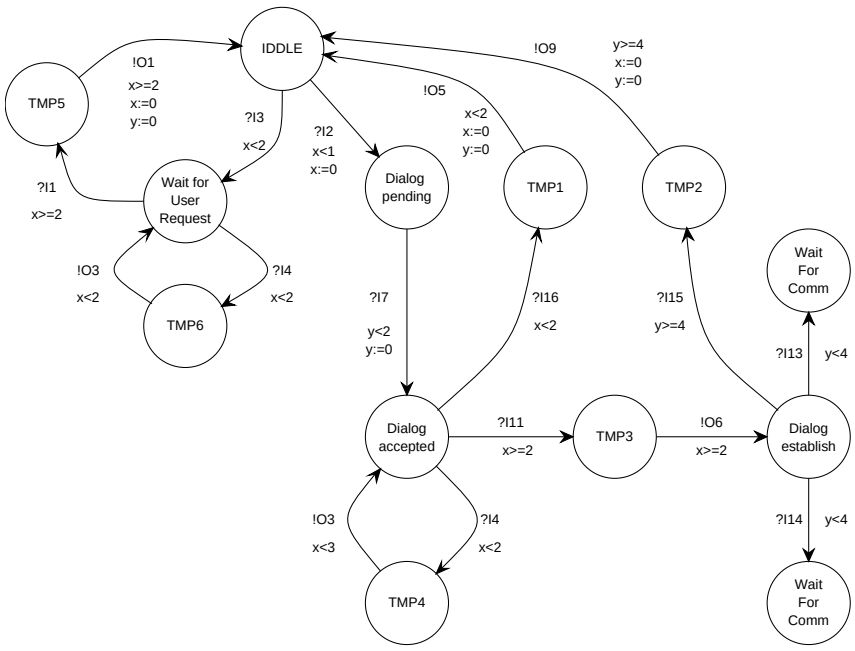
Parmi les protocoles utilisés par le protocole GSM (Global system for Mobile communication), neuf protocoles sont groupés dans le MAP (Mobile Application Part). Chacun correspond à une interface spécifique entre deux composants de services au niveau d'une

couche du GSM.

Le Dialog State Machine (DSM) génère des dialogues et simule la coordination entre les services MAP et leurs séquencements (ouverture, fermeture, continuité). Il gère les flots de données associés à des services MAP. Le DSM doit aussi instancier d'autres machines dans le but de démarrer certains services.

Une description détaillée de ces services et protocoles se trouve dans [CAR].

Étant donné que tous les algorithmes que nous avons précédemment décrits ne sont pas tous implantés, nous l'utiliserons dans ce qui suit qu'une partie de ce protocole avec ajout de deux horloges. Cette partie LDS, extraite de la modélisation LDS du GSM_MAP_DSM, est décrite ci-dessous. A partir de celle-ci, nous obtenons l'automate temporisé illustré en Figure A.50.



?I1 : map_u_abort_req
?I2 : tc_begin_ind
?I3 : map_open_req
?I4 : map_req
?I5 : map_delimiter_req
?I7 : map_open_rsp
?I11 : map_delimiter_rep
?I13 : tc_continue_ind
?I14 : tc_end_ind
?I15 : tc_p_abort_ind
?I16 : tc_p_abort_ind

!O1 : tc_u_abort_req; terminated
!O3 : service_invoked
!O5 : tc_ind_req;terminated
!O6 : tc_continue_req
!O9 : map_p_abort;terminated

FIG. A.50 – Protocole MAP-DSM

Cet automate temporisé peut aussi être traduit en graphe des régions illustré en Figure A.52. Les régions d'horloges sont décrites en Figure A.51.

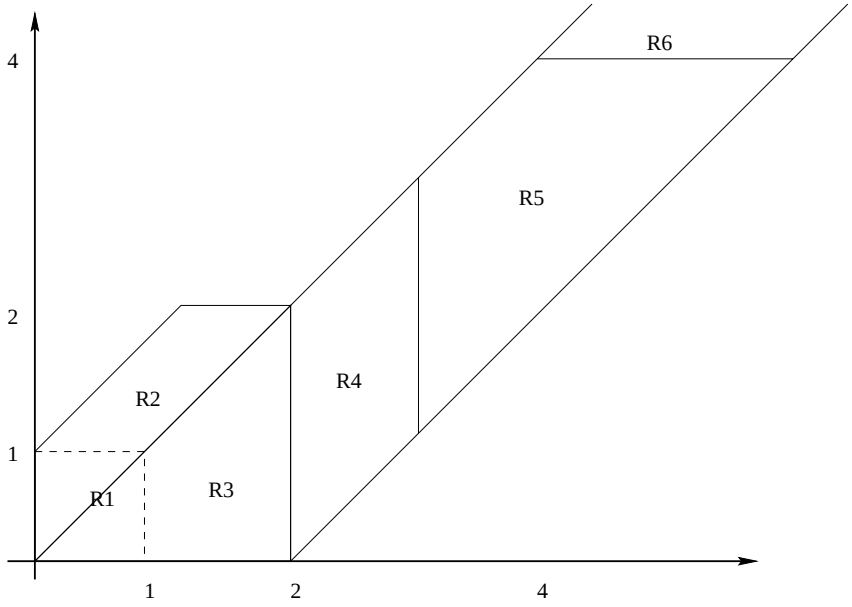


FIG. A.51 – Régions d'horloges

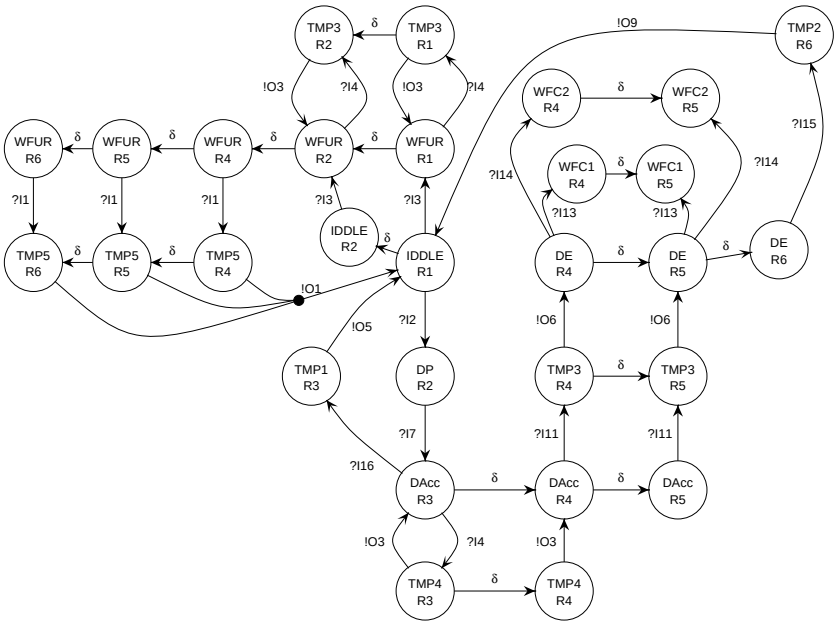


FIG. A.52 – Graphe des régions

Une partie de la specification LDS du GSM_MAP_DSM

SYSTEM GSM MAP DSM ;

STATE IDLE ;

INPUT MAP_OPEN_REQ(AC. Userlnfo) ;
TASK StoreAC :=AC COMMENT 'AC in V1 contextset ;
TASK STORE_Userlnfo :=Userlnfo ;
NEXTSTATE WAIT_FOR_USER_REQUESTS ;
INPUT TC_BEGIN_IND(AC) ;
DECISION 'DECODING_AC(AC)=SUPPORTED' COMMENT 'AC supported' ;
(TRUE) ;
NEXTSTATE DIALOGUE PENDING ;
(FALSE) ;
DECISION 'DECODING AC (AC) = ALTERNATIVE NAME'COMMENT 'Alternative name exists ? ' ; (TRUE) ;
TASK 'AC :=CONVERT(DECODING_AC(AC)) ;
(FALSE) ;
ENDDECISION ;
TASK ABORTreason :=l COMMENT'AC not supponed' ;
OUTPUT TC U ABORT_REQ(AC. ABORTreason) VIA route consomuner ;
NEXTSTATE- ;
ENDDECISION ;
ENDSTATE ;

STATE WAIT_FOR_USER_REQUESTS ;

INPUT MAP_REQ(ServicelD) ;
TASK nR InvokeID :=nR_InvokeID + I ;
TASK 'tab_REQ (nR_InvokeID) !PR ID := nR_InvokeID' ;
TASK 'tab_REQ (nR_InvokeID) !OP_CODE := EXTRACT OP_CODE(ServicelD) ' ;
OUTPUT SERVICE_INVOKED VIA route-consommer ;
NEXTSTATE ;
INPUT MAP_DELIMITER REQ ;
OUTPUT TC_BEGIN_REQ(StoreAC, STORE_UserInfo) VIA route-consommer ;
NEXTSTATE DIALOGUE_INITIATED ;
INPUT MAP_U_ABORT_REQ ;
TASK ABORTreason :=2 COMMENT 'user_defined' ;
TASK AC :=" ;
OUTPUT TC_U_ABORT_REQ(AC, ABORTreason) VIA route consommer ;
OUTPUT TERMINATED VIA route consommer ;
NEXTSTATE IDLE ;
ENDSTATE ;

STATE DIALOGUE_INITIATED ;

INPUT TC_END_IND(RESULT . AC COMPONENTS PRESENT) ;
DECISION 9ESULf COMMENT 'Refuse PDU ? ;
(1) ;
OUTPUT MAP_OPEN_CNF(AC) VIA route-consommer ;
COMMENT 'refused' ;

(0);
DECISION AC = StoreAC;
(TRUE);
OUTPUT MAP_OPEN_CNF(AC) VIA route-consommer COMMENT 'accepted';
DECISION COMPONENTS_PRESENT;
(TRUE);
NEXTSTATE WFC1;
ENDDECISION;
OUTPUT MAP_CLOSE_IND VIA route-consommer;
(FALSE);
OUTPUT MAP_P_ABORT_IND VIA route-consommer;
OUTPUT TERMINATED VIA route-consommer COMMENT 'All active SSMs';
OUTPUT TERMINATED VIA route-consommer COMMENT 'Allactive SSMs';
NEXTSTATE IDLE;
ENDDECISION;
ENDDECISION;
OUTPUT TERMINATED VIA route-consommer COMMENT 'All active SSMs';
OUTPUT TERMINATED VIA route-consommer COMMENT 'All active SSMs';
NEXTSTATE IDLE;
INPUT TC_CONTINUE_IND(COMPONENTS_PRESENT . AC);
DECISION AC= AC;
(TRUE);
OUTPUT MAP_OPEN_CNF(AC) VIA route-consommer COMMENT 'accepted';
DECISION COMPONENTS PRESENT;
(TRUE);
NEXTSTATE WFC2;
(FALSE);
ENDDECISION;
NEXTSTATE DIALOGUE ESTABLISHED;
(FALSE);
OUTPUT MAP_P_ABORT_IND VIA route-consommer;
TASK ABORTreason :=2 COMMENT 'user_defined';
TASK Userlnfo :='built MAP abort PDU' COMMENT *XXX pas un parametre de la primitive suivante';
OUTPUT TC_U_ABORT_REQ(AC. ABORTreason) VIA route-consommer';
OUTPUT TERMINATED VIA route-consommer COMMENT'All active SSMs';
OUTPUT TERMINATED VIA route-consommer COMMENT 'All active SSMs';
NEXTSTATE IDLE;
ENDDECISION;
INPUT TC_U_ABORT_IND(AC, ABORTreason, Userlnfo);
DECISION ABORTreason COMMENT "Which Cause";
(1) :
OUTPUT MAP_OPEN_CNF(AC) VIA route-consommer COMMENT 'refused, AC not supported';
(2) :
OUTPUT MAP_U_ABORT_IND(Userinfo) VIA route-consommer;

ENDDECISION COMMENT 'user_specific';
OUTPUT TERMINATED VIA route consommer COMMENT 'All active SSMs';
OUTPUT TERM INATED VIA route consommer COMMENT 'All active SSW';
NEXTSTATE IDLE;
INPUT TC_P_ABORT_IND;
OUTPUT MAP_P_ABORT_IND VIA route-consommer;
OUTPUT TERMINATED VIA route consommer COMMENT 'All active SSMs';
OUTPUT TERMINATED VIA route consommer COMMENT 'All active SSMs;
NEXTSTATE IDLE;
INPUT MAP_U_ABORT_REQ;
TASK ABORTreason :=2 COMMENT'user defined';
OUTPUT TC_U_ABORT_REQ(AC. ABORTreason) VIA route consommer;
OUTPUT TERMINATED VIA route consommer COMMENT All active SSMs' :
OUTPUT TERMINATED VIA route consommer COMMENT 'All active SSMs';
NEXTSTATE IDLE;
INPUT MAP_CL0SE_REQ(UserInfo);
OUTPUT TC_END_REQ(UserInfo) VIA route_consommer COMMENT'Pre arranged';
OUTPUT TERMINATED VIA route consommer COMMENT 'All active SSMs';
OUTPUT TERMINATED VIA route consommer COMMENT 'All active SSMs';
NEXTSTATE IDLE;
INPUT TC_L_CANCEL_IND(InvokeID);
OUTPUT MAP_OPEN_CNF(StoreAC) VIA route consommer COMMENT 'Accepted,
The dialogue is considered implicitly accepted when nothing is received';
TASK STRUCT1 :=tab_REQ(InvokeID);
DECISION 'STRUCT1!PR_ID /= 0' COMMENT ' and SSM active';
('TRUE');
OUTPUT TIMER_EXPIRY VIA route consommer; ('FALS E');
ENDDECISION;
NEXTSTATE DIALOGUE_INITIATED;
ENDSTATE;

STATE DIALOGUE_PENDING;

INPUT MAP OPEN RSP(RESULT);
DECISION RESULT;
(0);
NEXTSTATE DIALOGUE_ACCEPTED;
(1);
TASK UserInfo :='Built MAP_Refuse PDU' COMMENT 'Refused';
OUTPUT TC_END_REQ(UserInfo) VIA route consommer;
OUTPUT TERMINATED VIA route-consommer COMMENT 'All active SSMs';
OUTPUT TERMINATED VIA route-consommer COMMENT 'All active SSMs';
NEXTSTATE IDLE;
ENDDECISION;
INPUT MAP_U_ABORT_REQ;
TASK ABORTreason :=2 COMMENT 'user_defined';

OUTPUT TC_U_ABORT_REQ(AC. ABORTreason) VIA route consommer ;
OUTPUT TERMINATED VIA route consommer COMMENT 'All active SSMs' ;
OUTPUT TERMINATED VIA route consommer COMMENT 'All active SSMs' ;
NEXTSTATE IDLE ;
ENDSTATE ;

STATE DIALOGUE ACCEPTED ;

INPUT MAP_REQ(ServicelD)) ;
TASK nR InvokeID :=nR_InvokelD + 1 ;
TASK 'tab_REQ (nR_InvokeID) !PR ID := nR_InvokeID' ;
TASK 'tab_REQ (nR_InvokeID) !OP_CODE := EXTRACT_OP_CODE(ServicelD) ;
OUTPUT SERVICE INVOKED VIA route consommer ;
NEXTSTATE ;
INPUT MAP_RSPInvokeID) ;
OUTPUT RESPONSE_ISSUED VIA route-consommer ;
NEXTSTATE ;
INPUT MAP_CLOSE_REQ(UserInfo) ;
OUTPUT TC_END_REQ(Userinfo) VIA route-consommer ;
OUTPUT TERMINATED VIA route consommer COMMENT 'All active SSMs' ;
OUTPUT TERMINATED VIA route consommer COMMENT 'All active SSMs' ;
NEXTSTATE IDLE ;
INPUT MAP_DELIMITER_REQ ;
OUTPUT TC_CONTINUE_REQ VIA route-consommer ;
, NEXTSTATE DIALOGUE ESTABLISHED ;
ENDSTATE ;

STATE DIALOGUE ESTABLISHED ;

INPUT TC_CONTINUE_IND(COMPONENTS PRESENT, AC) ;
DECISION COMPONENTS PRESENT ;
(TRUE) ;
NEXTSTATE WFC3 ;
(FALSE) ;
ENDDECISION ;
NEXTSTATE ;
INPUT TC_END_IND(RESULT, AC, COMPONENTS PRESENT) ;
DECISION COMPONENTS PRESENT ;
(TRUE) ;
(FALSE) ;
NEXTSTATE WFC4 ;
ENDDECISION ;
OUTPUT MAP_CLOSE_IND VIA route-consommer ;
OUTPUT TERMINATED VIA route-consommer COMMENT 'All active SSMs' ;
OUTPUT TERMINATED VIA route-consommer COMMENT 'All active SSMs' ;
NEXTSTATE IDLE ;

INPUT TC_U_ABORT_IND(AC. ABORTreason, UserInfo);
DECISION ABORTreason COMMENT 'Which Cause';
(1);
OUTPUT MAP_P_ABORT_IND VIA route-consommer COMMENT 'Cause=map';
(2);
OUTPUT MAP_U_ABORT_IND(UserInfo) VIA route-consommer;
ENDDECISION COMMENT'Cause=user';
OUTPUT TERMINATED VIA route-consommer COMMENT 'All active SSMs';
OUTPUT TERMINATED VIA route-consommer COMMENT 'All active SSMs';
NEXTSTATE IDLE;
INPUT TC_P_ABORT_IND;
OUTPUT MAP_ABORT_IND VIA route-consommer;
OUTPUT TERMINATED VIA route-consommer COMMENT 'All active SSMs';
OUTPUT TERMINATED VIA route-consommer COMMENT 'All active SSMs';
NEXTSTATE IDLE;
INPUT MAP_REQ(ServiceID);
TASK nR_lnvokelD :=nR_InvokelD + 1;
TASK 'tab_REQ (nR_InvokeID) 'PR_ID := nR_InvokeID';
TASK 'tab_REQ (nR_InvokeID) !OP_CODE := EXTRACT_OP_CODE(ServiceID)
';
OUTPUT SERVICE_INVOKED VIA route-consommer;
NEXTSTATE -;
INPUT MAP_RSP(InvokeID);
OUTPUT RESPONSE_ISSUED VIA route-comommer;
NEXTSTATE -;
INPUT MAP_CLOSE_REQ(UserInfo);
OUTPUT TC_END_REQ(UserInfo) VIA route-consommer;
OUTPUT TERMINATED VIA route-consommer COMMENT 'All active SSMs';
OUTPUT TERMINATED VIA route-consommer COMMENT 'All active SSMs';
NEXTSTATE IDLE;
INPUT MAP_DELIMITER_REQ;
OUTPUT TC_CONTINUE_REQ VIA route-consommer;
NEXTSTATE -;
INPUT TC_L_CANCEL_IND(InvokeID);
TASK 'STRUCT1 := tab REQ(InvokeID) ';
DECISION 'STRUCT1! PR_ID /= 0 COMMENT 'and SSM active';
('TRUE');
OUTPUT TIMER_EXPIRY VIA route-consommer;
('FALSE');
ENDDECISION;
NEXTSTATE -;
ENDSTATE;

A.2 Qualité de test de l'automate temporisé

Soit l'automate temporisé de la Figure A.50, représentant une partie du protocole MAP-DSM avec ajout de deux horloges. L'automate traduit, contenant les intervalles de temps est illustré en Figure A.53

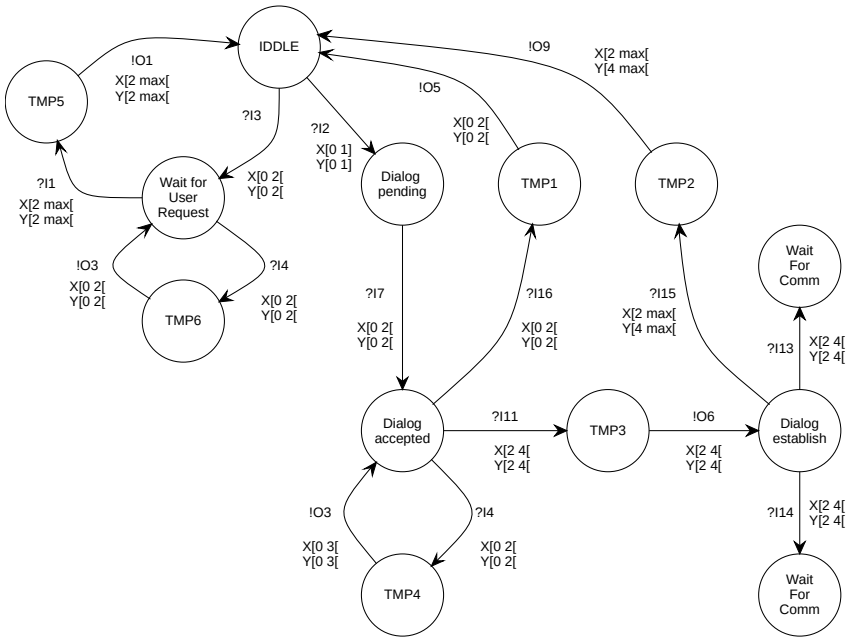


FIG. A.53 – Contraintes d'horloges modélisées par des Intervalles de temps

Degré de Forme Temporelle

En mesurant le Degré de Forme Temporelle sur cet automate, nous obtenons $S_t(\mathcal{A}) = \frac{13}{17}$. En effet, les transitions $WUFR \xrightarrow{?I1} TMP5$, $TMP5 \xrightarrow{!O1} IDDLE$, $DE \xrightarrow{?I15} TMP2$ et $TMP2 \xrightarrow{!O9} IDDLE$ contiennent des intervalles de temps infinis et $Card(I_{\mathcal{A}}) = 34$. Nous pouvons en conclure que le système a peu d'intervalles infinis mais qu'il sera, malgré cela, partiellement testé et que l'exécution des actions précédentes pourra générer des erreurs non détectables par le test.

Degré d'accessibilité temporelle

Les résultats suivants illustrent l'accessibilité temporelle de chaque ensemble d'intervalles de temps de l'automate temporisé décrit en Figure A.53.

Transition t	Intervalles de temps I_t	$card(SEQ_t)$	$RT(I_t)$
$IDDLE \xrightarrow{?I2} DP$	X[0 1] Y[0 1]	0	1
$IDDLE \xrightarrow{?I3} WFUR$	X[0 2] Y[0 2]	0	1
$WFUR \xrightarrow{?I4} TMP3$	X[0 2] Y[0 2]	1	1
$TMP3 \xrightarrow{!O3} WFUR$	X[0 2] Y[0 2]	1	1
$WFUR \xrightarrow{?I1} TMP5$	X[2 10] Y[2 10]	2	1
$TMP5 \xrightarrow{!O1} IDDLE$	X[2 10] Y[2 10]	2	1
$DP \xrightarrow{?I7} DAcc$	X[0 2] Y[0 2]	1	1
$DAcc \xrightarrow{?I4} TMP4$	X[0 2] Y[0 2]	1	1
$TMP4 \xrightarrow{!O3} DAcc$	X[0 3] Y[0 3]	1	1
$DAcc \xrightarrow{?I16} TMP1$	X[0 2] Y[0 2]	2	$\frac{1}{2}$
$TMP1 \xrightarrow{!O5} IDDLE$	X[0 2] Y[0 2]	2	$\frac{1}{2}$
$DAcc \xrightarrow{?I11} TMP3$	X[2 4] Y[2 4]	2	1
$TMP3 \xrightarrow{!O6} DE$	X[2 4] Y[2 4]	2	1
$DE \xrightarrow{?I15} TMP2$	X[4 10] Y[4 10]	2	1
$TMP2 \xrightarrow{!O9} IDDLE$	X[4 10] Y[4 10]	2	1
$DE \xrightarrow{?I13} WFC1$	X[2 4] Y[2 4]	2	1
$DE \xrightarrow{?I14} WFC2$	X[2 4] Y[2 4]	2	1

Pour le système, en sa globalité, nous obtenons $R_t(\mathcal{A}) = \frac{16}{17}$. Nous pouvons en conclure que la plupart des intervalles de temps sont complètement accessibles par les horloges, mais que le système sera partiellement testé. En effet, les intervalles de temps des transitions consécutives $TMP4 \xrightarrow{!O3} DAcc \xrightarrow{?I16} TMP1$ ne respectent pas la première proposition. Par conséquent, l'exécution successive des deux actions ne pourra pas toujours être testée.

Degré de Contrôle

La mesure du Degré de Contrôle sur l'automate temporisé de la Figure A.53 donne les résultats suivants. Nous avons choisi un ensemble de préambules décrit dans le tableau ci-dessous. La séquence de $Q_{\mathcal{A}}$, qui demande le plus grand temps d'exécution, est $?I2?I7?I11!O6?I15!O9?I3?I1$ et donne $E_{min} = 12$ et $E_{max} = 21$.

Préambule	Nombre de transitions atteintes	E_{min}	E_{max}	CT
ϵ	2	0	0	1
$?I3$	2	0	2	$\frac{31}{33}$
$?I3?I4$	1	0	2	$\frac{31}{33}$
$?I3?I1$	1	2	10	$\frac{21}{33}$
$?I2$	1	0	1	$\frac{32}{33}$
$?I2?I7$	3	0	2	$\frac{31}{33}$
$?I2?I7?I16$	1	0	3	$\frac{30}{33}$
$?I2?I7?I4$	1	0	3	$\frac{30}{33}$
$?I2?I7??I4!O3I11$	1	3	5	$\frac{26}{33}$
$?I2?I7?I11!O6$	3	4	5	$\frac{24}{44}$
$?I2?I7?I11!O6?I15$	1	4	11	$\frac{18}{33}$

Comme $Card(E_{\mathcal{A}}) = 17$, nous obtenons $CT(\mathcal{A}) = \frac{41}{51}$. Ceci montre que l'ensemble P, que nous avons choisi, permet d'atteindre les transitions du système rapidement en comparaison avec les séquences du système. En effet, la plupart des préambules demandent un coût faible pour être exécutés. Cependant, le préambule $?I2?I7?I4!O3?I11$ pourrait être remplacé par $?2?I7?I11$ car cette séquence est exécutée plus rapidement et offre une meilleure qualité de test au système.

Degré d'Indépendance Temporelle

En appliquant ce facteur sur l'automate temporisé de la Figure A.53, nous obtenons le résultat suivant : $IND_t(\mathcal{A}) = \frac{15}{16}$. La déduction qui peut en être faite est que le coût du test sera peu influencé par le nombre d'horloges, sachant que deux horloges pour tout un système temporisé semble un nombre raisonnable et que ce nombre pourra difficilement être réduit ([DY96].

A.3 Qualité de Test du Graphe des Régions

Degré de Forme Temporelle

Si nous considérons le graphe des régions, illustré en Figure A.52 et ses six régions d'horloges, illustrées en Figure A.51, nous obtenons les résultats suivants :

Region d'horloges	Nombre d'horloges non bornées	$card(U_R)$	ST_{rg}
R_1	0	4	1
R_2	0	4	1
R_3	0	4	1
R_4	0	7	1
R_5	0	6	1
R_6	2	4	0

De plus, $Card(AT) = 29$, donc nous obtenons un facteur $S_{rg} = (\mathcal{RA}) = \frac{25}{29}$. Celui-ci implique que le graphe des régions $\mathcal{RA}$ possède des régions d'horloges qui ne sont pas complètement formées temporellement. En effet, la région d'horloges R_6 contient un nombre infini de valuations d'horloges, ce qui implique que le test d'actions, pouvant s'exécuter dans R_6, sera partiel. Ainsi des erreurs potentielles risquent de ne pas être trouvées.

Degré d'Accessibilité Temporelle

Les résultats suivants illustrent l'application de ce facteur sur le graphe des régions représenté en Figure A.52.

Région d'horloges	$card(SEQ_{R_i})$	RT_{rg}	$Card(U_{R_i})$
R_1	46	$\frac{45}{92}$	4
R_2	9	$\frac{15}{18}$	4
R_3	4	$\frac{7}{8}$	4
R_4	27	$\frac{11}{18}$	7
R_5	60	$\frac{17}{30}$	6
R_6	48	$\frac{11}{16}$	4

Le Degré obtenus est tel que $R_t(\mathcal{A}) \simeq \frac{2}{3}$. Nous pouvons en conclure que le système n'est pas complètement accessible temporellement, car aucune région d'horloges ne peut toujours être atteinte quelquesoit l'exécution du système. En effet, certaines transitions étiquetées par les symboles $!O1$ $!O3$ et $!O6$ ne peuvent, suivant les exécutions, être passée et donc certaines régions d'horloges ne peuvent être atteintes. Ainsi la couverture du test sera partielle et quelques transitions devront être testées plusieurs fois pour obtenir un verdict.

Degré de Contrôle

L'application du degré de contrôle sur le graphe des régions, illustré en Figure A.52 donne les résultats suivants, sachant que la séquence de $Q_{\mathcal{RA}}$ qui demande le plus grand temps d'exécution est $?I2?I7?I11!O6?I15!O9?I3?I1$ et donne $E_{min} = 12$ et $E_{max} = 21$.

Préambule	Nb de transitions atteintes	E_{min}	E_{max}	CT_{rg}
$?I3$	5	0	2	$\frac{31}{33}$
$?I3?I4$	2	0	2	$\frac{31}{33}$
$?I3?I1$	3	2	10	$\frac{21}{33}$
$?I3?I1!O1$	3	2	10	$\frac{21}{33}$
$?I2$	1	0	1	$\frac{32}{33}$
$?I2?I7$	4	0	2	$\frac{31}{33}$
$?I2?I7?I16$	1	0	3	$\frac{30}{33}$
$?I2?I7?I4$	2	0	3	$\frac{30}{33}$
$?I2?I7?I11$	2	2	5	$\frac{26}{33}$
$?I2?I7?I11!O6$	5	4	5	$\frac{24}{33}$
$?I2?I7?I11!O6?I15$	1	4	11	$\frac{18}{33}$

Comme $Card(AT) = 29$, nous obtenons le facteur final $CT_{rg}(\mathcal{RA}) = \frac{219}{319}$. Ces résultats montrent que l'ensemble P des préambules choisis, donne un coût faible par rapport au comportement du système. Cependant, le préambule $?I3?I1!O1$ pourrait être remplacé par ϵ qui est un préambule demandant un temps d'exécution nul, et qui améliore donc la qualité de test de $\mathcal{RA}$.

Degré d'Indépendance Temporelle

Les résultats sont les suivants :

Transition $t \in E_{\mathcal{A}}$	$N_R(t)$	$IND(t)$
$IDDLE \xrightarrow{?I2} DP$	1	1
$IDDLE \xrightarrow{?I3} WFUR$	2	$\frac{4}{5}$
$WFUR \xrightarrow{?I4} TMP3$	2	$\frac{4}{5}$
$TMP3 \xrightarrow{!O3} WFUR$	2	$\frac{4}{5}$
$WFUR \xrightarrow{?I1} TMP5$	3	$\frac{2}{3}$
$TMP5 \xrightarrow{!O1} IDDLE$	1	1
$DP \xrightarrow{?I7} DAcc$	1	1
$DAcc \xrightarrow{?I4} TMP4$	1	1
$TMP4 \xrightarrow{!O3} DAcc$	2	$\frac{4}{5}$
$DAcc \xrightarrow{?I16} TMP1$	1	1
$TMP1 \xrightarrow{!O5} IDDLE$	1	1
$DAcc \xrightarrow{?I11} TMP3$	2	$\frac{4}{5}$
$TMP3 \xrightarrow{!O6} DE$	2	$\frac{4}{5}$
$DE \xrightarrow{?I15} TMP2$	1	1
$TMP2 \xrightarrow{!O9} IDDLE$	1	1
$DE \xrightarrow{?I13} WFC1$	2	$\frac{4}{5}$
$DE \xrightarrow{?I14} WFC2$	2	$\frac{4}{5}$

Comme $Card(E_{\mathcal{A}}) = 17$, nous obtenons un facteur tel que $I_{rg}(\mathcal{RA}) = \frac{167}{255}$. Nous pouvons en conclure que le coût du test du graphe des régions demande un peu plus d'efforts que celui du test de l'automate temporisé, plus précisément, il demande $\frac{1}{I_{rg}(\mathcal{RA})} \simeq 1.52$ fois plus d'efforts. Ceci est dû aux actions qui sont testées dans plusieurs régions d'horloges, donc pour lesquelles N_R est plus grand que 1.

A.4 Génération de séquences de test (méthodologie orientée objectif de test)

Illustrons dans cette section, l'exemple de génération de séquences de test à partir d'un objectif de test sur la spécification décrite en Figure A.50. L'objectif de test temporisé que nous proposons est celui décrit en Figure A.54. Celui-ci permet de vérifier si une connexion peut être établie entre le serveur et un fournisseur de services (le serveur émet le message !O6 = tc_continue_req) à partir d'une initialisation de connexion (le serveur reçoit ?I7 = map_open_rsp). Notons que l'objectif de test n'est pas une partie de la spécification, mais est composé de certaines de ces propriétés comportementales et temporelles.

A partir de la spécification (Figure A.50) et après l'avoir transformée en graphe des régions minimal, nous obtenons différents chemins contenant les symboles de l'objectif de test dans le même ordre. Ceux-ci sont décrits en Figure A.55. Les régions d'horloges sont illustrées en Figure A.51.

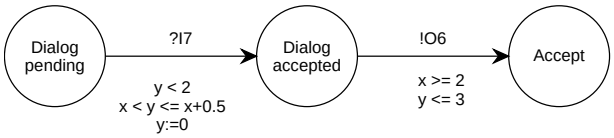


FIG. A.54 – Un objectif de test temporisé

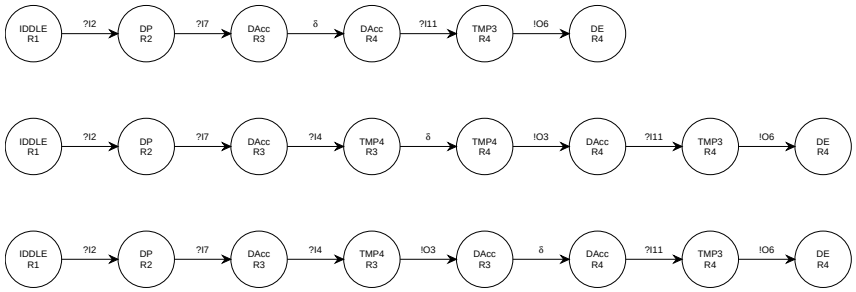


FIG. A.55 – Chemins du graphe des régions obtenus depuis le protocole GSM_MAP_DSM

De même, nous obtenons, depuis l'objectif de test (Figure A.54) le graphe des régions illustré en Figure A.57 et les régions d'horloges illustrées en Figure A.56.

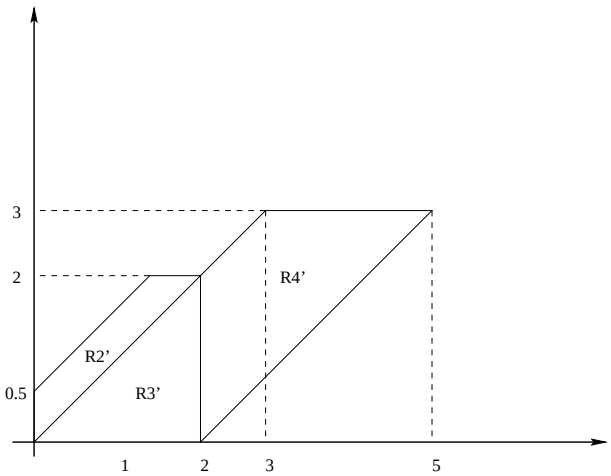


FIG. A.56 – Régions d'horloges obtenues à partir de l'objectif de test

Finalement, chaque chemin de la spécification (Figure A.55) est synchronisé avec l'objectif de test (Figure A.57), ce qui donne plusieurs séquences de test qui sont illustrées en Figure A.58.

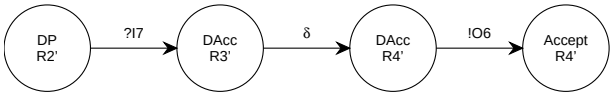


FIG. A.57 – Graphe des régions généré à partir de l'objectif de test

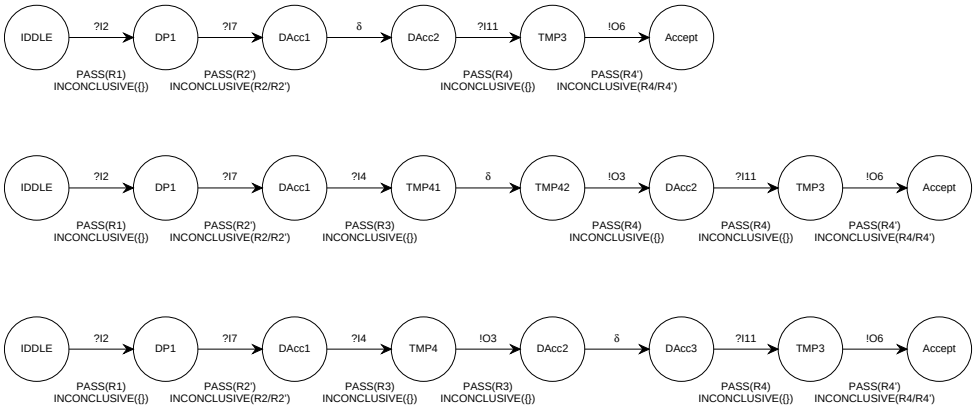


FIG. A.58 – Sequences de test

A.5 Génération de séquences de test (méthodologie basée sur la caractérisation d'états temporisés)

Cette section décrit les différents ensembles de séquences d'actions temporisées obtenus, depuis l'automate temporisé modélisant une partie du protocole GSM_MAP_DSM, grâce à la méthodologie basée sur la caractérisation d'états temporisés de graphe des régions. Plus précisément, elle décrit l'ensemble de caractérisation de chaque état ainsi que l'ensemble des préambule Q, l'ensemble de couverture de transitions P et l'ensemble $R = P - Q$.

Ensembles de Caractérisation d'états :

$TW_{TMP_3,R_1} : \{\langle !O3, R_1, accept\rangle\}$
$TW_{TMP_3,R_2} : \{\langle !O3, R_2, accept\rangle\}$
$TW_{TMP_5,R_4} : \{\langle !O1, R_4, accept\rangle\}$
$TW_{TMP_5,R_5} : \{\langle !O1, R_5, accept\rangle\}$
$TW_{TMP_5,R_6} : \{\langle !O1, R_6, accept\rangle\}$
$TW_{TMP_1,R_3} : \{\langle !O5, R_3, accept\rangle\}$
$TW_{TMP_4,R_3} : \{\langle !O3, R_3, accept\rangle\}$
$TW_{TMP_4,R_4} : \{\langle !O3, R_4, accept\rangle\}$
$TW_{TMP_2,R_6} : \{\langle !O9, R_6, accept\rangle\}$
$TW_{TMP_3,R_4} : \{\langle !O6, R_4, accept\rangle\}$
$TW_{TMP_3,R_5} : \{\langle !O6, R_5, accept\rangle\}$
$TW_{DE,R_6} : \{\langle !O9, R_6, refuse\rangle, \langle ?I15, R_6, accept\rangle\langle !O9, R_6, accept\rangle\}$
$TW_{DAcc,R_3} : \{\langle !O5, R_3, refuse\rangle, \langle ?I16, R_3, accept\rangle\langle !O5, R_3, accept\rangle\}$

$TW_{DAcc,R_4} : \{\langle !O6, R_4, refuse\rangle, \langle ?I11, R_4, accept\rangle\langle !O6, R_4, accept\rangle\}$
$TW_{DAcc,R_5} : \{\langle !O6, R_5, refuse\rangle, \langle ?I11, R_4, accept\rangle\langle !O6, R_4, refuse\rangle,$
$\langle ?I11, R_5, accept\rangle\langle !O6, R_5, accept\rangle\}$
$TW_{DP,R_2} : \{\langle ?I16, R_3, accept\rangle\langle !O5, R_3, refuse\rangle, \langle ?I7, R_2, accept\rangle\langle ?I16, R_3, accept\rangle$
$\langle !O5, R_3, accept\rangle\}$
$TW_{IDDLE,R_1} : \{\langle ?I7, R_2, accept\rangle\langle ?I16, R_3, accept\rangle\langle !O5, R_3, refuse\rangle,$
$\langle ?I2, R_1, accept\rangle\langle ?I7, R_2, accept\rangle\langle ?I16, R_3, accept\rangle\langle !O5, R_3, accept\rangle\}$
$TW_{WFUR,R_6} : \{\langle !O1, R_6, refuse\rangle, \langle ?I1, R_5, accept\rangle\langle !O1, R_5, refuse\rangle,$
$\langle ?I1, R_6, accept\rangle\langle !O1, R_6, accept\rangle\}$
$TW_{WFUR,R_5} : \{\langle !O1, R_5, refuse\rangle, \langle ?I1, R_4, accept\rangle\langle !O1, R_4, refuse\rangle,$
$\langle ?I1, R_5, accept\rangle\langle !O5, R_5, accept\rangle\}$
$TW_{WFUR,R_4} : \{\langle !O1, R_4, refuse\rangle, \langle ?I1, R_4, accept\rangle\langle !O1, R_4, accept\rangle\}$
$TW_{WFUR,R_1} : \{\langle !O3, R_1, refuse\rangle, \langle ?I4, R_1, accept\rangle\langle !O3, R_1, accept\rangle\}$
$TW_{WFUR,R_2} : \{\langle !O3, R_2, refuse\rangle, \langle ?I4, R_1, accept\rangle\langle !O3, R_1, refuse\rangle,$
$\langle ?I4, R_2, accept\rangle\langle !O3, R_2, accept\rangle\}$
$TW_{IDDLE,R_2} : \{\langle ?I3, R_2, accept\rangle\langle ?I4, R_2, accept\rangle\langle !O3, R2, accept\rangle,$
$\langle ?I4, R_2, accept\rangle\langle !O3, R_2, refuse\rangle, \langle ?I3, R_1, accept\rangle\langle ?I4, R_2, accept\rangle\langle !O3, R_1, refuse\rangle\}$

Ensemble des Préambules Q :

$IDDLE, R_1 : \epsilon$
$IDDLE, R_2 : \langle \delta, R_1\rangle$
$WFUR, R_1 : \langle ?I3, R_1\rangle$
$TMP3, R_1 : \langle ?I3, R_1\rangle\langle ?I4, R_1\rangle$
$TMP3, R_2 : \langle ?I3, R_1\rangle\langle ?I4, R_1\rangle\langle \delta, R_1\rangle$
$WFUR, R_2 : \langle ?I3, R_1\rangle\langle \delta, R_1\rangle$
$WFUR, R_4 : \langle ?I3, R_1\rangle\langle \delta, R_1\rangle\langle \delta, R_2\rangle$
$WFUR, R_5 : \langle ?I3, R_1\rangle\langle \delta, R_1\rangle\langle \delta, R_2\rangle\langle \delta, R_4\rangle$
$WFUR, R_6 : \langle ?I3, R_1\rangle\langle \delta, R_1\rangle\langle \delta, R_2\rangle\langle \delta, R_4\rangle\langle \delta, R_5\rangle$
$TMP5, R_4 : \langle ?I3, R_1\rangle\langle \delta, R_1\rangle\langle \delta, R_2\rangle\langle ?I1, R_4\rangle$
$TMP5, R_5 : \langle ?I3, R_1\rangle\langle \delta, R_1\rangle\langle \delta, R_2\rangle\langle \delta, R_4\rangle\langle ?I1, R_5\rangle$
$TMP5, R_6 : \langle ?I3, R_1\rangle\langle \delta, R_1\rangle\langle \delta, R_2\rangle\langle \delta, R_4\rangle\langle \delta, R_5\rangle\langle ?I1, R_6\rangle$
$DP, R_2 : \langle ?I2, R_1\rangle$
$DAcc, R_3 : \langle ?I2, R_1\rangle\langle ?I7, R_2\rangle$
$TMP1, R_3 : \langle ?I2, R_1\rangle\langle ?I7, R_2\rangle\langle ?I16, R_3\rangle$
$TMP4, R_3 : \langle ?I2, R_1\rangle\langle ?I7, R_2\rangle\langle ?I4, R_3\rangle$
$TMP4, R_4 : \langle ?I2, R_1\rangle\langle ?I7, R_2\rangle\langle ?I4, R_3\rangle\langle \delta, R_3\rangle$
$DAcc, R_4 : \langle ?I2, R_1\rangle\langle ?I7, R_2\rangle\langle \delta, R_3\rangle$
$DAcc, R_5 : \langle ?I2, R_1\rangle\langle ?I7, R_2\rangle\langle \delta, R_3\rangle\langle \delta, R_5\rangle$
$TMP3, R_4 : \langle ?I2, R_1\rangle\langle ?I7, R_2\rangle\langle \delta, R_3\rangle\langle ?I11, R_4\rangle$
$TMP3, R_5 : \langle ?I2, R_1\rangle\langle ?I7, R_2\rangle\langle \delta, R_3\rangle\langle \delta, R_5\rangle\langle ?I11, R_5\rangle$
$DE, R_6 : \langle ?I2, R_1\rangle\langle ?I7, R_2\rangle\langle \delta, R_3\rangle\langle ?I11, R_4\rangle\langle \delta, R_4\rangle\langle !O6, R_5\rangle\langle \delta, R_5\rangle$
$TMP4, R_6 : \langle ?I2, R_1\rangle\langle ?I7, R_2\rangle\langle \delta, R_3\rangle\langle ?I11, R_4\rangle\langle \delta, R_4\rangle\langle !O6, R_5\rangle\langle \delta, R_5\rangle\langle ?I15, R_6\rangle$

Ensemble de couverture de transitions P :

$\langle ?I2, R_1\rangle$
$\langle ?I2, R_1\rangle\langle ?I7, R_2\rangle$
$\langle ?I2, R_1\rangle\langle ?I7, R_2\rangle\langle ?I16, R_3\rangle$
$\langle ?I2, R_1\rangle\langle ?I7, R_2\rangle\langle ?I16, R_3\rangle\langle !O5, R_3\rangle$
$\langle ?I2, R_1\rangle\langle ?I7, R_2\rangle\langle ?I4, R_3\rangle$
$\langle ?I2, R_1\rangle\langle ?I7, R_2\rangle\langle ?I4, R_3\rangle\langle !O3, R_3\rangle$
$\langle ?I2, R_1\rangle\langle ?I7, R_2\rangle\langle ?I4, R_3\rangle\langle \delta, R_3\rangle$
$\langle ?I2, R_1\rangle\langle ?I7, R_2\rangle\langle ?I4, R_3\rangle\langle \delta, R_3\rangle\langle !O3, R_4\rangle$
$\langle ?I2, R_1\rangle\langle ?I7, R_2\rangle\langle \delta, R_3\rangle$
$\langle ?I2, R_1\rangle\langle ?I7, R_2\rangle\langle \delta, R_3\rangle\langle ?I11, R_4\rangle$
$\langle ?I2, R_1\rangle\langle ?I7, R_2\rangle\langle \delta, R_3\rangle\langle \delta, R_5\rangle$
$\langle ?I2, R_1\rangle\langle ?I7, R_2\rangle\langle \delta, R_3\rangle\langle \delta, R_5\rangle\langle ?I11, R_5\rangle$
$\langle ?I2, R_1\rangle\langle ?I7, R_2\rangle\langle \delta, R_3\rangle\langle ?I11, R_4\rangle\langle !O6, R_4\rangle$
$\langle ?I2, R_1\rangle\langle ?I7, R_2\rangle\langle \delta, R_3\rangle\langle \delta, R_5\rangle\langle ?I11, R_5\rangle\langle !O6, R_5\rangle$
$\langle ?I2, R_1\rangle\langle ?I7, R_2\rangle\langle \delta, R_3\rangle\langle ?I11, R_4\rangle\langle \delta, R_4\rangle\langle !O6, R_5\rangle\langle \delta, R_5\rangle$
$\langle ?I2, R_1\rangle\langle ?I7, R_2\rangle\langle \delta, R_3\rangle\langle ?I11, R_4\rangle\langle \delta, R_4\rangle\langle !O6, R_5\rangle\langle \delta, R_5\rangle\langle ?I15, R_6\rangle$
$\langle ?I2, R_1\rangle\langle ?I7, R_2\rangle\langle \delta, R_3\rangle\langle ?I11, R_4\rangle\langle \delta, R_4\rangle\langle !O6, R_5\rangle\langle \delta, R_5\rangle\langle ?I15, R_6\rangle\langle !O9, R_6\rangle$
$\langle ?I3, R_1\rangle$
$\langle ?I3, R_1\rangle\langle ?I4, R_1\rangle$
$\langle ?I3, R_1\rangle\langle ?I4, R_1\rangle\langle !O3, R_1\rangle$
$\langle \delta, R_1\rangle$
$\langle \delta, R_1\rangle\langle ?I3, R_2\rangle$
$\langle ?I3, R_1\rangle\langle \delta, R_1\rangle\langle ?I4, R_2\rangle$
$\langle ?I3, R_1\rangle\langle \delta, R_1\rangle\langle ?I4, R_2\rangle\langle !O3, R_2\rangle$
$\langle ?I3, R_1\rangle\langle \delta, R_1\rangle\langle \delta, R_2\rangle$
$\langle ?I3, R_1\rangle\langle \delta, R_1\rangle\langle \delta, R_2\rangle\langle ?I1, R_4\rangle$
$\langle ?I3, R_1\rangle\langle \delta, R_1\rangle\langle \delta, R_2\rangle\langle ?I1, R_4\rangle\langle !O1, R_4\rangle$
$\langle ?I3, R_1\rangle\langle \delta, R_1\rangle\langle \delta, R_2\rangle\langle \delta, R_4\rangle$
$\langle ?I3, R_1\rangle\langle \delta, R_1\rangle\langle \delta, R_2\rangle\langle \delta, R_4\rangle\langle ?I1, R_5\rangle$
$\langle ?I3, R_1\rangle\langle \delta, R_1\rangle\langle \delta, R_2\rangle\langle \delta, R_4\rangle\langle ?I1, R_5\rangle\langle !O1, R_5\rangle$
$\langle ?I3, R_1\rangle\langle \delta, R_1\rangle\langle \delta, R_2\rangle\langle \delta, R_4\rangle\langle \delta, R_5\rangle$
$\langle ?I3, R_1\rangle\langle \delta, R_1\rangle\langle \delta, R_2\rangle\langle \delta, R_4\rangle\langle \delta, R_5\rangle\langle ?I1, R_6\rangle$
$\langle ?I3, R_1\rangle\langle \delta, R_1\rangle\langle \delta, R_2\rangle\langle \delta, R_4\rangle\langle \delta, R_5\rangle\langle ?I1, R_6\rangle\langle !O1, R_6\rangle$

Ensemble $R = P - Q$:

$\langle ?I2, R_1\rangle\langle ?I7, R_2\rangle\langle ?I16, R_3\rangle\langle !O5, R_3\rangle$
$\langle ?I2, R_1\rangle\langle ?I7, R_2\rangle\langle ?I4, R_3\rangle\langle !O3, R_3\rangle$
$\langle ?I2, R_1\rangle\langle ?I7, R_2\rangle\langle ?I4, R_3\rangle\langle \delta, R_3\rangle\langle !O3, R_4\rangle$
$\langle ?I2, R_1\rangle\langle ?I7, R_2\rangle\langle \delta, R_3\rangle\langle ?I11, R_4\rangle\langle !O6, R_4\rangle$
$\langle ?I2, R_1\rangle\langle ?I7, R_2\rangle\langle \delta, R_3\rangle\langle \delta, R_5\rangle\langle ?I11, R_5\rangle\langle !O6, R_5\rangle$
$\langle ?I2, R_1\rangle\langle ?I7, R_2\rangle\langle \delta, R_3\rangle\langle ?I11, R_4\rangle\langle \delta, R_4\rangle\langle !O6, R_5\rangle\langle \delta, R_5\rangle\langle ?I15, R_6\rangle\langle !O9, R_6\rangle$
$\langle ?I3, R_1\rangle\langle ?I4, R_1\rangle\langle !O3, R_1\rangle$
$\langle \delta, R_1\rangle\langle ?I3, R_2\rangle$
$\langle ?I3, R_1\rangle\langle \delta, R_1\rangle\langle ?I4, R_2\rangle$
$\langle ?I3, R_1\rangle\langle \delta, R_1\rangle\langle ?I4, R_2\rangle\langle !O3, R_2\rangle$

$\langle ?I3, R_1\rangle\langle \delta, R_1\rangle\langle \delta, R_2\rangle\langle ?I1, R_4\rangle\langle !O1, R_4\rangle$
$\langle ?I3, R_1\rangle\langle \delta, R_1\rangle\langle \delta, R_2\rangle\langle \delta, R_4\rangle\langle ?I1, R_5\rangle\langle !O1, R_5\rangle$
$\langle ?I3, R_1\rangle\langle \delta, R_1\rangle\langle \delta, R_2\rangle\langle \delta, R_4\rangle\langle \delta, R_5\rangle\langle ?I1, R_6\rangle\langle !O1, R_6\rangle$

Index

Bibliographie

[ACD90] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *Proceedings* 5^{th} *Annual Symposium on Logic in Computer Science,* Philadelphia, USA, pages 414–425. IEEE Computer Society Press, 1990.

[ACH+92] R. Alur, C. Courcoubetis, N. Halbwachs, D. Dill, and H. Wong-Toi. Minimization of timed transition systems. In R. Cleaveland, editor, *Proceedings CONCUR 92,* Stony Brook, NY, USA, volume 630 of *Lecture Notes in Computer Science*, pages 340–354. Springer-Verlag, 1992.

[AD92] R. Alur and D. Dill. The theory of timed automata. In J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg, editors, *Proceedings REX Workshop on Real-Time : Theory in Practice,* Mook, The Netherlands, June 1991, volume 600 of *Lecture Notes in Computer Science*, pages 45–73. Springer-Verlag, 1992.

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

[AFH99] R. Alur, L. Fix, and T. A. Henzinger. Event-clock automata : a determinizable class of timed automata. In *Theoretical Computer Science*, pages 253–273, 1999.

[AH90] L. Aceto and M. Hennessy. Adding action refinement to a finite process algebra. Report 6/90, Computer Science, University of Sussex, Brighton, 1990.

[AKA00] R. Dssouli A. Khoumsi, A. En-Nouaary and M. Akalay. A new method for testing real time system. In *Proceedings of the 7th International Conference on Real-Time Computing Systems and Applications, RTCSA'00 (Cheju Island, South Korea),IEEE Computer Society*, pages 441–449, December 2000.

[AS96] Ahmet F. Ates and Behcet Sarikaya. Test Sequence Generation and timed Testing. *Computer networks ans ISDN systems*, 29 :107–131, 1996.

[BB88] J.C.M. Baeten and J.A. Bergstra. Global renaming operators in concrete process algebra. *Information and Computation*, 78(3) :205–245, 1988.

[BD88] S. Budkowski and P. Dembinski. An introduction to estelle : A specification language for distribued systems. *Computer Networks and ISDN Systems*, 14 :3–24, January 1988.

[BDDD91] G.v. Bochmann, G. Das, R. Dssouli, and M. Dubuc. Fault Models in Testing. In *Proceedings of the International Workshop on Testing of Communicating Systems IWTCS'91*, 1991.

[Ben94] R. G. Bennetts. Progress in design for test : A personal view. In *IEEE Design an Test of Computers*, 1994.

[BFG+00] M. Bozga, J.C. Fernandez, L. Ghirvu, C. Jard, T. Jéron, A. Kerbrat, P. Morel, and L. Mounier. Verification and test generation for the sscop protoocl. *Science of Computer Programming*, 36(1) :27–52, 2000.

[BGM91] G. Bernot, M. Gaudel, and B. Marre. Software testing based on formal specifications : a theory and tool. In *Software Engineering Journal*, 1991.

[BL95] J. Bengtsson and F. Larsson. *UPPAAL - a Tool Suite for Symbolic and Compositional Verification of Real-Time Systems (Draft)*, 1995.

[BLL+95] J. Bengtsson, K.G. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL - a tool suite for automatic verification of real-time systems. In *4th. DIMACS Workshop on Verification and Control of Hybrid Systems*, New Brunswick, New Jersey, October 1995.

[Bri87] E. Brinksma. A theory for the derivation of tests. In S. Aggarwal, editor, *Proceedings of the Eighth International Conference on Protocol Specification, Testing and Verification*. North-Holland, 1987.

[Bri88] E. Brinksma. *On the design of Extended LOTOS – a specification language for open distributed systems*. PhD thesis, Department of Computer Science, University of Twente, 1988.

[CA92a] S. Chamberlain and P. Amer. Formal Specification of Real-Time Constraints in ESTELLE. *Réseaux et Informatique Répartie*, 2, 1992.

[CA92b] W. Chung and P. Amer. Improved on UIO Sequence Generation and Partial UIO Sequences. In R.J. Linn and M.U. Uyar, editors, *Protocol Specification, Testing, and Verification, XII,* Lake Buena Vista, Florida, USA. North-Holland, June 1992.

[CAR] N. CARRERE. Dsm specification in lotos and test cases generation. *INT (French Telecommunication National Institute).*

[CCKS95] R. Castanet, C. Chevrier, O. Koné, and B. Le Saec. An Adaptive Test Sequence Generation Method for the User Needs. In *Proceedings of IWPTS'95, Evry, France*, 1995.

[CdO95] J.-P. Courtiat and R. de Oliveira. RT-LOTOS and its formal design of multimedia protocol. In *Annals of telecommunications*, volume 50, 1995.

[CE81] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In D. Kozen, editor, *Proceedings of the Workshop on Logic of Programs,* Yorktown Heights, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer-Verlag, 1981.

[CGPT95] M. Clatin, R. Groz, M. Phalippou, and R. Thummel. Two approaches linking a test generation tool with verification techniques. In *Proceedings of IWPTS'95, Evry, France*, 1995.

[Cho78] T.S. Chow. Testing software design modeled by finite-state machines. *IEEE Transactions on Software Engineering*, SE-4(3) :178–187, 1978.

[CL97] D. Clarke and I. Lee. Automatic generation of tests for timing constraints from requirements. In *Proceedings of the Third International Workshop on*

Object-Oriented Real-Time Dependable Systems, Newport Beach, California, February 1997.

[COG98] R. Cardel-Oliver and T. Glover. A practical and complete algorithm for testing real-time systems. In *Proc. of the 5th. Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 1486 of Lecture Notes in Computer Science, pages 251–261. SpringerVerlag, 1998.

[CSLO99] J-P. Courtiat, C.A.S. Santos, C. Lohr, and B. Outtaj. Experience with rt-lotos, a temporal extension of the lotos formal description technique. In *LAAS-CNRS*, 1999.

[DAdSS01] K. Drira, P. Azema, and P. de Saqui Sannes. Testability analysis in communicating systems. *Computer Networks*, 36 :671–693, 2001.

[DNH84] R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34 :83–133, 1984.

[DOTY95] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool Kronos. In R. Alur, T.A. Henzinger, and E.D. Sontag, editors, *Hybrid Systems III*, volume 1066 of *Lecture Notes in Computer Science*, pages –. Springer-Verlag, 1995.

[DY96] C. Daws and S. Yovine. Reducing the number of clock variables of timed automata. In *Proceedings of the 1996 IEEE Real-Time Systems Symposium, RTSS'96,* Washington DC, USA. IEEE Computer Society Press, 1996.

[EFH83] H. Ehrig, W. Fey, and H. Hansen. Act one : An algebraic language with two levels of semantics. Technical Report 83-03, Technische Universität Berlin, 1983.

[EM85] H. Ehrig and B. Mahr. *Fundamentals of algebraic specifications I*, volume 6 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1985.

[ENDE97] A. En-Nouaary, R. Dssouli, and A. Elqortobi. Génération de tests temporisés. In *Proceedings of the* 6^{th} *bi-Annual Colloque Francophone de l'ingénierie des Protocoles,* Lièges, Belgique, 1997.

[ENDKE98] A. En-Nouaary, R. Dssouli, F. Khendek, and A. Elqortobi. Timed test cases generation based on state characterization technique. In *19th IEEE Real Time Systems Symposium (RTSS'98)* Madrid, Spain, 1998.

[ENFDE97] A. En-Nouaary, H. Fouchal, R. Dssouli, and A. Elqortobi. Test derivation for timed systems. Report LERI-97-09-01, LERI-RS (Université de Reims), 1997.

[FBK+91] S. Fujiwara, G. von Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi. Test selection based on finite-state models. *IEEE Transactions on Software Engineering*, 17(6) :591–603, June 1991.

[Fis96] S. Fischer. Real time estelle. In *Reihe Informatik, University of Mannheim*, March 1996.

[FJJV96] J. Cl. Fernandez, C. Jard, T. Jéron, and C. Viho. Using on-the-fly verification techniques for the generation of test suites. In *CAV'96. LNCS 1102 Springer Verlag*, 1996.

[FPS00] H. Fouchal, E. Petitjean, and S. Salva. Testing Timed Systems with Timed Purposes. In *Proceedings of the 7th International Conference on Real-Time Computing Systems and Applications, RTCSA'00 (Cheju Island, South Korea),IEEE Computer Society*, pages 166–171, December 2000.

[Fre91] R. S. Freedman. Testability of software components. *IEEE transactions on Software Engineering*, 17(6), june 1991.

[Ga00] J. Grabowski and al. On the design of the new testing language ttcn-3. In *Testing of Communicating Systems*, volume 13, pages 161–176, 2000.

[GFS90] M.C. Gaudel, C. Froidevaux, and M. Soria. Types de données et algorithmes. In *Mc Graw Hill*, 1990.

[Gil62] A. Gill. *Introduction to the theory of finite-state machines.* Mc Graw-Hill, New York – USA, 1962.

[GJ95] L. Gallon and G. Juanole. Critical Time Distributed Systems : Qualitative and Quantitative Analysis based on Stochastic Petri Nets. In *Proceedings of the 8th International Conference on Formal Description Techniques FORTE'95 (Montreal, Canada)*, October 1995.

[Gon80] G. Gonenc. A method for the design of fault detection experiment. *IEEE transactions on Computers*, C-19 :551–558, 1980.

[Gra00] J. Grabowski. Ttcn-3 - a new test specification language for black-box testing of distributed systems. In *17th International Conference and Exposition on Testing Computer Software (TCS'2000),Washington D.C.*, June 2000.

[HHWT95] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech : The next generation, 1995. Manuscript.

[HM01] R.B. Yehezkael H.G. Mendelbaum. Using 'parallel automaton' as a single notation to specify, design and control small computer based systems. *IEEE-ECBS2001 conf, Washington*, April 2001.

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

[Hol90] Gerard J. Holzmann. *Design and Validation of Computer Protocols.* Prentice Hall, November 1990.

[ISO87] ISO. *Information processing systems – open systems interconnection – LOTOS – a formal description technique based on the temporal ordering of observational behaviour* ISO/TC97/SC21/N DIS8807, 1987.

[ISO88] ISO. ESTELLE — A formal description technique based on extended state transition model. International Standard 9074, International Organization for Standardization — Information Processing Systems — Open Systems Interconnection, Genève, September 1988.

[ISO91a] ISO. Conformance Testing Methodology and Framework. International Standard 9646, International Organization for Standardization — Information Technology — Open Systems Interconnection, Genève, 1991.

[ISO91b] ISO. Conformance Testing Methodology and Framework-part 1-5. International Standard 9646, International Organization for Standardization — Information Technology — Open Systems Interconnection, Genève, 1991.

[Kar97] K. Karoui. *Conception de logiciels de communication testables.* PhD thesis, Departement d'Informatique et de Recherche Opérationnelle, Montréal, june 1997.

[KC95] M. Kim and S. T. Chanson. Design for testability of protocols based on formal specifications. In *Proceedings of the 8th Internationnal Workshop on Protocol Test Systems, Evry, France*, September 1995.

[KDC96] K. Karoui, R. Dssouli, and O. Cherkaoui. Specification transformations and design for testability. In *IEEE Globecom'96, Londre*, November 1996.

[KDCK94] K. Karoui, R. Dssouli, O. Cherkaoui, and A. Khoumsi. Estimation de la testabilité d'un logiciel modélisé par les relations. 1994. research report #921.

[KGD96] K. Karoui, A. Ghedamsi, and R. Dssouli. A study of some influencing factors in testability and diagnostics based on fsms. 1996. rapport de recherche #1048.

[Lau99] P. Laurençot. *Integration du temps dans les tests de protocoles de communication.* PhD thesis, Univ. of Bordeaux 1, January 1999.

[LC97] Patrice Laurencot and Richard Castanet. Integration of Time in Canonical Testers for Real-Time Systems. In *International Workshop on Object-Oriented Real-Time Dependable Systems, California.* IEEE Computer Society Press, 1997.

[LLdFQ95] G. Leduc, L. Leonard, D. de Frutos, and J. Quemada. Time Extended LOTOS. In *Revised Working Draft on Extended LOTOS,* ISO/IEC JTC1/SC 21/WG7, 1995.

[LPB93] G. Luo, A. Pentrenko, and G.v Bochman. Selecting test sequences for partially-specified nondeterministic finite state machines. feb 1993. research report #864.

[LPvB94] G. Luo, A. Petrenko, and G. v. Bochmann. Selecting Test Sequences for Partially-specified Non-deterministic Finite State Machines. In *Proceedings of the 7th International Workshop on Protocol Test System IWPTS'94 (Tokyo, Japan)*, Amsterdam, september 1994. North-Holland.

[MF76] P. Merlin and D.J. Faber. Recoverability of Communication Protocols. *IEEE Trans Comm*, 24, 1976.

[Mil80] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science.* Springer-Verlag, 1980.

[Mil90] A. Mili. Program fault tolerance. *Prentice Hall*, 1990.

[MY96] O. Maler and S. Yovine. Hardware timing verification using KRONOS. In *Proceedings of the IEEE 7th Israeli Conference on Computer Systems and Software Engineering, ICCBSSE'96.* IEEE Computer Society Press, June 1996.

[NS01] Brian Nielsen and Arne Skou. Automated Test Generation from Timed Automata. In T. Margaria and W. Yi, editors, *Proceedings of the Workshop on Tools and Algorithms for the Construction and Analysis of Systems,* Genova, Italy, volume 2031 of *Lecture Notes in Computer Science*, pages 343–357. Springer-Verlag, April 2001.

[NT81] S. Naito and M. Tsunoyama. Fault Detection for Sequential Machines by Transition- Tours. *Proceedings of Fault Tolerant Computer Systems*, pages 238–243, 1981.

[oSET90] ANSI/IEEE Standard Glossary of Software Engineering Terminology. Ansi/ieee standard 610.12-1990. In *IEEE Press, New York*, 1990.

[Ost92] J.S Ostroff. Formal methods for the specification and design of real-time safety critical systems. *Systems and Software*, 1992.

[Ost94] J.S Ostroff. Specifying and verifying real-time reactive systems in ttm/rttl. august 1994. Technical report, York University.

[OY93] A. Olivero and S. Yovine. KRONOS *: A Tool for Verifying Real-time Systems. User's Guide and Reference Manual (Draft 0.0)*, August 1993.

[PDK93] A. Petrenko, R. Dssouli, and H. Koenig. On evaluation of testability of protocol structures. In *Proc. Int. Workshop on Protocol Test System (IFIP), Pau, France*, 1993.

[Pet00] Eric Petitjean. Etude des méthodes de test sur les systèmes temporisés, thèse de doctorat, université de reims champagne ardenne. November 2000.

[PF99a] Eric Petitjean and Hacène Fouchal. From Timed Automata to Testable Untimeed Automata. In *24th IFAC/IFIP International Workshop on Real-Time Programming, Schloss Dagstuhl, Germany*, 1999.

[PF99b] Eric Petitjean and Hacène Fouchal. A Realistic Architecture for Timed Systems. In *5th IEEE International Conference on Engineering of Complex Computer Systems, Las Vegas, USA*, pages 109–118, 1999.

[PF00] Eric Petitjean and Hacène Fouchal. A fault detection on timed systems. Report : Resycom-2000-01-01, LERI-RESYCOM (Université de Reims), 2000.

[Pha91] M. Phalippou. Tveda : an experiment in computer-aided test case generation from formal specifications of protocols. In *Note Technique NT/LAA/SLC/347, CNET*, January 1991.

[Pha94] M. Phalippou. *Relation d'implantation et hypothèses de test sur des automates à entrées et sorties.* PhD thesis, Univ. of Bordeaux, September 1994.

[PLC98] O. Koné P. Laurencot and R. Castanet. On the Fly Test Generation for Real Time Protocols. In *International Conference on Comnputer Communications and Networks, Louisiane U.S.A*, 1998.

[Pro82] R. L. Probert. Life-cycle/ grey box testing, winnipeg, canada. In *Congressus Numerantium*, volume 34, 1982.

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

[Ray87] D. Rayner. OSI conformance testing. *Computer Networks and ISDN Systems*, 14 :79–98, 1987.

[RBC92] N. Rico, G.v. Bochmann, and O. Cherkaoui. Model-checking for real-time systems specified in LOTOS. In *Computer Aided Verification, LNCS 663*, 1992.

[RLP+98] J. Rengsston, K. Larsen, P. Pettersson, W. Yi, F. Larsson, and C. Weise. New generation of uppaal. *BRICS, Aalborg University, Denmark, Departement of Computer systems, Uppsala University, sweden*, 1998.

[RNHW98] P. Raymond, X. Nicollin, N. Halbwatchs, and D. Waber. Automatic testing of reactive systems, madrid, spain. In *Proceedings of the 1998 IEEE Real-Time Systems Symposium, RTSS'98*, pages 200–209. IEEE Computer Society Press, December 1998.

[SD88a] K. Sabnani and A. Dabhura. A protocol test generation procedure. *Computer networks ans ISDN systems*, 15 :285–297, 1988.

[SD88b] K. Sabnani and A. Dahbura. A protocol test generation procedure. *Computer Networks and ISDN Systems*, 15 :285–297, 1988.

[SF01a] S. Salva and H. Fouchal. Paramètres pour la testabilité des systèmes temporisés. In *3eme Colloque sur la Modélisation et Simulation des Systèmes, MOSIM'01 (Troyes, France)*, April 2001.

[SF01b] S. Salva and H. Fouchal. Some Paramètres for Timed System Testability. In *ACS/IEEE International Conference on Computer System and Applications, AICCS'01 (Beirut, Lebanon)*, June 2001.

[SF01c] S. Salva and H. Fouchal. Timed test execution and ttcn generation. In *ICIS, International Conference on Computer and Information Systems*, August 2001.

[SFB00] S. Salva, H. Fouchal, and S. Bloch. Metrics for Timed Systems Testing. In *4th OPODIS International Conference on Distribued Systems, Paris*, December 2000.

[Sif80] J. Sifakis. Performance Evaluation of Systems Using Nets. *Lecture Notes in Computer Science*, 84, 1980.

[SLD92] Y.N. Shen, F. Lombardi, and A.T. Dabuhra. Protocol Conformance Testing by Multiple uio Sequences. *IEEE Transactions on Communications*, 40 :1282–1287, 1992.

[SPF01] S. Salva, E. Petitjean, and H. Fouchal. A simple approach to testing timed systems. In *FATES01 (Formal Approaches for Testing Software), Aalborg, Denmark*, August 2001.

[SVD01] J. Springintveld, F.W. Vaandrager, and P. R. D'Argenio. Testing Timed Automata. Technical Report 254, 2001.

[SY96] J. Sifakis and S. Yovine. Compositional specification of timed systems. In *Proceedings of the 13th Annual Symp. on Theoretical Aspects of Computer Science, STACS'96*, volume 1046 of *Lecture Notes in Computer Science*, pages 347–359, Grenoble, France, 1996. Springer-Verlag.

[Tar41] A. Tarski. On the calculus of relations. *The Journal of Symbolic Logic*, 6 :73–98, 1941.

[TB99] J. Tretmans and A. Belinfante. Automatic testing with formals methods. *EuroSTAR'99, 7th European International Conference of Software Testing*, November 1999.

[TC96] S. Tripakis and C. Courcoubetis. Extending promela and spin for real time. In T. Margaria and B. Steffen, editors, *Proceedings of the Second Workshop on Tools and Algorithms for the Construction and Analysis of Systems,* Passau, Germany, volume 1055 of *Lecture Notes in Computer Science*, pages 329–348. Springer-Verlag, April 1996.

[TPB96] A. Tan, A. Petrenko, and G.v. Bochmann. A Test Generation Tool for Specifications in the Form of State Machine. Report, 1016, DIRO, Université de Montréal, Canada, 1996.

[Tre96] J. Tretmans. Conformance testing with labelled transition systems : Implementation relations and test generation. *Computer Networks and ISDN Systems*, 29 :49–79, 1996.

[VCI89] S. Vuong, W. Chan, and M. Ito. The UIOv-Method for Protocol Test Sequence Generation. In *2nd IWPTS International Workshop on Protocol Test Systems, Berlin*, 1989.

[VM95] J. M. Voas and K. W. Miller. Software testability : The new verification. In *IEEE Software*, May 1995.

[WG97] T. Walter and J. Grabowski. Real-time ttcn for testing real time and multimedia systems. In *International Workshop on Testing of Communicating Systems, Corée*, 1997.

[WG99] T. Walter and J. Grabowski. A framework for the specification of test cases for real-time systems. In *Towards the third edition of TTCN, 11th Journal of Information and Software Technology*, 1999.

[YL93] M. Yannakakis and D. Lee. An efficient algorithm for minimizing real-time transition systems (Extended abstract). In C. Courcoubetis, editor, *Proceedings of the 5th International Conference on Computer Aided Verification,* Elounda, Greece, volume 697 of *Lecture Notes in Computer Science*, pages 210–224. Springer-Verlag, 1993.

[Yov93] S. Yovine. *Méthodes et outils pour la vérification symbolique de systèmes temporisés*. PhD thesis, Institut National Politechnique de Grenoble, France, May 1993.

[YPKP95] N. Yevtushenko, A. Petrenko, K. Karoui, and S. Prokopenko. On the design for testability of communication protocols. *IWPTS'95*, 1995.

[Z.198] CCITT Recommendation Z.100. Specification and Description Language SDL. *Annexes A–F to Z.100*, VI.20–VI.24, 1998.

Étude de la qualité de test des systèmes temporisés

Résumé

Le coût de la validation d'un système dépasse souvent 70% du coût total de son développement, et ne cesse de croître de par la complexité de plus en plus accrue de ces systèmes et l'ajout de concepts tel que le temps. Il est par conséquent nécessaire de proposer des méthodes permettant de réduire ce coût tout en obtenant des systèmes pouvant être complètement testés. Pour ce faire, il est nécessaire d'évaluer ce coût et d'évaluer les parties du système qui pourront être testées. Cette évaluation est appelée la **Qualité de test**.

Cette thèse a pour but de répondre aux problèmes de coût et de qualité.

Une première partie concerne l'étude de la qualité de test des automates temporisés d'Alur et Dill et des graphes des régions. Cette évaluation s'effectue à travers des paramètres de testabilité. Nous montrons comment modifier les systèmes en vue d'obtenir une meilleure qualité, et donc en vue de réduire les coûts de test et d'améliorer la détection des erreurs.

Une seconde partie a pour but de montrer comment réduire le coût du test, en proposant de nouvelles méthodes de test. La première est une méthode orientée Objectif de test. Cette méthode permet de vérifier n'importe quelle propriété sur l'implantation. La seconde méthode est basée sur la caractérisation d'états de graphe des régions. L'originalité de cette méthode est de caractériser des états temporisés sans apporter aucune modification sur la spécification.

Une dernière partie propose une approche au problème de la génération du graphe des régions : la méthode décrite, permet de minimiser les automates temporisés en graphe des régions.

Mots-clés : test de conformité, automate temporisé, graphe des régions, qualité de test, méthodologies de test.