Déterminisation des automates temporisés avec durées d'actions pour le test formel

Ilhem KITOUNI

Université de Mentouri, 25000 Constantine

A toute personne fière dans ce monde...

Remerciements

Je remercie Monsieur KOLLADI Mohamed Khireddine, Docteur à l'Université Mentouri de Constantine, qui m'a fait l'honneur de présider ce jury.

Je remercie Monsieur SAIDOUNI Djamel-Eddine, Docteur à l'Université Mentouri de Constantine, d'avoir bien voulu accepter la charge de rapporteur. Merci pour toutes ces remarques qui m'ont permis d'améliorer ce travail.

Je tiens également à remercier Monsieur CHAOUI Allaoua, Docteur à l'Université Mentouri de Constantine et Monsieur CHIKHI Salim, Docteur à l'Université Mentouri de Constantine, qui ont accepté d'examiner ce travail.

Je remercie particulièrement Monsieur SAIDOUNI Djamel-Eddine, qui a encadré ce travail et qui m'a fait confiance dès le premier entretien. Une grande reconnaissance pour sa disponibilité, son soutien et sa compréhension, qui ont permis à ce travail d'aboutir malgré toutes mes contraintes personnelles.

Je tiens également à témoigner du soutien inconditionnel de Monsieur DRIS Djamel, Président Directeur Général du Laboratoire des Travaux Publics de l'Est, ainsi que mes collègues : Monsieur BENZEGOUTA Abdelbaki, Monsieur ZIOUANI Med Salah, Madame BENTCHIKOU Mimosa et enfin mais pas le moindre de Mademoiselle BOUARIOUA Lamia.

Je dédie ce modeste résultat à ma famille, mon mari Toufik, mes filles et particulièrement à ma sœur, pour leur appui inébranlable.

Résumé

La complexité croissante des systèmes informatiques et les risques d'erreurs inhérents à leur conception sont des problèmes de plus en plus croissants. En effet, des conséquences dramatiques sont souvent causées par un comportement erroné dans une implémentation critique. Cependant ils ont toutes un point commun : La disproportion entre le coût des dégâts et la simplicité des erreurs à la source du problème. Diverses techniques, modèles formels et outils existent, permettent de garantir le bon fonctionnement du système spécifié, parmi celles offertes aujourd'hui : le Model-Checking , le Theorem-Proving et l'approche Test.

Dans ce travail on s'intéresse au test de conformité et au souci de formalisation de la génération du test. Cette étude concerne la phase de spécification des systèmes concurrents en vue d'une possible automatisation de la génération des tests. Les exigences de délais de plus en plus fortes, justifies le choix du modèle des automates temporisés avec durées d'actions (DATAs). Ce modèle étant aux frontières des modèles discrets et ceux temporels, il est temporisé, manipulant les durées des actions explicitement sous forme de contraintes temporelles. Cet aspect qui lui est spécifique a orienté notre étude du modèle.

Les travaux présentés dans ce mémoire portent sur les points suivants : (i) Le déterminisme, (ii) L'implémentabilité, (iii) La robustesse par rapport aux propriétés attendues du modèle des DATAs.

Les deux qualités souhaitables pour l'implémentation des modèles sont : La conservation des propriétés par passage à l'implémentation et la propriété dite « faster is better ». Dans cette vision des modèles, on est certain que la maîtrise des durées des actions (quelle que soit leurs natures) permettra de conserver les deux propriétés lors du passage à l'implémentation.

Par souci de praticabilité du modèle des DATAs, sujet de notre étude, nous avons proposé un modèle servant à la production (génération) des tests à base de refus directement générés à partir d'une structure de DATA déterministe. Cette approche de test classique est une technique utilisée pour l'analyse de la testabilité des systèmes et le diagnostic de fautes dans le domaine non temporisé. Dans ce travail nous avons proposé l'extension du modèle des Graphes de refus par la considération de contraintes temporelles en utilisant le modèle des DATAs comme modèle de base et qui a abouti à un graphe nommé Graphe de Refus Temporisés (GRTs).

Mots-clés Test de conformité, Durées d'actions, Spécification formelle, le modèle des DATAs, Graphes de refus, Graphes de refus mixtes.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

1.1 Contexte

La complexité croissante des systèmes informatiques et les risques d'erreurs inhérents à leur conception sont des problèmes de plus en plus croissants. En effet, des conséquences dramatiques sont souvent causées par un comportement erroné dans un logiciel critique. Ils existent de nombreux exemples, tristement célèbres, des pannes ou problèmes ayant entraînés des pertes humaines, des dégâts matériels ou financiers[Pri03][Her01][Fle01]. On rapporte que, en 1990, le réseau téléphonique d'une grande compagnie a rencontré une panne d'une durée de neuf heures. Un logiciel installé sur 114 routeurs s'est déclaré en panne sur toutes ces installations au même moment à cause d'une instruction " break " qui ne s'était pas exécutée correctement. Le nombre d'appels bloqués a été estimé à 5 millions, le coût global de cette panne étant alors considérable.

De même, le crash d'Ariane 5 et le désastre de la Bank of New York, sont d'autres exemples de cas qui auraient pu être évités, grâce à une vérification plus rigoureuse. Ces pannes ont toutes un point commun : La disproportion entre le coût des dégâts qu'elles ont engendrés et la simplicité des erreurs à la source du problème.

Diverses techniques s'appuyant sur des modèles formels ont vu le jour, offrant ainsi un ensemble d'outils permettant de garantir le bon fonctionnement du système spécifié. Selon que l'on souhaite vérifier la partie du système spécifiée formellement ou le produit final, différentes techniques de validation s'appliquent au processus de développement. Les méthodes de vérification d'un ensemble de propriétés sur le modèle formel sont, par exemple, le Model-Checking ou le Theorem-Proving, qui permettent l'analyse du comportement prévu par certaines parties du modèle formel établi à partir de la spécification et de vérifier que le comportement modélisé correspond aux attentes.

- Le **Model-Checking** consiste à parcourir exhaustivement le modèle du système pour vérifier la satisfaction des propriétés comportementales. De cette manière, il est possible de s'assurer que la spécification formelle du système respecte les exigences du cahier des charges.

- Le **Theorem-Proving**, quant à lui, revient à exprimer le modèle et la propriété par un

ensemble d'expressions d'une logique mathématique. La preuve de cette propriété est obtenue à partir d'un certains nombres d'axiomes établis.

Lorsque la validation de la spécification est terminée, le système est implémenté. Ce produit final doit encore être soumis à une phase de validation pour s'assurer qu'il correspond au modèle formel. Cette activité est dite phase de test, elle consiste à soumettre le produit final à un ensemble d'exécutions simulant son environnement après mise en oeuvre : les réponses et réactions du système dans cet environnement d'exécution sont observées et analysées. Si aucun comportement erroné n'est détecté, le produit est prêt pour sa mise en oeuvre définitive.

L'objectif de l'activité de test est donc de vérifier qu'un produit réalisé répond aux exigences définies. Il existe plusieurs catégories de test pour vérifier qu'une implémentation sera capable de fonctionner "correctement" dans une situation réelle (opérationnelle). Parmi ces tests, les tests dits fonctionnels qui s'intéressent aux fonctionnalités d'un ou plusieurs systèmes sans prendre en compte leur structure interne. Les systèmes testés sont dits des "boîtes noires" dont le comportement n'est connu que par les interactions à travers leurs interfaces avec l'environnement. Parmi ces différents types de tests fonctionnels, on peut distinguer :

Le test de conformité, le test d'interopérabilité, le test de robustesse et le test de performances (délais, débit, etc.).

Quelque soit le type de test effectué, l'activité de test fait intervenir un système sous test (*SUT* pour System Under Test), un testeur et l'environnement via lequel le *SUT* et le testeur communiquent. Selon le contexte de test, le *SUT* est composé d'une ou plusieurs implémentations sous test (*IUT* pour « Implementation Under Test »).

L'objectif de l'activité de test est alors de générer à partir des spécifications des systèmes à tester les cas de test (*TC* pour Test Case) exécutables sur le SUT. Généralement, cette génération de test est basée sur un objectif de test (*TP* pour Test Purpose). Un objectif de test décrit une propriété à vérifier lors des tests. Un cas de test est un test élémentaire destiné à tester cet objectif de test. Un ensemble de cas de test compose une suite de test. L'exécution d'un cas de test renvoie un verdict qui peut prendre les valeurs "Pass" si l'objectif de test a été atteint, "Fail" si une erreur a été observée lors des tests, ou "Inconclusive" si le comportement observé est correct par rapport à la ou les spécifications mais ne correspond pas à l'objectif visé par le test.

Plusieurs étapes sont nécessaires avant d'aboutir à l'exécution des tests sur un *SUT*.

Deux phases principales, pour le processus de test, peuvent être distinguées, Figure1.1.

- La première est la phase de spécification ou de génération des tests abstraits (ATS, Abtract Test Suite) pendant laquelle les tests sont spécifiés sous une forme abstraite.
- La seconde phase est celle de la réalisation des tests. Cette phase regroupe l'étape de compilation des tests abstraits en tests exécutable (ETS, Executable Test Suite), l'étape d'exécution des tests exécutables obtenus ainsi que l'analyse des résultats des tests.

Le travail présenté dans ce mémoire concerne la première phase, et particulièrement le

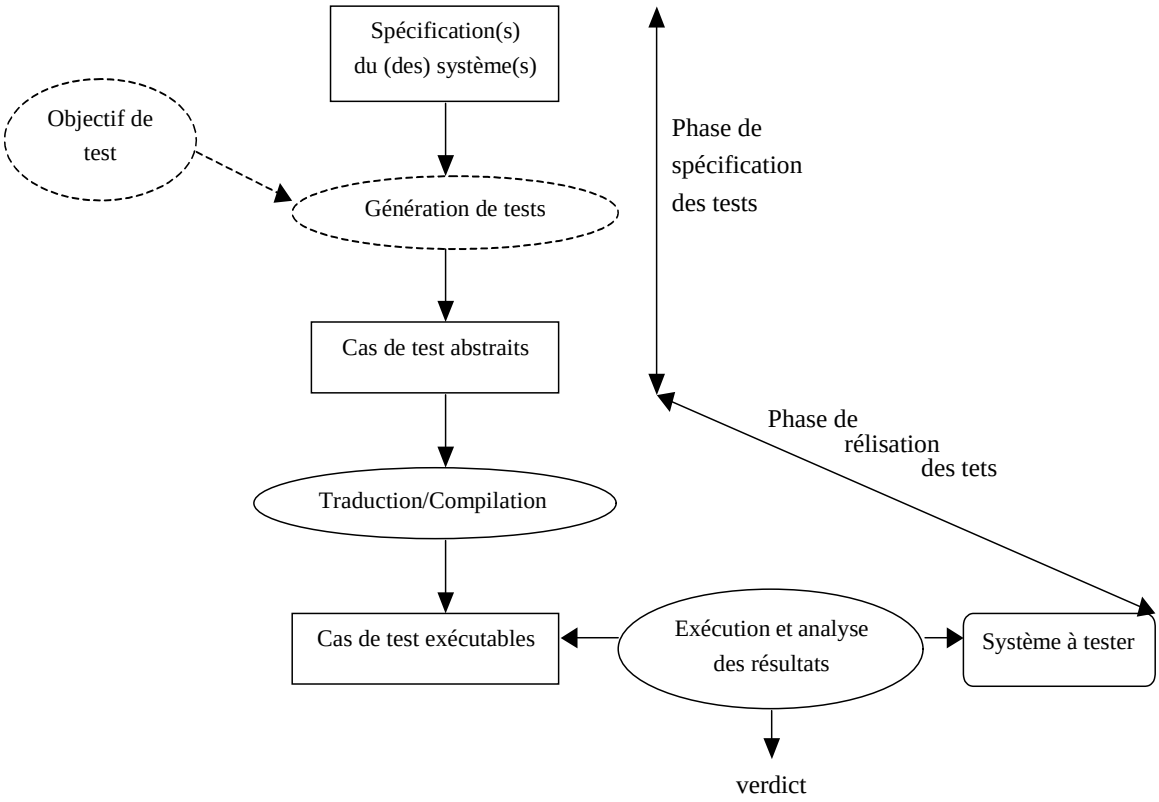

FIG. 1.1 – Etapes de l'activité de test

modèle formel de spécification et modélisation des systèmes en vue d'une possible automatisation de la génération des tests.

1.2 Motivations

L'interconnexion, de plus en plus forte, des systèmes pose souvent une exigence sur les délais d'exécutions du système ou de l'un de ses composants. Un système est dit temps réel si la correction de son fonctionnement est liée au respect de ces temps d'exécutions.

Des modèles et des représentations, formels temps réels, ont été développés. Ainsi les techniques de validation de la spécification et de test du produit final ont été étendues au cas des systèmes temps réel. Cette activité étant très coûteuse dans le cadre d'un développement industriel, des équipes de recherche se penchent sur de nouvelles méthodes pour l'automatisation de la phase de test à partir de la spécification formelle.

A l'heure actuelle, on peut considérer que le test de conformité est relativement précisément défini. En effet, il existe de nombreux travaux dans ce domaine, mais également une norme (norme ISO/9646[ISO94]), définissant un cadre méthodologique pour ce test. De plus, différents travaux dans le domaine de la conformité ont permis de définir formellement la notion de conformité grâce à des relations d'implémentations.

L'apparition des méthodes formelles de modélisation telles que Estelle, SDL ou Lotos a mené l'ISO et l'Union Internationale de Télécommunication (ITU-TS) à conduire un projet sur l'utilisation des méthodes formelles dans le contexte du test de conformité. Le groupe de recherche FMCT (Formal Methods in Conformance Testing) a été créé dans ce but. Les conclusions de ce groupe de travail ont permis de produire une norme « Z500 » qui balise la formalisation du test de conformité par trois grands axes :

- La définition d'une relation d'implémentation,
- L'indéterminisme dans le modèle de spécification choisi (Le test d'un système requiert que la spécification soit déterministe)
- Et l'influence de l'architecture de test. Les directives de la norme précisent l'importance de la prise en compte des contraintes du test lors de la spécification (i.e. adaptation du modèle de spécification à la génération des tests – dite aussi propriété d'implémentabilité), ainsi que la prise en compte de l'architecture de test lors de la dérivation des séquences de test (i.e. adaptation du modèle de spécification aux différentes architectures – dite aussi propriété de Robustesse)

Ces aspects du modèle étant primordiales pour la suite de la génération automatique du test. Notre travail s'inscrit dans cette perspective par le choix du modèle des DATAs (pour Duration Action Timed Automata, en français Automates Temporisés avec Durées d'Actions).

1.3 Contribution

Dans ce travail nous nous sommes intéressés au test de conformité et la modélisation des systèmes utilisant le modèle des DATAs. Ce modèle étant aux frontières des modèles discrets et ceux temporels, il est temporisé et ce situe dans le cadre des modèles très en vogue (scientifiquement), car manipulant les durées des actions explicitement sous forme de contraintes temporelles. Cet aspect qui lui est spécifique a orienté notre étude du modèle.

Nous nous proposons d'étudier le modèle des DATAs pour le destiner au domaine du test formel en respect de la norme Z500[Z5096], qui se présente comme la norme de formalisation du test de conformité, en proposant des définitions formelles et des études comparatives de différents modèles dans le même contexte.

Les travaux présentés dans ce mémoire portent sur les points suivants :

1. Le déterminisme : la proposition d'une méthode formelle de déterminisation du modèle des automates temporisés avec durées d'actions pour la génération automatique de séquences de test et ainsi prouver que le modèle est déterminisable.

2. L'implémentabilité : cette propriété des modèles de spécification à fait l'objet de beaucoup d'études voir Section6.4.4, elle consiste à chercher un cadre d'implémentation à l'issue d'un travail de modélisation. Il est à remarquer que Les modèles de spécification sont des formalismes mathématiques. La sémantique de ces modèles est en effet extrêmement précise, il est évident qu'un certain nombre de propriétés liées au caractère « idéal » du modèle mathématique est impossible de reproduire dans la réalité de l'exécution. On peut cependant se poser la question de la pertinence de la vérification ou du test formel de ces objets mathématiques, dès lors qu'ils représentent des programmes s'exécutant sur des machines réelles. Aucune plateforme d'exécution ne permet d'implémenter aussi précisément de tels modèles, et rien ne permet d'assurer a priori que les propriétés vérifiées sur le modèle théorique seront préservées lors de l'implémentation.

3. La robustesse des propriétés attendues. i.e. indifféremment de l'environnement d'exécution, ces propriétés restent valides. Deux qualités sont souhaitables pour l'implémentation d'un modèle :

   - La conservation des propriétés par passage à l'implémentation.

   - La propriété « faster is better » : si l'implémentation sur une plateforme est satisfaisante alors elles doit l'être sur une autre plateforme plus rapide ou plus performante.

   Il est certain que la maîtrise des durées des actions (quelle que soit leurs natures) permettra de conserver les deux propriétés lors du passage à l'implémentation.

4. Praticabilité du modèle des DATAs : Sur la base de cette formalisation, nous avons proposé un modèle servant à la production (génération) des tests à base de refus directement générés à partir d'une structure de DATA déterministe. Cette approche de test,

classique basée sur le traitement par le refus, est une technique utilisée pour l'analyse de la testabilité des systèmes et le diagnostic de fautes dans le domaine non temporisé. Dans ce travail nous avons proposé l'extension du modèle des Graphes de refus par la considération de contraintes temporelles en utilisant le modèle des DATAs comme modèle de base et qui a abouti à un graphe nommé graphe de refus temporisés (GRTs).

1.4 Plan

En plus de ce chapitre Introduction, le mémoire est organisé comme suit :

Le Chapitre 2 sera consacré aux différentes notions sur le test, il y sera présenté les différents types de test ainsi que leur pouvoir de validation des systèmes.

Le 3^{eme} Chapitre sera dédié à la notion de standard pour le test de conformité ainsi que les nouvelles recommandations pour la formalisation et l'automatisation du processus de génération de test.

Le 4^{eme} Chapitre concernera les modèles formels pour le test, il présentera les langages de spécification ainsi que les modèles à base de systèmes de transitions et enfin la modélisation des systèmes temporisés et sera introduit le modèle, base de notre étude.

Alors que le 5^{eme} Chapitre sera consacré aux grandes familles de génération de test de conformité. Ils y seront détaillées, les notions de base tel que l'indéterminisme, le choix, les blocages comme présentées dans la littérature. Enfin une pratique du test y sera exposée.

Le Chapitre 6, sera considéré comme principal, il introduira formellement le modèle ainsi que l'opération de déterminisation.

Une étude sur l'implémentabilité et la robustesse du modèle avec d'autres propriétés importantes par rapport aux travaux existants y sera présentée

Le Chapitre 7, traitera le modèle des graphes de refus ainsi que ses extensions actuelles, le problème de décidabilité des graphes de refus mixtes et présentra enfin une extension temporisée qui traite ce problème. Cette structure est générée à partir d'un DATA déterministe.

Le dernier Chapitre se présentera sous forme d'une conclusion générale et la donnée de plusieures perspectives.

Chapitre 2

Notions pour le test

Le test fait partie intégrante du développement du logiciel et/ou du matériel ; il ne s'intéresse qu'au produit final, c'est-à-dire au code du programme ou au comportement de l'exécutable correspondant. Cette activité a pour objectif d'exécuter le système, dans l'intention de détecter une erreur[Mye80]. Son exécution consiste à appliquer au système sous test un ensemble d'entrées et à observer les sorties afin de déterminer si elles correspondent au cahier des charges.

2.1 Caractéristiques du test :[Pri03]

2.1.1 Le pouvoir du test.

L'objectif de la phase de test est de vérifier que l'implémentation satisfait les exigences spécifiées par le cahier des charges, avant la mise en oeuvre finale du système. Il faut cependant noter que le test ne permet pas de prouver l'absence d'erreurs ; il met seulement en évidence des dysfonctionnements éventuels.

Cependant le test et la vérification d'un système sont deux approches jointes mais ne produisent pas les mêmes résultats. La vérification consiste à s'assurer que la spécification du système est correcte. Puis, le test de conformité permet de vérifier que le comportement externe d'un système est équivalent à sa spécification formelle initiale.

Même si la spécification ne vérifie pas une exigence particulière, l'implémentation sera conforme car les séquences de tests sont générées à partir du modèle de la spécification. Autrement dis le test de conformité assure qu'il n'y a pas de comportement erroné engendré par la réalisation et non pas la correction du modèle lui-même ou l'assurance d'absence de fautes.

2.1.2 Le coût du test.

Il est communément admis que le test couvre près de 50 % du coût total du développement d'un système. La construction des séquences de test, à partir du cahier des charges, est une

phase laborieuse qui peut être source d'erreurs. De plus, les séquences de test écrites sont produites à partir de la liste des exigences du système. Ainsi, rien ne garantit l'absence de redondance ou la complétude de la suite de tests, c'est-à-dire l'absence d'un test nécessaire à la validation.

2.1.3 Catégories de tests.

Le choix du test correspond au type de validation que l'on souhaite effectuer. Ainsi, une différence est faite entre test fonctionnel et test structurel.

1. **Le Test structurel** ou Test boîte blanche vise à analyser la structure interne du système. Celle-ci étant connue et c'est en fonction d'elle que s'effectue la sélection des cas de test. L'une des méthodes possibles est d'exécuter tous les chemins possibles du code. Le critère de sélection exhaustif consiste à passer dans toutes les branches d'exécution du programme, ce qui est impossible à atteindre. Il définit les éléments du code parcourus par le test : on parle alors de "couverture". Cela permet, entre autres, de détecter des morceaux de programmes qui, présents dans le code source, ne sont jamais réellement exécutés. Il s'agit de code mort qui doit être supprimé avant mise en service du logiciel.

2. **Le Test Fonctionnel** ou Test boîte noire a pour objectif d'analyser les comportements externes du système. Sa structure interne n'est pas connue. L'objectif est de déterminer si le produit développé correspond aux fonctionnalités prévues. Il s'agit de soumettre l'implémentation sous test à des entrées et d'observer les sorties afin de déterminer la validité des comportements. Les tests fonctionnels sont définis à partir de la spécification. Il est donc nécessaire que celle-ci soit précise et ne renferme pas d'erreur.

3. **Le Test boîte grise** est une forme hybride de test où les tests de types boîte blanche et boîte noire sont utilisés de manière complémentaire. On applique des stratégies de boîtes noires en ayant connaissance de la structure du programme. Ce type d'approche est intéressant dans le cas des structures modulaires.

2.1.4 Les outils du test.

Il existe des outils permettant d'automatiser la phase d'exécution des tests. Par exemple, Rational system test (anciennement ATTOL system test) permet une exécution automatique des séquences de test sur une implémentation[RHW]. D'autres outils opèrent différemment , ils permettent de sauvegarder les séquences de test exécutées par un intervenant humain et rejouer ces séquences de test sur le système. Cependant, aucun des outils utilisés actuellement dans le cadre de développements industriels ne permet une génération automatique des tests.

2.1.5 Les modes d'exécution du test[Mal07]

Dans le cadre du test, on distingue généralement deux modes d'exécution :

- **Le test actif**, dans lequel le testeur stimule (contrôle) l'implémentation et observe ses réactions. En fonction de celles-ci, il décide du prochain stimulus à émettre.

- **Le test passif**, consiste à examiner le comportement (messages d'entrée/sortie) d'une implémentation sans pour autant imposer les entrées, ainsi le testeur passif n'aura pas à définir une interface d'entrée sur l'IUT (**Implementation Under Test**) , il se contentera dans un premier lieu de récupérer une trace d'exécution . Une fois cette trace obtenue, on va déterminer si elle est conforme à la spécification. Si c'est le cas, on ne peut rien affirmer à propos de la validité de l'implémentation, en effet une autre trace non observée pourrait révéler des erreurs. Sinon, on sait que l'implémentation n'est pas conforme. L'analyse de cette trace permet au testeur d'émettre un verdict soit à l'issue du test, soit pendant le test si une faute se produit.

- **Comparaison :** Le principe du test actif est de stimuler une implémentation sous test (IUT) au moyen d'un testeur qui envoie des messages (i.e. les entrées censées faire réagir l'IUT) et qui collecte les observations (i.e. les sorties émises par l'IUT) dans le but d'émettre un verdict sur la conformité de l'implémentation. La difficulté est de sélectionner un ensemble de tests pertinents pour la détection de la non-conformité.

  Les méthodes de test actifs basées sur les FSM (**Finite States Machine**) ou de façon duale, d'autres méthodes de test actives basées sur les modèles type LTS (**Labelled Transitions System**) utilisent des relations de conformité qui caractérisent l'ensemble des implémentations conformes. Les suites de tests produites par ces méthodes, sont injectés dans l'implémentation sous test. Cependant, ce contrôle des entrées limite la détection de certaines erreurs qui peuvent se produire dans un environnement hostile.

  Dans le test passif, contrairement au test actif, le testeur ne contrôle pas l'implémentation sous test. Il se limite à un rôle d'observateur et d'analyseur : le testeur collecte les informations, i.e. les messages entrants et sortants de l'IUT et analyse ses échanges par rapport à ceux attendus par la spécification. Cette forme de test n'est pas intrusive, en le sens où l'IUT n'est pas stimulée et ainsi elle n'influence pas le comportement interne de l'implémentation. De ce fait, la force majeure du test passif sur le test actif est sa capacité à être utilisé sur un réseau en fonctionnement. En effet, le test actif peut parfaitement perturber un réseau en cours de fonctionnement puisqu'on impose au système de produire un certain comportement par l'injection d'entrées spécifiques, tandis que le test passif se contente d'observer sans intervenir. Bien que le test passif soit cité comme une alternative au test actif dans certains documents, il doit être considéré comme une forme de test complémentaire, car certains comportements peu fréquents en utilisation normale ne peuvent être observés.

2.1.6 Types de test

Il existe plusieurs types de test adaptés aux comportements erronés que l'on souhaite détecter sur l'implémentation. Dans [ABH99] une classification a été proposée, en fonction de l'aspect du système à tester. De ce fait, nous pouvons nous poser les questions suivantes :

• Le système se comporte-t-il de manière fonctionnelle, comme cela est prévu par sa spécification ?

• Le système a-t-il les performances requises ?

• Comment le système se comporte-t-il si son environnement lui fournit des données non prévues ?

• Le système peut-il traiter un grand nombre d'entrées ?

• Pendant quelle durée est-il possible de faire confiance au système ?

• Quelle est la disponibilité du système ?

A partir de cette distinction sur la connaissance du système pour le testeur, plusieurs types de tests permettent de vérifier des propriétés différentes et peuvent être classifié suivant leur appartenance aux types suivants :

Test de conformité ;
Test d'une politique de sécurité ;
Test de performance ;
Test d'interopérabilité/test d'inter fonctionnement ;
Test de robustesse.

- **Le test de conformité** : Vise à déterminer si l'implémentation est conforme à sa spécification en vérifiant que le comportement observable d'une implémentation correspond à celui prévu par sa spécification, ce qui le rattache à la classe des tests fonctionnels ayant pour but de vérifier que le système implémente correctement les fonctionnalités prévues par la spécification. C'est le type de tests défini dans ISO/9646 [Voir Chapitre4]. Il est clair que c'est le type de tests que nous allons chercher à générer et, ou nous allons nous placer pour ce travail dans un cadre formel.

- **Le test d'une politique de securité**[VD04] : Il vise le respect d'une politique de sécurité, c'est un test de conformité d'une mise en oeuvre (déployée sur l'ensemble des machines en réseau) d'une spécification de la sécurité. Il s'agit d'un test fonctionnel à posteriori après déploiement d'un ensemble d'éléments logiciels et matériels.

C'est un test de conformité particulier pour des réseaux qui a fait l'objet de nombreuses études depuis la mise en place des interconnexions de machines au sein d'architectures ouvertes, en particulier dans le cadre des architectures de type ISO et Internet. Celui-ci s'appuie sur des notions essentielles pour une démarche formelle qui a permit de développer des outils d'analyse automatique des logiciels déployés pour la sécurité.

Les étapes suivantes sont à respecter dans le cas du test d'une politique de sécurité :

- Une modélisation formelle de la politique de sécurité ou une formalisation de la relation de conformité entre le système à tester et la spécification qu'il doit respecter.

- Une définition de l'architecture du test, en référence à l'architecture du système, qui exprime les capacités d'interaction (observation et contrôle) entre le système de test et le système à tester.

Dans le cas du test de politique de sécurité, certains événements observables (interaction entre deux machines distantes, variables d'environnement mises à jour par le système d'exploitation, etc.) ne peuvent pas toujours être connus du testeur pendant l'exécution du test. Ces événements, stockés au fur et à mesure dans un journal de bord, ne sont alors disponibles qu'a posteriori. Le contrôle effectué par le testeur n'est donc que partiel et un certain nombre de décisions doivent être prises arbitrairement, sans connaître complètement l'état du système sous test. De même, le verdict de test ne pourra être émis qu'après analyse de la séquence d'exécution, en fonction de l'état atteint dans l'arbre de test. Contrairement au principes du test actif. C'est pour quoi, dans le cadre du test de la politique de sécurité, les deux modes d'exécutions des tests restent envisageables.

- **Le test de performance :** Dès lors que des performances sont exigées par le cahier des charges des systèmes sous test, elles font de toute évidence partie des éléments à tester des implémentations. Le test de performance est une spécialisation du test de conformité qui vise en fait à tester les aspects temporels d'une spécification comportementale. Il est alors indiqué dans le cas d'une formalisation d'utiliser un modele capable de prendre en compte l'aspect temporel permettant de spécifier des propriétés temporelles.

- **Le test d'interopérabilité/test d'inter fonctionnement** : Ce type de tests est une autre spécialisation du test de conformité qui vise en fait à tester les aspects d'interopérabilité ou d'inter fonctionnement de ces systèmes. Cependant il faut dissocier ces deux types de spécifications : le test d'interopérabilité vise à vérifier si deux ou plusieurs composants d'un système réparti peuvent communiquer entre eux ; le test d'inter fonctionnement nécessite d'une part l'interopérabilité et d'autre part le test du comportement dynamique des systèmes répartis.

- **Le test de robustesse :** Il consiste à vérifier la réaction du système face aux tentatives d'utilisation abusives. Cette notion, informelle, peut aussi bien inclure la réaction face aux pannes, que face aux fraudes ou aux simples erreurs de manipulation. Depuis quelques années, il est apparu un besoin au niveau de la robustesse des systèmes : comment le système réagit-il face à des événements imprévus ? Cette réponse ne peut pas être apportée par le test de conformité qui ne tient pas compte, en général, du caractère hostile de l'environnement. La littérature concernant le test de robustesse est beaucoup moins conséquente que le test de conformité. La robustesse est définie comme étant : "la capacité d'un système à adopter un comportement acceptable en présence d'entrées invalides ou d'environnement stressant" [BAL04] [Kho06]. Dans [RC02] l'approche proposée considère le test de robustesse comme une extension du

test de conformité, en conséquence, il propose de définir une relation de robustesse, notée *Rob*, étendant la relation $Conf$ utilisée dans le cadre du test de conformité [Voir Chapitre4] : $I\ Rob\ S$ si $\forall \sigma \in L^*, \forall A \subseteq L$, I après σ refuse $A \Rightarrow S$ après σ refuse A ; où I et S sont respectivement l'impléméntation (ou son modèle) et la spécification, σ est une trace, L^* est le langage sur l'alphabet d'entrée (pour prendre en compte les événements inopportuns) et L^*_θ est le langage étendu avec des événements corrompus ou inconnus. Dans cette approche il est fait l'hypothèse que la robustesse est plus forte que la conformité : $I\ Rob\ S \Rightarrow I\ Conf\ S$.

Une méthodologie assez intéressante est proposée dans [RC02] pour la construction du testeur de robustesse.

2.2 Le test de conformité

Venons-en maintenant au test de conformité, celui-ci est né du besoin des administrateurs réseaux de vérifier l'adéquation des équipements commerciaux sur le premier réseau public de données. Dans les années 1980, les premières attentes de la construction des séquences de test étaient de permettre d'établir si l'implémentation réalisait toutes les fonctions de la spécification sur l'ensemble des valeurs des paramètres. Une autre attente était de vérifier la capacité d'une implémentation à rejeter des entrées erronées comme prévu dans la spécification initiale.

Rappelons ici que le test de conformité s'intéresse uniquement aux comportements observables de l'implémentation. Cette dernière est considérée comme une boîte noire dont on ne connaît pas le fonctionnement interne. Il a pour objectif de montrer que la mise en œuvre d'un système (ou implémentation) correspond à sa spécification initiale. Ce type de vérification consiste à appliquer un ensemble de séquences de test à l'implémentation et d'observer les réponses du système. L'analyse de ces réponses permet de déterminer si l'IUT est conforme à sa spécification grâce au verdict établi par le testeur à la fin de l'exécution du test. Les cas de test utilisés par le testeur sont le plus souvent construits manuellement. Cette démarche engendre d'importants coûts de développement.

2.2.1 Standardisation et formalisation du test de conformité

Le test de conformité est particulièrement utilisé dans le domaine des protocoles de télécommunication. Une norme de référence pour la standardisation de la démarche de test de conformité à été développée pour produire un cadre méthodologique, répondant à toutes les problématiques qui peuvent être posées.

La norme, **ISO**/9646 [Voir Chapitre3] qui décrit les ingrédients principaux du test de conformité, donne un ensemble de définitions pour les suites de test, la notion de conformité, la phase d'exécution des tests et la signification des verdicts.

Mais le test de conformité n'est pas confiné au domaine des télécoms, ainsi le même type

de test peut être utilisé pour des systèmes réactifs quelconques, et en particulier pour les systèmes critiques.

La construction des séquences de test étant une phase coûteuse et source d'erreurs, des travaux ont été proposés pour construire automatiquement les suites de test à partir d'une spécification formelle. Ces méthodes de génération automatique des cas de test permettent d'alléger les coûts de construction des tests. Certaines d'entres-elles s'appuient sur des algorithmes de parcours exhaustif des états accessibles du modèle (Model-Checking).

Le gain potentiel en terme de réduction du coût du test, apporté par l'automatisation de la génération de test de conformité, a suscité un grand intérêt à la fois chez les industriels et les universitaires depuis de nombreuses années. Cependant, bien que la génération automatique de tests connaisse un réel succès industriel dans le domaine du matériel, elle a eu plus de mal à percer dans le domaine du logiciel. Une des raisons, est peut être la trop grande distance entre les techniques universitaires et la pratique industrielle [LL95] . Un effort reste à faire pour rendre ces techniques plus utilisables sur des cas réels.

Chapitre 3

Standardisation du test de conformité

3.1 le test de conformité entre le standard et le formel[San97][Pri03]

Dans le domaine des télécommunications, le test de conformité est souvent réalisé par des laboratoires de test indépendants, sur des systèmes développés par des fournisseurs, afin de fournir un certificat de conformité utile pour des clients éventuels. Dans ce cas, le fait que le code ne soit pas connu est dû à des raisons de confidentialité évidentes.

En pratique, le test de conformité consiste à faire interagir l'implémentation sous test (**IUT pour Implementation Under Test**) avec son environnement. Cet environnement, qui est constitué par un ou plusieurs processus ou utilisateurs, est simulé par un ou un ensemble de testeurs. Ces testeurs exécutent des tests élémentaires appelés cas de test qui consistent en une suite d'interactions. Cette suite d'interactions contrôle l'**IUT** par des stimuli (des envois de messages, des appels de fonctions, de méthodes, etc...) et observe ses réactions (messages reçus, résultats d'appels, etc....). Les comportements de l'**IUT** sont contrôlés et observés à travers des interfaces (appelées **PCOs** pour "Points of Control and Observation" dans le monde télécoms). Généralement, chaque cas de test a pour objectif le test d'une fonctionnalité précise, décrite dans ce qu'on appelle un objectif de test.

Le test de conformité est un type de test qui s'intéresse à la correction fonctionnelle d'une implémentation d'un système vis à vis d'une spécification de référence. C'est donc un test de type fonctionnel de type boite noire, car le code de l'implémentation n'est en général pas connu.

Dans ce genre de développement, la notion de standard a une importance particulière. Il est en effet nécessaire de pouvoir garantir que le comportement d'un composant s'adaptera aux autres composants standardisés, avec lesquels il sera exécuté.

Afin de garantir que des tests de conformité d'une même implémentation, effectués par des équipes différentes, fournissent les mêmes résultats, il est important de définir clairement les notions de conformité, de séquences de test et de verdict. Ces éléments doivent être

définis et standardisés. C'est dans cet objectif que **l'ISO**, en accord avec **l'ITU** (l'Union Internationale de Télécommunication), a développé un standard pour le test de conformité des systèmes ouverts. Le standard, désigné sous le sigle **ISO**/9646, produit un cadre méthodologique de description et d'exécution des suites de test permettant d'établir la conformité d'un composant. C'est pourquoi les techniques de génération automatique de tests, basées sur une spécification formelle du système, sont un apport majeur pour les développements industriels.

Dans la réalité du test la dérivation de ces séquences de test est une étape lourde et coûteuse du processus de développement d'un système.

Ce Chapitre est divisé en deux parties, dans un premier temps, nous présentons le concept de test de conformité et la norme qui le standardise, l'**ISO**/9646 ; La seconde partie de ce Chapitre présente les conclusions de la norme Z500, qui n'est pas moins d'une étape vers l'automatisation de la construction des séquences de test.

3.2 La norme ISO/9646 relative au test de conformité

La norme, **ISO** "Conformance Testing Methodology and framework", définit un cadre méthodologique de test de conformité des systèmes. Selon l'ISO9646, il est possible de distinguer trois phases dans le processus de test de conformité. Une première phase correspond à la dérivation des séquences abstraites de test. Ensuite, ces séquences abstraites sont transformées en séquences de test exécutables : c'est l'étape d'implémentation des tests. Enfin, l'exécution des séquences de test sur l'**IUT** et l'analyse du résultat, permettent d'obtenir le verdict sur la conformité. Les résultats de l'exécution sont documentés par le **PCTR** (Protocol Conformance Test Report). Tretmans propose dans [Tre01] une synthèse des concepts définis par cette norme. Nous en donnons ici les grands axes.

La norme **ISO**/9646 a été définie pour des protocoles spécifiés en langage naturel. En fait au départ, elle a été développée pour les protocoles **ISO**, mais elle est utilisée pour tester tous les types de protocoles (ISDN, GSM). On la retrouve bien entendu dans le domaine des systèmes à objets répartis et plus particulièrement dans la gestion de réseaux **ISO** [Kab95][Bae93]. Cette norme comprend cinq parties :

la partie 1 [IT92a] est une introduction et traite des concepts généraux ;

la partie 2 [IT92b] décrit le processus de spécification des suites de tests ;

la partie 3 [IT92c] définit la notation TTCN (Tree and Tabular Combined Notation) ;

la partie 4 [IT92d] traite de la réalisation du test.

3.2.1 La conformité

La première étape consiste à définir précisément la relation de conformité entre la spécification et l'implémentation. Les contraintes de conformité sont explicitement indiquées dans la norme du protocole. Elles indiquent ce qu'une implémentation conforme doit faire et ce qu'elle ne doit pas faire.

Toutes les options qui ont été réalisées dans une implémentation donnée sont listées dans les **PICSs** (Protocol Implementation Conformance Statement). Des restrictions dans la sélection sont données par l'intermédiaire de deux types de contraintes : les contraintes statiques de conformité et les contraintes dynamiques de conformité.

Les contraintes statiques définissent les capacités minimales qu'une implémentation doit avoir (les combinaisons possibles des options) ;

Les contraintes dynamiques (la plus grande et la plus importante part de la norme) définissent le comportement observable de l'implémentation lors de la communication avec son environnement : l'ordre des événements, le codage des informations dans les **PDU** (Protocol Data Unit) et les relations entre les PDU. Le test de conformité a alors pour fonction de s'assurer que l'implémentation satisfait les exigences dynamiques et statiques de conformité spécifiées par le PICS.

3.2.2 Etapes du test de conformité

Dérivation des séquences de test

La première étape du processus de test de conformité est de générer les séquences de test abstraites à partir de la spécification du protocole. Les séquences de test sont construites à partir des exigences de conformité dynamiques. Dans un premier temps, les objectifs de test sont dérivés à partir de toutes les exigences de conformité. Un objectif de test correspond à une description détaillée de ce qui doit être testé pour la vérification d'une exigence particulière de conformité.

Deux niveaux de cas de test :

1. Cas de test générique qui une séquence de test contenant des opérations nécessaire a la vérification de l'objectif de test.

2. Cas de test abstrait ne tient pas compte de l'environnement ni de la methode de test choisi la norme **ISO/** 9646 recomande par chaque objectif de test la construction d'un cas de test générique, à partir du quel un cas de test abstrait est génèré.

Implémentation des séquences de test.

La seconde étape est l'implémentation des séquences de test permettant de rendre exécutable la séquence de test pour une **IUT** spécifique.

1. **Sélection des tests.** La première étape est une sélection des tests parmi l'ensemble des séquences de test abstraites obtenues. Cette sélection concerne les tests correspondant à des comportements qui n'ont pas été implémentés pour l'**IUT** que nous souhaitons tester. Les tests pertinents sont sélectionnés en fonction des **PICS**.

2. **Implémentation des tests.** Pour construire les séquences de test exécutables, il est indispensable de tenir compte des particularités de l'**IUT** et de son environnement.

Le **PIXIT** (Protocol Implementation eXtra Information for Testing) contient les informations nécessaires. Les tests implémentés sont les tests obtenus après la phase de sélection des tests auxquels on a ajouté les informations contenues dans le **PIXIT**.

3. **Format des séquences de test**. ISO9646 recommande l'utilisation du langage **TTCN** (Tree and Tabular Combined Notation) pour la notation des séquences de test[ISO91]. Une suite de tests de ce format est composée de quatre parties : un résumé, une zone de déclaration, une partie contraintes et une partie dynamique. Un cas de test est un arbre dans lequel chaque branche décrit une séquence d'interaction entre le testeur et l'**IUT**.

Exécution des tests

L'exécution des tests applique les séquences de test implémentées pour une **IUT** particulière afin d'établir le verdict de conformité. La première étape est une vérification de la conformité statique. Les **PICS** sont observés pour déterminer s'ils répondent aux exigences de conformité du standard. La seconde étape de l'exécution est d'appliquer les séquences de test exécutables sur l'implémentation. Chaque séquence de test exécutée produit un verdict. Ce verdict peut être :

- Pass : le test a été exécuté avec succès et la propriété exprimée par l'objectif de test est vérifiée.

- Fail : le test a permis de conclure que l'implémentation n'était pas conforme à la spécification.

- Inconclusive : l'exécution de la séquence de test ne permet pas d'établir la non conformité, mais la propriété correspondante à l'objectif de test n'est pas respectée par l'implémentation.

Les résultats de l'exécution de tous les tests statiques ou dynamiques sont combinés pour établir le verdict final de conformité. Un verdict Pass est obtenu seulement dans le cas ou aucun des tests individuels effectués n'a produit de résultat Fail. Les résultats individuels de chaque test et le verdict final sont rapportés dans un document : le **PCTR** (Protocol Conformance Test Report). La Figures3.1 illustre cette technique.

3.2.3 Architecture de test à la norme ISO/9646

Les points où le testeur contrôle l'implémentation s'appellent des **PCOs** (Points of Control and Observation). L'implémentation sous test (**IUT**) est observable uniquement par l'intermédiaire de ces **PCOs**. Selon la norme ISO9646, il existe deux types d'architecture de test :

L'architecture locale : Le testeur et l'**IUT** sont exécutés sur une même machine. Ce type de campagne de test est appliqué, par exemple, au test de logiciel.

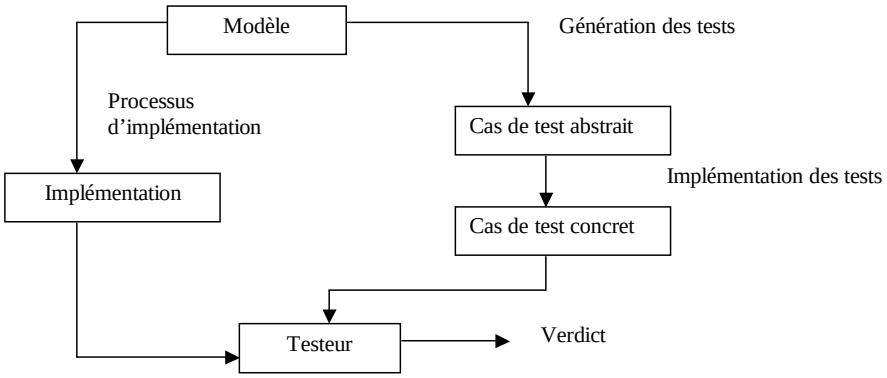

FIG. 3.1 – Méthode de test

L'architecture globale : pour un système distribué, réparti et distant. Pour ce type d'architecture, il est nécessaire de quantifier les délais et de les prendre en compte lors de la création des tests.

3.3 Le test de conformité et la formalisation

Le croissant engouement pour la modèlisation formels des systèmes a permis, depuis plusieurs années, d'utiliser des méthodes de vérification automatique dans les techniques de génération de tests. aussi de montrer et profiter du pouvoir de la modélisation formelle pour générer automatiquement les séquences de test. Celle ci présente l'avantage de réduire le risque d'erreurs inhérent à la construction manuelle des suites de test.

Test et vérification. Dans [ABH99] il est expliqué que le test et la vérification sont deux techniques complémentaires garantissant le bon fonctionnement d'un système. En effet, la construction des séquences de test, à partir d'une modélisation, s'appuie sur l'hypothèse que la modélisation est correcte. Une séquence de test construite sur un modèle erroné sera fausse et le verdict ne pourra donc pas être interprété correctement. par conséquent il est nécessaire de garantir la correction du modèle.

L'apport principal du test formel est que la modélisation soit le point de départ de la génération de test. Cette phase peut être automatisée car ces techniques de génération ont pour fondement les algorithmes issus des méthodes de Model-Checking.

L'application des méthodes **formelles** dans le cadre du test de conformité consiste à vérifier qu'une implémentation est conforme à une modélisation **formelle** par rapport à une relation de conformité **formelle**.

3.3.1 La norme Z500 : Une formalisation de l'ISO/ 9646[Pri03]

L'apparition des méthodes formelles de modélisation telles que **Estelle**, **SDL** ou **Lotos** a mené l'**ISO** et l'Union Internationale de Télécommunication (**ITU-TS**) à conduire un projet

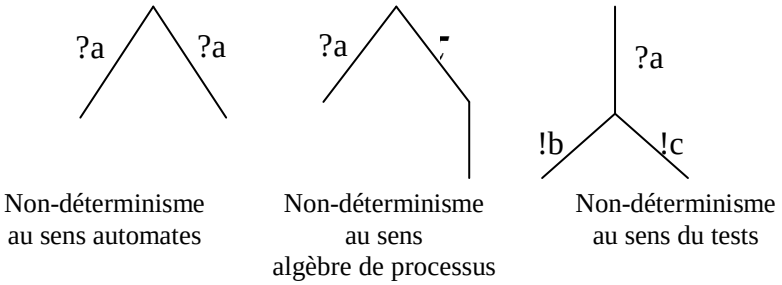

FIG. 3.2 – Différents types de non-déterminisme

sur l'utilisation des méthodes formelles dans le contexte du test de conformité. Le groupe de recherche **FMCT** (Formal Methods in Conformance Testing) a été crée dans le but d'étudier les problématiques suivantes :

■ Déterminer quels sont les apports de la théorie du test pour la norme **ISO**/9646 et dans quel cadre les concepts de conformité, de suite de tests et d'exécution des tests pouvaient être appliqués aux modélisations formelles telles que **SDL** ou **Lotos**.

■ Evaluer la possibilité de dériver automatiquement les séquences de test à partir de telles spécifications.

Les conclusions de ce groupe de travail ont permis de produire une norme [Z5096]. Le document [ARC93] présente ces résultats et les perspectives de l'application des langages formels dans le test et les méthodes de génération pour le test, il définit un modèle d'application de test dans le cas où la spécification est décrite par un langage formel.

On résume dans la suite les résultats du groupe FMCT.

1. **La relation d'implémentation** : La première conclusion du groupe de recherche est que la modélisation formelle de la spécification n'a pas d'intérêt dans le cadre du test en l'absence d'une définition précise de la relation liant la spécification à l'implémentation. Cette relation est nommée relation d'implémentation ou encore relation de conformité.

2. **L'indéterminisme et la modélisation** : Le test d'un système requiert que la spécification **soit déterministe**. La notion d'indéterminisme étant différente suivant le contexte, La Figure3.2 présente plusieurs types d'indéterminisme. Dans le Chapitre6, cette notion sera largement étudiée, elle doit alors être établie avant de choisir le modèle ainsi que la méthode du test. Cet aspect de la modélisation étant primordiale pour la suite de la génération automatique du test.

3. **Influence de l'architecture de test :** Les études théoriques sur le pouvoir du test ont montré que l'architecture de test peut limiter la contrôlabilité du système et son observabilité. Cela a une influence sur les exigences de conformité pouvant effectivement être vérifiées. Les conclusions du groupe FMCT à ce propos sont les suivantes :

• Les contraintes du test devraient être prises en compte au moment de la phase de spécification. Il n'est pas nécessaire de spécifier des comportements qui ne seront pas testables en raison des contraintes de l'exécution des tests. Il est nécessaire de construire prudemment la spécification de manière à ne pas limiter les comportements pouvant être testés.

• Certaines séquences de test exécutables sur une architecture particulière peuvent devenir non exécutables sur d'autres architectures de test. Contrairement aux recommandations de l'**ISO**/9646, les séquences de test abstraites ne peuvent donc pas être dérivées des séquences de test génériques. Elles devraient être construites à partir de l'objectif de test et de l'architecture de test.

4. **Manipulation de spécifications synchrones :** L'un des problèmes mis en évidence est la différence entre une spécification synchrone et une spécification asynchrone. En effet, les suites de test TTCN procèdent par une communication asynchrone entre le testeur et l'implémentation via les **PCOs**, tandis qu'une spécification synchrone prévoit une communication par rendez-vous (cas de Lotos). Une solution à ce problème est l'utilisation d'une interface synchrone/asynchrone entre l'**IUT** et le testeur.

3.4 Conclusion

Depuis plus d'une trentaine années, la génération automatique de tests est perçue comme l'une des applications les plus rentables des techniques de spécifications formelles. Vu l'importance en terme de temps et de coût des phases de test, les chercheurs et praticiens du test ont cherché depuis plusieurs années à automatiser cette phase. Ainsi, de nombreuses méthodes de génération automatique de test de conformité ont vu le jour issues du test de circuit [voir par exemple la synthèse et les références du Chapitre4]. Malheureusement, peu d'entre elles sont utilisées en pratique et les tests sont encore écrits à la main à partir de spécifications informelles. Cette situation est particulièrement vraie dans le contexte des systèmes temporisés qui forment le cadre de travail de cette thèse.

En respect avec la norme **Z500**, et dans le cadre du test des systèmes temporisés, nous avons choisi le modèle des automates temporisés avec durées d'actions les DATAs pour la spécification des systèmes, aussi la possibilité de sa déterminisation le promet pour la modélisation de l'implémentation sous test ainsi que les testeurs. Nous avons eu l'intuition que la capacité des DATAs à modéliser la réalité concernant la durée des actions lui accord un niveau d'implémentabilité et de robustesse assez intéressant d'étudier, en respect des recommandations de la norme le modèle des DATAs se prête naturellement à la génération automatique du test formel pour le système temporisés.

Chapitre 4

Modèles formels pour le test

4.1 Modèles formels

La modélisation mathématique des systèmes informatiques est devenue de plus en plus une nécessité méthodologique, notamment pour le développement et la validation des systèmes complexes. Au fil des années, un bon nombre de modèles ont été introduits pour décrire les composants électroniques ou les systèmes informatiques. Dans le cadre de notre étude nous allons nous concentrer sur les modèles les plus utilisés pour la validation des protocoles de communication. Rappelons que pour qu'un système soit parfaitement utilisable, il doit être validé sur la base de fondements mathématiques, et par conséquent sur une sémantique correcte. Le choix du modèle dépend du type du système à modéliser et des conditions de son évolution, ainsi il doit être ouvert pour accepter de nouvelles contraintes de modélisation.

Dans ce Chapitre, nous rappelons les concepts de base relatifs aux modèles utilisés dans le cadre de la modélisation des systèmes discrets (non-temporels) et ceux temporisés : Machines à états finis, Systèmes de transitions. Bien entendu, la liste est incomplète si on ne cite qu'à titre d'exemple, les réseaux de Petri [Pet62][Pet63] utilisant les notions : places, transitions, flux et jetons pour modéliser et analyser les comportements dynamiques d'un système discret.

La spécification doit permettre de décrire le comportement d'un système en décrivant ses propriétés importantes de façon abstraite, sans détails inutiles sans dire comment il sera obtenu (non-algorithmique). Pour la spécification des propriétés, des exigences et le respect des contraintes ; des langages de spécification basés sur les algèbres de processus sont traités tel que : ACP de Bergstra et al. [BK84], CSP de Tony Hoare[Hoa85], CCS de Robin Milner[Mil80][Mil89], MEIJE de R.de Simone [dS85] et LOTOS [Bri88]. Ces langages formels ont une syntaxe et une sémantique bien définies ; les diagrammes syntaxiques peuvent etre une syntaxe possible. La sémantique est décrite en algèbres, automates et systèmes de transitions, fonctions, relations et prédicats ; etc. . .

Dans ces algèbres, la spécification du système contient un certain nombre de processus élémentaires et un certain nombre d'opérations qui, appliquées à des processus, construisent d'autres processus. A chaque processus, il est possible de construire un système de transitions

représentant son comportement.

4.2 Les langages de spécification

Afin d'étudier la possibilité d'utiliser des spécifications formelles pour la définition des protocoles et des services de l'ISO, des langages de spécification tels que SDL, LOTOS ou Estelle, dis aussi techniques de description formelles ou (TDFs) ont été proposés, ils sont recommandés pour la spécification et la description des systèmes de télécommunications, des services et protocoles de réseaux et des systèmes réactifs, distribués ou temps réel. LOTOS a été adopté par les organismes de normalisation ISO(International Standardization Organisation) et ITU (International Telecommunication Union).

Les TDFs ont été développées pour assurer la clarté, la complétude, la consistance et la traçabilité des spécifications. On s'intéresse ici au langage de spécification LOTOS avec ses extensions temporisées. Pour plus de détails voir[Bel05].

4.2.1 LOTOS (Language Of Temporal Ordering Specification)

LOTOS est une norme internationale[FDT87]. Conçu pour la spécification, la conception, et l'analyse fonctionnelle des systèmes concurrents. La spécification LOTOS possède deux parties clairement séparées. La première offre un modèle comportemental dérivé des algèbres de processus, principalement de CCS (Calculus of Communicating Systems [Mil89]), mais aussi de CSP (Communicating Sequential Processus [Hoa85]). La deuxième partie du langage permet de décrire des types de données abstraites et des valeurs, et est basée sur le langage des types de données abstraits ACT-ONE [EM85]. LOTOS utilise les concepts de processus, événements et expressions comportementales comme concepts de base pour la modélisation. Basic LOTOS le sous-ensemble de LOTOS où les processus interagissent entre eux par synchronisation pure, sans échange de valeurs. En Basic LOTOS, les actions correspondent aux portes de synchronisation des processus.

Formellement Basic LOTOS se presente comme suit : Soit PN l'ensemble des variables de processus parcouru par X et soit G l'ensemble des noms de portes définies par l'utilisateur (ensemble des actions observables) parcouru par g .Une porte observable particulière $\delta \notin G$ est utilisée pour notifier la terminaison avec succès des processus. L dénote tout sous-ensemble de G , l'action interne est désignée par i . B parcouru par $E, F, ...$dénote l'ensemble des expressions de comportement dont la syntaxe est :

$E ::= stop|exit|X[L]|g; E|i; E|E[]E|E|[L]|E|hideLinE|E \gg E|E[> E$

Etant donné un processus dont le nom est P et dont le comportement est E ,la définition de P est exprimée par $P := E$. L'ensemble de toutes les actions est désigné par Act ($Act = G \cup \{i, \delta\}$).[Bel05]

4.2.2 Le langage LOTOS et les extensions temporelles

Le besoin de l'expression explicite du temps dans LOTOS a généré plusieurs variantes du langage. Les plus connues sont listées dans ci-dessous . Les plus connues sont listées dans[Loh02], elles différent principalement par :

Choix du domaine temporel Soit discret, soit dense et dénombrable , soit réel.

Temporisation des actions

Hypothèses d'urgence des actions Une action est dite urgente lorsqu'elle doit se réaliser immédiatement, sans progression possible du temps, dès qu'elle est sensibilisée. Pour plus de détails sur les extensions temporelles de LOTOS, le lecteur peut se référer à[Loh02].

4.2.3 Autres langage

D'autres langages ont été proposés pour specifier les systèmes en général et les protocoles en particulier. on peut citer :

SDL : (Specification and Description Language) normalisé par l 'ISO et le CCITT (Comité Consultatif International Téléphonique et Télégraphique)

est recommandé par le ITU pour spécifier les systèmes de télécommunication d'une manière non ambiguë.

Estelle (Extended State Transition Language) normalisée par le CCITT et ISO

UML (Unified Modeling Language)de l'OMG (Object Management group).

IF (Intermediate Format) est un langage permettant la description d'arbres abstraits. Ce langage est une représentation à base d'automates temporisés communicants.

Promela est un langage de spécification de systèmes asynchrones ou systèmes concurrents.

4.3 Machines à états finis

Les machines à états finis (en anglais FSM pour Finite State Machine) sont parmis les modèles les plus anciens. Elles sont issues de la modélisation des circuits logiques. Une FSM est une machine ayant une quantité finie de mémoire pour représenter les états et distingue entre une sortie (signal émis) et une entrée (signal reçu). Il existe deux types de machines à états finis séquentielles : machine de Moore et machine de Mealy. Dans une machine de Moore, les sorties dépendent des états seulement, tandis que dans une machine de Mealy les sorties dépendent des états et des entrées. Dans[Gil62] une théorie complète sur les FSMs peut être trouvée. Nous rappelons ici les notions de base relatives aux machines de Mealy.

Machine de Mealy. C'est un graphe décrivant le comportement de la spécification. Les nœuds décrivent les états internes du système. Les arcs (transitions) sont étiquetées par un ensemble de symboles d'entrée et un ensemble de symboles de sortie. Les sorties changent immédiatement en réaction à un changement d'entrée. Formellement [Mor00]

Définition 4.1 *Une Machine de Mealy est un t-uplet* $(S, I, O, \sigma, \omega)$

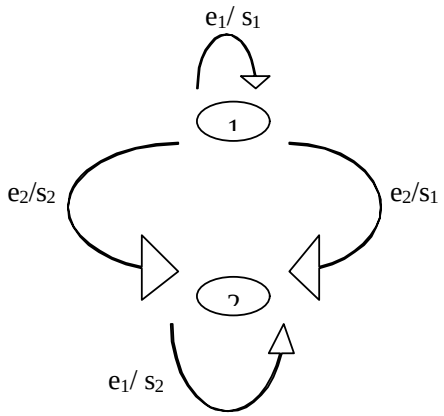

FIG. 4.1 – Un exemple de Machine de Mealy

• S est un ensemble fini d'états
•I est un alphabet fini d'entrées
•O est un alphabet fini de sorties
• $\delta : S \times I \rightarrow S$ est une fonction de transfert étendue en $\delta^* : S \times I^* \rightarrow S^*$
• $\omega : S \times I \rightarrow O$ est une fonction de sortie étendue en $\omega^* : S \times I^* \rightarrow O^*$

Une Machine de Mealy est déterministe et complète. voir exemple de machine de Mealy Figure4.1.

Définition 4.2 *(Equivalence d'états) Deux états s et s' sont équivalents si :* $\forall \sigma \in I^*, \omega^*(s, \sigma) = \omega^*(s\prime, \sigma)$

Définition 4.3 *(Equivalence de deux machines de Mealy) Deux machines de Mealy M et $M\prime$ sont équivalentes si pour tout état de M, il existe un état équivalent dans M' et inversement.*

Définition 4.4 *Une machine $M = (S, I, O, \sigma, \omega)$ est fortement connexe si et seulement si pour toute paire d'états s et t de M, il existe un chemin de t vers t contenant s*

Définition 4.5 *Une machine $M = (S, I, O, \sigma, \omega)$ est complète si et seulement si* $\forall s \in S$ $\forall a \in I$ *;* $\sigma(s, a) existe.$

Définition 4.6 *Une machine $M = (S, I, O, \sigma, \omega)$ est minimale si et seulement si* $\forall s, t \in S$ *tels que $s \neq t$, s n'est pas équivalent à t*

Définition 4.7 *Une machine $M = (S, I, O, \sigma, \omega)$ est déterministe si et seulement si* $\forall s, t, t' \in S$ *et* $\forall i \in I$, $\forall o \in O$ *tels que* $s \xrightarrow{i/o} t, et\ s \xrightarrow{i/o} t' alors\ t = t'$

Le modèle de base FSM a été étendu de plusieurs façons :CFSM qui modélise les communications par des canaux, EFSM qui enrichi le modèle de base par les variables, paramètres et prédicats[Gil62] et CEFSM qui combine entre les deux extensions précédentes.

4.4 Systèmes de transitions

Un Système de transitions étiquetées(LTS) (Labelled Transitions System) est défini en terme d'états et de transitions, celles ci sont étiquetées. Les étiquettes indiquent le comportement produit durant la transition. Lorsque l'on se trouve dans l'un des états du système de transitions, il est possible de changer d'état en effectuant une action qui étiquette l'une des transitions sortantes de cet état.

4.4.1 Système de transitions étiquetées (LTS)

Définition 4.8 *(Labelled Transition System) LTS [Arn92][Arn94][BT00]est un quadruplet* $(Q, q_0, \Sigma, \rightarrow)$ *tel que :*

- Q est un ensemble dénombrable d'états,
- $q_0 \in Q$ est l'état initial,
- Σ est un alphabet dénombrable d'actions,
- $\rightarrow \subseteq Q \times \Sigma \cup \{\tau\} \times Q$ est un ensemble de transitions. Un élément (s, a, s')de $\rightarrow$ sera simplement noté $s \xrightarrow{a} s'$.

Un alphabet Σ est un ensemble fini d'actions. Une action d'un LTS peut prendre plusieurs formes : une entrée, une sortie, un appel de méthode, etc. Les états sont souvent une abstraction d'états du système (habituellement, il est impossible de représenter tous les états du système). Les actions de Σ sont dites actions observables .Une transition peut être étiquetée soit par une action observable, ou par une action interne (inobservable, ou encore silencieuse) τ .

Une séquence ou chemin est une suite finie d'actions que l'on note simplement par juxtaposition :$\sigma = a_1 a_2 ....a_n$, $a_i \in \Sigma$. L 'ensemble des séquences finies et non vides deΣ est noté Σ^+. La séquence vide est noté ε . Σ^* dénote l'ensemble des séquences de Σ , c'est à dire l'ensemble $\Sigma^+ \cup \{\varepsilon\}$. De plus τ désigne une action qui n'appartient pas à Σ, l'ensemble $\Sigma \cup \{\tau\}$est noté Σ_τ

La longueur d'une séquence σ , notée $|\sigma|$, est le nombre d'actions qui la composent. σ_i dénote la ième action de σ (l'indice i est compris entre 1 et $|\sigma|$).

Le produit de concaténation de deux séquences $\sigma_1 = a_1 a_2 ....a_n$ et $\sigma_2 = b_1 b_2 ....b_m$ est la séquence $\sigma = a_1 a_2 .....a_n b_1 b_2 .....b_m$

Bien entendu, $\varepsilon.\sigma = \sigma.\varepsilon = \sigma$ et $|\sigma_1, \sigma_2| = |\sigma_1| + |\sigma_2|$

Une exécution sur un système de transitions est alors une séquence d'actions $(a_i)_{i \in [1,n]}$.

4.4.2 Notations standards des LTSs :

Soient $S = (Q, q_0, \Sigma, \rightarrow)$ un LTS, a,$\mu_{(i)} \in \Sigma \cup \{\tau\}$, $\alpha_{(i)} \in \Sigma$ une action abservable, $\sigma \in \Sigma^*$ une séquence d'actions observables et $q, q' \in Q$ deux états. Les notations standards des LTSs sont rappelées ci-dessous :

- $q \xrightarrow{a}$ $\stackrel{\Delta}{=}$ $\exists q' \mid q \xrightarrow{a} q'$.
- $q \xrightarrow{\mu_1 \ldots \mu_n} q'$ $\stackrel{\Delta}{=}$ $\exists q_0 \ldots q_n | q = q_0 \xrightarrow{\mu_1} q_1 \xrightarrow{\mu_2} \ldots \xrightarrow{\mu_n} q_n = q'$.

Le comportement observable de S est décrit par la relation $\Rightarrow$:

- $q \stackrel{\varepsilon}{\Rightarrow} q'$ $\stackrel{\Delta}{=}$ $q = q'$ ou $q \xrightarrow{\tau \ldots \tau} q'$.
- $q \stackrel{\alpha}{\Rightarrow} q'$ $\stackrel{\Delta}{=}$ $\exists q_1, q_2 | q \stackrel{\varepsilon}{\Rightarrow} q_1 \xrightarrow{\alpha} q_2 \stackrel{\varepsilon}{\Rightarrow} q'$

- $q \stackrel{\alpha_1 \ldots \alpha_n}{\Rightarrow} q'$ $\stackrel{\Delta}{=}$ $\exists q_0 \ldots q_n | q = q_0 \stackrel{\alpha_1}{\Rightarrow} q_1 \stackrel{\alpha_2}{\Rightarrow} \ldots \stackrel{\alpha_n}{\Rightarrow} q_n = q'$.
- $q \stackrel{\sigma}{\Rightarrow}$ $\stackrel{\Delta}{=}$ $\exists q' | q \stackrel{\sigma}{\Rightarrow} q'$.
- q after σ $\stackrel{\Delta}{=} \left\{ q' \in Q | q \stackrel{\sigma}{\Rightarrow} q' \right\}$, l'ensemble des états atteignables à partir de q par σ. Par extension, Q after $\sigma \stackrel{\Delta}{=} q_0$ after σ.
- Traces (q) $\stackrel{\Delta}{=} \left\{ \sigma \in \Sigma^* | q \stackrel{\sigma}{\Rightarrow} \right\}$, l'ensemble des séquences observables tirables à partir de q. Par convention les traces d'un LTS sont ceux de sont état initial : Traces$(Q) \stackrel{\Delta}{=}$ Traces(q_0) .

4.4.3 Système de transitions à entrée/sortie.

La distinction entre les entrées et les sorties nécessite un modèle plus riche. Plusieurs modèles ont été proposés dont les automates à entrée/sortie (IOA pour Input Output Automata)de Lynch [Lyn88], les automates à entrée/sortie (IOSM pour Input Output State Machines) de Phalippou [Pha94] et les systèmes de transitions à entrée/sortie (IOLTS Input Output Labelled Transition Systems) de Tretmans [Tre96b]. Contrairement aux machines de Mealy dont les transitions portent à la fois une entrée et une sortie, les transitions de ces modèles sont soit des entrées, soit des sorties, soit des actions internes. Ces modèles présentent beaucoup de similarités ainsi que les travaux qui en ont découlé. Le modèle qui nous semble le plus intéressant de part qu'il se rapproche de l'étude de notre intérêt et d'autre part de la quantité de travaux y afférente est celui des IOLTSs que nous présenterons ci-aprés.

4.4.4 Système de transitions étiquetées à entrée/sortie.

Les IOLTSs sont simplement des LTSs où l'on distingue deux types d'actions observables : les entrées et les sorties. Formellement,

Définition 4.9 ***(IOLTS)*** *: Un IOLTS $S = (Q, q_0, \Sigma, \rightarrow)$ est un LTS dont l'alphabet Σ est partitionnée en deux ensembles $\Sigma = \Sigma_o \cup \Sigma_I$, avec Σ_o l'alphabet de sortie et Σ_I l'alphabet d'entrée. L'ensemble des IOLTS S définis sur Σ est noté $IOLTS(\Sigma)$ ou encore $IOLTS(\Sigma_I, \Sigma_o)$ .Une entrée $a \in \Sigma_I$ est notée $?a$ et une sortie $a \in \Sigma_o$ est notée $!a$.*

Exemple : La Figure4.2 ci dessus illustre un exemple d'un IOLTS. Il est composé de trois états et six transitions. L'alphabet Σ correspond à $\{!a, ?b\}$ avec $\Sigma_I = \{?b\}$ et $\Sigma_0 = \{!a\}$.

Notations des IOLTS : En plus des notations standards des LTS , les notations suivantes pour les $IOLTS$ sont utilisées :

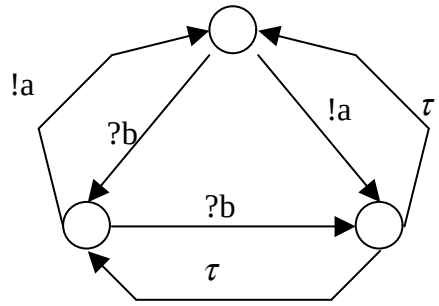

FIG. 4.2 – IOLTS S

-$Out\,(q)$ $\triangleq \left\{a \in \Sigma_o | q \stackrel{a}{\Rightarrow}\right\}$, l'ensemble des sorties possibles de q.
-$Out\,(P)$ $\triangleq \{Out\,(q)\,|q \in P\}$, l'ensemble des sorties possibles de$P \subseteq Q$.
-$Out\,(q, \sigma)$ $\triangleq Out\,(q \text{ after } \sigma)$.

4.4.5 Systèmes de transitions étiquetées maximales (MLTS) [Saï96]

Les modèles de spécifications existant reposent sur une hypothèse primordiale qui suppose l'atomicité structurelle est temporelle des actions. Un des modèle pionnier dans la levée de cette hypothèse est celui introduit dans [Saï96] Ce modèle considère que toute action dure dans le temps. Dans la pratique de la modélisation et du test il est non réaliste de maintenir l'hypothèse de l'atomicité des actions.

Nous allons présenté le modèle qui n'est qu'une variante des systèmes de transitions étiquetées, défini sur un ensemble d'événement qui concrétise le début de l'exécution des action avec les deux fonctions sur les configurations ainsi obtenues, permettant de suivre l'évolution des actions en cours d'exécution. Cette nouvelle approche permet de modéliser le vrai parallélisme ainsi que l'autoconcurrence. Un système de transitions étiquetées maximales n'est autre qu'un graphe d'état bi-étiqueté, tel que une transition est étiquetée par un nom d'action. Un état est étiqueté par l'ensemble des noms d'événements identifiant les actions qui sont potentiellement en cours d'exécution au niveau de cet état là. Ces actions sont dites maximales. Les noms des événements sont choisis à partir d'un ensemble dénombrable noté $\mathcal{M}$.

Définition 4.10 *L'ensemble des atomes de support Act est* $Atm = 2^{\mathcal{M}}_{fn} \times Act \times \mathcal{M}$. $2^{\mathcal{M}}_{fn}$ *étant l'ensemble des parties finies de l'ensemble dénombrable des noms des événements* $\mathcal{M}$. *Cet ensemble est parcouru par* $...x, y, z.M, N$, *... dénotent des sous ensembles finis de* $\mathcal{M}$. *Un atome* (M, a, x) *sera noté* ${}_M a_x$.

Définition 4.11 *Soit* $\mathcal{M}$ *un ensemble dénombrable de noms d'événements, un système de transitions étiquetées maximales de support* $\mathcal{M}$ *est un quintuplés* $(\Omega, \lambda, \mu, \xi, \psi)$ *avec :*

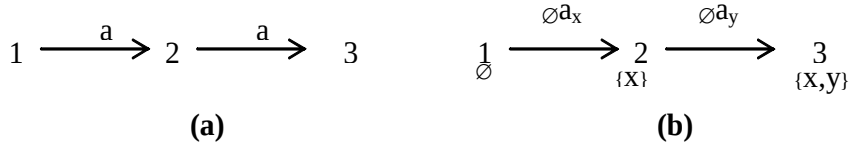

FIG. 4.3 – Système de transitions étiquetées maximales

$\Omega = \langle S, T, \alpha, \beta, s_0 \rangle$ un système de transitions tel que :

• S est un ensemble non vide d'états du système, cet ensemble peut être fini.

• T est un ensemble de transitions indiquant le changement d'état que peut réaliser le système, cet ensemble peut être fini ou infini.

• α et β sont deux applications de T dans S tel que pour toute transition t on a : $\alpha(t)$ est l'origine de la transition et $\beta(t)$ sa déstination.

• s_0 est l'état initial du système de transitions Ω.

(Ω, λ) est un système de transitions étiquetées par la fonction λ sur un alphabet A appelé support de (Ω, λ). $(\lambda : T \rightarrow A)$.

• $\psi : S \rightarrow 2^{\mathcal{M}}_{fn}$ est une fonction qui associe à chaque état l'ensemble fini des noms des événements maximaux présents à son niveau.

• $\mu : T \rightarrow 2^{\mathcal{M}}_{fn}$ est une fonction qui associe à chaque transition l'ensemble fini des noms des événements correspondant aux actions qui ont commencé leur exécution et dont la terminaison sensibilise cette transition.

• $\xi : T \rightarrow \mathcal{M}$ est une fonction qui associe à chaque transition le nom de l'événement qui identifie son occurrence.

tel que $\psi(s_0) = \emptyset$ et pour toute transition t, $\mu(t) \subseteq \psi(\alpha(t))$, $\xi(t) \notin \psi(\alpha(t)) - \mu(t)$ et $\psi(\beta(t)) = (\psi(\alpha(t)) - \mu(t)) \cup \{\xi(t)\}$.

Notation 4.1 *Soit* $Stem = (\Omega, \lambda, \mu, \xi, \psi)$ *un système de transitions étiquetées maximales tel que* $\Omega = \langle S, T, \alpha, \beta, s_0 \rangle$, $t \in T$ *étant une transition telle que* $\alpha(t) = s$, $\beta(t) = s'$, $\lambda(t) = a$, $\mu(t) = E$ *et* $\xi(t) = x$. *La transition* t *sera notée aussi* $s \xrightarrow{E^{a_x}} s'$.

Soit l'expression de comportement Basic LOTOS $E \equiv a; stop|||a; stop$. La Figure 4.3.(a) donne le système de transitions étiquetées obtenu par la sémantique d'entrelacement. Ce système de transition est exactement le même que celui de l'expression de comportement $F \equiv a; a; stop$ représentant l'exécution séquentielle de deux actions a. L'application de la sémantique de maximalité génère le système de transitions de la Figure 4.3.(b) dans le quel l'ensemble $\{x, y\}$ exprime le fait que deux exécutions de l'action a peuvent avoir lieu en parallèle dans l'état final.

4.4.6 Modélisation et spécification des systèmes temporisés

La vérification des systèmes temporisés nécessite un formalisme adapté aux contraintes temporelles quantitatives nous rappelons dans la suite les systèmes de transitions temporisés ainsi que le modèle des automates temporisés avec durées d'actions DATAs.

1- Domaine du temps et sémantique temporisée.

Les domaines traditionnellement utilisés pour le temps sont les ensembles $\mathbb{N}$, $\mathbb{Q}_+$,$\mathbb{R}_+$ le premier est un cas particulier du domaines discret, parfois modélisé par des séquences d'événements lors des quelles il est possible d'observer les états du systèmes, les deux autres représente un temps dense. Dans[NSY93] une proposition d'un modèle unifier du domaine du temps sous la forme d'un monoïde commutatif $(\mathbb{T}, +)$ ou l'élément neutre est noté (0) vérifiant les propriétés suivantes :

$\forall t, t' \in \mathbb{T} \;\; t + t' = t \Leftrightarrow t' = 0$ la realtion $\leq$ définie par $\{t \leq t' si\ \exists t" \in \mathbb{T} telquet\prime = t + t"\}$ est un ordre total sur $\mathbb{T}$.

Aussi les éléments de $\mathbb{T}$ représentent des dates et la différence entre deux dates correspond à une durée également dans $\mathbb{T}$.

2- Systeme de transitions temporisé.

Les systèmes de transitions temporisés (TLTS) permettent de décrire la dynamique des systèmes combinant trois types d'actions : actions "passage du temps", actions observables et les actions internes τ, toutes les actions excepter celles du passage du temps sont instantanées (i.e. sans consommation de temps).

Un système de transitions temporisé est un quadruplet $\mathcal{T} = (S, L, T, s_0)$ avec :

• S l'ensemble des états non vide ou de configurations dont lesquels peut se trouver le système, cet ensemble peut être fini ou infini.

• $L \stackrel{def}{=} Act \cup \{\tau\} \cup \mathbb{T}$ représente l'ensemble des actions observables, internes et les actions "passage du temps".

• s_0 est l'état initial du système de transitions $\mathcal{T}$

• $T \subset S \times (L) \times S$ est l'ensemble de transitions indiquant le changement d'état que peut réaliser le système, cet ensemble peut être fini ou infini.

les transitions doivent de plus vérifier les propriétés ci-dessous :

• Déterminisme temporel : si $s \xrightarrow{d} s'$et $s \xrightarrow{d} s"$, avec $d \in \mathbb{T}$ alors $s\prime = s"$

• Additivité : si $s \xrightarrow{d} s'$et $s' \xrightarrow{d'} s"$, avec $d, d' \in \mathbb{T}$ alors $s\prime \xrightarrow{d+d'} s"$

• Continuité : si $s \xrightarrow{d} s'$alors pour tout $d\prime, d" \in \mathbb{T}$ avec $d = d\prime + d", \exists s" \in S$ tel que : $s \xrightarrow{d'} s" \xrightarrow{d"} s\prime$

• Durée 0 : pour tout $s \in S, s \xrightarrow{0} s$.

D'autres définitions sur le modèle sont disponibles dans[LBB05].

4.4.7 Discussion

Dans le domaine temporisé les langages de spécification ainsi que le modèle sémantique sous jacent en l'occurrence TLTS manipulent le temps sous forme d'actions distinctes, ainsi des propriétés quantitatives des systèmes peuvent être prise en compte lors de la modélisation et/ou du test. Les actions autres que celles exprimant le passage du temps sont toute fois instantanées. Des modèles plus élaborés tels que les Automates temporisés[AD90][AD94] manipulent le temps sous forme de contraintes sur les transitions ou des gardes. Des travaux important ont été réalisés sur ce modèle ainsi que ses extensions temps réels.

Néanmoins l'impléméntabilité de ses systèmes reste un inconvénient jusque là non amoindris car l'hypothèse de l'atomicité des actions demeure non réaliste[KA05].

Le modèle des DATAs présenté dans la section suivante et aux frontières des modèles discrets et ceux temps réels. Car il considére les durées des actions sous forme de contraintes temporelles sans pour autant manipuler les contraintes temporelles proprement dites du système, il se présente comme une alternative pour une modélisation plus robuste et implémentable.

4.4.8 Le modéle des DATAs "Durational Action Timed Automata"- Introduction

Les modèles déjà cités dans le contexte non temporisé supposent la non-atomicité des actions (temporelle et structurelle) que se soit ceux basés sur l'algèbre de processus ou issue des systèmes de transitions étiquetées la sémantique sous jacente est celle de l'entrelacement. Intuitivement elle suppose l'atomicité des actions et réduit l'exécution des actions parallèles à leur exécution entrelacer dans le temps. Cette hypothèse peut rendre invalides certaines spécifications et même le test exécuter sur la base de cette modélisation, de même les modèles temporisés qui formellement traitent trois types d'actions (i) les actions observables, (ii) les actions internes, (iii) et le passage de temps matérialisé par une durée[MOK06][LBB05][BDGP98], ils se résignent à exécuter des actions instantanées. La durée des actions n'a jamais été prise en compte explicitement.

Dans ce contexte le modèle des DATAs pour (Durational Action Timed Automata model) [ref acit05] résulte de la combinaison des modèles de transition temporisés et de la sémantique de maximalité[Dev92][CS95][Saï96].

Le modèle des DATAs considère que chaque action dure dans le temps, cette quantité est capturée par la condition de durée sur un état, destination d'une transition étiquetée par cette action, toutes les transitions offertes sont possibles alors que toutes celles qui dépendent de la terminaison de l'action en cours sont inhibées jusqu'à ce que la condition d'exécution attribuée à ces transitions soit vérifiée.

Les actions qui doivent s'exécuter en parallèle sont au contraire lancer sans être concerné par les conditions de durées. Dans ce cas la condition de durée offre explicitement l'information sur l'évolution du système. La condition de durée d'une action a décore chaque état

atteint par l'exécution de l'action a. Alors que toute transition est gardée par une condition d'exécution, c'est une conjonction sur des conditions de durée, elles concernent une transition dont le passage est conditionné par la terminaison d'un ensemble d'actions (dépendance causale dans le sens d'événements maximaux).

Ce modèle est aux frontières des modèles temporisés est ceux non temporisés car les contraintes temporelles qu'il manipule expriment la réalité physique de tout système sans pour autant traiter les contraintes temps réel de celui ci.

Le modèle ainsi défini ouvre des perspectives très intéressantes dans des axes d'utilisation importants ceci est dû à trois facteurs qui seront détaillés dans la discussion de fin de Chapitre.

1. L'action interne est considérée comme toute autre action est possède une durée. Il a été prouvé que dans le contexte temporel la considération des actions internes par les modèles temporisés augmente leur expressivité[BB96][VD97].

2. L'implementabilité ainsi que la robustesse du modèle des DATAs par essence même de la sémantique qui lui est associée[KA05].

3. Le modèle est déterminisable, ce qui le prête fortement à l'automatisation de la génération du test.

Jusqu'à présent la spécification et l'implémentation pour le test restent des modèles mathématiques parfait, la réalisation du test se heurte aux réalités physiques des plates formes d'exécution. Nous avons eu l'intuition que le modèle des DATAs se présente comme une alternative intéressante aux modèles existants du point de vu implémentabilité et robustesse.

Il est bien connu que la modélisation des systèmes complexes nécessite des modèles non-déterministes, alors que dans la pratique de la vérification formelle ou du test il est nécessaire de passer au modèle déterministe s'il existe. Cependant plusieurs modèles temporisés ne possèdent pas d'équivalent déterministe[Alu99][AD94].

L'opération de déterminisation reste une phase importante dans le processus d'utilisation de n'importe quel modèle destiné à spécifier les systèmes. Comme vu dans le Chapitre3, il consiste à éliminer les actions non observables ainsi que le non-déterminisme introduit par le choix des actions observables.

Dans le domaine non temporisé la déterminisation des systèmes de transition est exactement la même que l'opération définie sur les automates d'états finis[HU79], en interprétant les transitions par les actions internes comme des $\varepsilon - transitions$ [JER04]. Plusieurs algorithmes sont proposés dans ce but[KD01][Bri88][PSS93][Dri92][F.M92].

Dans ce Chapitre on se propose de formaliser le processus de déterministaion spécifique au modèle des DATAs. Cette opération manipule les contraintes temporelles exprimant la durée des actions. Durant le processus de déterminisation le modèle des DATAs ne perd aucune information sur les durées des actions. Aussi, tous les types de blocages (quiescence) existants dans la spécification sont préservés sans aucun autre traitement particulier.

4.4.9 Le modéle des DATAs - Intuition

La modélisation des durées associées aux actions est inspirée de la sémantique de maximalité. Dans une représentation par les systèmes de transitions, celle-ci représente le début d'exécution d'une action. Dans l'état résultant on dit que l'action est éventuellement en cours d'exécution, aucune conclusion ne peut être tirée concernant la fin de son exécution, cependant cette information peut être déduite dans un état ultérieur dans lequel une action qui lui est causalement dépendante est exécutée. L'association de durées explicites aux actions va nous permettre de matérialiser et le début et la fin d'exécution des actions.

Considérons l'exemple d'un système S composé de deux sous-systèmes $S1$ et $S2$ s'exécutant en parallèle et se synchronisant sur une action d . Le sous-système $S1$ exécute l'action a suivie de d , tandis que $S2$ exécute b puis d , représenté par la Figure4.4. A partir de l'état initials_0 , les deux actions a et b peuvent commencer leur exécution indépendamment l'une de l'autre. Puisque nous pouvons avoir le cas où ces deux actions s'exécutent en parallèle, nous allons attribuer à chacune d'elles une horloge,x et y respectivement, pour distinguer leurs occurrences. Donc, à partir de l'états_0 , les deux transitions suivantes sont possibles :$s_0 \overset{a,x:=0}{\longrightarrow} s_1$ et $s_0 \overset{b,y:=0}{\longrightarrow} s_2$. Une transition étiquetée par a désigne le début d'exécution de l'action a , l'horloge qui lui est associée comptabilise l'évolution dans le temps de cette action.

En suivant le même raisonnement, les deux transitions suivantes sont possibles $s_1 \overset{b,y:=0}{\longrightarrow} s_3$et $s_2 \overset{a,x:=0}{\longrightarrow} s_3$ le comportement du système S jusqu'ici est illustré par la Figure4.4.(a) . A partir de l'état s_3, l'action d ne peut évidemment commencer son exécution que si les deux actions a et b ont terminé leurs exécutions. La transition relative à d ne peut être tirer que si une condition portant sur les exécutions de a et de b est satisfaite. Cette condition, appelée condition d'exécution, est construite en fonction des durées de a et de b . Après le tirage de la transition $s_0 \overset{a,x:=0}{\longrightarrow} s_1$, nous avons besoin d'une information sur l'exécution éventuelle de l'action a dans l'état s_1. On est sûre que l'action a termine son exécution lorsque l'horloge correspondante x atteint la valeur qui représente la durée de l'action a par exemple 10 unités de temps, alors, on ajoute à l'état s_1 la condition sur la durée de a , $\{x \geq 10\}$, qui signifie que si la valeur de x est supérieure ou égale à 10, on est sûre que l'action a a fini de s'exécuter. Le même traitement pour l'état s_2 qui sera étiqueté par $\{y \geq 12\}$ relativement à la durée de l'action b qui est de 12 unités de temps, y représente l'horloge correspondante. A l'état s_0, aucune action n'est en exécution, ce qui explique que l'ensemble des conditions sur les durées soit vide. A l'état s_3, les actions a et b peuvent éventuellement s'exécuter en parallèle, seule une valuation des deux horloges permettra de juger de la terminaison de l'une ou l'autre des deux actions. D'où l'ensemble des conditions sur les durées $\{x \geq 10, y \geq 12\}$. La condition d'exécution de l'action d devient alors $x \geq 10 \wedge y \geq 12$. A l'état s_3 la condition sur les durées des actions a et b signifie la possibilité de leurs évolutions parallèles, Figure4.4.(b). une telle structure est dite un des DATAs (*Durational Action Timed Automata).*

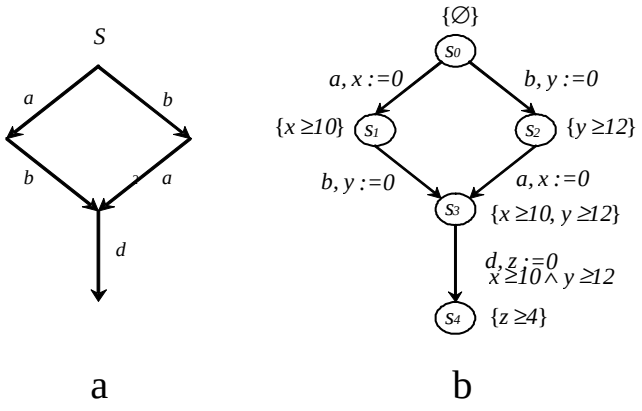

FIG. 4.4 – Le système S et sa représentation dans le modèle des DATAs

4.5 Conclusion

Dans ce Chapitre nous avons présenté un état de l'art concernant les modèles les plus utilisés pour spécifier les systèmes logiciels ou matériels. Les machines à état finis (FSMs) sont souvent utilisées dans le domaine du matériel pour décrire le fonctionnement des circuits logiques.

Dans le domaine des protocoles, les langages de spécification (SDL, Lotos, Estelle, etc), normalisés par l 'ISO et ITU, sont les formalismes les plus utilisés pour écrire les spécifications. Ces langages reposent généralement sur une sémantique des systèmes de transitions étiquetées à entrée/sortie (IOLTS).

Aussi la sémantique sous-jacente à tous ces langages est la sémantique d'entrelacement, qui se résume en l'atomicité structurelle et temporelle des actions ainsi que l'exécution entrelacée dans le temps des actions parallèles.

Le modèle qui semble le mieux approché la réalité physique est celui des STEMs ou MLTSs (Maximality based labelled Transition System) car ils abordent la modélisation d'une autre manière. Cette approche nous semble plus réaliste est prometteuse ce qui nous a incité à faire la suite de se travail sur le même axe de réalisme est de facilité opérationnelle en voulant affiner un modèle qui prolonge les STEMs par explicitation des durées des actions, en l'occurrence le modèle des DATAs (Durational Action Timed Automata). Une formalisation du modèle des DATAs ainsi que le processus de déterminisation sont présentés dans le Chapitre6.

Chapitre 5

Génération de test

5.1 Les grandes familles d'algorithmes de génération de test

Historiquement, on peut distinguer deux grandes familles de méthodes de génération de séquences de test de conformité. La première est issue des méthodes de test de circuits séquentiels. Cette méthode s'appuie sur une modélisation par automates.

La seconde famille de méthodes est issue des algèbres de processus et construit les séquences de test à partir d'un système de transition. Ces deux familles de méthodes diffèrent principalement par le modèle utilisé pour représenter la spécification. De ce fait, les algorithmes de génération à partir de ces modèles diffèrent eux aussi. Dans la suite, nous présentons les principaux concepts de ces deux méthodes. Un état de l'art plus complet est disponible dans[Jér01].

5.2 Méthodes de test basées sur les automates

La première classe de méthodes est basée sur une spécification représentée à base de Machines de Mealy. Le principe du test pour ce type de modèle est de déterminer si les machines$I$ et S, représentant respectivement l'implémentation et la spécification, sont équivalentes. Plusieurs algorithmes permettent de détecter si deux machines sont équivalentes. Nous présentons ici les techniques suivantes :

1-Tour de transitions [NT81].

2- séquence de distinction DS-Method [G.G70].

3-séquence de caractérisation W-method[Cho78] et enfin

4- séquence unique d'entrée/sorties UIO-method[LY96].

Définition 5.1 ***(Le test de conformité des machines de Mealy).*** *Soit S une Machine de Mealy représentant la spécification et I une Machine de Mealy correspondant à l'implémentation, la conformité consiste à déterminer si I et S sont équivalentes.*

Le modèle de faute permet de déterminer les comportements erronés qui montrent que deux machines de Mealy ne sont pas équivalentes.

Les deux machines de Mealy I et S sont équivalentes si aucune des fautes suivantes n'a été détectée :

- Faute de sortie. La sortie produite après une entrée n'est pas celle attendue.
- Faute de transfert. L'état d'arrivée après une transition n'est pas celui attendu.

Séquence d'identification. Les tests construits doivent pouvoir détecter les fautes de sorties et les fautes de transferts. Il existe plusieurs méthodes de dérivation de tests basées sur le même principe et qui diffèrent par l'identification de l'état d'arrivée. Une séquence d'identification est une séquence d'entrées/sorties qui doit permettre d'identifier certainement l'état d'arrivée. Dans la suite on va présenter les différentes séquences d'identification utilisées en fonction de la méthode de test choisie.

5.2.1 Méthode T : Tour de transitions

Cette méthode s'applique uniquement sur une spécification minimale, fortement connexe et complètement spécifiée décrite par une machine de Mealy. Une suite de test pour cette spécification consiste à passer par toutes les transitions du modèle au moins une fois, En général, les fautes de transfert ne sont pas détectées par cette méthode. Cependant, les machines testées ont parfois des transitions spéciales en chaque état qui renvoient le numéro de l'état courant.

Trouver un tour de transitions de longueur minimale se ramène à trouver un circuit Eulérien. Un circuit Eulérien est un chemin passant une fois et une seule par toutes les transitions et revenant dans son état initial.

La complexité des opérations sur le système ainsi modélisé est polynomiale.

5.2.2 Méthode DS : Séquence de Distinction

Une séquence de distinction d'une machine M est une séquence unique d'entrées qui produit une séquence de sortie différente pour chacun des états de M. Cette technique permet l'identification de l'état initial de la machine.

L'algorithme consiste à examiner l'arbre de successeurs appelé arbre distinguant, dont la longueur est de la taille de la séquence de distinction. La longueur d'une séquence est exponentielle et l'algorithme est P-space complet [Gil62]. Cette séquence n'existe pas sur une machine non minimale. De plus, elle est souvent longue et existe rarement, même sur une machine minimale.

Formellement : $\exists DS \in I*, \forall s_i \in S, \forall s_j \in S, s_i \neq s_j, \omega^*(s_i, DS) \neq \omega^*(s_j, DS)$ qui signifie : Il existe une séquence d'entrées DS capable de différencier toutes les paires d'états.

5.2.3 Méthode U : Unique input output (UIO)

Une **UIO** d'un état s d'une machine M est une séquence qui distingue s de tout autre état de la machine, de telle manière qu'aucun autre état n'a le même comportement que s, i.e. la même séquence de sortie, après l'application de la séquence UIO. Cette technique consiste à vérifier que la machine est bien dans un état particulier.

La complexité du calcul des UIO est exponentielle, ainsi que leur taille[LY94], en pratique, une séquence UIO est courte si elle existe.

Un test est constitué d'une transition de la machine concaténée à l'UIO identifiant son état d'arrivée. Le problème est alors de trouver un enchaînement de ces tests de longueur minimale. Ce qui est réalisé en trouvant un chemin eulérien.

Formellement : $\exists s_i \in S, \exists UIO \in I^*, \forall s_j \in S, \omega^*(s_i, UIO_i) \neq \omega^*(s_j, UIO_i)$: Il existe une séquence d'entrées qui différencie s_i de tous les s_j.

5.2.4 Méthode W : séquences caractéristiques

Une séquence caractéristique est une séquence qui distingue deux états d'une machine de Mealy. Un ensemble de caractérisation est un ensemble de séquences caractéristiques d'une machine M. Chaque couple d'états de M a une séquence caractéristique dans l'ensemble de caractérisation.

Formellement : $\exists ls_i \in S, \forall s_j \in S, \exists c_{ij} \in I^*, \omega^*(s_i, c_{ij}) \neq \omega^*(s_j, c_{ij})$ autrement dit :

Toutes paires d'états, s_i, s_j peuvent être différenciées par une séquence d'entrées c_{ij}

L'existence d'un ensemble de caractérisation est assurée aussitôt que la machine est minimale. La complexité de la construction de séquences caractéristiques est en $O(n^2)$[Cho78]. Comme pour la méthode **U**, un test correspond à une transition suivie de séquences caractéristiques. Une suite de tests est produite en concaténant, de manière optimale, l'ensemble des tests.

5.2.5 Synthèse

Les méthodes basées sur les automates ont certains avantages et certains inconvénients. Cette théorie est bien fondée et la couverture des fautes est complète modulo le modèle de fautes (fautes de sortie et de transfert). Mais beaucoup d'hypothèses sont faites restreignant ainsi la classe des spécifications qui peuvent être traitées et des IUTs qui peuvent être testées. La spécification doit être dans beaucoup de cas fortement connexe et complète, et de plus l'IUT doit être minimale. Les algorithmes connus de ces méthodes sont souvent polynomiaux en temps et en mémoire, et ceci réduit d'autant plus la taille des spécifications en entrée.

5.3 Méthodes de test basées sur les systèmes de transitions

La synthèse faite dans cette section est globalement tirée du document[Kho06]. Les principaux travaux sur les méthodes basées sur les systèmes de transitions sont fondés sur la

comparaison entre les modèles de la spécification et de l'implémentation à l'aide d'une relation formelle. Cette relation est dite relation de conformité, relation d'implémentation ou aussi relation d'équivalence.

Les relations de conformité sont un moyen de comparer deux systèmes, elles permettent d'exprimer ce qu'est la conformité d'une implémentation par rapport à une spécification.

5.3.1 Le concept de la relation de conformité

"Qu 'est ce que l'on entend par une implémentation valide ?" La réponse peut s'obtenir à l'aide du concept de validité par équivalence communément rencontré dans les techniques basées sur les algèbres de processus. Etant données une spécification S et une relation d 'équivalence R , on peut décrire l'ensemble des implémentations valides de S par l'ensemble des systèmes qui lui sont équivalents par R , c'est à dire l'ensemble : $\{I \mid I\ R\ S\}$, diverses relations d'équivalence ont été proposées dans[dH84][Hen88].

Les relations de conformité sont souvent (mais pas toujours) des préordres, (des relations réflexives et transitives pas nécessairement symétriques). elles sont les plus souvent utilisée[Bri88] car elles s'accordent assez bien au contexte de conformité et de test de l'ISO.

Quelques relations de conformité

- **Relation** $\leq_{tr}$ Parmi les notations standards des LTSs, nous avons défini $Traces(S)$ comme étant l'ensemble des séquences de S à partir de l'état initial. La première relation de conformité est celle dite de l'inclusion des traces. Cette relation vise à vérifier que toute trace de l'implémentation est une trace de la spécification. Formellement :

Définition 5.2 *Soient* $S, I \in LTS(\Sigma) : I \leq_{tr} S =_{\Delta} Traces(I) \subseteq Traces(S)$.

Il est facile de voir que $\leq_{tr}$ est réflexive et transitive, donc c'est un préordre.

- **Relation** $conf$ Cette relation a été proposée par Brinksma[BSS87], $conf$ est basée sur l'observation de l'évolution et des blocages d'un système. Formellement :

Définition 5.3 B_1, B_2 *deux systèmes :* $B_1 conf\ \ B_2$ *si* $\forall \sigma \in Traces(B_2)$,

$$\forall A \subset \Sigma \ \begin{cases} si \quad \exists B_1' \forall a \in A\ B_1 \stackrel{\sigma}{\Longrightarrow} B_1' \stackrel{a}{\not\Rightarrow} \\ alors \quad \exists B_2' \forall a \in A\ B_2 \stackrel{\sigma}{\Longrightarrow} B_2' \stackrel{a}{\not\Rightarrow} \end{cases}$$

$B_1 conf\ \ B_2$ implique que si le système B_1 évolue après l'exécution de la trace σ alors, le système B_2 devrait aussi évoluer. De plus, si B_1 se bloque après une action a alors, le système B_2 devrait se bloquer aussi après la même action.

- **Relation** $ioconf$ **:** La relation $conf$ est basée sur les traces et ne fait pas de distinction entre les entrées qui sont contrôlables par l'environnement du système et les sorties qui sont contrôlables par le système lui même. Tretmans[Tre96b] a introduit la relation

ioconf qui fait une telle distinction. Cette relation établit qu'une implémentation I est conforme à une spécification S si, après une trace de S , les sorties produites par I sont prévues par S .

Rappelons que $Out(S, \sigma)$ est l'ensemble des événements (actions observables) produits par S après l'application de la trace σ. Formellement :

Définition 5.4 *Soient* $S, I \in IOLTS(\Sigma)$:
I *ioconf* $S =_{\Delta} \forall \sigma \in Traces(S) \Rightarrow Out(I, \sigma) \subseteq Out(S, \sigma)$.

La relation *ioconf* permet la spécification partielle, par conséquent, l'implémentation peut ajouter un traitement additionnel non prévue par la spécification.

- **Relation** *ioco* : Une évolution dans la théorie du test, permet d'observer le blocage d'un système (le système s'arrête d'évoluer) par l 'utilisation de temporisateurs. Le blocage ou l'absence d'action dans un système peut avoir plusieurs causes.

  La théorie *ioco* de Tretmans[Tre96b][Tre96a] considère les sorties et les blocages possibles d'une spécification, pour ce faire, ceux ci sont modélisés dans la spécification. En effet, l'observation d'un blocage de l'implémentation ne doit pas forcément produire un verdict **Fail** , qui correspond au rejet de l'implémentation, si celui ci est prévu dans la spécification. A partir de l'IOLTS S de la spécification, il est proposé la construction d'un IOLTS S^{δ}, appelé Automate suspendu , obtenu par l'ajout des informations de blocage. Dans S^{δ}, un blocage est modélisé par un événement de sortie $!\delta$ visible par l'environnement et ne faisant pas partie des événements de la spécification. S^{δ} est construit en ajoutant des boucles $q \xrightarrow{!\delta} q$ pour chaque état de blocage q [Voir section 5.6.4]. La relation de conformité *ioco* est définie comme suit :

Définition 5.5 *Soient* $S, I \in IOLTS(\Sigma)$:
I *iocoS* $=_{\Delta} \forall \sigma \in Traces(S^{\delta}) \Rightarrow Out(I^{\delta}, \sigma) \subseteq Out(S^{\delta}, \sigma)$.

ioco étend *ioconf* en considérant non seulement les traces de S mais aussi les traces avec blocages de S, c'est à dire les traces de l'automate S^{δ}.

5.3.2 Approche de génération de test

Les méthodes de génération des tests de conformité qui sont basées sur les systèmes de transitions peuvent être classées en deux sous-catégories : la génération aléatoire des tests et la génération à travers des objectifs de test.

1- Génération aléatoires.

Dans[Tre96b][Tre96a] ont été décrit des algorithmes pour de génération de test aléatoires pour les relations *ioconf* et *ioco*. Lidée principale de ces algorithmes se base sur le parcourt du système de transition S (ou S^{δ} pour la relation *ioco*). A chaque pas, soit :

1. L'arrêt et le verdict associé à l'état courant est Pass .

2. Le choix d'une entrée i de S (ou S^{δ} pour la relation *ioco*) et application de l'algorithme récursivement sur S after i (ou S^{δ} after i pour la relation *ioco*).

3. Pour toutes les sorties (et les blocages pour la relation *ioco*)de l'alphabet. Dans le cas, d'une sortie ou d'un blocage x de S (ou S^{δ}) application de l'algorithme récursivement sur S after x (ou S^{δ} after x pour la relation *ioco*), pour chaque sortie ou blocage non autorisé dans S ou (ou S^{δ}) le verdict associe est Fail .

L'algorithme peut produire un nombre infini de cas de test. L'outil TorX a été produit dans le cadre de cette technique conçus en collaboration entre l'université de Twente, l'université de Technologie de Eindhoven et les industriels Philips, Research Laboratories et Lucent Technologies.

2- Génération à travers des objectif de test.

Un des ingrédients principaux de cette technique est l'utilisation d'objectifs de test pour générer les cas de test à partir d'une spécification formelle d'un système. Chaque objectif de test est considéré comme une description abstraite du comportement à tester (une partie de la spécification). Formellement, un objectif de test est modélisé par un IOLTS équipé de deux états spéciaux Accept et Reject pour décrire la satisfaction de cet objectif.

Il sert à sélectionner des séquences de la spécificationdites cas de test . La dérivation des cas de test se décrit fonctionnellement en plusieurs étapes :

1. Un produit synchrone entre la spécification et l'objectif de test ;

2. La suppression des actions internes et la déterminisation du modèle ;

3. La synthèse d'un sous-graphe et la résolution des conflits de contrôlabilité.

Les algorithmes utilisés fonctionnent à la volée (on-the-fly)ce qui signifie qu'une exécution de l'algorithme principal va activer l'exécution de chacune des étapes intermédiaires. Le terme à la volée signifie également que pour produire le résultat, il ne sera pas nécessaire de construire les résultats complets de chaque étape, ni même exiger la complitude en entrée.

Un cas de test est une expérience réalisée par le testeur sur l'implémentation. Il est modélisé par un $IOLTS$ dont le graphe associé est un arbre. Les branches d'un cas de test décrivent les séquences d'interactions entre le testeur et l'IUT . Le rôle d'un cas de test est de détecter si l'IUT est conforme à sa spécification (la conformité étant donnée par une relation de conformité : $ioconf, ioco$,...).

D'autres techniques utilisent des objectif de test pour la sélection des tests ceux ci on été fondateur pour la création d'outils :

Notons par exemple,

1- L'outil Samstag où les objectifs sont fournis sous forme de MSC (Message Sequence Charts) et la spécification est donnée en SDL.

2- L'outil STG (Symbolic Test Generator)[DCZ02][DC01] qui prend en entrée une spécification et un objectif de test symboliques.

3-Cette technique a mené au développement de l'outil TGV (Test Generation with Verification technologies)[TJ99][Jér02].

5.4 Synthèse

Dans cette partie du Chapitre nous avons passé en revue un état de l'art sur les différentes approches et outils développés pour la génération des tests de conformité. La majorité des méthodes basées sur les FSMs permettent l'identification de l'état d'arrivée. En revanche, leur applicabilité est limitée d'une part, à cause des hypothèses portées sur les modèles (complétude, forte connexité, déterminisme,...) et, d'autre part à cause de la complexité de leurs algorithmes. De plus, la sémantique du modèle FSM est différente de celle utilisée dans les langages de spécification, ceux-ci sont en général basées sur la sémantique des systèmes de transitions.

Les méthodes basées sur les systèmes de transitions sont plus proches des langages de spécification, de plus, elles ne font à priori aucune hypothèse sur les modèles ce qui élargit leurs champs d'application. D'un point de vue algorithmique, ces techniques présentent une meilleure complexité (linéaire excepté la déterminisation qui est exponentielle) [Jér02]. En revanche, ces méthodes sont non-exhaustives et notamment à cause de l'utilisation d'objectifs de test.

Un inconvénient commun entre les deux familles de méthodes précédentes est l'absence totale de la considération des données dans le test. Pour ce faire, les méthodes basées sur les EFSM et les graphes de flux de données[JER04] ont ouvert une autre vision pour coupler les contrôles et les données dans les séquences de test. Même si la limite de leur applicabilité reste à étudier.

5.5 L'indeterminisme

Le possible non déterminisme de la spécification fait qu'une même trace peut mener à plusieurs états distincts dans lesquels les ensembles d'entrées possibles, (impossibles ou non spécifiés selon ou on se place dans l'étude) sont différents. Lorsque l'on observe une implémentation sous test, on ne connaît d'elle que la trace observable qui est exécutée, On ne sait pas exactement dans quel état l'implémentation se trouve par rapport à la spécification.

On ne peut que considérer l'ensemble des états de la spécification atteignables après cette trace pour définir quelles sont les actions autorisées.

Dans la suite on va tenter de clarifier ce qui est désigné dans la littérature par les termes Choix, Non déterminisme et complétude, qui sont souvent employés avec des acceptions

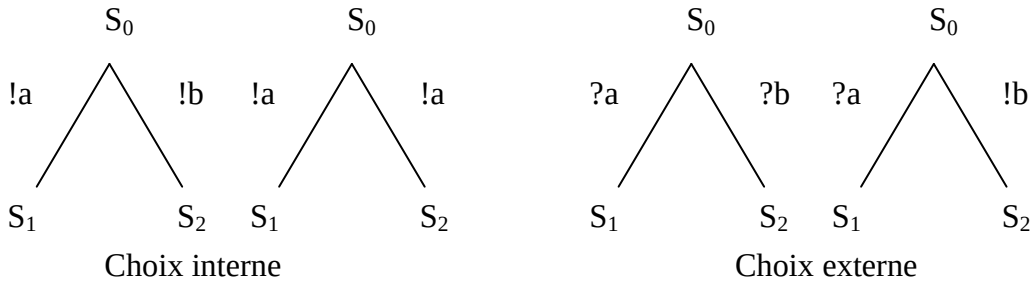

FIG. 5.1 – Différents types de choix

variables.

5.5.1 Choix et indeterminisme[Pha94]

On dit qu'on a **un choix**, lorsque plusieurs transitions sont possibles à partir d'un état donné de l'automate. Ce choix peut se faire entre des transitions qui sont toutes des émissions, toutes des réceptions ou une combinaison des deux. A partir d'un état donné on peut avoir un choix entre des transitions ayant la même interaction Figure5.1.

• Lorsque les transitions concernées sont toutes des émissions on dit qu'on a un **choix interne** puisque la décision de déclencher une transition particulière est locale à l'automate.

• Si on a affaire à un choix entre des réceptions différentes, on dit qu'on a un **choix externe**, la transition qui est déclenchée est choisie par l'environnement, c'est à dire le monde extérieur qui envoie des interactions à l'automate.

• On peut aussi avoir un choix entre des émissions et des réceptions. Vu l'interprétation que l'on donne à la progression du temps, on considère qu'une réception est déclenchée dès qu'elle arrive, la machine pouvant décider en interne de faire une émission avant.

Le terme de **non-déterminisme** est employé en informatique avec des sens différents selon le contexte dans lequel on l'utilise. Intuitivement, un système est non-déterministe lorsqu'il ne permet pas de contrôler ses évolutions futures possibles. Trois formes de non-déterminisme sont intéressantes Figure5.2 :

• Dans la théorie des automates, le futur, c'est la transition suivante que l'on peut choisir : on parle de non-déterminisme lorsqu'on a plusieurs transitions partant du même état et étiquetées par la même action[HU79]. Selon cette définition, dans un modèle de transitions le non-déterminisme, peut résulter soit de deux actions identiques(émission ou réceptions), ou deux transitions internes.

• Dans les théories d'algèbres de processus, le futur, c'est l'action observable suivante qui va se produire dans le système. Donc on parle de non-déterminisme dans le cas décrit ci-dessus, mais aussi dans le cas où un choix est possible entre une transition étiquetée par une action observable et une transition étiquetée par l'action interne[Klo92]. On utilise cette

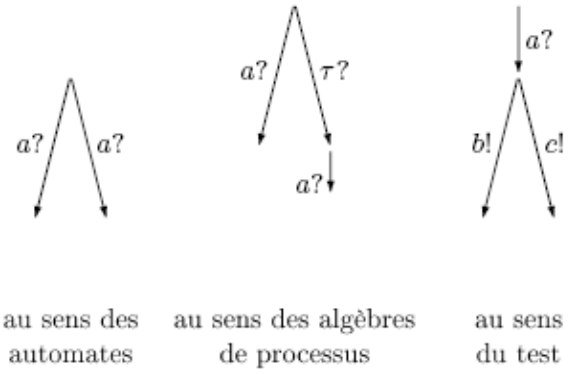

FIG. 5.2 – Différents types de non-déterminisme

appellation de "non-déterminisme" car l'environnement ne contrôle pas le comportement de l'automate, qui peut décider localement de faire la transition interne.

• En ce qui concerne le test, le futur c'est : que va répondre le système si on lui envoie telle interaction ? En sus des deux formes de non-déterminisme déjà citées, on parle de non-déterminisme dès que le comportement du système en réponse à une sollicitation n'est pas unique. C'est le sens utilisé habituellement dans le contexte du test[SF].

Le modèle de la spécification est d'autant plus expressif s'il est non-déterministe, mais la déterminisation reste une tache primordiale du moment que le modèle utiliser pour l'implémentation sous test ainsi que le testeur doivent être déterministe(ne gardant que l'aspect observable).

D'un point de vue historique, les travaux sur les méthodes de test d'automates finis ont été basés sur des modèles déterministes [Voir [Cho78] [SF] [DS89]]. Cependant la nécessité de considérer des modèles non-déterministes pour traiter des applications réelles apparaît de plus en plus clairement, ce qui explique une abondance récente de travaux sur ce thème[SF91] [Klo92] [PT92].

Les automates à entrées et sorties ont été interprété comme des machines vers les quelles on envoie des interactions et qui répondent en retour. Il est clair qu'il est toujours possible dans la réalité d'envoyer vers une implémentation n'importe quelle interaction. Nous avons affaire à une boîte noire, qui, dans tous les cas, doit avoir une réaction (qui peut être, bien sûr, de ne rien répondre). Or selon la définition un automate ne possède pas nécessairement, dans un état donné, de transition de réception pour toutes les interactions du langage L : on a affaire dans ce cas à une spécification incomplète. Formellement :

Définition 5.6 *Un automate est complèt si pour chaque état s et pour chaque interaction a il existe une transition correspondant à la réception de a dans s. On dit que l'automate est incomplèt sinon.*

Les implémentations doivent toujours être modélisées par des automates complets. Par contre les spécifications peuvent être décrites par des automates complets ou incomplets. Nous considérons nécessaire de prendre en compte le cas de spécifications incomplètes, car dans la réalité ce phénomène est courant dans le cas des applications visées par la théorie du test. Les spécifications incomplètes, tout comme les spécifications non-déterministes, sont d'ailleurs un sujet d'études très en vogue actuellement[Pet91][GL93]. Il est important de remarquer que cette définition de la complétude correspond à la notion généralement employée dans le domaine du test, elle diffère de la définition usuelle de la complétude des automates d'états finis.

En effet, la complétude ci-dessus est définie uniquement par rapport aux réceptions de l'automate à entrées et sorties. Dans la théorie des automates finis un automate ayant un ensemble d'états S et un alphabet L est complet si pour tout $a \in L$ et $s \in S$ il existe une transition issue de s et étiquetée par a [HU79]. Cette définition de la complétude est cohérente avec celle utilisée pour les machines de Mealy.

5.6 Dans la pratique du test[JER04]

5.6.1 Modèle de spécification

Un système réactif est généralement décrit dans un langage de description spécialisé (par exemple SDL, LOTOS, UML). La sémantique opérationnelle de ces langages définit les comportements possibles de la spécification. Dans bien des cas cette sémantique peut être décrite par un LTS où les états représentent les configurations possibles (états de contrôle, valeur des variables, contenu des files, etc.) et les actions sont les transitions atomiques. Les simulateurs de tels langages implémentent cette sémantique. Par exemple un simulateur interactif calcule l'état initial et propose l'ensemble des transitions tirables. Le choix d'une transition résulte en un nouvel état, etc. Les transitions atomiques peuvent être complexes et contenir plusieurs actions (par exemple une entrée suivie de sorties et d'actions internes). L'interprétation de la sémantique LTS en IOLTS nécessite parfois un découpage des transitions et d'identifier les entrées, sorties et actions internes. Pour un système réactif, ceci est généralement aisé. Par exemple, Lotos ne différencie pas les entrées et les sorties, mais parle de synchronisation sur des portes.

Alors que la modélisation en Lotos d'un système réactif fait en général apparaître clairement les entrées et sorties du système par rapport à son environnement.

Les comportements de la spécification sont modélisés par un IOLTS (S), il est supposé aussi non-divergent, i.e. il n'a pas de séquence infinie d'actions internes passant par une infinité d'états distincts.

5.6.2 Modèle d'implémentation

En réalité l'Implémentation Sous Test (IUT)est une boîte noire, logicielle ou matérielle. Ce n'est pas un objet formel, mathématique. Cependant, pour pouvoir raisonner formellement sur la conformité d'une implémentation vis à vis d'une spécification, il faut se baser sur des modèles formels, seule possibilité pour définir une relation mathématique entre implémentation et spécification.

On fait donc l'hypothèse de test suivante. Bien qu'ils soient inconnus, les comportements possibles de l'implémentation sont supposés modélisables par un IOLTS IUT, il est toujours supposé que les alphabets de l'IUT sont compatibles avec ceux de S. On supposera de plus que l'IUT est complète en entrée (faiblement) sur son alphabet d'entrée.

L'hypothèse de compatibilité des alphabets reflète le fait que l'alphabet de l'IUT soit inconnu, mais que ses entrées contiennent celles de la spécification, en réalité les seules avec lesquelles on le stimule dans le cadre de la conformité.

5.6.3 Comportements visibles

Dans la pratique, les tests observent les réactions d'un système à des stimuli du testeur, donc les traces d'un système (i.e. les séquences d'actions visibles) et les comparent aux comportements attendus, décrits par les traces de la spécification. Il est donc nécessaire d'identifier l'ensemble des traces de la spécification ou tout au moins les traces prévisibles résultant des stimuli du testeur.

L'ensemble des traces d'un IOLTS (ou d'un LTS) S peuvent être déterminés à partir d'un IOLTS (LTS) **déterministe**.

5.6.4 Les Blocages

Dans la modèlisation par des IOLTS (LTS), plusieurs types de blocages peuvent apparaître, définies dans la littérature comme suit :

(i) le blocage de sortie (*outputslock*) : le système attend d'être stimulé pour produire une action observable (une sortie).

(ii)Le blocage complet (*deadlock*) : le système ne peut plus évoluer.

(iii)Le blocage vivant (*livelock*) : le système diverge par une suite infinie d'actions internes (boucle d'actions internes) i.e. $\exists \tau_1, \tau_2, \tau_3$....tel que $s \overset{\tau_1,\tau_2,\tau_3.....}{\rightarrow} s$.

Si on note $deadlock(S)$ l'ensemble des états de deadlock, $livelock(S)$ l'ensemble des états livelock et $outputlock(S)$ les états de blocage de sortie on définit :

$Quiescent(S) = deadlock(S) \cup livelock(S) \cup outputlock(S)$.

Un blocage de la spécification n'est pas forcement une erreur. C'est évident pour les blocages de sortie, mais c'est vrai aussi pour les livelocks (par composition parallèle et masquage des interactions internes par exemple), et même les deadlocks. Par conséquent, un test appliqué à une IUTdoit pouvoir distinguer entre un blocage de l'IUT valide (existant

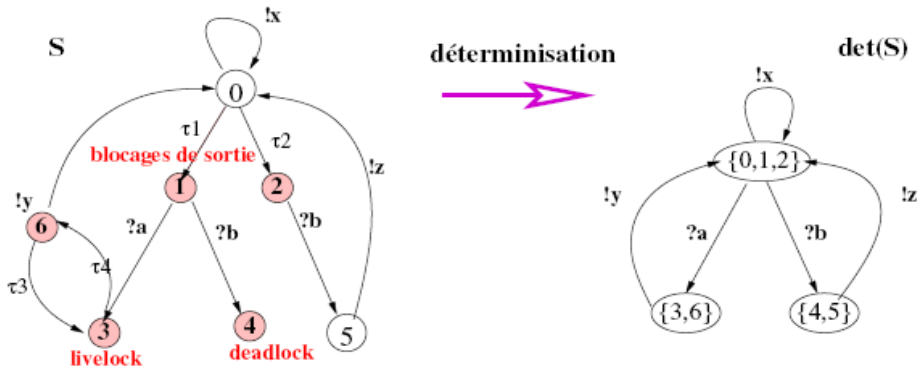

FIG. 5.3 – Perte des blocages par déterminisation

dans la spécification) ou non valide. Il faut donc détecter les blocages possibles dans les comportements de la spécification.

Dans la pratique du test, en plus des traces observables, les tests doivent permettre la détection des blocages (quiescence). Pour cela il est utilisé des temporisateurs (timers) qui sont armés en attente d'une réponse de l'IUT. On fait alors l'hypothèse que si la valeur du temporisateur est suffisamment grande, l'expiration du temporisateur est assimilée à un blocage de l'IUT.

Il est important de constater que la déterminisation ne préserve pas nécessairement les blocages, comme l'illustre dans la Figure5.3 :

Pour conserver les blocages par déterminisation, il faut les expliciter sur l'IOLTS (LTS) de la spécification, après les avoir détectés. En fait, il suffit de rajouter, dans chaque état de blocage, une boucle étiquetée par une nouvelle action δ . L'action δ est considérée comme une sortie de la spécification , puisqu'elle est observable mais non _contrôlable par l'environnement, elle correspondra à l'expiration d'un temporisateur. L'IOLTS (LTS) résultant de l'ajout des actions δ est appelé automate de suspension. La définition ci-dessous formalise la transformation de l'automate.

Définition 5.7 *(Automate de suspension) Soit un IOLTS (LTS)* $S = (Q, A, \rightarrow_S, q_0)$ *, l'automate de suspension de* S *est un IOLTS (LTS)* $\Delta(S) = (Q, A^\Delta, \rightarrow_{\Delta(S)}, q_0^\Delta)$ *tel que :*

$$\begin{cases} A^\Delta = A \cup \{\delta\} \, (avec \ \delta \ une \ sortie \ pour \ le IOLTS) \ et \\ La \ relation \ de \ transition \ \rightarrow_{\Delta(S)} \ d\acute{e}finiepar \ \rightarrow_{\Delta(S)} = \rightarrow_S \cup q \xrightarrow{\delta} q \ / q \in Quiescent(S). \end{cases}$$

La déterminisation à posteriori de l'automate de suspension (i.e. après ajout des δ), illustrée par la transformation définie précédemment (Définition5.7), maintient alors les comportements visibles en conservant toutes les informations de blocages Figure5.4.

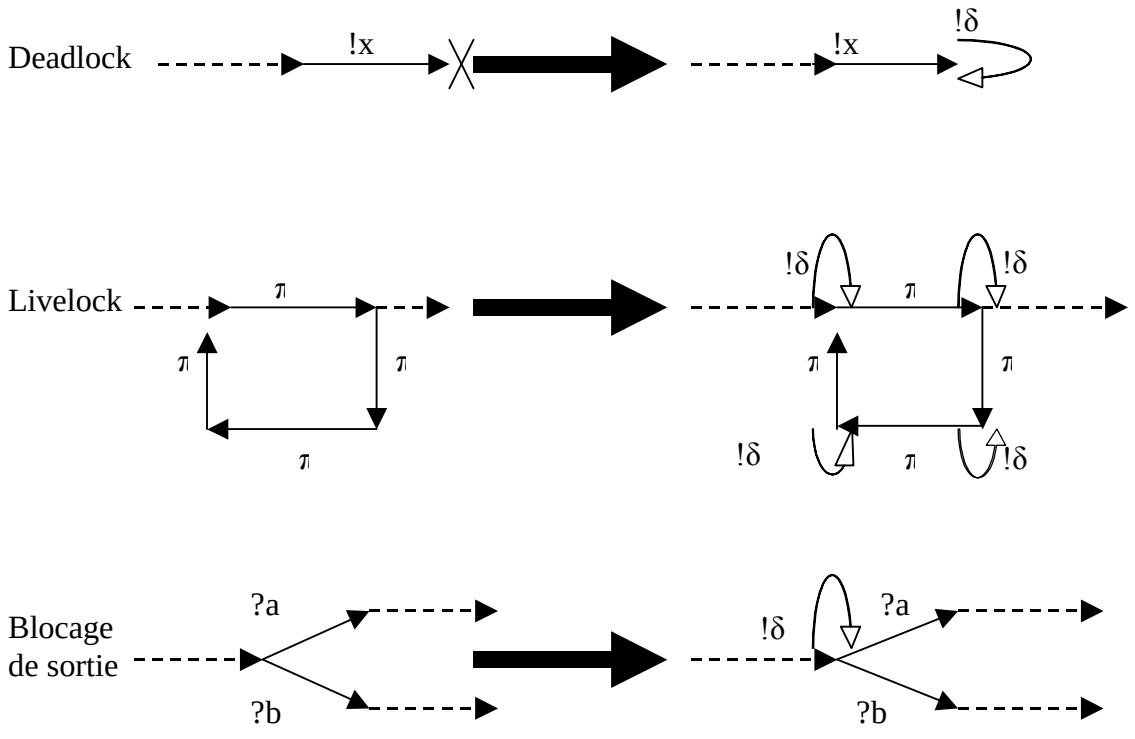

FIG. 5.4 – Construction de l'automate de suspension

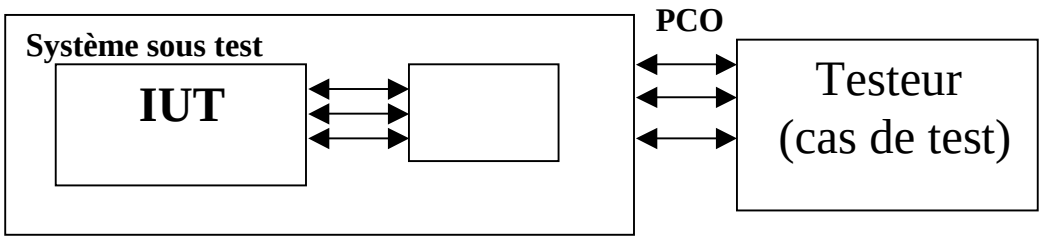

FIG. 5.5 – Test dans un contexte

Ces comportements visibles, ou traces de l'automate suspendues, sont appelées les traces suspendues (suspension traces) par Tretmans[Tre96b]. Tretmans dans [Tre96b] définit directement l'automate de suspension comme l'automate déterminisé résultant.

Il est évident que les deux opérations sont indépendante mais liées, dans la littérature du test les travaux sur les modèles séparent les deux étapes, à noter la déterminisation et la suspension par ajout des δ.

5.6.5 La relation de conformité

Pour les IOLTS, plusieurs relations sont envisageables suivant les observations permises par le test (i.e. traces, blocages, transitions tirables, refus, etc.) et la notion de correction voulue. différentes relations ont été définies dans ce même Chapitre.

5.6.6 Algorithme de génération de test avec objectifs de test ayant des actions internes

Dans le domaine du test il à été montré que seul l'aspect observable est traité dans l'IUT et les objectifs de test ; Alors que les objectifs de test avec actions interne permettent plus de précision dans la sélection des tests. Ceux ci reconnaissent, non pas des traces suspendues, mais des séquences d'actions de la spécification, y compris les actions internes. La définition formelle des objectifs s'en trouve modifiée, mais aussi quelques aspects des algorithmes[JER04].

Justification : Rappelons que le rôle d'un objectif de test est de sélectionner des comportements visibles de la spécification dans un cas de test qui observera les traces suspendues de l'IUT. Le cas de test n'aura donc que des actions visibles définies à partir de celles de la spécification. Dans[JER04] il est souligné que a priori tous les moyens sont bons pour sélectionner des comportements du cas de test, afin de focaliser sa production. Il n'y a aucune raison de se limiter à décrire des propriétés sur les traces suspendues de la spécification et on doit pouvoir utiliser toutes les informations contenues dans la spécification, en particulier les actions internes comme dans l'outil TGV, qui c'est avéré utile, particulierement dans le cas du test dans un contexte Figure5.5.

Dans cette architecture de test, le système sous test SUT (system under test) est composé de plusieurs processus. L'implémentation sous test IUT est un composant logiciel qui

n'est accessible qu'à travers un contexte (i.e. un autre composant logiciel, un médium de communication). On peut supposer que le contexte est implémenté correctement. Bien qu'on souhaiterait tester l'IUT par rapport à sa spécification, les interfaces disponibles pour le test se trouvent entre le contexte et l'environnement. Les comportements visibles du système sont donc à cette interface et ne permettent de tester que le système sous test SUT, c'est à dire l'IUT dans son contexte, par rapport à la spécification de ce composant dans son contexte (ou dans la spécification de celui-ci) Si les objectifs permettent de décrire seulements des comportements visibles, il faut pouvoir les imaginer à cette interface, ce qui implique de considérer le comportement du système composé. Mais comme on souhaite cibler le test sur les comportements de l'IUT, l'usage des actions internes permet de décrire des objectifs sur l'IUT, sans faire référence à son contexte.

5.7 Conclusion

Dans ce Chapitre une présentation de la pratique du test formel, ou toutes les phases du test reposent les modèles de systèmes de transitions. Il à été question de modèle déterministe et de déterminisation. Les outils de test ainsi que les algorithmes de base repose sur l'hypothèse de modèle déterministe.

Dans des travaux plus récent une approche qui révèle la considération des actions internes particulièrement dans les objectifs de test se présente comme une nouveauté dans ce domaine Ceci est justifié par la complexité croissante des systèmes à modélisé.

Dans le cadre de notre travail, le modèle DATA est considéré. on vas prouver qu'il est déterminisable et que la manipulation des actions internes est particulière du fait qu'elles sont considérer comme toute autre action par rapport à leur durée même dans le cas de leur élimination par déterminisation, Distance temporelle qu'elle induit est ainsi gardée et respectée.

Se qui vas permettre une plus grande précision lors de sont utilisation pour le test que se soit dans sont modèle original ou celui qui vas être proposé comme alternative dans le test par le refus détaillé dans le Chapitre7.

Chapitre 6

Déterminisation des automates temporisés avec durées d'actions

A ce stade de l'étude on rappelle le formalisme des DATAs et on propose une méthode de déterminisation ainsi que l'élimination des actions internes, dans la perspective de son utilisation dans le domaine de génération automatique du test de conformité.

6.1 Formalisation du modèle des DATAs

Définition 6.1 *Soit $\mathcal{H}$, parcouru par x, y... un ensemble d'horloges de valeurs non negatives dans (un domaine temporel $\mathbb{T}$, tel que $\mathbb{Q}^+$ ou $\mathbb{R}^+$). l'ensemble $\Phi_t(\mathcal{H})$ de contraintes temporelles γ sur $\mathcal{H}$ est défini par la syntaxe $\gamma ::= x \sim t$, où x est une horloge dans $\mathcal{H}$, $\sim \in \{=, <, >, \leq, \geq\}$ et $t \in \mathbb{T}$. F_x est utilisé pour représenter une contrainte de la forme $x \sim t$. soit l'ensemble $\{F_{x_1}, F_{x_2}, \ldots, F_{x_n}\}$ de contraintes temporelles , $\wedge\{F_{x_1}, F_{x_2}, \ldots, F_{x_n}\}$évoque une contrainte de la forme $\bigwedge_{i=1,n} F_{x_i}$.*

Une valuation ou interprétation v sur $\mathcal{H}$ est une fonction qui associe à toute horloge $x \in \mathcal{H}$ une valeur dans $\mathbb{T}$. une valuation v sur $\mathcal{H}$ satisfait une contrainte temporelle γ sur $\mathcal{H}$ ssi γ est vérifiée lors de la valuation des horloges par v. pour $\mathcal{I} \subseteq \mathcal{H}$, $[\mathcal{I} \mapsto 0]\, v$ désigne la valuation sur $\mathcal{H}$ qui affecte la valeur 0 à toute horloge $x \in \mathcal{I}$, est vérifie toutes les contraintes sur les autres horloges de $\mathcal{H}$. l'ensemble de toutes le valuations de $\mathcal{H}$ est noté $\Xi(\mathcal{H})$. la relation de satisfaction $\models$ sur les contraintes temporelles est définie sur l'ensemble des valuations sur $\mathcal{H}$ par : $v \models x \sim t \Longleftrightarrow v(x) \sim t$ tel que $v \in \Xi(\mathcal{H})$.

On note 2^T_{fn} l'ensemble fini des sous ensembles de T.

Définition 6.2 *Un DATA -Durational Action Timed Automaton- $\mathcal{A}$ est un tuple $(S, L_S, s_0, \mathcal{H}, T)$ tel que :*

- S est un ensemble fini d'états,

• $L_S : S \rightarrow 2^{\Phi_t(\mathcal{H})}_{fn}$ est une fonction qui correspond à chaque état s l'ensemble F des conditions de durées des actions potentiellement en exécution dans l'état s .

• $s_0 \in S$ est l'état initial,

• $\mathcal{H}$ est un ensemble fini d'horloges.

• $T \subseteq S \times \wedge 2^{\Phi_t(\mathcal{H})}_{fn} \times Act \times \mathcal{H} \times S$ est l'ensemble des transitions. Une transition $(s, \gamma, a, x, s\prime)$représente le passage de l'état s à l'état $s\prime$, en lançant l'exécution d'une action a et en initialisant x à zero. γ est la contrainte temporelle qui conditionne le passage de la transition. (s, γ, a, x, s') est aussi notée $s \xrightarrow{\gamma,a,x} s\prime$. l'ensemble de transition T parcouru par τ_0, τ_1, ...étant donnée une transition $\tau_i = (s, \gamma, a, x, s')$, $\alpha(\tau_i)$ (resp. $\beta(\tau_i)$) représente l'état initial de τ_i (resp. l'état final de τ_i). (i.e. $\alpha(\tau_i) = s$ et $\beta(\tau_i) = s'$). L'étiquette de la transition τ_i est $\lambda(\tau_i) = a$. L'horloge associée à la transition τ_i est donnée par la fonction $\xi(\tau_i) = x$.

Définition 6.3 *La sémantique d'un DATA $\mathcal{A} = (S, L_S, s_0, \mathcal{H}, T)$ est définie par un système de transition infini $\mathcal{S}_\mathcal{A}$ sur $Act \cup \mathbb{T}$, Un état de $\mathcal{S}_\mathcal{A}$ (ou* configuration*) est un couple $\langle s, v\rangle$ tel que s est un état de $\mathcal{A}$ et v une valuation sur $\mathcal{H}$. une configuration $\langle s_0, v_0\rangle$ est dite initiale ssi s_0 est l'état initial de $\mathcal{A}$ et $\forall x \in \mathcal{H}, v_0(x) = 0$. Deux types de transitions entre deux configurations de $\mathcal{S}_\mathcal{A}$ sont possibles, est qui correspondent respectivement au passage du temps la règle (*RA *rule) , une transition de $\mathcal{A}$ la régle (*RD *rule).*

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

Dans [BS05], il a été défini pour le langage Basic Lotos une sémantique opérationnelle étendue par la fonction qui associe à toute action sa durée ; se qui permet de correspondre à toutes expressions de comportement Basic Lotos une structure dans le modèle des DATAs.

Définition 6.4 *Soit $a \in Act$, $x \in \mathcal{H}$, $\Gamma(a)$ la fonction qui associe une durée à l'action a; une condition de durées est définie comme suit : $DC(a) = \{x \geq \Gamma(a)\}$. l'ensemble de toutes les conditions de durées sera noté $[DC]$.*

Définition 6.5 *Une condition d'exécution γ d'une transition $(s, \gamma, a, x, s\prime)$ est définie par , $\wedge_{b_i}(DC(b_i))$, la conjonction des conditions de durées des actions b_i tel que la terminaison de ces actions conditionne la transition. L'ensemble de toutes les conditions d'execution sera noté $[EC]$.*

Définition 6.6 *Soit D_f un fonction définie de $[EC]$ vers $[DC]$ qui associe à toute condition d'exécution l'ensemble des conditions de durées qui la composent. en d'autres termes : $D_f(\wedge\{(x_1 \geq c_1), \ldots, (x_i \geq c_i)\}) = \{(x_1 \geq c_1), \ldots, (x_i \geq c_i)\}$.*

Définition 6.7 *Si $\mathcal{B}$ est un ensemble , $\mathcal{B}^*$ (resp. $\mathcal{B}^\omega$) représente l'ensemble des séquences d'éléments finie (resp. infinie) de $\mathcal{B}$. L'ensemble des séquences finie et infinie de $\mathcal{B}$ est noté $\mathcal{B}^\infty$ (i.e. $\mathcal{B}^\infty = \mathcal{B}^* \cup \mathcal{B}^\omega$). Une séquence temporelle sur $\mathbb{T}$ est une séquence croissante $(t_i)_{i\geq 1}$ $\in \mathbb{T}^\infty$.*

- *Une exécution r sur un DATA est une succession de transitions valuées définie comme suit :*

$$r = (s_0,v_0) \xrightarrow[t_0]{\gamma_0, a_0, x_0} (s_1, v_1) \xrightarrow[t_1]{\gamma_1, a_1, x_1} ... \xrightarrow[t_n]{\gamma_n, a_n, x_n} (s_n, v_n) ...$$

où $(v_i)_{i\geq 0}$ est une succession de valuations d'horloges tel que pour tout $x \in \mathcal{H}$, $v_0(x) = 0$ et pour tout $i \geq 0$ $\begin{cases} v_{i+1} = [x_{i+1} \mapsto 0](v_i + t_{i+1} - t_i) \\ v_i + t_{i+1} - t_i \models \gamma_{i+1} \end{cases}$.

- *On suppose que $t_0 = 0$, la date de début de l'exécution du système. Une trace temporisée Ttrace associée à une exécution r est une séquence $w = (a_i, t_i)_{i\geq 1}$ dans $(Act \times \mathbb{T})^*$ tel que $(a_i)_{i\geq 1} \in Act^*$ et $(t_i)_{i\geq 1} \in \mathbb{T}^*$ représente l'instant à partir de laquelle la condition d'exécution est vérifiée, et $a_i \neq \tau$ (action interne non observable). Une trace temporisée capture l'aspect observable d'une exécution.*

- *Une paire de deux transitions τ_1 et τ_2 d'un DATA sont dites contiguës ssi : $\beta(\tau_1) = \alpha(\tau_2)$*

Définition 6.8 *Un DATA (S, L_S, s_0, H, T) est dit déterministe ssi il satisfait les deux conditions suivantes :*

1. *Le DATA ne comporte pas de transitions internes,*
2. *Pour tout $s \in S$ et pour toutes paires de transitions distinctes issues de s et étiquetées par la même action a , i.e., $t_1 = (s, \gamma_1, a, x_1, s_1)$ $t_2 = (s, \gamma_2, a, x_2, s_2)$ la conjonction des deux conditions d'exécutions $\gamma_1 \wedge \gamma_2$ n'est pas satisfaite.*

Note : Dans le modèle des DATAs les actions internes qui sont issues soit de la composition parallèle des systèmes ou par abstraction des actions, ont la même considération vis-à-vis du temps et des durées d'actions. L'opération de détermination que nous proposons est, plus délicate car les durées des actions internes doivent être capturées est gardées après élimination de celles ci.

La deuxième problématique à laquelle on s'attaque lors de la déterminisation d'une structure dans le modèle des DATAs manipulant les actions internes est, le traitement des cycles ou circuits de $\tau - transitions$.

Dans tous les cas les circuits $\tau - transitions$ causent les divergences dans le système (dites aussi blocages de type « livelock ») par conséquence, la majorité des modèles pour le test imposent une hypothèse de forte convergence c'est à dire le système ne présente pas de $\tau-$cycle, hors cette restriction semble être trop sévère[CC07].

Le processus de déterminisation que nous proposons traite les séquences de $\tau-transitions$ ainsi que les cycles tout en gardant l'information temporelle qui été véhiculée par ceux ci. La dernière étape du processus de déterminisation consiste en le traitement de l'indéterminisme causé par le choix des actions observables.

6.2 L'opération de déterminisation d'un DATA

L'opération de déterminisation d'un DATA $\mathcal{A}$ signifie le calcul d'un DATA déterministe ayant les mêmes traces temporisées que $\mathcal{A}$ en deux étapes :

1. L'élimination des actions internes ($\tau - actions$) et
2. L'élimination du non déterminisme causé par les actions observables.

6.2.1 Conditions générales sur la pratique du Modèle

1. Les actions internes sont urgentes : Cette condition est importante car l'idée sur laquelle repose l'élimination des actions internes est le cumule des durées de ces actions ensuite l'attribution de cette mesure temporelle à la transition dont le franchissement on est dépendant. Pour que ce calcul soit valide dans le temps et que les contraintes temporelles du DATA résultant soient cohérentes, il faut qu'il n'y ai pas de délais entre le moment où l'action interne est disponible et celui de son exécution (ou lancement). Autrement dit l'action interne dés quelle est disponible elle est exécutée.

   Cette condition reste réaliste car l'action interne se présente, sans dépendance de facteurs extérieurs qui peuvent la retardée, en principe si c'est une action qui matérialise une synchronisation du système elle est sans durée sinon, c'est une abstraction d'actions observables ce qui signifie que l'action est d'un niveau plus faible et qu'elle est encapsulée dans une action de plus haut niveau et dont l'exécution est sans retard ; Formellement cette règle se presente comme suit : la règle suivante est appliquée (RU rule) :

$$(\mathrm{RU})\ \frac{\forall\,(s,\gamma,\tau,x,s') \in T \qquad \forall d, d' \in \mathbb{T}\ 0 \leq d' \leq d\ \ v + d' \not\models \gamma}{\langle s, v\rangle \xrightarrow{d'} \langle s, v + d'\rangle}$$

2. Toute spécification de système doit être gardée : i.e. à partir de l'état initial du système les transitions possibles sont toutes observables. Formellement :

   $\forall \tau_i \in T$ tel que $\alpha\,(\tau_i) = s_0$ alors $\lambda\,(\tau_i) \neq \tau$

Dans notre cas le non respect de cette condition signifie le cas de figure suivant :

Dans l'exemple Figure6.1, avant l'exécution de l'action a , une durée minimale à attendre est de 1 unités de temps la durée de τ or le but, lors de la déterminisation après élimination des actions internes est de garder l'intégrité des contraintes et la sémantique des conditions d'exécutions. Dans le cas présent l'élimination de τ causera un problème quant à la condition d'exécution des actions a et b qui, d'un point de vue sémantique, le système est bloqué durant l'exécution de τ alors que ce blocage n'est pas perçut entant que tel par l'observateur (le testeur par exemple) qui, selon la spécification déterministe, attend l'observation d'une action (soit a, soit b). Aussi d'un point de vue structurel les conditions d'exécution de a et b dépendront d'un nom d'horloge désormais non existant.

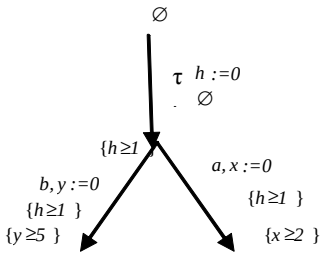

FIG. 6.1 – un DATA non gardé

6.2.2 L'élimination des actions internes

L'idée principale pour l'élimination des actions internes, consiste en le calcul de l'effet temporel de la séquence de transitions internes qui se termine par une action observable ensuite d'affecter cet effet à la condition d'exécution de celle ci.

Les opérations et les substitutions suivantes sur les contraintes temporelles sont définies :

Définition 6.9 *Soit* $\gamma_1 = \wedge\{F_{x_1}, \ldots, F_{x_n}\}$ *et* $\gamma_2 = \wedge\{F_{y_1}, \ldots, F_{y_m}\}$ *deux conditions d'exécution ,* $F = \{F_{x_1}, \ldots, F_{x_n}\}$ *et* $F' = \{F_{y_1}, \ldots, F_{y_m}\}$ *deux ensembles de contraintes temporelles,* F_{x_i} *une contrainte temporelle et* d *un élément de* $\mathbb{T}$. $Dom(F)$ *est l'ensemble des noms d'horloges de* F $F = \{F_{x_1}, \ldots, F_{x_n}\}$ *alors* $Dom(F) = \{x_1, x_2, \ldots, x_n\}$; *les opérations suivantes sont définies :*

- $(\wedge\{F_{x_1}, \ldots, F_{x_n}\}) \wedge (\wedge\{F_{y_1}, \ldots, F_{y_m}\}) = \wedge\{F_{x_1}, F_{x_2}, \ldots, F_{x_n}, F_{y_1}, F_{y_2}, \ldots, F_{y_m}\}$. *La conjonction de conditions d'exécutions.*
- $(\wedge\{F_{x_1}, \ldots, F_{x_n}\}) - F_{x_i} = \wedge(\{F_{x_1}, \ldots, F_{x_n}\} - F_{x_i})$. *L'élimination d'une contrainte à partir d'une condition d'exécution*
- $(\wedge\{F_{x_1}, \ldots, F_{x_n}\}) + d = \wedge\{F_{x_1} + d, \ldots, F_{x_n} + d\}$ *où* $(x \sim t) + d = x \sim (t + d)$ *pour toute contrainte temporelle* $x \sim t$.

Forme normale d'un ensemble de contraintes temporelles

Un ensemble de contraintes est dit Normal s'il ne contient pas deux èlèments portant sur la même horloge ; Formellement :

$\forall x_i, x_j \in Dom(F)$, si $F_{x_i}, F_{x_j} \in F$ alors $x_i \neq x_j$.

Normalisation d'un ensemble de contraintes temporelles

Intuitivement deux contraintes temporelles sur le même nom d'horloge concerne la même action en cours d'exécution ; si les deux contraintes sont dans le même ensemble de conditions de durées d'un état ou dans la même condition d'exécution d'une transition cela signifie

qu'elles seront vérifiées à la fois ou l'une après l'autre (si leur valeurs sont différentes) leur conjonction ne sera vérifié que si la contrainte ayant la valeur maximale est vérifiée.

L'opération de normalisation sur un ensemble de contraintes temporelles permettra de minimaliser cet ensemble en ne gardant que la contrainte maximale.

Définition 6.10 *Soit F un ensemble de contraintes temporelles $Norm(F)$ est défini comme suit :*

$$Norm(F) = Norm(\{F_{x_1}, \ldots, F_{x_n}\}) = \left\{ \begin{array}{ll} \forall x_i, x_j \in Dom(F), & \\ F - F_{x_i} - F_{xj} \cup Max(F_{x_i}, F_{x_j}) si & x_i = x_j \\ F & si \quad x_i \neq x_j \end{array} \right\}$$

où Max est une fonction qui rend la contrainte maximale de deux contraintes sur la même horloge.

Définition 6.11 *La substitution de contraintes temporelles est définie comme suit :*

$$\gamma_2[\gamma_1/x] = \wedge \left(Norm \left(\bigcup_{i=1}^{m} F_{y_i}[\{F_{x_1}, \ldots, F_{x_n}\}/x] \right) \right) \text{ tel que :}$$

$$\left\{ \begin{array}{ll} F_{y_i}[\{F_{x_1}, \ldots, F_{x_n}\}/x] = F_{y_i} & si \;\; y_i \neq x \\ F_{y_i}[\{F_{x_1}, \ldots, F_{x_n}\}/x] = \{F_{x_1}, \ldots, F_{x_n}\} & si \; y_i = x \end{array} \right\}$$

Définition 6.12 *L'augmentation des ensembles des conditions de durées d'un état est définie comme suit : $Aug(F, F') = Aug(\{F_{x_1}, \ldots, F_{x_n}\}, \{F_{y_1}, \ldots, F_{y_m}\}) = Norm(F \cup F')$.*

6.2.3 Fonction de translation en avant (Forward Translation)

Intuitivement la fonction de translation en avant traite toutes les paires de transitions contiguës tel que la première soit une τ-transition en trois étapes : (1) Le traitement de l'état de sortie de la τ-transition (2) Le traitement de la condition d'exécution de la transition suivantes si elle dépend de la terminaison de τ, et cela en translatant la durée de l'action interne, ainsi que sa condition d'exécution. (3) L'élimination effective de la τ-transition. Ce schéma est détaillé comme suit :

1. Traitement de l'état de sortie de la $\tau - transition$: Cela est réalisé lors de la substitution de la condition de durée de l'action τ dans l'état destination par la contrainte temporelle qui conditionne la $\tau - transition$. formellement il y'a substitution de la durée de l'action τ par les contraintes temporelles formant sa condition d'exécution.

2. Toutes les transitions qui se suivent directement la $\tau - transition$(formant avec elle une paire de transitions contiguës) : Le traitement consiste en le remplacement de la condition de durée de l'action τ dans la condition d'exécution de la transition suivante, si elle existe, par les contraintes temporelles conditionnant l'exécution de τ augmentées de la durée de τ ainsi le traitement est terminé.

Sinon le traitement s'effectue sur la branche débutant par la deuxième transition de la paire avec tous ses successeurs en respectant le même scénario. Le processus s'arrête, soit par la détection d'une dépendance soit par la détection d'une transition sans successeur (atteignant un état final).

3. Enfin , élimination de la $\tau - transition$.

4. Refaire l'opération jusqu'à élimination de toutes les $\tau - transitions$ du DATA.

Définition 6.13 *Soit η une famille de relations définie comme suit : $\eta = \{\eta_0, \eta_1, \eta_2, ...\eta_n\}$ tel que $\eta_k = \{(\tau_k, \tau_j) \in T \times T / k \neq j$ et τ_k, τ_j deux transitions contiguës$\}$, si à chaque transition correspond une relation de contiguïté , au maximum le nombre de relation est celui des transitions de T plus la relation $\eta_0, |\eta| < |T| + 1$*

$\eta_0 \subset T \times \emptyset$ est la relation qui regroupe l'ensemble des transitions n'ayant pas de transitions contiguës : $\eta_0 = \{(\tau_i, \varnothing) \in T \times \emptyset / \tau_i$ une transition terminale$\}$, τ_i est dite terminale ssi il n'existe pas de transition $\tau_j \neq \tau_i$ tel que $\beta(\tau_i) = \alpha(\tau_j)$.

Intuitivement cet ensemble regroupe toutes les transitions terminales. Lors du processus de déterminisation il sera traité de façon particulière.

Il est à noter que les cycles de $\tau - transitions$ ne sont pas autorisés. La divergence sera étudiée dans la seconde partie de ce chapitre.

Définition 6.14 *Soit $\eta_i = \{(\tau_i, \tau_j) \in T \times T, \quad j = 1..n\}$; $F = \{x \geq d\}$ la condition de durée de l'action $\lambda(\tau_i)$, γ la condition d'exécution de τ_i ; on définit la fonction ϕ sur η_i dite translation en avant comme suit :*

$$\phi(\eta_i, \gamma, F) = \begin{cases} L'_S = Ls[\beta(\tau_i) \mapsto Aug((Ls(\beta(\tau_i)) - F), D_f(\gamma + d))] & (1) \\ \forall \tau_j \in T \text{ tel que } (\tau_i, \tau_j) \in \eta_i : \begin{cases} \gamma_j[(\gamma + d)/x] & (2) \\ \phi(\eta_j, \gamma, F) & (3) \end{cases} \end{cases}$$

on définit la fonction ϕ_0 sur η_0 comme suit :

$$\phi_0(\eta_0) = \begin{cases} \forall \tau_i \in T \text{ tel que } (\tau_i, \varnothing) \in \eta_0 , \\ L'_S = Ls[\beta(\tau_i) \mapsto Aug((Ls(\beta(\tau_i)) - F), D_f(\gamma + d))] \end{cases}$$

Intuitivement cette fonction transfert la condition d'exécution γ d'une transition dans son état destination par la première substitution (phase 1). Et pour toute transition contiguë à celle ci, effectue la substitution d'une contrainte temporelle relative à un nom d'horloge x (représentant une condition de durée) par une condition d'exécution γ (phase 2). Et enfin l'itération du même traitement récursivement pour toute les branches du DATA initiées par η_i (phase 3).

Considérons η^τ la famille des relations suivantes :

$\eta^\tau_k = \{(\tau_k, \tau_j) \in T \times T / j = 1..n$ et $\lambda(\tau_k) = \tau$ une action interne$\}$ représentant l'ensemble des relations de contiguïté sur les transitions internes.

Dans la suite du processus de déterminisation , nous nous intéressons à l'ensemble η^τ dans l'élimination des $\tau - transitions$.

Définition 6.15 *Soit* $\mathcal{A} = (S, L_S, s_0, \mathcal{H}, T)$ *un DATA ,* $\eta_i^\tau = \{(\tau_i, \tau_j) \in T \times T, \quad j = 1..n\}$ *une relation de contiguïté tel que :* $\tau_i = (s_i, \gamma_i, \tau, x_i, s_{i+1})$ *et* $F_i = \{x_i \geq d_i\}$ *la condition de durée de l'action* τ *: on définit* $Telim_1(\eta_i^\tau, \mathcal{A})$*, noté aussi* $Telim_1^{\eta_i^\tau}(\mathcal{A})$ *un DATA* $\mathcal{A}' = (S', L'_S, s_0, \mathcal{H}, T')$ *résultant du traitement suivant sur* $\mathcal{A}$ *:*

- $\phi(\eta_i^\tau, \gamma_i, F_i)$
- $\forall \tau_k \in T$ tel que $\alpha(\tau_k) = s_i$ ou $\beta(\tau_k) = s_{i+1}$ remplacement de s_i par s_{i+1}
- $T' = (T - \{\tau_i\})$
- $S' = S - s_i \quad$ si $\quad s_i \neq s_{i+1}$ .....(1)

Définition 6.16 *Soit* $\eta_0^\tau = \{(\tau_i, \varnothing) \in T \times \emptyset / \tau_i$ *est une* $\tau -$ *transition terminle*$\}$

$\tau_i = (s_i, \gamma_i, \tau, x_i, s_{i+1})$ *on définit* $Telim_0\left(\eta_0^\tau, \mathcal{A}\right)$ *noté aussi* $Telim_0^{\eta_0^\tau}(\mathcal{A})$ *un DATA* $\mathcal{A}' = (S', L'_S, s_0, \mathcal{H}, T')$

résultant du traitement suivant sur $\mathcal{A}$ *:*

- $\phi_0\left(\eta_0^\tau\right)$
- $T' = (T - \{\tau_i\})$
- $S' = S - s_i \quad$ si $\quad s_i \neq s_{i+1}$.....(2)

Note : Les conditions (1) et (2) assurent l'intégrité de l'opération d'élimination de l'état source de la $\tau -$ *transition* dans le cas où celle-ci forme un cycle sur le même état. Il est à noter aussi que cette situation peut être originaire du système lui-même ou héritée du processus de déterminisation.

Définition 6.17 *Soit un DATA* $\mathcal{A} = (S, L_S, s_0, \mathcal{H}, T)$ *et* $\eta^\tau = \{\eta_i^\tau\}_{0<i\leq n}$ *l'ensemble des relations de contiguïté définies sur les* $\tau -$ *transitions de* $\mathcal{A}$*, on définit*

$$\Phi_1(\eta^\tau, \mathcal{A}) = \begin{Bmatrix} \mathcal{A} & if & \eta^\tau = \emptyset \\ Telim_1^{\eta_n^\tau} \circ \cdots \circ Telim_1^{\eta_1^\tau}(\mathcal{A}) & & \end{Bmatrix}$$

Définition 6.18 *Soit un DATA* $\mathcal{A} = (S, L_S, s_0, \mathcal{H}, T)$ *et* $\eta_0^\tau \in \eta$ *la relation de contiguïté définie sur les* $\tau -$ *transitions terminales de* $\mathcal{A}$*, on définit*

$$\Phi_2(\eta_0^\tau, \mathcal{A}) = \begin{Bmatrix} \mathcal{A} & if & \eta_0^\tau = \emptyset \\ & Telim_0^{\eta_0^\tau}(\mathcal{A}) & \end{Bmatrix}$$

Algorithme d'élimination des actions internes Soit $\mathcal{A} = (S, L_S, s_0, \mathcal{H}, T)$ un DATA

Tr :=ensemble des relations de contiguïté sur les $\tau -$ *transitions* de $\mathcal{A}$

Repeat

Comput $\Phi_1(Tr, \mathcal{A})$ /* La fonction Φ_1 modifie le DATA $\mathcal{A}$*/

Tr := ensemble des relations de contiguïté sur les $\tau -$*transitions* du DATA resultant.

until $Tr = \varnothing$

Tr := la relation de contiguïté sur les $\tau -$ *transitions* terminales de $\mathcal{A}$

If $\quad Tr \neq \varnothing$

Then

Compute $\Phi_2(Tr, \mathcal{A})$ /*La fonction Φ_2 modifie le DATA $\mathcal{A}$*/
End

Lemme 6.1 *Soit $\mathcal{A} = (S, L_S, s_0, \mathcal{H}, T)$ un DATA et $\mathcal{A}' = (S', L_{S'}, s'_0, \mathcal{H}, T')$ un DATA sans actions internes, obtenu par application de l'algorithme d'élimination des action internes alors $\mathcal{A}'$vérifie le premier critère de déterminisme [Définition 6.8].*

Preuve. Par construction à partir de l'algorithme 6.2.3 . ■

Lemme 6.2 *Soit $\mathcal{A} = (S, L_S, s_0, \mathcal{H}, T)$un DATA n'ayant pas de séquence infinie de $\tau -$ transitions,L'algorithme d'élimination des $\tau -$transitions appliqué sur $\mathcal{A}$ se termine en un nombre fini d'étapes. Le DATA résultat a les mêmes traces temporisées que le DATA initial.*

Preuve. Par construction à partir de l'algorithme 6.2.3 . ■

6.2.4 La résolution de l'indéterminisme dû au choix des actions observables

Par définition 6.8 l'indéterminisme sur les actions observables est celui créé dans la situation où à partir d'un même état deux transitions sont étiquetées par la même action tel que les conditions d'exécution des deux transitions soient vérifiées à la fois.

La sémantique d'un DATA suppose que γ_1 et γ_2 sont deux ensembles de conjonction de contraintes temporelles relatives aux actions en cours et dont la validité engage le lancement des transitions. Dans notre exemple, la condition d'exécution vérifiée va lancer l'action a et passer dans un état s_2 ou s_3 de façons déterministe, par la suite les actions possibles sont soit b, soit c, soit les deux à la fois.

Cependant cette situation ne peut se présenter que quant les deux actions a sont elles même en parallèle, i.e. la possibilité de l'autoconcurrence. En respect de la sémantique du modèle aucun traitement ne peut être effectué afin de ne pas altérer la validité de la sémantique des DATAs concernant le vrai parallélisme.

Lemme 6.3 *(Traces du DATA déterministe). Soit $\mathcal{A} = (S, L_S, s_0, \mathcal{H}, T)$ un DATA et $\mathcal{A}_{\text{det}} = (S', L_{S'}, s'_0, \mathcal{H}, T')$ un DATA déterministe associer à $\mathcal{A}$, alors $Ttraces(\mathcal{A}_{\text{det}}) = Ttraces(\mathcal{A})$.*

Preuve. Soit la séquence d'exécution suivante sur un DATA :
$r = (s_i, v_i) \xrightarrow[t_i]{\gamma_i, a, x_i} (s_{i+1}, v_{i+1}) \xrightarrow[t_{i+1}]{\gamma_{i+1}, \tau, x_{i+1}} (s_{i+2}, v_{i+2}) \xrightarrow[t_{i+2}]{\gamma_{i+2}, b, x_{i+2}}$ La trace temporisée qui lui correspond est : $(a, t_i), (\tau, t_{i+1}), (b, t_{i+2})$. Cette trace signifie qu'à l'instant t_i la valuation v_i vérifie la contrainte γ_i, l'action a, peut être exécutée et x_i est initialisée à zéro. La valuation v_{i+1} concernant la $\tau-$*transition* est la suivante : $\begin{cases} v_{i+1} = [x_{i+1} \mapsto 0](v_i + t_{i+1} - t_i) \\ v_i + t_{i+1} - t_i \models \gamma_{i+1} \end{cases}$
l'instant t_{i+1} correspond à l'instant de tir de τ (l'hypothèse de l'urgence de τ). A l'instant t_{i+2} la valuation v_{i+2} vérifie la contrainte γ_{i+2},l'action b, peut être exécuter et x_{i+2} est initialisée à zéro ce qui donne : $\begin{cases} v_{i+2} = [x_{i+2} \mapsto 0](v_{i+1} + t_{i+2} - t_{i+1}) \\ v_{i+1} + t_{i+2} - t_{i+1} \models \gamma_{i+2} \end{cases}$

$$or\begin{cases} v_{i+2} = [x_{i+2} \mapsto 0](v_i + t_{i+1} - t_i + t_{i+2} - t_{i+1}) \\ v_i + t_{i+1} - t_i + t_{i+2} - t_{i+1} \models \gamma_{i+2} \end{cases}$$

$$d'où\begin{cases} v_{i+2} = [x_{i+2} \mapsto 0](v_i + t_{i+2} - t_i) \\ v_i + t_{i+2} - t_i \models \gamma_{i+2} \end{cases}$$

Ce qui garantie la validité des instants t_i correspondant aux actions a et b et la trace $(a, t_i), (b, t_{i+2})$ reste valide indépendamment de la $\tau - transition$, l'elimination de la $\tau - transition$ affecte la condition d'exécution de la transition qui lui est successive de la façons suivante : $\gamma_{i+2}\left[(\gamma_{i+1} + (t_{i+2} - t_{i+1}))/x_{i+1}\right]$ dans ce cas $t_{i+2} - t_{i+1}$correspond à la durée de τ or l'opération de substitution des contraintes temporelles basée sur la normalisation remplace une partie de la contrainte γ_{i+2} par $(\gamma_{i+1} + (t_{i+2} - t_{i+1})$ par définition :

$$\left\{ \begin{array}{l} v_{i+2} = (v_{i+1} + t_{i+2} - t_{i+1}) \models \gamma_{i+2} \text{.....(1)} \\ v_{i+1} \models \gamma_{i+1} \Rightarrow (v_{i+1} + t_{i+2} - t_{i+1}) \models \gamma_{i+1} + t_{i+2} - t_{i+1} \text{.....(2)} \end{array} \right\}$$

Par (1) toute partie de γ_{i+2} est vérifiée lors de v_{i+2} et par (2) la partie substituée qui est augmentée par la durée de l'action τ l'est aussi, d'où $v_{i+2} \models \gamma_{i+2}\left[(\gamma_{i+1} + (t_{i+2} - t_{i+1}))/x_{i+1}\right]$. L'ensemble des valuations restent ainsi valides après élimination des $\tau - transitions$. ■

Cas particulier : Un cas particulier qui peut exister est celui où une séquence de $\tau - transitions$ dont la dernière est une $\tau - transition$ terminale, cette séquence est négligeable car sont exécution ne produit aucun effet sur aucune action susceptible d'être observée.

6.3 Concept de divergence

La divergence caractérise un état du système dans le quel aucune évolution n'est perçu par l'utilisateur, la divergence dans les systèmes à base de transitions étiquetées se traduit par la présence de séquence infinie de $\tau - transitions$.

Dans les sections précédentes on a émi l'hypothèse de la non existence de telle situation dans la spécification du système, les blocages possibles sont ceux de sorties ou de terminaison dit aussi cas de deadlocks spécifiés. Cette restriction est considérée sévère car la divergence est causée en générale par la composition des systèmes parallèle ainsi que le masquage d'interactions[KD01], de ce fait il est important de déceler les divergences de la spécification est de les considérer lors du processus de déterminisation du modèle. Formellement :

Soit D un DATA, s_i un état de S ; s_i est dit divergent si $s_i \xrightarrow{\tau^\infty}$. Le DATA diverge ssi un état de S diverge, alors $s \xrightarrow{\tau^\infty}$ signifie qu'une exécution infinie et possible sur une infinité de transitions interne à partir de s sans aucune observation. Dans le cas où le nombre d'états du système est fini la divergence a une autre signification $\exists n \geq 1 : s \xrightarrow{\tau^n} s$, où s est un état par le quel passe un cycle de $\tau - transitions$.

6.3.1 Le traitement de la divergence lors de la déterminisation du Modèle des DATAs

Dans le modèle des DATAs une $\tau - transition$ boucle sur le même état, $\alpha(\tau_k) = \beta(\tau_k)$, deux cas se présentent :

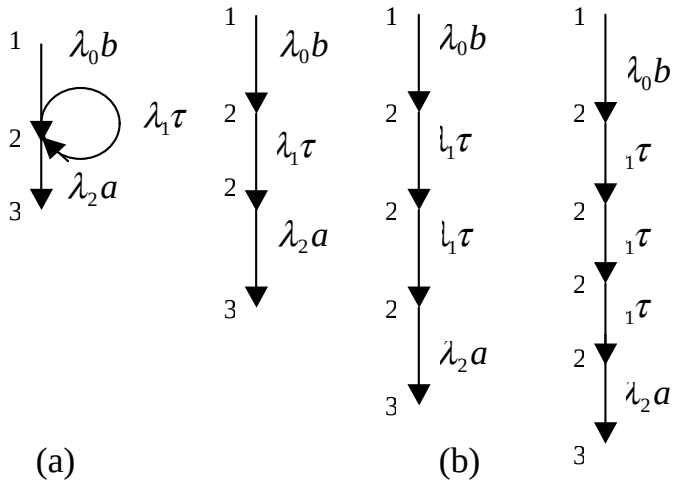

FIG. 6.2 – Système divergent

1. Soit le $\tau - cycle$ est induit par une élimination de $\tau - transitions$, premier cas de figure, quant les deux transitions sont parallèles ; le cas échéant le lancement de la $b - transition$ dépend des deux, alors la condition d'exécution de celle-ci synthétisée par l'algorithme précèdent est complète et valide. (La $b - transition$ doit attendre la durée maximale des deux) .

2. Le deuxième cas de figure, est celui où le $\tau - cycle$ est inhérent au système, la procédure déjà élaborée pour l'élimination des $\tau - transitions$ le traite de la façons suivante :

- Il y a cumule de la durée de la séquence $\tau - transitions$ dans l'état d'arrivé par l'opération de substitution (i.e. une contrainte temporelle sera générée tel qu'elle soit composée de toutes les durées des $\tau - transitions$), par normalisation la contrainte maximale sur toute horloge sera gardée.
- Sur les transitions issues de cet état toutes celles qui dépendent de cette séquence, se retrouvent avec une condition d'exécution qui cumule les durées, aussi une normalisation est nécessaire pour ne garder que les maximales.

La sémantique associée au $\tau - cycle$ est représentée dans la Figure6.2 et qui s'articule autour de l'exécution infini sur un ensemble de transition fini (chemin fini) Figure6.2.(a)

Ce graphe peut être déplier comme présenté dans la Figure6.2.(b).après execution de l'action b, la transition par a sera lancée après la terminaison de l'action τ dans ce cas celle ci (la transition par a) doit attendre autant de fois que boucle l'action τ.

La contrainte temporelle ainsi définie sera un incrémentation (peut être à l'infini) de la durée de l'action τ.

On définit une contrainte temporelle incrémenté $F_i^\omega = \{x_i \geq d_i\}^\omega$ comme étant un contrainte incrémentale sur d_i.Formellement :

Définition 6.19 *Une contrainte temporelle de la forme* $F_i = \{x_i \geq d_i\}$ *est dite incrémentale sur* d_i *ssi :* $F_i^{\omega} = \{x_i \geq d_i\}^{\omega} = \begin{cases} \omega = 0 & alors \quad F_i = \{x_i \geq 0\} \\ \omega \geq 1 & alors \quad F_i^{\omega} = \{x_i \geq \omega d_i\} \end{cases}$

Les fonctions de translations en avant seront modifiées pour la considération des cycle $\tau - transitions$ comme suit :

Définition 6.20 *Soit* $\eta_i = \{(\tau_i, \tau_j) \in T \times T, \quad j = 1..n\}$; $F = \{x \geq d\}$ *la condition de durée de l'action* $\lambda(\tau_i)$ *,* γ *la condition d'exécution de* τ_i *; on définit la les deux fonctions* ϕ^{∇} *sur* η_i $i \neq 0$ *et* ϕ_0^{∇} *sur* η_0 *comme suit :* $\phi^{\nabla}(\eta_i, \gamma, F) =$

$$\begin{cases} si \quad \alpha(\tau_i) = \beta(\tau_i) \begin{cases} L_S^{'} = Ls[\beta(\tau_i) \mapsto Aug((Ls(\beta(\tau_i)) - F), D_f([\gamma + d]^{\omega}))] & (1) \\ \forall \tau_j \in T \ tel \ que \ (\tau_i, \tau_j) \in \eta_i : \begin{cases} \gamma_j \left[(\gamma + d)^{\omega} / x\right] & (2) \\ \phi^{\nabla}\left(\eta_j, \gamma, F\right) & (3) \end{cases} \end{cases} \\ sinon \qquad \phi(\eta_i, \gamma, F) \end{cases}$$

$$\phi_0^{\nabla}(\eta_0) = \begin{cases} \forall \tau_i \in T \quad tel \ que \quad (\tau_i, \varnothing) \in \eta_0 = \{(\tau_i, \varnothing) \in T \times \emptyset \diagup \tau_i \ une \ transition \ terminale\} \\ il \ n'existe \ pas \ de \ transition \ \tau_j \neq \tau_i \ et \ \beta(\tau_i) = \alpha(\tau_j) \\ \begin{cases} si \ \alpha(\tau_i) = \beta(\tau_i) \ \ alors \ L_S^{'} = Ls[\beta(\tau_i) \mapsto Aug((Ls(\beta(\tau_i)) - F), D_f([\gamma + d])) \\ sinon \qquad \phi_0(\eta_0) \end{cases} \end{cases}$$

Note : Le reste du traitement demeure sans changement, en effet le traitement des cycles de $\tau - transitions$ fait un ajustement sur la substitution dans les conditions d'exécutions et de durées dans le cas où la $\tau - transition$ boucle sur le même état.

6.4 Etude des blocages, l'implémentabilité et de la robustesse &Travaux connexes

Cette partie de l'étude concerne le modèle des DATAs par rapport à trois aspects différents, mais dont la portée est de taille, qui sont des exigences de la norme sur la formalisation des modèles de spécification Z500 [voir section3.3.1]. Cela touche à l'implémentabilité, la robustesse ainsi que la manipulation des actions internes.

6.4.1 L'étude des blocages (quiescence)

Un concept central dans la théorie non temporisée et la notion de quiescence qui caractérise un état du système qui ne produit plus aucune action observable, aussi la notion de trace de comportement peut être généralisée par l'observation de la quiescence. Informellement une implémentation et conforme à la spécification si les sorties et la quiescence sont correctement prédits, définits.

Un blocage dans la spécification n'est pas forcement une erreur, le cas livelock (dont l'origine est le masquage des actions, la composition parallèle des systèmes, etc. ....), ainsi le deadlock. Par conséquent lors de la modélisation, déjà il faut considérer les blocages valides.

Un test appliqué à une IUT doit pouvoir distinguer entre un blocage de l'IUT valide existant dans la spécification et celui non valide.

Treatmans, dans[Tre96b]les nomes aussi la suspension des sorties ou traces après lesquelles aucune action observable n'est observer. La définition originale de la propriété de quiescence sur les automates à entrées /sorties suppose l'absence d'observation des actions ainsi que l'absence des actions internes et ne considérant pas le blocage de type *livelock*, il se limite au système fortement convergent. Treatmans introduit une nouvelle action observable particulière δ pour spécifier l'arrêt du système c'est-à-dire aucune sortie n'est observable jusqu'à ce qu'une entrée le stimule.

Brinksma et al.dans [LBB05] proposent une extension de la relation de conformité pour les systèmes temps réels celle-ci paramétré par l'observation du quiescence avec durée. La notion de quiescence des états dans le domaine temporisée est définie comme un état ou le système est incapable de produire des actions observables dans l'immédiat ou dans le future sans avoir au préalable reçu des stimuli. Il est supposé que sous l'hypothèse de l'absence d'interaction entre le système est son environnement pour une durée de M unités de temps implique un état de quiescence, il a définit une approche de génération de test $M - parametree$.

Du fait que M est un paramètre à spécifier sans aucune indication sur la source de sa valeur, dans un système un état de quiescence qui nécessite une durée N : ($N < M$ ou $N > M$) ne peut être détecter est le verdict émit peut être non valide. Un état de quiescence qui dure N unité de temps moins que les M unités fixées dans la spécification n'est pas observé dans les délais, d'autant plus que si à cause d'une non-conformité on observe une action dans l'intervalle $[N, M]$ alors le verdict sera nécessairement invalide car l'observation est survenue après un état de blocage spécifié.

Dans notre proposition le traitement du blocage se fait implicitement sans modification de la spécification, donc aucune augmentation sur le nombre de transitions ni le nombre d'actions.

6.4.2 Le modéle des DATAs et le concept de quiescence

De part la prise en compte des durées d'action selon la sémantique de maximalité dans le modèle des DATAs, l'absence d'observation dans le cas d'un état quiescent est implicitement prise en charge, car les conditions des durées présentes sur tout état donne une information réelle sur l'avancement d'exécution du système (i.e. information sur les actions en cours d'exécutions) l'absence d'une réponse peut être quantifiée et mesurée.

Les deadlocks peuvent être considérés comme un cas particulier des blocages de sortie, lors de l'implémentation du test la valeur du temporisateur est calculée de façons précise à partir des conditions d'exécutions.

Aussi une méthode de déterminisation adéquate [Jér02] qui prend en compte la durées des actions internes pourrait résoudre le problème des cycles d'actions internes ainsi que le livelock dans le modèle, chose qui a été traiter dans le ce même Chapitre6.

Tout cela procure au modèle DATA une **Fiabilité** i.e une confiance dans les verdicts,

basés sur des valeurs de durées qui soient les plus colées à la réalité et une **Robustesse** dans la modélisation i.e l'indépendance par rapport à l'environnement d'exécution et au choix aléatoire des paramètres.

6.4.3 Les actions silencieuses ($\varepsilon - transitions$)

Les $\varepsilon - transitions$ sont des transitions étiquetées par des actions internes ou silencieuses τ, celle-ci est dite silencieuse car son apparition dans un processus est totalement invisible de l'extérieur, pratiquement cela signifie qu'un système effectue une action qui ne fait rien et ne prend aucun délai ceci parait non réaliste dans le monde physique.

Lors de la spécification cette action est très importante par la largeur de sont utilisation, elle spécifie l'indéterminisme, la synchronisation ou encore un niveau de confidentialité pour un système, tout ce qui est restreint et donc invisible. Dans la modélisation des systèmes non temporisés les actions silencieuses ont juste une représentation technique, alors que dans les systèmes temporisés, la situation et complètement différente.

Une autre propriété très importante sur les processus est la bissimulation de ceux ci. Deux forme de bissimulation sont connues « Faible » et « Forte», elles se distinguent par leur comportement vis à vis de l'action silencieuse τ. Lors des études des équivalences entre les systèmes la bissimulation forte est utilisée, celle ci considère les τ comme toute autre action observable[SA03]. D'ou l'idée que cette action est certes silencieuse, observationnellement elle ne fait rien mais si elle s'exécute elle doit se faire sentir par le biais d'un écoulement de temps, ce comportement différent se fera sentir concrètement par la suite dans les domaines de la vérification ou du test qui sont la principale motivation de cette théorie. Par conséquent la manipulation des dates doit être prise en compte impérativement.

Dans [BB96][VD97][BDGP98] les auteurs s'intéressent au modèle original des Automates Temporisés où les transitions sont étiquetées pour des actions non observables. Il est proposé une étude de la robustesse et de l'expressivité des Automates Temporisés d'Alur et Dill [AD90][AD94] avec les $\varepsilon - transitions$. Tel que la cloture par rapport à la composition(l'union et l'intersection des langauges temporisés), et au passage au complément ainsi que les opérations de raffinement, d'abstraction et le masquage des actions (Hide). un résultat intéressant dans le cas des Automates Temporisés et celui qui stipule que si les $\varepsilon - transitions$ ne remettent pas les horloges à zéro alors il existe un Automate Temporisé sans les $\varepsilon - transitions$ équivalent.

La sémantique sous-jacente au modèle DATAs considère les $\varepsilon - transitions$ et que les actions internes ont une durée, cette remarque est de taille car on ne peut considérer une modélisation sans passer par les étapes de raffinement successifs ou d'abstraction ; Aussi structurellement le Modèle DATAs peut être considéré comme une variante des Automates Temporisés classiques avec une sémantique particulière[Tre96b]. Toutes les propriétés vérifiées structurellement sur les Automates Temporisés sont acquises ; Ainsi structurellement il existe un DATA déterministe sans $\varepsilon - transitions$ qui lui est équivalent, ce qui rejoint l'étude faite dans le Chapitre 6. Ce résultat est significatif pour la déterminisation des DATAs est sont

grand potentiel.

6.4.4 L'implémentabilité et la robustesse

L'implémentabilité des modèles de spécification à fait l'objet de beaucoup d'études [DWM04a] [DWM04b] [AK05], cela consiste, à chercher un cadre d'implémentation à l'issue d'un travail de modélisation. Il est à remarquer que Les modèles de spécification sont des formalismes mathématiques. La sémantique des modèles comme les systèmes de transitions temporisés ou le Automates Temporisés, est en effet extrêmement précise, il est évident qu'un certain nombre de propriétés liées au caractère « idéal » du modèle mathématique est impossible à reproduire dans la réalité de l'exécution tel que : (i) la synchronisation parfaite des différentes composantes du système, réception en temps nul des entrées, émission en temps nul des sorties, (ii) précision infinie des horloges, (iii) temps d'exécution des actions nul (vitesse infinie du système lors de l'évaluation des gardes des transitions, et du choix de l'action à déclencher). Un exemple simple celui du protocole d'exclusion mutuelle de Fischer, prouvé valide par des outils tel que UPPAAL[G.B04], alors que ce résultat est étroitement liée à l'aspect infiniment précis du modèle des automates temporisés. On peut cependant se poser la question de la pertinence de la vérification ou du test formel de ces objets mathématiques, dès lors qu'ils représentent des programmes s'exécutant sur des machines réelles. Aucune plateforme d'exécution ne permet d'implémenter aussi précisément de tels modèles, et rien ne permet d'assurer a priori que les propriétés vérifiées sur le modèle théorique seront préservées lors de l'implémentation.

Plusieurs approches sont proposées, celles basées sur la modélisation dans la quelle les caractéristiques réelles de la plateforme sont modélisées[AK05], ou encore l'approche dite sémantique qui consiste à étendre la sémantique des Automates temporisées par un type particulier de plateforme dans ce cas la sémantique du modèle ainsi augmenté est dite robuste, intuitivement le terme robuste s'oppose aux fragilités déjà cités.

Nous pensons que le modèle des DATAs qui prend en compte explicitement les durées des actions est naturellement plus apte à une implémentation confirmée et sûre. Aussi l'implémentation du modèle va satisfaire de façon robuste les propriétés attendues. i.e. indifféremment de l'environnement d'exécution les ces propriétés reste valide.

Deux qualités sont souhaitables pour l'implémentation d'un modèle :

1- la conservation des propriétés par passage à l'implémentation ;

2- la propriété « faster is better » : si l'implémentation sur une plateforme est satisfaisante alors elles doit l'être sur un autre plateforme plus rapide, plus performante.

Il est certain que la maîtrise des durées des actions (quelle que soit leurs natures) permettras de conserver les deux propriétés lors du passage à l'implémentation.

6.5 Conclusion & synthèse

Dans ce chapitre qu'on considère principale, on propose une formalisation de la déterminisation du modèle des DATAs sans cycles de $\tau - transitions$ ensuite en considérant la divergence.

L'intérêt majeur de cette formalisation et la possibilité d'utilisation du modèle des DATAs dans les différents axes de recherche sur la formalisation des processus de vérification et du test.

Dans le domaine de la vérification une théorie importante propose la recherche d'un complément d'une spécification (i.e. un automate qui accepte tout ce qui est rejeté par la spécification initiale) et qui repose sur la condition de l'existence d'un équivalent déterministe du modèle[TJ06].

Aussi dans le domaine du test ou du diagnostic des erreurs, le testeur déterministe est calculé à partir du modèle d'une spécification non déterministe. Ce qui conforte notre théorie sur le large mais raisonnable spectre d'utilisation du modèle des DATAs.

Chapitre 7

Le modèle des DATAs et le test

7.1 Introduction

Comme il a été déjà exposé dans les chapitres précédents, la génération du test est considérablement influencée par le modèle de spécification choisi. Pour clarifier les idées, prenons l'exemple classique des deux machines $M1$ et $M2$ pouvant offrir deux tasses de café après l'introduction d'une pièce de monnaie et dont les comportements sont les suivants : La machine $M1$ possède un seul dispositif délivrant le café, alors que la machine $M2$ de dispose de deux dispositifs. Que ce soit la machine $M1$ ou la machine $M2$, un client doit introduire une pièce de monnaie p et interagir deux fois sur le bouton c de demande de café. Il est clair qu'un client utilisant ces deux machines peut s'apercevoir de la différence entre celles-ci, en effet dans la machine $M1$, après l'introduction d'une pièce de monnaie p et après la première interaction sur c, la deuxième interaction sur c est bloquante jusqu'à ce que la première tasse de café soit délivrée. Cependant, dans la machine $M2$, après l'introduction de la pièce de monnaie et la première interaction sur c, un client peut interagir une deuxième fois sur c et se voir délivrer les deux tasses de café en parallèle. Remarquons que cette différence a pu être observée du fait que l'opération de livraison de café n'est pas instantanée. Il est donc impératif de distinguer les deux machines dès les premières étapes de conception afin de rester en harmonie avec le monde réel [SB05].

Dans ce chapitre, nous présenterons les modèles existants pour le génération des testes basés sur les refus, nous citons les Graphes de Refus, et les Graphes de Refus Mixtes.

Nous mettrons en évidence les problèmes posés par les deux modèles du fait de l'absence du temps et nous proposerons une approche répondant à cette problématique. Celle-ci est basée sur l'utilisation du modèle des DATAs pour la génération des tests temporisés.

7.2 Le test et les Graphes de Refus

7.2.1 Graphes de Refus et la sémantique d'entrelacement[SG06]

Dans le cas de la sémantique d'entrelacement, les transitions sont des événements qui correspondent à l'exécution entière des actions. Construire des tests permettant de distinguer de tels comportements a conduit à l'apparition de nombreuses théories, parmi lesquelles on peut citer [Hen85][BSS87][Bri88][Dri92]. La mise en oeuvre de ces théories a permit en la définition d'un modèle entièrement abstrait appelé Graphe de Refus permettant une représentation des systèmes communicant se basant sur une sémantique proche de la sémantique de refus "failure semantics" de CSP[Hoa85] et un formalisme similaire au "failure tree" de [Bri88] mais permet de décrire les comportements récursifs et supporte des définitions opérationnelles des relation de conformité[Dri92]. L'approche des graphes de refus s'inscrit dans les méthodes formelles pour le test de conformité des systèmes non temporisés. Elle consiste en la création d'un testeur capable de détecter si l'implémentation viole la relation de conformité et cela par l'exécution en parallèle de l'implémentation sous test et du testeur. Le testeur dans ce cas est fabriqué à partir d'une spécification sous forme de système de transitions en utilisant un graphe de refus, ce dernier est un système de transitions déterministe où les nœuds sont augmentés par les actions interdites.

Intuitivement un graphe de refus est un système de transitions étiquetées déterministe tel que à chaque état est associé un ensemble de refus incluant les ensembles d'actions sur lesquelles le système peut se bloquer. Dans [Dri92] le modèle des graphes de refus est étudié ainsi que toutes les propriétés le concernant sont prouvées.

A titre d'exemple, reconsidérons l'exemple des distributeurs de café $M1$ et $M2$ dont les spécifications respectives sont données par les expressions de comportement Basic LOTOS suivantes : $M1 \equiv p; c; c; stop$ et $M2 \equiv p; (c; stop|||c; stop)$. Les deux machines $M1$ et $M2$ admettent le même système de transitions étiquetées (LTS) et par voie de conséquence elles admettent le même graphe de refus, représentés respectivement par les Figures 7.1.(a) et 7.1.(b).

Ceci a motivé la proposition du modèle des Graphes de Refus Mixtes (GRMs) incluant à la fois les blocages classiques, qualifiés de permanents, et les blocages temporaires. Il est évident que cette notion de blocage temporaire introduite est intrinsèquement liée à la non atomicité temporelle des actions. Cette remarque a conduit à l'adoption de la sémantique de maximalité qui permet d'échapper à l'hypothèse d'atomicité des actions. [Dev92][CS95][Saï96].

7.2.2 Graphes de Refus et la sémantique de maximalité [SG06]

Dans l'approche basée sur la sémantique de maximalité, le modele utilisé est celui des STEMs[voir Chapitre4]. Le principe de basesur lequel repose ce modèle consiste à associer à chaque état du système les actions qui sont potentiellement en cours d'exécution. Dans cette approche les transitions sont des évènements qui représentent le début d'exécution des

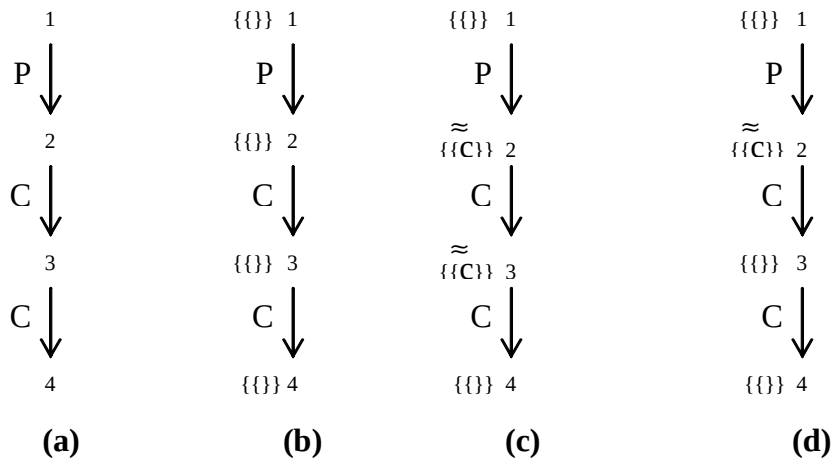

FIG. 7.1 – GRs et GRMs associés aux machines $M1$ et $M2$

actions. Les graphe de refus ont été étendu par de nouveaux blocages comme suit :

Graphes de Refus mixtes[SG06]

Définition 7.1 *Un Graphe de Refus Mixte (GRM) est une structure de graphe Bi-étiqueté déterministe : $Gm = (G, \Sigma, \Delta, Ref, g_0)$ où :*

- G est un ensemble fini d'états, $g_0 \in G$ est appelé état initial.
- $\Sigma \subseteq Act$ est un ensemble fini d'actions : alphabet de grm.
- $\Delta \subseteq (G \times \Sigma \times G)$ est une relation de transitions. Un élément $(g, a, g\prime) \in \Delta$ est noté $g \stackrel{a}{\Longrightarrow} g\prime$.

Etant donné qu'un GRM est déterministe, Δ est donc une famille de fonctions $\{f_a : G \longrightarrow G\}$ tel que :

$a \in \Sigma$, $g\prime = f_a(g)$. i.e. $\forall g \in G$, $\forall a \in \Sigma$; $\exists$ au plus un $g\prime \in G$ tel que $g \stackrel{a}{\Longrightarrow} g\prime$.

Dans ce qui suit il est supposer que les actions peuvent être de durées non nulles. Soient $\widetilde{L}$ et $\widetilde{\widetilde{L}}$ deux ensembles définis comme suit : ($\forall a \in L \Longrightarrow \widetilde{a} \in \widetilde{L}$ et $\widetilde{\widetilde{a}} \in \widetilde{\widetilde{L}}$) réciproquement ($\forall \widetilde{a} \in \widetilde{L} \Longrightarrow a \in L$) et ($\forall \widetilde{\widetilde{a}} \in \widetilde{\widetilde{L}} \Longrightarrow a \in L$).

- $Ref : G \longrightarrow P(P(\widetilde{L} \cup \widetilde{\widetilde{L}}))$ est une application qui définit pour tout état $g \in G$, l'ensemble des parties d'actions pouvant être refusées. Etant donné un état g tel que $A \in Ref(g)$, $\widetilde{a} \in A$ signifie que l'action a peut être refusée en permanence à l'état g. Ces blocages permanents sont similaires à ceux des graphes de refus. $\widetilde{\widetilde{a}} \in A$ signifie que l'action a est temporairement refusée à l'état g . Ce cas de blocage se présente lorsque au moins une action de durée non nulle qui a causé a n'a pas encore terminé son exécution.

En reprenant l'exemple des machines $M1$ et $M2$. Leurs graphes de refus mixtes sont représentés respectivement par les Figures7.1.(c) et 7.1.(d). La différence entre les deux machines

est capturée par le blocage temporaire sur l'action c. Après la trace $p;c$, aucun blocage temporaire sur c n'est observable à l'état 3 de la Figure7.1.(d). Cependant, un blocage temporaire sur l'action c est observable à l'état 3 de la Figure7.1.(c).

Remarque 7.1 *Dans ce qui suit, nous adoptons les notations suivantes :*

- $g \stackrel{{}_M a_x}{\Longrightarrow}_m$signifie que g admet une dérivation par l'atome ${}_M a_x$: $\exists\ g\prime$tel que $g \stackrel{{}_M a_x}{\Longrightarrow}_m g\prime$.
- $g \stackrel{{}_M a_x}{\not\Rightarrow}_m$ signifie $\urcorner(g \stackrel{{}_M a_x}{\Longrightarrow}_m)$.
- La séquence vide est notée ε.
- Si $g \stackrel{{}_M \tau_x}{\longrightarrow}_m g\prime$ alors $g \stackrel{\varepsilon}{\Longrightarrow}_m g\prime$
- Si $g \stackrel{\varepsilon}{\Longrightarrow}_m g\prime \stackrel{{}_M \tau_x}{\longrightarrow}_m g'' \stackrel{\varepsilon}{\Longrightarrow}_m g'''$ alors $g \stackrel{\varepsilon}{\Longrightarrow}_m g'''$

Etant donné que les transitions d'un GRM sont $G \times \Sigma \times G$, les mêmes notations de traces que celles définies sur les GRs seront utilisées. La différence concerne les ensembles de refus et leur interprétation.

Génération d'un GRM à partir d'un STEM[SG06]

Etant donné un $STEM\ S$, l'application $newstate$ associe à chaque état du $GRM\ \ Gm$, une partie de l'ensemble des états du $STEM$. $newstate$ est définie de la manière suivante :

- $newstate(g_0) = \{s, s_0 \stackrel{\varepsilon}{\Longrightarrow} s\}$.

- Pour chaque état créé du $GRM\ Gm$, pour toute action $a \in Act$, un nouvel état $g\prime$ est créé, tel que :

$newstate(g\prime) = \{s', \exists s \in newstate(g), s \stackrel{{}_M a_x}{\Longrightarrow}_m s\}$ ainsi qu'une transition $g \stackrel{a}{\longrightarrow} g\prime$.

Dans le cas où il existe déjà un g'' (vérifiant $g \leq g''$ tel que $newstate(g'') = \{s, \exists s \in newstate(g) :\ s \stackrel{{}_M a_x}{\Longrightarrow}_m s\}$), alors l'algorithme crée simplement une transition $g \stackrel{a}{\longrightarrow} g''$.

La construction de $Ref(g)$ (g étant un état du G) consiste en les étapes suivantes : $Ref(g) := \{\widetilde{c}/c \in \widetilde{out}(g) \backslash out(s), s \in newstate(g)\} \cup \{\widetilde{\widetilde{c}}/c \in out(s_i)$ **telque** $s_i \stackrel{{}_M c_x}{\longrightarrow}_m s_j$ **est une transition du** $STEM\ S\ \ et\ \ M \neq \emptyset\}$**, avec** $\widetilde{out}(g) = \{\widetilde{a} \backslash a \in out(g)\}$ **où** $out(g) = \cup_{s \in newstate(g)} out(s)$**.**

L'algorithme termine lorsque aucun nouvel état ne peut être créé.

Propriétés

1. Préservation des traces : $Tr(S) = Tr(Gm)$.

2. Déterminisme du GRM **:** $\forall\ \sigma \in Tr(S)$**,** il existe un unique noeud $g_\sigma \in G : g_\sigma = Gm\ after\ \sigma$**.**

3. Après la trace σ, un ensemble d'action peut être refusé, s'il est contenu dans un élément de l'ensemble de refus associé au noeud g_σ.$\forall\ \sigma \in Tr(S)$, $S\ after\ \sigma\ ref\ A \Longleftrightarrow A \in Ref(g_\sigma)$.

Proposition 7.1 *Soit une expression Basic LOTOS E, et S le système de transitions étiquetées maximales qui le modélise alors, si pour toute action $a \in Act$, a est considérée de durée nulle, le graphe de refus mixte Gm associé à S coïncide avec le graphe de refus de S.*

Preuve : [SG06]

Définition 7.2 *Etant donné un graphe de refus mixte Gm, la procédure de minimisation de Gm est définie inductivement sur l'ensemble des noeuds de G comme suit :*

- Pour tout noeud g de G, $Min(Ref(g))$ minimise l'ensemble de refus $Ref(g)$ vis à vis de la relation d'inclusion, en d'autres termes $\forall X, Y \in Min(Ref(g)) : (Y \subseteq X) \Longrightarrow (X = Y)$. Notez que $Min(Ref(g))$ contient toute l'information relative aux refus définis par $Ref(g)$.

- $Min(Gm) = Gm^{'} \ / \ \forall \ g \in G^{'} / \begin{cases} g \in G & \text{et} \\ Ref(g) & \text{est minimal.} \end{cases}$

Plus de détails sur les propriétés des GRMs, tel que la relation de bissimulation, et l'algorithme de réduction, dans [SG06].

Remarque 7.2 *Il est important de noter que lors de la déterminisation des LTSs les ensembles de refus sont calculés pour générer les Graphes de refus [KD01][KD93], il est de même pour les GRMs, qui ne sont que des STEMs déterministe ou chaque état est augmenté par les ensembles de refus des deux types.*

7.2.3 Décidabilité des refus temporaires dans les graphes de refus mixtes

Problématique

La prise en compte de la non atomicité temporelle et structurelle des actions dans le modèle des graphes de refus mixtes représente une avancée considérable dans le monde du test formel. Cependant, le problème de la décidabilité de la fin des refus temporaires pose un réel problème dans la mise en pratique du modèle. Afin de situer le problème de décidabilité et de mettre en lumière la solution proposé, considérons la donnée de la spécification d'une machine à café dont le comportement est le suivant : Après l'introduction de la pièce de monnaie deux cas se présentent ; Soit que la pièce de monnaie est valide, dans ce cas la machine délivre la tasse de café, dans le cas contraire la machine rejette la pièce. Supposons maintenant qu'une implémentation sous test (IUT) de cette machine nous est proposée. Le comportement de cette implémentation est décrit par le système de transitions étiquetées maximales de la Figure 7.2.(a) (l'action p : pour la pièce, l'action c : pour café, et l'action r : pour rejet de la pièce). Intuitivement, dans cette implémentation, après l'introduction de la pièce de monnaie, soit la machine sert une tasse de café, soit elle rejette la pièce ou bien elle peut de manière non observable passer dans un état dans le quel elle n'évolue plus. En d'autres termes cette implémentation vole de l'argent.

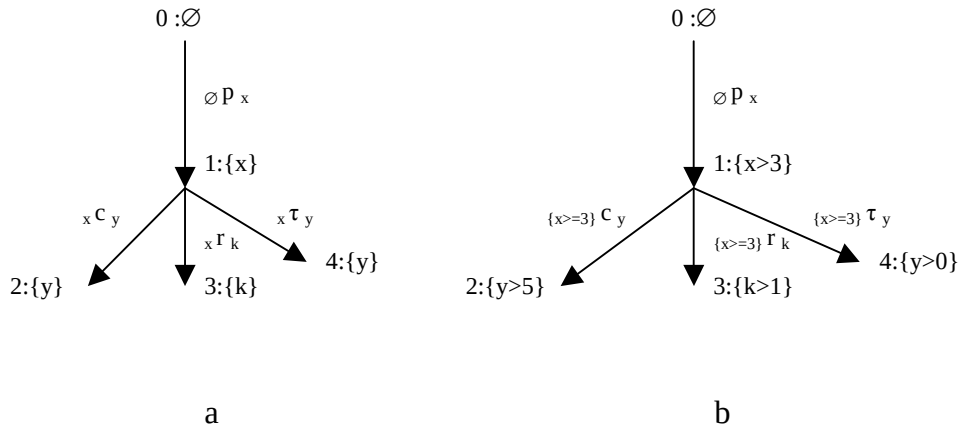

FIG. 7.2 – STEM et DATA représentants la machine à café qui vole de l'argent

L'observateur une fois qu'il a introduit sa pièce de monnaie, il s'attend à un blocage temporaire sur l'action c ou r, visiblement le blocage temporaire aura une durée égale à la durée de l'action p (reconnaissance de la pièce) si au moment du test le blocage persiste (ni tasse de café ni rejet de la pièce), le testeur est incapable de statuer sur le résultat du test car il ne peut décider de la durée du blocage temporaire et à quel moment celui-ci devient une défaillance ou une non conformité. Cette situation a déjà été étudier sous un autre aspect celui de la quiescence par Treatmans et brinksma [voir Section5.6.4 Chapitre5], dans le modèle des $LTSs$ ($IOLTs$) ; La solution retenue été la suspension des systèmes de transitions étiquetées et l'utilisation des temporisateurs lors de l'implémentation du test.

Notre approche et différente pour repérer les blocages non spécifiés ; elle repose sur l'utilisation du modèle des $DATAs$ et la temporisation des actions.

Approche proposée

Nous avons eu l'intuition suivante : l'association des durées explicites aux actions pourrait résoudre ce problème. L'utilisation d'un modèle sémantique pour la spécification des systèmes permettant de capturer la notion de durées d'actions sous forme de contraintes temporelles régissant l'évolution temporelle des systèmes serai adapté. Ainsi le modèle des automates temporisés avec durées d'actions (DATAs) nous semble approprié.

Pour l'exemple précèdent, la Figure7.2 (b) représente l'automate temporisé avec durées d'actions correspondant. En considérant la fonction de durée suivante : $d(p) = 3, d(c) = 5, d(r) = 1$.[1]

Après introduction de la pièce et durant les trois premières unités de temps aucune action n'est observable, c'est le blocage temporaire qui dure exactement trois unités de temps (le blocage temporaire de c et r), après cela soit que la tasse de café est servie avec une durée

[1] Les notations sont reprises de [SC03]

de cinq unités de temps soit que la pièce est rejetée, décision qui durera une unité de temps. Enfin, le cas échéant, l'implémentation passe dans la branche silencieuse ; Dans ce cas, après une durée maximale de trois unités de temps si aucune observation, le testeur peut déclarer le blocage permanent de l'implémentation.

7.2.4 Pour la suite...

Dans la suite de ce travail, nous proposons une extension temporisée des graphes de refus mixtes, fixant les délais des refus temporaires tout en capturant l'effet temporel des actions non observables, l'aspect déterministe du model DATA permet un passage assez technique (i.e. facile) vers un nouveau type de graphes de refus permettant aussi la manipulation de relation de conformité élaborée tel que celle proposée dans[KD01] ainsi que le traitement de la divergence de façons assez élégante.

7.3 La prise en compte des durées d'actions dans les graphes de refus

La prise en compte des durées d'actions de manières quantitative (explicite) dans la modélisation motive l'extension des GRMs par des contraintes temporelles. Ainsi le refus temporaire d'une action peut être quantifié.

7.3.1 Les Graphes de Refus Temporisés GRTs

Dans ce qui suit l'approche qui va être développée et dans le prolongement des graphes de refus [Dri92] et les graphes de refus mixtes[SG06]. Par l'attribution explicite d'une durée d'exécution à une action dans la modélisation, il devient possible de doter les refus temporaires de conditions de durée correspondant aux actions refusées temporairement cela permet lors de dépassement de ces délais de statuer sur la conformité de l'implémentation par rapport à la spécification.

7.3.2 Proposition d'une formalisation d'un GRT

Définition 7.3 *Un graphe de refus temporisé (GRT) est une structure de graphe Bi-étiquetée déterministe : $Gt = (G, g_0, \Sigma, \mathcal{H}, \Delta, Ref)$ où :*

- G est un ensemble fini d'états,
- $g_0 \in G$ est l'état initial.
- $\Sigma \subseteq Act$ est un ensemble fini d'actions : alphabet du GRT { actions observables }
- $\mathcal{H}$ ensemble fini d'horloges
- $\Delta \subseteq S \times \wedge 2_{fn}^{\Phi_t(\mathcal{H})} \times \Sigma \times \mathcal{H} \times S$ est une relation de transitions. Un élément de Δ sera noté $(s, \gamma, a, x, s\prime)$

• $Ref : G \rightarrow P\left(P\left(\Sigma \times \wedge 2_{fn}^{\Phi_t(H)}\right)\right)$est une application qui définit pour tout état l'ensemble des parties d'actions pouvant être refusées.

$Ref(g)$ est un ensemble d'éléments de la forme (c, γ) ou $c \in \Sigma$ et γ une conditions d'exécution.

$$Ref(g) = \{(c,\gamma)/\begin{cases} \text{Si } \gamma{=}\emptyset \text{ alors } c \text{ est refusée en permanence} \\ \text{Sinon } \gamma \neq \emptyset \text{ alors } c \text{ est refusée temporairement et } \gamma\text{est} \\ \text{la condition d'exécution associée au refus temporaire de } c \end{cases}$$

7.3.3 Génération d'un Graphe de Refus Temporisé à partir d'un DATA

La génération du graphe de refus temporisé à partir d'un DATA déterministe se fait par le calcul des ensembles de refus de chaque états du DATA comprenant les refus temporaire est ceux permanant en associant à chaque action refusée, des deux types les contraintes temporelles assurant leurs validités.

Un graphe de refus temporisé n'est en réalité qu'un DATA déterministe muni des ensembles de refus spécifiques

Calcule des ensembles de refus :

Soit D_{det} un $DATA$ déterministe, $D_{\text{det}} = (S, L_S, s_0, \mathcal{H}, T)$ et soit Gt le GRT généré : $Gt = (G, g_0, \Sigma, \mathcal{H}, \Delta, Ref))$ tel que :

$S = G$
$s_0 = g_0$
$T = \Delta$

$$Ref\,(g) = Ref_p\,(g) \cup Ref_t\,(g) \quad \text{pour tout } g \in G$$

$$Ref_p\,(g) = \{(c,\phi)\,/c \in out\,(g)\}$$

$$Ref_t\,(g) = \left\{ (c,\gamma)\,/c \in out\,(g)\, et \begin{pmatrix} \exists \tau_i \in T \\ \tau_i = g \xrightarrow{\gamma,c,x} g' \\ \gamma \neq \emptyset \end{pmatrix} \right\}$$

L'ensemble des refus d'un état g du graphe ainsi construit est composé de deux sous ensembles, le premier regroupe les refus permanents tel que définis dans les GRs classiques augmenté de l'ensemble vide pour la contrainte temporelle à respecter. Contrairement au deuxième ensemble où chaque action refusée est dotée d'une conditions d'exécution qui lors de l'implémentation du test limite le temps de blocage sur cette action. Et en fin permet de distinguer le dépassement qui nuit à la conformité.

Propriétés

Aussi on prouve par construction que :

- Il y'a préservation de traces entre le DATA et le GRT correspondant.

- Le déterminisme du GRT qui decoule du déterminisme du DATA source.

Discussion

Si on considère que toute action à une durée positive ou nulle (sans durée dans le temps) la condition de durée sera la suivante $\{x\} = \{x >= 0\}$, le DATA se réduit à un STEM et toute horloge sera assimilée à un événement déclenchant une action, par voie de consequence le GRT sera réduit au GRM correspondant.

Chapitre 8

Conclusion et perspectives

Le travail présenté dans ce mémoire est une contribution à la spécification des systèmes avec durées explicites des actions. Nous avons eu l'intuition que le modèle des DATAs se prête naturellement à la spécification des systèmes concurrents grâce à la puissance des contraintes temporelles qui capturent les durées des actions.

Ce modèle qui se situe aux frontières des modèles discrets et ceux temporisés sans pour autant œuvrer sur les contraintes temporelles des systèmes temps réels. Cette faculté, en sus des propriétés de déterminisme, d'implémentabilité et de robustesse étudiées, lui confère une place assez prometteuse pour son utilisation future dans le domaine de la vérification et du test formel.

Ce travail a débouché sur la définition formelle du processus de déterminisation du modèle des DATAs, une étude d'implémentabilité et de robustesse a permis de le positionner par rapport aux modèles temporisés existants. Ce résultat nous a permis d'étendre un modèle basé sur les refus par les durées des actions.

8.1 Dans le domaine du test

Une étude de l'état de l'arts concernant les modèles les plus couramment utilisés nous a fournit un scénario, bien ajusté, relatif à la pratique du test. Que ce soit le test de conformité, d'interopérabilité ou de robustesse, le schéma est identique, plus explicitement :

Un système réactif est généralement décrit dans un langage de description spécialisé (par exemple SDL, LOTOS, UML). La sémantique opérationnelle de ces langages définit les comportements de la spécification, elle peut être décrite par un système de transitions. Les attributs du test peuvent être résumés en ce qui suit :

1. **Modèle d'implémentation** : L'Implémentation Sous Test est une boîte noire, pour pouvoir raisonner formellement sur la conformité, il faut se baser sur des modèles formels, seule possibilité pour définir une relation mathématique entre implémentation et spécification. On a donc l'hypothèse de test suivante : Les comportements possibles de l'implémentation sont supposés modélisables par un système de transitions.

2. **Comportements visibles** : Dans la pratique, les tests observent les réactions d'un système à des stimuli du testeur, les séquences d'actions visibles et les comparent aux comportements attendus. Il est donc nécessaire d'identifier l'ensemble des traces de la spécification ou tout au moins les traces prévisibles résultant des stimuli du testeur. L'ensemble des traces de la spécification peut être déterminé à partir d'un système de transitions déterministe.

3. **Les Blocages** : Dans la modélisation par système de transitions, les blocages sont définies dans la littérature comme suit : Le blocage de sortie (outputslock), le blocage complet (deadlock) et le blocage vivant (livelock). Dans la pratique du test, en plus des traces observables, les tests doivent permettre la détection des blocages (quiescence). Pour cela deux traitements sont nécessaires :

   - La spécification est augmentée par une transition libellée par une action observable particulière afin de situer les blocages specifiés.
   - Lors de l'exécution du test, des temporisateurs (timers) sont utilisés, ces derniers sont armés en attente d'une réponse de l'IUT.

4. **La relation de conformité** : Pour les systèmes de transitions, plusieurs relations sont envisageables suivant les observations permises par le test, (traces, blocages, transitions tirables, refus, etc...) et la notion de correction voulue.

5. **Test avec objectifs de test ayant des actions internes** : Dans le domaine du test, seul l'aspect observable est traité dans l'IUT et les objectifs de test ; Alors que les objectifs de test avec actions internes permettent plus de précision dans la sélection des tests. Dans des travaux plus récents, une approche qui révèle la considération des actions internes particulièrement dans les objectifs de test se présente comme une nouveauté dans ce domaine, ceci est justifié par la complexité croissante des systèmes à modéliser.

Dans le cadre de notre travail, le modèle des DATAs étant déterminisable, et que la manipulation des actions internes est particulière du fait qu'elles sont considérées comme toute autre action par rapport à leurs durées, même après élimination par la déterminisation, la distance temporelle qu'elle correspondante est gardée et respectée. Ces propriétés prédisposent le modèle à être utilisé comme modèle de spécification et de modélisation de l'implémentation sous test. Aussi vont permettre une plus grande précision lors de la génération des suites de test.

8.2 Par rapport à l'existant :

Nous avons étudié la littérature consacrée aux modèles sémantiques pour la spécification des systèmes en vu de valider, et la spécification et la modélisation. Et en confrontant l'état de l'art avec notre approche il s'est dégagé ce qui suit :

Le Modèle des DATAs est un modèle déterminisable contrairement aux modèles existant basés sur les systèmes de transitions temporisés (Travaux de B. BERARD sur les automates temporisés[Bér04]), où de larges contraintes sont posées sur le modèle pour décider de sa déterminisation.

8.3 Perspectives.

Le travail présenté dans ce document peut être continué dans plusieurs directions. Nous proposons trois voies possibles qui sont forment une suite logique des travaux précédents et qui nous intéressent particulièrement :

1- Le test de conformité et la vérification formelle sont deux approches ayant des bases solides pour la validation des systèmes réactifs, les deux approches consistent en la comparaison de deux vues (Niveau d'abstraction) différentes.

La vérification se penche sur la comparaison de la spécification et un ensemble de propriétés que le système doit vérifier ; alors que le test de conformité compare le comportement observable et une implémentation sous forme de boîte noire, utilisant une spécification formelle et une relation de conformité.

Sûrement les deux techniques sont complémentaires, aussi elles reposent sur les mêmes algorithmes et techniques de base[AG99] [CJ04] [JB04] [HH02]. Cependant, les utiliser l'une à la suite de l'autre risque de compromettre la validité des propriétés. La question devient alors : est-il possible d'assurer que l'implémentation vérifie les propriétés attendues ?

Utiliser le modèle des DATAs comme modèle de spécification et modélisation nous semble assez naturel, reste alors l'expression des propriétés dans une logique et la définition d'une relation de conformité pour répondre à cette question.

2- L'implémentation des spécifications pour la vérification formelle ou la génération des suites de test ou enfin pour exécuter le test, nécessite des transformations sur la spécification initiale. La raison la plus invoquée dans la littérature est le passage de machines avec actions instantanées à des machines avec délais pour les actions.[DWM04a][GP07][JC08], une question qui reste posée est la décidabilité de l'existence d'une telle implémentation[DWM04a].

Le modèle des DATAs, comme il se présente, nous semble assez prometteur dans cette axe de recherche du faite qu'il peut être implémenté sur une machine réelle avec des durées d'actions. On se pose le cas d'étude suivant :

L'étude du raffinement temporelle des actions sera alors judicieuse pour coller au schéma du processus de spécifications successives des systèmes.

3- Plusieurs modèles temporisés existent dans la littérature, l'abondance des études prouve l'intérêt de cette dernière approche. Il nous semble intéressant de pouvoir situer le modèle des DATAs par rapport au pouvoir d'expression et l'inclusion de langages, parmi eux.

Chapitre 9

Annexe

9.1 Glossaire inspiré de ISO/9646

implémentation sous test (IUT pour Implementation Under Test) : c'est l'implémentation réelle du système à tester. C'est en général un code exécutable qui n'est pas connu du testeur. L'implémentation peut également être matérielle. On parle aussi de système sous test (SUT pour System Under Test) quand l'objet à tester est composite.

Testeur : programme exécutant des tests sur l'**IUT**, dans certaines architectures de test, plusieurs testeurs peuvent être nécessaires.

Protocol Data Unit (PDU) : élément d'information entre entités d'un protocole (message).

Abstract Service Primitive (ASP) : élément de service du protocole, (appel de fonction).

Points de contrôle et d'observation (PCO) : ils définissent l'interface entre le testeur et l'**IUT**, les points auxquels le testeur pourra contrôler l'**IUT** et/ou l'observer via l'échange de PDU et d'ASP.

Architecture de test : description abstraite (modèle) de la situation du testeur vis à vis de l'**IUT**, c'est-à-dire quelles sont les **PCOs**, comment s'effectue la communication entre testeur et **IUT**. L'architecture de test est donc un modèle de l'architecture réelle de test. L'écriture des tests dépend donc étroitement de l'architecture choisie. Plusieurs architectures types sont normalisées : locale, distante, multi partie, etc...

Suite de test : ensemble des tests élémentaires (cas de test) d'une **IUT**, en TTCN (Tree and Tabular Combined Notation), langage de description de tests issu de **ISO/** 9646), une suite de test contient plusieurs parties. Le sommaire définit l'architecture de test considérée et contient la structure c'est-à-dire comment la suite est organisée en groupes et sous-groupes et l'index qui décrit chaque objectif de test. La partie déclarations définit les types de données, les **PDUs** et **ASPs**, les temporisateurs, les **PCOs**, les variables de la suite de test, des opérations de base. La partie, contraintes définit les valeurs des différents champs des **PDUs** et **ASPs** utilisées dans les cas de test. La partie comportementale décrit le comportement

du testeur par des cas de test.

Objectif de test : il décrit de manière abstraite l'objectif recherché par l'exécution d'un cas de test c'est-à-dire le comportement qu'il est supposé tester sur l'**IUT**. En général, un objectif découle d'une exigence de conformité (conformance requirement) décrivant une propriété que le système (la spécification et son implémentation) est sensée vérifier.

Cas de test : décrit un programme constitué d'interactions entre le testeur et l'implémentation sous test, des opérations sur des temporisateurs de test, des verdicts. Il est généralement constitué d'un préambule, d'un corps de test, d'une séquence d'identification et d'un postambule. Il peut faire référence à des procédures qui sont des portions de cas de test communes à plusieurs cas de test. En particulier, les parties préambule, séquence d'identification et postambule sont décrites par des procédures.

Préambule : le préambule a pour but de mener l'**IUT** dans un état où on pourra appliquer le corps du test.

Corps du test : c'est la partie du cas de test servant à vérifier l'objectif de test.

Identification : souvent, une partie du cas de test sert à identifier l'état dans lequel se trouve l'**IUT** (ou plutôt se trouvait l'**IUT** juste avant la phase d'identification)

Postambule : d'après **ISO/** 9646, le postambule sert à revenir dans un état initial afin d'enchaîner un autre cas de test. En fait, il s'agit bien souvent plutôt de l'état de contrôle initial.

Verdicts : Les comportements décrits dans les différentes parties d'un cas de test sont équipées de verdicts qui déterminent l'acceptation ou le rejet de l'**IUT** en fonction de ses comportements observés. Plus précisément, on distingue trois verdicts principaux, FAIL, PASS et INCONCLUSIVE. En principe ces verdicts sont positionnés sur des observations du testeur, des entrées ou des échéances de temporisateurs. FAIL signifie le rejet de l'**IUT**. (PASS) signifie que l'objectif de test a été atteint. Après identification et postambule, et en l'absence d'erreur révélée dans ces phases, un verdict PASS est attribué. (INCONCLUSIVE) signifie que le comportement observé est correct mais ne permet pas de satisfaire l'objectif. Ceci est dû à l'impossibilité de contrôler l'**IUT** (on parle souvent de non-déterminisme). La solution est alors de rejouer le test jusqu'à l'obtention d'un autre verdict. Comme pour (PASS), un postambule après un verdict (INCONCLUSIVE) mènera à un verdict INCONCLUSIF si aucune erreur n'est détectée.

Bibliographie

[ABH99] R.G. de Vries J. Tretmans-N. Goga L. Feijs S. Mauw A. Belinfante, J. Feenstra and L. Heerink. Formal test automation : A simple experiment. *12th Int. Workshop on Testing of Communicating Systems,*, page $179_196.$, 1999.

[AD90] R. Alur and D. Dill. Automata for modeling real-time systems. In *Proc. 17th International Colloquium on Automata, Languages and Programming (ICALP'90)*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer-Verlag, 1990.

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

[AG99] C.L. Heitmeyer A. Gargantini. Using model checking to generate tests from requirements specifications. *European Software Eng. Conf. and ACM SIGSOFT Symp. Foundations of Software Eng. (ESEC/FSE '99)*, pages 146–162, 1999.

[AK05] TRIPAKIS S ALTISEN K. Implementation of timed automata : An issue of semantics or modelling. Technical Report Nř TR-2005, Verimag, Centre équation, 38610 Gières, France, Juin 2005.

[Alu99] R. Alur. Timed automata. In *Proc. 11th International Conference on Computer Aided Verification (CAV'99)*, volume 1633 of *LNCS*, pages 8–22. Springer-Verlag, 1999.

[ARC93] Marc Phalippou. Ana R. Cavalli, Jean Philippe Favreau. Formal methods for conformance testing : Results and perspectives. *Protocol Test Systems,*, page 3 17, 1993.

[Arn92] A. Arnold. *Systèmes de transitions finis et sémantique des processus communicants.* Masson, Paris, 1992.

[Arn94] A. Arnold. Hypertransition systems. In P. Enjalbert, E. W. Mayr, and K. W. Wagner, editors, *STACS'94*, volume 775 of *LNCS*, pages 327–338. Springer-Verlag, 1994.

[Bae93] Brigitte Baer. A conformance testing approach for managed objects. In *In 4th IFIP/IEEE International Workshop on Distributed System Operations and Management,*, Long Branch, New Jersey, USA„ octobre 1993.

[BAL04] D. Chen D. Khuu B. Alcalde, A. Cavalli and D. Lee. Network protocol system passive testing for fault management : A backward checking approach,. *Proceeding of FORTE/PSTV'2004,*, 2004.

[BB96] A Petit B Bérard, P Gastin. On the power of non observable actions in timed automata. proceeding of the 13th annual symposium on theoretical aspect of computer science (STACS'96) number 1046 Springer Verlag, 1996. lecture notes in computer science pages 257-268.

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

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

[Bér04] B. Bérard. *Vérification de Modèles Temporisés " (Mémoire D'habilitation À Diriger Les Recherches)*. PhD thesis, Laboratoire Spécification et Vérification ENS de Cachan CNRS UMR 8643, 2004.

[BK84] J.-A. Bergstra and J.-W. Klop. Process algebra for synchronous communication. *Information and Computation*, 60:109–137, 1984.

[Bri88] E. Brinksma. A theory for the derivation of test. *Protocol Specification, Testing and Verification, S.Aggarwal and Sabnani,eds*, VIII:63–74, 1988.

[BS05] N. Belala and D. E. Saïdouni. Non-atomicity in timed models. In *Proceedings of International Arab Conference on Information Technology (ACIT'2005)*. Al-Isra Private University, Jordan, December 6-8th 2005.

[BSS87] E. Brinksma, G. Scollo, and C. Steenbergen. Lotos specifications, their implementations and their tests. *Proceedings of the sixth international workshop on protocol specification, testing and verification, eds. Behcet Sarikaya ,North-Holland, Amsterdam*, VI:349–360, 1987.

[BT00] E. Brinksma and J. Tretmans. Testing transition systems : An annotated bibliography. *the proceeding of Modelling and Verification of Parallel Processes (MOVEP2k),*, LNCS 2067:187–195, 2000.

[CC07] H. Marchand V. Rusu C. Constant, T. Jéron. Integrating formal verification and conformance testing for reactive systems. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, 33, NO. 8, AUGUST 2007.

[Cho78] T. Chow. Testing software design modelled by finite state machines. *IEEE Transactions on Software Engineering,*, SE-4(3), May 1978.

[CJ04] T. Jéron C. Jard. TGV: Theory, principles and algorithms a tool for the automatic synthesis of conformance test cases for non-deterministic reactive systems. *Springer-Verlag 2004 Int J Softw Tools Technol Transfer (2005)7:297 -315 /Digital Object Identifier (DOI) 10.1007/s10009-004-0153-x*, 20 October 2004.

[CS95] J. P. Courtiat and D. E. Saïdouni. Relating maximality-based semantics to action refinement in process algebras. In D. Hogrefe and S. Leue, editors, *IFIP TC6/WG6.1, 7th Int. Conf. on Formal Description Techniques (FORTE'94)*, pages 293–308. Chapman & Hall, 1995.

[DC01] V. Rusu E. Zinovieva D. Clarke, T. Jéron. Automated test and oracle generation for smart-card applications. *International Conference on Research in Smart Cards (e-Smart'01)*, LNCS 2140:58–70, 2001.

[DCZ02] V. Rusu D. Clarke, T. Jeron and E. Zinovieva. STG : A symbolic test generation tool. *Lecture Notes in Computer Science*, LNCS 2280:470, 2002.

[Dev92] R. Devillers. Maximality preservation and the ST-idea for action refinement. In G. Rozenberg, editor, *Advances in Petri Nets*, volume 609 of *LNCS*, pages 108–151. Springer-Verlag, 1992.

[dH84] R. de Nicola and M. Hennessy. Testing equivalences for processes. *TCS*, 34:83–133, 1984.

[Dri92] K. Drira. *Transformation et Composition Des Graphes de Refus : Analyse de la Testabilité.* PhD thesis, Université Paul Sabatier de Toulouse, 1992.

[dS85] R. de Simone. Higher-level synchronising devices in MEIJE-SCCS. *Theoretical Computer Science,*, 37:245–267, 1985.

[DS89] T. Leung D. Sidhu. Formal methods for protocol testing : A detailed study. *IEEE transactions on software engineering*, 15, n. 4„ avril 1989.

[DWM04a] MARKEY N. RASKIN J.-F. DE WULF M., DOYEN L. Robustness and implementability of timed automata. *LAKHNECH Y., YOVINE S., Eds., Proceedings of the Joint Conferences Formal Modelling and Analysis of Timed Systems (FORMATS'04) and Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'04)*, 3253 de Lecture Notes in Computer Science, Springer:118–133, 2004.

[DWM04b] RASKIN J.-F DE WULF M., DOYEN L. Almost ASAP semantics : From timed models to timed implementations. In PAPPAS G. J ALUR R., editor, *7th International Workshop on Hybrid Systems : Computation and Control (HSCC'04)*, volume 2993 of *LNCS*, pages 296–310. Springer Verlag, 2004.

[EM85] H. Ehrig and B. Mahr. *Fondamentals of Algebraic Specification,Equations and Initial Semantics.* Springer-Verlag, 1985.

[FDT87] ISO/TC97/SC21/WG 1 FDT/C. Lotos,a formal description technique based on the temporal ordering. Technical report, Technical Report DIS 8807,ISO„ 1987.

[Fle01] E. Fleury. *Automates Temporisés Avec Mises À Jour.* PhD thesis, Ecole Normale supérieure de Cachan, 2001.

[F.M92] P. Azéma F.Michel. Apprentissage par scénarios. *Ergonomie et informatique Avancée, ERGO.IA 92 Biarritz, France*, Octobre 7-9,1992.

[G.B04] K.G. LARSEN G.BEHRMANN, A. DAVID. A tutorial on uppaal. *Revised Lectures of the International School on Formal Methods for the Design of Real-Time Systems, (SFM-RT'04)*, 3185 de Lecture Notes in Computer Science, Springer:200–236, septembre 2004.

[G.G70] G.Gonenc. A method for the design of fault detection experiment. *IEEE transactions on Computers*, C-19:551–558„ 1970.

[Gil62] A. Gill. Introduction to the theory of finite-state machines. *Mc Graw-Hill ,NewYork-USA,*, 1962.

[GL93] G.V. Bochmann G. Luo, A. Petrenko. Selecting test sequences for partially specified nondeterministic finite state machines. *Publication n. 864, Université de Montréal*, février 1993.

[GP07] S. Graf and A. Prinz. Time in state machines. *Fundamenta Informaticae*, 77(1- 2):143–174, 2007.

[Hen85] M. Hennessy. Acceptance trees. *ACM Journal*, 32(4):896–928, October 1985.

[Hen88] M. Hennessy. *Algebraic Theory of Processes.* MIT Press, 1988.

[Her01] F. Herbreteau. *Automates À Le Réactifs Embarqués Application À la Vérification de Systèmes Temps-Réel.* PhD thesis, Ecole Centrale de Nantes, décembre 2001.

[HH02] O. Sokolsky H. Ural H.S. Hong, I. Lee. A temporal logic based theory of test coverage and generation. *Proc. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS '02)*, pages 327–341, Apr. 2002.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to automata theory, languages, and computation.* Addison-Wesley, 1979.

[ISO91] CCITT. ISO. Information technology. open systems interconnection , conformance testing methodology and framework. Technical report, International Standard IS-9646, x.290-x.294 edition, partie 3, 1991.

[ISO94] ISO. information technology - open systems interconnection confor-mance testing methodology and framework. Technical report, - Parts 1-7. International Standard ISO/IEC 9646/1-7, 1994.

[JB04] B. Jonsson P. Pettersson J. Blom, A. Hessel. Specifying and generating test cases using observer automata. *Proc. Workshop Formal Approaches to Software Testing (FATES '04), J. Grabowski and B. Nielsen, eds*, 2004.

[JC08] A. SLISSENKO J. COHEN. Implementation of timed abstract state machines with instantaneous actions by machines with delays. Technical Report TR-LACL-2008-02, Laboratoire d'Algorithmique, Complexité et Logique (LACL) Département d'Informatique Université Paris 12 Val de Marne (Paris Est), Faculté des Science et Technologie, January 2008.

[Jér01] T. Jéron. Le test de conformité : état de l'art. *Rapport pour l'AEE (Architecture Electronique Embarquée)*, 2001.

[Jér02] T. Jéron. TGV : Théorie, principes et algorithmes. *Techniques et Sciences Informatiques, numéro spécial Test de Logiciels*, 21:108–109, 2002.

[JER04] T. JERON. *Habilitation À Diriger Des Recherches" Contribution À la Génération Automatique de Tests Pour Les Systèmes Réactifs "*. PhD thesis, Université de Rennes 1 Institut de Formation Supérieure en Informatique et en Communication, Version du 6 février 2004.

[KA05] Pierre-Alain REYNIER Stavros TRIPAKIS Karine ALTISEN, Nicolas MARKEY. Implémentabilité des automates temporisés. *Article rédigé dans le cadre du projet CORTOS de l'ACI " Sécurité Informatique " MSR 05*, 2005.

[Kab95] Paul Kaboré. *Une Approche de Test de Conformité Des Systèmes D'administration de Réseaux.* PhD thesis, Université Henri Poincaré, Nancy I, Centre de Recherche en Informatique de Nancy (CRIN)„ 1995.

[KD93] B. SOULS A-M. CHEMALI K. DRIRA, P. AZEMA. A formel assessment of synchronous testability for communicating systems. *IEEE*, pages 149–156, 1993.

[KD01] P. DE SAQUI SANNES K. DRIRA, P. AZEMA. Testability analysis in communicating systems. *elsevier science B. V. computer networks 36*, pages 671–693, 2001.

[Kho06] Farès Saad Khorchef. *Un Cadre Formel Pour Le Test de Robustesse Des Protocoles de Communication.* PhD thesis, l'université de bordeaux I„ 13/12/2006.

[Klo92] H. Kloosterman. Test derivation from non-deterministic finite state machines. *proceedings of the 5 th International Workshop on Protocol Test Systems, Montréal*, September 1992.

[LBB05] E Brinksma L Brandan Briones. Test generation framework for quiescent real-time systems. In J.Grabowski and B.Nielsen, editors, *FATES*, volume 3395 of *LNCS*, pages 64–78, Berlin Heidelberg, 2005. Springer-Verlag.

[LL95] R. Lai and W. Leung. Industrial and academic protocol testing : The gap and the means of convergence. *Computer Networks and ISDN Systems,*, 27:27:537–547„ 1995.

[Loh02] C. Lohr. *Contribution à la Conception de Systèmes Temps-Réel S'appuyant sur la Technique de Description Formelle RT-LOTOS.* PhD thesis, LAAS-CNRS, 7 avenue du colonel Roche, 31077 Toulouse Cedex France, 2002.

[LY94] D. Lee and M. Yannakakis. Testing finite state machines: State identification and verification. *IEEE Trans Computer*, pages 43(3):306–320, 1994.

[LY96] D. Lee and M. Yannakakis. Principles and methods of testing finite state machines- a survey. *Proceedings of the IEEE*, pages 84(8):1090–1123, August 1996.

[Lyn88] N.-A. Lynch. I/O automata : A model for discrete event systems. *Proc. of 22ndConf. on Information Sciences and Systems, Princeton, NJ, USA,*, pages 29–38, March1988.

[Mal07] W. Mallouli. Projet POLITESS, l4.1 définition d'une architecture générique pour la surveillance - version révisée 0.2-10/04/2007. version révisée 0.2-10/04/2007.

[Mil80] R. Milner. *Communication and Concurrency*, volume 92 of *LNCS*. Springer-Verlag, 1980.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[MOK06] Houda BEL MOKADEM. *Vérification Des Propriétés Temporisées Des Automates Programmables Industriels*. PhD thesis, L'ECOLE NORMALE SUPERIEURE DE CACHAN, septembre 2006.

[Mor00] P. Morel. *Une Algorithmique Efficace Pour la Génération Automatique de Tests de Conformité*. PhD thesis, Universite de Rennes 1 Equipe d'accueil : PAMPA Ecole Doctorale : Sciences Pour l'Ingénieur Composante universitaire : IFSIC/IRISA, 7 février 2000.

[Mye80] Glenford J. Myers. Review of "the art of software testing ", wiley-interscience,. *Year of Publication: 1980 ISSN:0095-0033*, 11(4 (Summer 1980)):23 – 23, 1980.

[NSY93] X. Nicollin, J. Sifakis, and S. Yovine. From ATP to timed graphs and hybrid systems. *Acta Informatica*, 30(2):181–202, 1993.

[NT81] S. Naito and M. Tsunoyama. Fault detection for sequential machines by transitions tours. *IEEE Fault Tolerant Comput. Symp, IEEE Computer Soc. Press*, pages 238–243, 1981.

[Pet62] C. Petri. *Kommunikation mit Automaten*. PhD thesis, Institut für Instrumentelle Mathematik, Bonn, Germany, 1962.

[Pet63] C.-A. Petri. Fundamentals of a theory of asynchronous information flow. Amsterdam, North Holland, 1963.

[Pet91] A. Petrenko. Checking experiments with protocol machines. *proceedings of the 4 th International Workshop on Protocol Test Systems, The Hague*, October 1991.

[Pha94] M. Phalippou. *Relations D'implantations et Hypothèses de Test sur Les Automates À Entrées et Sorties*. PhD thesis, Université de Bordeaux 1, 1994.

[Pri03] Armelle Prigent. *Le Test Des Systèmes Temps-Réel Paramétrés : Application À la Conception D'architectures Avioniques*. PhD thesis, délivré conjointement par l'Ecole Centrale de Nantes et l'université de Nantes Spécialité, 12 décembre 2003.

[PSS93] J.P. Courtiat P.Azéma P.de Saqui Sannes, K.Drira. Séquences de test et testeurs canoniques dérivables de spécifications estelle. *une expérience avec le protocole MMS, in : R. Dssouli, G.V. Bochmann (Eds), CFIP93 Ingénierie des Protocoles,Montréal, Canada*, Spetembre 1993.

[PT92] K. Naik P. Tripathy. Generation of adaptative test cases from nondeterministic finite state models. *proceedings of the 5 th International Workshop on Protocol Test Systems, Montréal*, September 1992.

[RC02] H. Waeselynck R. Castanet. Résumé de synthèse des travaux de l'Action spécifique AS 23 "test avancé de systèmes complexes, test de robustesse". Technical report, LaBRI, équipe MVT SI (Modélisation, Vérification et Test de systèmes informatisés), 2001-2002.

[RHW] 2003. Rational. Http ://Www.Rational.Com/Products/Tstudio/Prodinfo.Jsp.

[SA03] G. HAINS S. ANANTHARAMAN. A synchronous Bisimulationŋ Based approach for information flow analysis. Technical Report 2003-01, Université d'Orléans, France LIFO, 2003.

[Saï96] D. E. Saïdouni. *Sémantique de Maximalité: Application au Raffinement d'Actions en LOTOS*. PhD thesis, LAAS-CNRS, 7 av. du Colonel Roche, 31077 Toulouse Cedex France, 1996.

[San97] Mazziotta Sandro. *Spécification et Génération de Tests Du Comportement Dynamique Des Systèmes À Objets Répartis*. PhD thesis, Université de Nice-Sophia Antipolis u.f.r. faculté des sciences institut eurecom, 24 octobre 1997.

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

[SC03] D. E. Saïdouni and J. P. Courtiat. Prise en compte des durées d'action dans les algèbres de processus par l'utilisation de la sémantique de maximalité. In *Ingénierie Des Protocoles (CFIP'2003)*. Hermes, France, 2003.

[SF] F. Khendek M. Amalou A. Ghedamsi Test Selection Based on Finite State Models IEEE Transactions on Software Engineering Vol. 17 N. 6 Juin 1991 S. Fujiwara, G. Bochmann.

[SF91] G. Bochmann S. Fujiwara. Testing non-deterministic state machines with fault coverage. *proceedings of the 4th International Workshop on Protocol Test Systems, The Hague*, October 1991.

[SG06] D. E. Saïdouni and A. Ghenaï. Intégration des refus temporaires dans les graphes de refus. In *NOTERE*, Toulouse, France, 2006. Hermes.

[TJ99] P. Morel T. Jéron. Test generation derived from model-checking. *CAV'99, Trento, Italy*, LNCS 633- Springer-Verlag:108–122, July 1999.

[TJ06] V. RUSU. T. JERON, H. MARCHAND. Symbolic determinisation of extended automata. Technical Report Publication interne 1776, INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTEMES ALEATOIRES, JANUARY 2006.

[Tre96a] J. Tretmans. Test generation with inputs, outputs, and quiescence. Tools and Algorithms for the Construction and Analysis of systems, Springer, March 1996. lecture notes in computer science 1055 pages 127 (TACAS 96) march 1996.

[Tre96b] J. Tretmans. Test generation with inputs,outputs and repetitive quiescence. *In Software-Concepts and Tools,17(3)(1996)*, 1996. Also:Technical Report N0.96-26,Cen- ter for Telematics and Information Technology,University of Twente,The Nether- lands,pp.103 -120.

[Tre01] Jan Tretmans. An overview of OSI conformance testing. *Formal Methods Tools group University of Twente*, January 25, 2001.

[VD97] A Petit V Diekert, P Gastin. Removing e-transitions in timed automata. Proceeding of the 14th annual symposium on theoretical aspect of computer science (STACS'97) number 1200, 1997. lecture notes in computer science pages 583-594.

[VD04] Roland Groz Laurent Mounier Jean-Luc Richier Vianney Darmaillacq, Jean-Claude Fernandez. Eléments de modélisation pour le test de politiques de sécurité ;, 2004. Ce travail a été soutenu par le projet Protestat de l'ACI Sécurité et le projet IMAG- Modeste.

[Z5096] ITU. Z500. NORM: Formal methods in conformance testing,. Technical report, 1996.