VACSIM

Validation de la commande des systèmes critiques par couplage simulation et méthodes d'analyse formelles

Tâche 1

Validation par test progressif par parties de systèmes logiques

Livrable L1.1

Spécification et validation sur études de cas d'un algorithme de test progressif par parties de systèmes logiques non bouclés

Version C

Appel :	**PROGRAMME INGENIERIE NUMERIQUE & SECURITE 2011**
Numéro d'agrément :	**ANR-11-INSE-004**
Thématique:	**Systèmes embarqués et ingénierie du logiciel**
Objectif:	**Validation par test progressif par parties de systèmes logiques**
Date de démarrage du projet:	**01.10.2011**
Durée :	**42 mois**

Synthèse

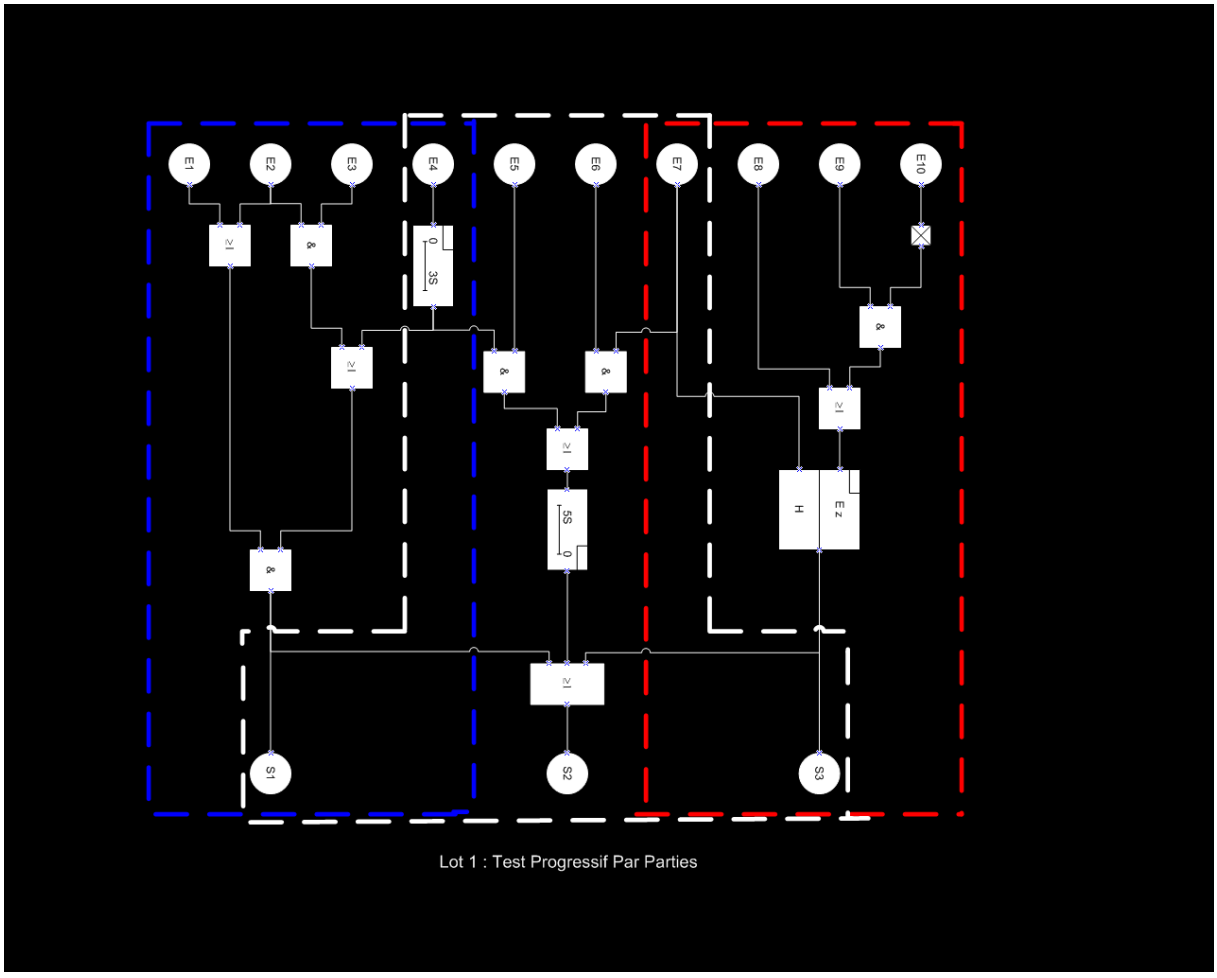
Le projet **VACSIM** (Validation de la commande des systèmes critiques par couplage simulation et méthodes d'analyse formelle), référencé ANR-11-INSE-004, étudie les avantages respectifs des techniques de simulation, en incluant des modèles des processus commandés, et des méthodes d'analyse formelles, pour la validation de la commande des systèmes critiques. Ce projet est structuré en 6 tâches.

La tâche 1 « Validation par test progressif par parties de systèmes logiques » vise à développer un prototype, intégré dans l'environnement ControlBuild (Dassault Systèmes), réalisant le test progressif par parties. Le projet a été l'occasion de formaliser un algorithme de génération de tests et de vérifications minimales qui utilise l'existence de sorties intermédiaires entre les sorties à tester et leurs entrées, autorisant un test progressif par parties. Le projet VACSIM sera l'occasion de vérifier le caractère réalisable et performant de ces nouvelles possibilités, qui correspondraient à la levée d'un verrou technique important pour le problème de la réduction de la combinatoire du test des fonctions logiques critiques non bouclées. Il sera aussi l'occasion de prototyper ces nouveaux algorithmes et de les mettre en œuvre sur des cas industriels.

Ce livrable L1.1 du projet VACSIM décrit une spécification et propose une validation sur études de cas d'un algorithme de test progressif par parties de systèmes logiques non bouclés.

Ce livrable L1.1 a été intégralement rédigé par EDF R&D.

Cette version C tient compte des évolutions des spécifications fonctionnelles suite à l'analyse des résultats de la version B précédente. Le livrable L1.3 du projet en version A correspond aux résultats obtenus à partir du développement qui utilise les spécifications du présent document.



Sommaire

SYNTHESE 2

1. INTRODUCTION 4

1.1. CADRE DE L'ETUDE 4
1.2. DEFINITION DU BESOIN 5
1.3. APPROCHE RETENUE POUR LE TEST D'APPLICATIONS A GRAND NOMBRE D'ENTREES 5
1.4. OBJECTIF DU DOCUMENT 5
1.5. LE PROTOTYPE TESTMINATOR 6

2. RAPPEL DES OBJECTIFS DE TEST DE TESTMINATOR POUR LES DIFFERENTS TYPES D'APPLICATIONS ET DES OBJECTIFS DE TEST DU TPPP 6

2.1. TYPES DE MEMOIRES ET DE TEMPORISATIONS 6
2.1.1. Temporisations 6
2.1.2. Mémoires 7
2.2. POUR LES APPLICATIONS PUREMENT COMBINATOIRES 8
2.3. POUR LES APPLICATIONS PUREMENT SEQUENTIELLES 9
2.4. POUR LES APPLICATIONS TEMPORISEES 9

3. SPECIFICATION DU TEST PARTIEL DE SYSTEMES N'APPARTENANT PAS A LA CLASSE DES SYSTEMES TESTES PAR TESTMINATOR 10

4. METHODOLOGIE PROPOSEE POUR LE TEST PROGRESSIF PAR PARTIES DE SYSTEMES LOGIQUES PROGRAMMES 12

4.1. AMELIORATIONS APPORTEES PAR LE TEST PROGRESSIF PAR PARTIES 12
4.2. DEFINITIONS 13
4.2.1. Entrées directes/indirectes 13
4.2.2. Sorties directes/indirectes 14
4.2.3. Mémoires directes/indirectes 15
4.2.4. Temporisations directes/indirectes 16
4.3. SPECIFICATION DES VERIFICATIONS EN BOITE BLANCHE POUR LE TPPP 16
4.4. ALGORITHME DE POSITIONNEMENT DU SYSTEME VERS UN ETAT STABILISE (BTP) 18
4.4.1. Objectifs 18
4.4.2. Algorithme BTP 19
4.4.3. Exemple 20
4.5. ALGORITHME DE GENERATION DES SEQUENCES DE TESTS (ATP) 26
4.5.1. Objectif et principe 26
4.5.2. Algorithme ATP 28
4.5.3. Exemple 29
4.6. OPTIMISATION DU PARCOURS DES SEQUENCES DE TEST 37
4.6.1. Algorithme TPPP optimisé 37
4.6.2. Exemple 39
4.7. OBSERVABILITE DES TEMPORISATIONS 40

5. CONCLUSION 41

6. DOCUMENTS DE REFERENCE 41

7. GLOSSAIRE 42

8. ANNEXES 43

8.1. ANNEXE 1 : EXEMPLE D'ETAT STABLE QUE TESTMINATOR NE PERMET PAS DE DETECTER 43
8.2. ANNEXE 2 : JUSTIFICATION DU CHOIX FAIT DE POSITIONNER E ET NON E_P DANS L'ALGORITHME ATP 45
8.3. ANNEXE 3 : OPTIMISATION DE L'ENCHAINEMENT DES VECTEURS DE TEST DES SEQUENCES DE TEST RISI…RSRS 46

1. Introduction

1.1. Cadre de l'étude

Testminator est un logiciel de génération automatique de tests pour des systèmes combinatoires, séquentiels et temporisés. Il est issu de travaux engagés à EDF R&D, à partir de 2002 dans le cadre du projet VD3 900, sur la génération de jeux de tests avec une combinatoire limitée par des vérifications minimales sur la réalisation. Ces travaux s'intéressaient plus particulièrement à la preuve de la bonne réalisation d'un système programmé au regard de ses spécifications. L'ambition de cette étude exploratoire amont était de proposer une méthode qui pouvait générer et exécuter tous les scénarios de test présentant un intérêt dans la preuve de la bonne programmation d'un système[1].

L'approche retenue complète un certain nombre de vérifications sur le programme par l'exécution de jeux de tests, soit sur un système simulé, soit sur le système réel, par excitation des entrées et observation des sorties. L'approche proposée combine l'étude du système en « boîte blanche » et le test du système en « boîte noire ». L'étude du système en boîte blanche vise à vérifier un certain nombre de propriétés de la réalisation du système au regard de ses spécifications fonctionnelles, afin de rendre le test significatif et de réduire le problème combinatoire du test. Il s'agit de pouvoir n'exécuter, dans la preuve de la bonne programmation du système, qu'un sous-ensemble de vecteurs d'entrée possibles pour le test du système en boîte noire.

Ces réflexions ont conduit à la réalisation de Testminator qui traite le cas des systèmes combinatoires, séquentiels et temporisés non bouclés. Testminator ne traite pas le test des systèmes temporisés avec le même niveau de confiance que pour des systèmes combinatoires et séquentiels. Les tests des systèmes temporisés, en effet, ne bénéficiaient pas, contrairement aux systèmes combinatoires et séquentiels, d'une théorie aboutie au moment du développement de Testminator. Le projet ANR TESTEC (TLOG07-022) a été l'occasion de travailler sur ce manque de théorie. Le temps étant une grandeur analogique, il est aussi moins évident de garantir totalement la bonne programmation d'un système avec un nombre limité de tests.

La première version de Testminator réalise le test d'un système temporisé en sollicitant chaque temporisation suivant plusieurs combinaisons des entrées et des états internes dont elle dépend. Dans ce test, on limite le nombre de scénarios, dans le parcours des états des mémoires et autres temporisations du système, aux valeurs des entrées et des mémoires directes de la temporisation testée, sans mettre en œuvre la combinatoire de l'ensemble des entrées et des mémoires dont la temporisation testée dépend. L'objectif de test est restreint aux mémoires et aux entrées directes dont dépend la temporisation testée, ces mémoires / sorties étant testées par ailleurs. Il s'agit du seul cas de ce premier développement où on utilise l'existence de sorties intermédiaires entre la fonction à tester et les entrées dont elle dépend. La suite du document propose une généralisation de ce test progressif par parties.

Les techniques du premier développement Testminator sont utilisables pour des systèmes logiques critiques qui peuvent réaliser plusieurs milliers de fonctions simples, en utilisant typiquement de quelques entrées à une quinzaine d'entrées. Au-delà d'une quinzaine d'entrées, par exemple pour des fonctions logiques complexes qui utiliseraient plus de quarante ou cinquante entrées, les techniques prototypées dans le premier développement ne permettent pas une réduction du nombre de scénarios d'essais à un niveau acceptable.

Ce document propose un algorithme de génération de tests et de vérifications minimales qui utilise, en plus des techniques maintenant maîtrisées, l'existence de sorties intermédiaires entre les sorties à tester et leurs entrées, autorisant un test progressif par parties.

[1] La méthode ne s'appuie donc pas sur des critères de taux de couverture de test de la spécification, qui sont plus classiques en génération automatique de test, mais s'inspire plus des bonnes pratiques manuelles de V&V des systèmes programmés critiques.

1.2. Définition du besoin

Le test d'un système logique purement combinatoire en « boîte noire » ne peut être considéré comme complet que dans la mesure où tous les $2^{\text{nombre d'entrées}}$ vecteurs sont exécutés, ce qui devient vite impraticable si le nombre d'entrées du système devient important. Pour un système séquentiel, le test exhaustif en « boîte noire » est encore plus contraignant, car il nécessite d'exécuter $2^{\text{nombre d'entrées + nombre d'états internes}}$ vecteurs.

La preuve de la bonne programmation d'un système logique au regard de ses spécifications fonctionnelles commence par une vérification en « boîte blanche » des hypothèses sur le type de système à tester, entre le système et ses spécifications.

Dans Testminator, une fois ces vérifications en « boîte blanche » réalisées, la combinatoire des vecteurs de test se limite aux seules entrées dont dépendent les sorties et les mémoires. Ceci permet de réduire le jeu de tests tout en garantissant un test complet. Cependant, des considérations pratiques (temps d'exécution, occupation mémoire…) restreignent la classe des systèmes auxquels cette approche peut s'appliquer. Elle est en effet bien adaptée pour des logiques de sécurité spécifiées dans des Diagrammes Fonctionnels Logiques (DFL) EDF, qui ont généralement de faibles dépendances en nombre entre les entrées et les sorties et dont le comportement séquentiel est limité à la prise en compte d'ordres fugitifs et de mémorisation d'ordres de protection.

L'expérience a montré que Testminator est particulièrement adapté pour le test de nombreuses applications dont les sorties dépendent au plus d'une quinzaine d'entrées. Cependant, dans le cas de systèmes plus conséquents pour lesquels une sortie dépend d'un grand nombre d'entrées, Testminator, dans sa première version, ne parvient pas à maîtriser l'explosion combinatoire.

1.3. Approche retenue pour le test d'applications à grand nombre d'entrées[2]

L'objectif de l'intégration du Test Progressif Par Parties (TPPP) est de diminuer le nombre de tests nécessaires pour couvrir le même objectif de test que celui de la première version de l'outil Testminator. L'approche utilise l'existence d'autres sorties (observables) entre la sortie à tester et les entrées dont elle dépend pour réduire le nombre de tests à utiliser. Ces sorties que l'on qualifie dans la suite de « sorties intermédiaires » sont elles-mêmes testées par ailleurs. La combinatoire du test d'une sortie peut donc être réduite aux valeurs des entrées qui lui sont directement reliées et aux valeurs de ces sorties intermédiaires. L'approche consiste donc à réaliser un test progressif des applications, permettant d'abord de vérifier, par exemple, toutes les conditions d'un critère instrumenté d'activation d'une protection, puis de considérer cette logique comme validée, pour continuer les tests de la suite de la chaîne mêlant plusieurs autres critères, en limitant la combinatoire à l'activation d'une seule des conditions de chaque critère. Cette approche est mise en œuvre manuellement pour le test des protections des réacteurs. L'objectif de ce document est de la formaliser et de spécifier un algorithme pour automatiser la démarche.

1.4. Objectif du document

Ce document a un double objectif.

Premièrement, il s'agit d'identifier les optimisations à apporter à Testminator pour ne pas rejeter en bloc des spécifications comportant des boucles ou des mémoires non instrumentées (ce que fait le premier prototype), mais pour tester et vérifier les sorties non bouclées testables, en indiquant par un message la liste des sorties non testées.

Deuxièmement, il s'agit de spécifier les modifications à apporter aux algorithmes de l'outil Testminator pour y intégrer le test progressif par parties. Ce document explicite également certains choix qui permettent l'optimisation du test progressif par parties.

[2] Sera considérée comme application à grand nombre d'entrées, une application dont une sortie dépend d'au moins 20 entrées.

1.5. Le prototype Testminator

La première version Testminator dispose d'un module d'extraction des fichiers DFL au format Visio dans un format texte développé en Visual Basic, d'un prototype (boîte blanche + boîte noire) développé en C++ et de toute la documentation associée à son développement (cahier des charges [3] [6], dossier de spécification [4], dossier de conception [5]). L'intégration de ce prototype dans un environnement ControlBuild a été travaillée par la société Dassault Systèmes à l'occasion du projet ANR TESTEC (TLOG07-022).

Une maquette permettant d'exécuter le plan de test obtenu par le prototype Testminator sur une cible de type automate Premium Schneider a aussi été réalisée par EDF R&D. Cette maquette a permis de tester complètement les logiques d'un système CRF d'une centrale nucléaire REP 900 MWe.

Les modifications apportées à ce premier prototype, qui sont spécifiées dans ce document, feront l'objet d'un développement par Dassault Systèmes dans l'environnement ControlBuild (livrable L1.2). Ces techniques seront ensuite évaluées sur études de cas industriels (livrable L1.3).

2. Rappel des objectifs de test de Testminator pour les différents types d'applications et des objectifs de test du TPPP

Les systèmes testés par Testminator doivent être non bouclés et comporter exclusivement des opérateurs logiques (ET, OU et NON), des mémoires (à EN prioritaire ou à HORS prioritaire), des temporisations (à l'excitation ou à la désexcitation) et des seuils sur des grandeurs analogiques. Testminator vérifie en boîte blanche que le système soumis au test respecte bien ces contraintes.

2.1. Types de mémoires et de temporisations

2.1.1. Temporisations

La norme internationale CEI 61131-3 [7] définit deux blocs de base pour les temporisations.

Temporisation à l'excitation

La temporisation à l'excitation est notée TON et son chronogramme est le suivant (l'entrée est repérée IN, la sortie Q) :

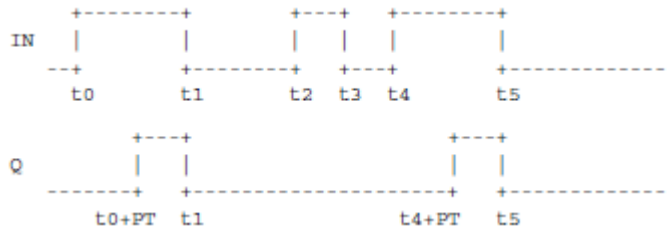
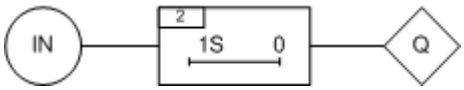


Figure 1 : Chronogramme d'une temporisation à l'excitation

Dans la suite et lors de l'édition des spécifications d'un système logique, une temporisation à l'excitation est représentée graphiquement de la façon suivante (le délai de la temporisation n°2 est d'une seconde) :



Une temporisation à l'excitation est dite activée (ACTIVE) quand sa sortie vaut 1. Elle est dite désactivée (DESACTIVE) quand son entrée vaut 0. Quand son entrée vaut 1 et sa sortie vaut 0 (la temporisation n'est pas encore arrivée à échéance), son état est dit transitoire (TRANSITOIRE).

Temporisation à la désexcitation

La temporisation à la désexcitation est notée TOF et son chronogramme est le suivant (l'entrée est repérée IN, la sortie Q) :

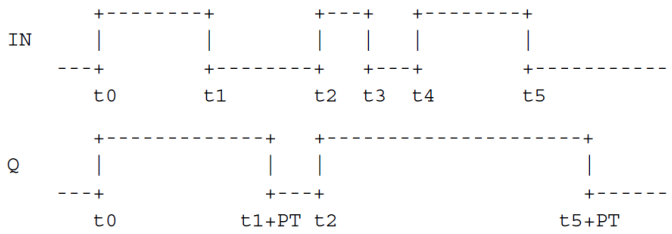
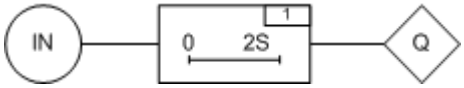


Figure 2 : Chronogramme d'une temporisation à la désexcitation

Dans la suite et lors de l'édition des spécifications d'un système logique, une temporisation à la désexcitation est représentée graphiquement de la façon suivante (le délai de la temporisation n°1 est de deux secondes) :



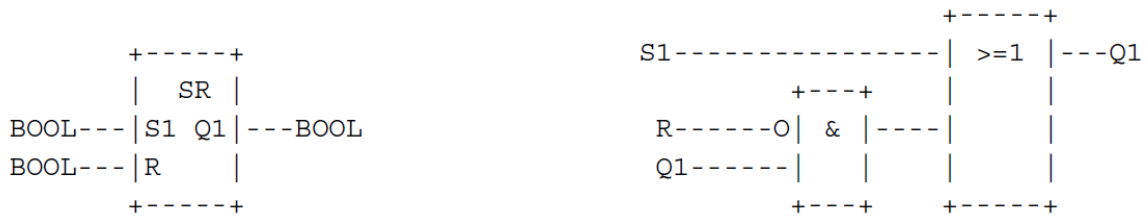
Une temporisation à la désexcitation est dite activée (ACTIVE) quand sa sortie vaut 0. Elle est dite désactivée (DESACTIVE) quand son entrée vaut 1. Quand son entrée vaut 0 et sa sortie vaut 1 (la temporisation n'est pas encore arrivée à échéance), son état est dit transitoire (TRANSITOIRE).

2.1.2. Mémoires

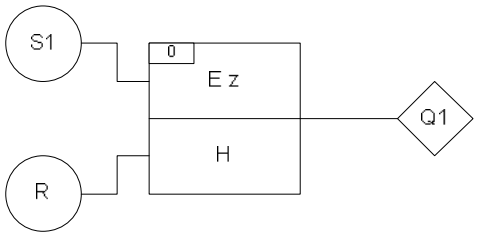
La norme internationale [7] définit deux blocs de base pour les mémoires bistables.

Mémoire bistable à enclenchement (EN) prioritaire

La mémoire à EN prioritaire (set) est appelée Sr et est définie par la fonction logique suivante :

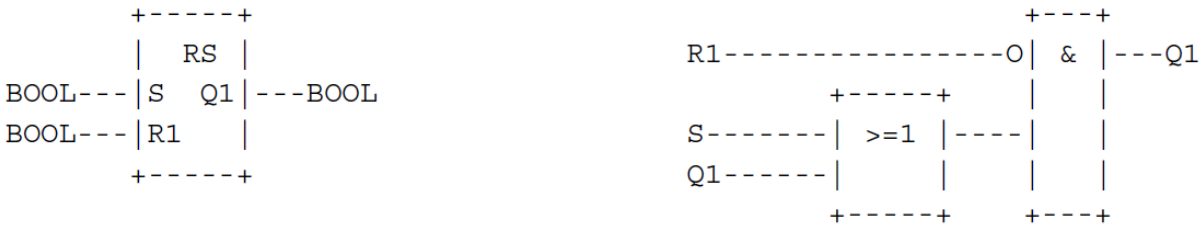


Dans la suite et lors de l'édition des spécifications d'un système logique, une mémoire bistable à EN prioritaire Sr est représentée graphiquement de la façon suivante (le Z indique la priorité à l'enclenchement - E) :

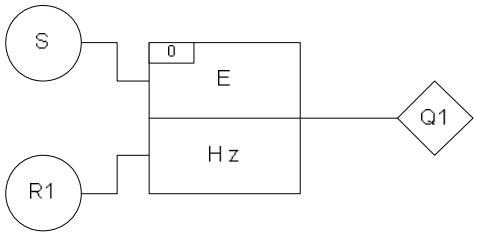


Mémoire bistable à déclenchement (HORS) prioritaire

La mémoire à HORS prioritaire (reset) est appelée Rs et est définie par la fonction logique suivante :



Dans la suite et lors de l'édition des spécifications d'un système logique, une mémoire bistable à HORS prioritaire Rs est représentée graphiquement de la façon suivante (le Z indique la priorité au déclenchement - H) :



2.2. Pour les applications purement combinatoires[3]

Une application purement combinatoire est une application comprenant exclusivement des ET, des OU et des NON logiques. Les variables d'entrée sont logiques. L'application peut néanmoins comporter des seuils sur des grandeurs d'entrée analogiques.

L'objectif de test Testminator d'une sortie d'une application purement combinatoire est son test par l'ensemble des combinaisons possibles des valeurs des entrées dont elle dépend.

Un vecteur d'entrée peut mettre la valeur de la sortie à un (vecteur S comme set) ou à zéro (vecteur R comme reset). Les vecteurs de test sont joués en séquences de type RSRS…RSRS afin de tester les variations de valeurs de la sortie.

Les entrées de l'application dont ne dépend pas une sortie ne sont pas à prendre en compte dans son test, car une vérification est faite préalablement sur l'application, lors des contrôles en boîte blanche, pour s'assurer des dépendances entre les sorties et les entrées du type $S = f(e_i)$.

Cependant, pour des sorties totalement indépendantes de la sortie testée, il ne coûte presque rien de vérifier, lors du test de la réalisation, qu'elles ne changent pas de valeur lors du test[4]. Pour réaliser cette vérification redondante, le nom de la variable à tester est suivi, dans le fichier de scénario de test, du nom des sorties dont la valeur ne doit pas varier lors du test.

Une optimisation par classes d'équivalence permet de tester plusieurs sorties purement combinatoires en même temps [6].

[3] Applications sans mémoires, ni temporisations

[4] Pour une sortie partageant au moins une entrée avec la sortie testée, on ne peut rien vérifier.

2.3. Pour les applications purement séquentielles[5]

Une application purement séquentielle est une application comprenant exclusivement des ET, des OU, des NON logiques, des mémoires à En Prioritaire (EP) et des mémoires à Hors Prioritaire (HP). Les variables d'entrée sont logiques. L'application peut néanmoins comporter des seuils sur des grandeurs d'entrée analogiques.

La sortie de chaque mémoire (EP ou HP) de l'application correspond à une sortie de l'application par hypothèse (chaque mémoire doit être instrumentée, en ajoutant si besoin des sorties au schéma à tester). Cette propriété est vérifiée par l'outil.

L'objectif de test d'une sortie de type mémoire, qui peut comporter d'autres fonctions séquentielles dans les calculs des entrées EN et HORS, est de pouvoir solliciter effectivement cette mémoire pour tous les vecteurs d'entrée permettant de la mettre à 1 (vecteur S comme Set), de la mettre à 0 (R comme Reset) ou de laisser sa valeur inchangée (vecteur I comme inchangée, les deux entrées EN et HORS de la mémoire sont à 0), dans tous les cas possibles des états internes intervenant dans son calcul.

Les vecteurs sont joués en séquences de type RISIRIS..RSRS dans le premier développement. Le document référencé [6] du projet ANR TESTEC contient le détail des algorithmes qui utilisent des vecteurs, R, S et I pour leur organisation en séquence. Par rapport à ce premier développement du prototype Testminator, la seule évolution proposée ici dans l'organisation de ces séquences est de regrouper tous les vecteurs de type I, pour utiliser des séquences de type RI..ISI..IRSRSRS, qui pourraient être moins longues en temps d'exécution.

Dans le premier développement [6], tous les vecteurs d'entrée n'étaient pas utilisés à partir de l'ensemble des états atteignables du système lors de son test. Une propriété (prouvée) permettait de diminuer le temps de test en n'utilisant que des vecteurs laissant inchangé un état interne en amont de la mémoire testée. Dans le nouveau développement, l'objectif de test étant réduit à une évolution d'états partiels, cette possibilité n'est dans un premier temps non retenue, afin de ne pas trop complexifier les algorithmes.

Les vecteurs étaient aussi regroupés pour minimiser le nombre de ces états internes à positionner. Ce document contient une nouvelle spécification de ce traitement.

2.4. Pour les applications temporisées

Une application temporisée est une application comprenant exclusivement des ET, des OU, des NON logiques, des mémoires à En Prioritaire (EP) et des mémoires à Hors Prioritaire (HP), des temporisations à l'excitation (TON), des temporisations à la désexcitation (TOF). Les variables d'entrée sont logiques. L'application peut néanmoins comporter des seuils sur des grandeurs d'entrée analogiques.

L'objectif de test d'un système temporisé du premier développement était de solliciter chaque temporisation suivant plusieurs combinaisons des entrées, des états internes et des autres temporisations dont elle dépendait. Dans ce test, on limite le nombre de scénarios, dans le parcours des états des mémoires et autres temporisations du système, aux valeurs des entrées et mémoires directes de la temporisation testée, sans mettre en œuvre la combinatoire de l'ensemble des entrées et des mémoires dont la temporisation testée dépend.

L'objectif de test est restreint aux mémoires et entrées directes dont dépend la temporisation testée, ces mémoires / sorties étant testées par ailleurs. Il s'agit du seul cas du premier développement où on utilise l'existence de sorties intermédiaires entre la fonction à tester et les entrées dont elle dépend. La suite du document propose une généralisation de ce test progressif par parties.

Le premier prototype Testminator diminue les temps de test par un test du système dans une version non temporisée (paramètres des temporisations à zéro), qui est complété par un test temporisé avec

[5] sans temporisation

l'objectif de test décrit précédemment. Pour certains schémas fonctionnels qui comportent des impulsions, cette approche peut limiter l'intérêt du test dans sa version non temporisée (une impulsion de durée nulle « efface » les spécifications qui la génèrent). Le premier prototype reconnaît automatiquement ces cas de figure et complète en conséquence les tests temporisés pour atteindre un niveau de confiance équivalent.

Pour le nouveau prototype, cette approche du test d'une temporisation particulière qui complète le test d'une version non temporisée du système n'est pas retenue. Le nouveau système est toujours testé globalement en observant s'il y a lieu les évolutions de ses états internes et de ses temporisations.

Le temps est un paramètre analogique, ce qui signifie qu'il faudrait en théorie une infinité de scénarios d'essai pour tester « complètement » un système temporisé. Pour les systèmes combinatoires et séquentiels, une preuve de la suffisance des tests du premier prototype, au vu des vérifications réalisées, a été formulée [6]. L'objectif d'une telle preuve est difficilement atteignable pour un système temporisé. L'objectif d'un test « complet » d'un système temporisé a été abandonné.

Dans le premier développement, le système est positionné dans tous les états internes des mémoires et des temporisations qui permettent d'activer une temporisation à tester, puis un vecteur d'entrée sert au test de la temporisation. Dans ce test, les états internes des mémoires et temporisations du système peuvent évoluer et la stratégie de test consiste à observer la conformité de cette évolution aux spécifications, sans faire varier les valeurs des entrées tant que toutes les temporisations ne sont pas arrivées à échéance.

La variation simultanée des entrées et des états internes, sans attendre l'échéance des temporisations, conduirait à un nombre de test trop important. Pour dix entrées, un système qui évolue et se stabilise en dix secondes et un temps de cycle de rafraîchissement des entrées du test de 100 millisecondes, on aurait 2^{10} possibilités de valeurs d'entrées par cycle, soit 2^{10} choix au premier cycle, 2^{20} au second, 2^{30} au troisième… sur cent cycles (ce qui conduit à un total de tests inconséquent de 2^{1000}).

La stratégie du changement des valeurs des entrées uniquement dans des états stables du système testé est de ce fait retenue pour les évolutions proposées dans ce qui suit. Par contre, cette stratégie conduit à des limites théoriques du test et certains schémas comportant des impulsions concourantes et décalées dans le temps peuvent être difficile à tester (le premier prototype avertit toutefois l'utilisateur de difficultés potentielles de ce type).

L'annexe 1 du chapitre 8.1 est un exemple de schéma pour lequel des combinaisons de vecteurs d'entrée mènent à des états stables non atteignables par Testminator au vu de la stratégie de test des systèmes temporisés retenue. Le lecteur remarquera qu'il s'agit aussi de cas un peu complexes qui seront difficiles à tester manuellement de façon classique.

3. Spécification du test partiel de systèmes n'appartenant pas à la classe des systèmes testés par Testminator

La première version de Testminator est destinée au test de conformité de systèmes combinatoires, séquentiels et temporisés non bouclés. Si, lors de la vérification en boîte blanche du système l'outil détecte une boucle[6] ou des blocs interdits par hypothèse[7] (type opérande sur des grandeurs analogiques, régulation…) dans le système, il émet une erreur bloquante informant l'utilisateur de la non-testabilité du système par l'outil.

Plusieurs réflexions ont permis d'identifier une optimisation pour le test de ces systèmes. Il s'agit d'appliquer la vérification d'absence de boucle du système (ou d'absence de blocs interdits), non plus

[6] Rappel de [3] [6] §3.1.2 Hypothèse n° 5 : Les spécifications ne font pas apparaître de boucle logique pure.

[7] Rappel de [3] [6] §3.1.2 Hypothèse n° 2 : Les spécifications des traitements logiques ne font appel qu'à un jeu réduit d'instructions : fonctions logiques « et », « ou », « non », mémoires à enclenchement (set) prioritaire (Sr) et à déclenchement (reset) prioritaire (Rs) ; temporisations à l'excitation (TON) et à la désexcitation (TOF).

globalement à tout le système, mais individuellement à chacune des sorties. En effet, si une sortie de l'application est bouclée (si elle dépend d'entrées par l'intermédiaire d'une ou plusieurs boucles), rien n'empêche de générer des tests pour les autres sorties non bouclées de l'application. La même remarque est valable pour une sortie qui dépend d'entrées par l'intermédiaire de blocs non autorisés par hypothèse.

Le schéma Visio ci-dessous illustre cette modification[8]. Le système est effectivement bouclé. La modification consiste à isoler la partie bouclée du système (en rouge) et à générer les tests pour la partie non bouclée du système (en vert).

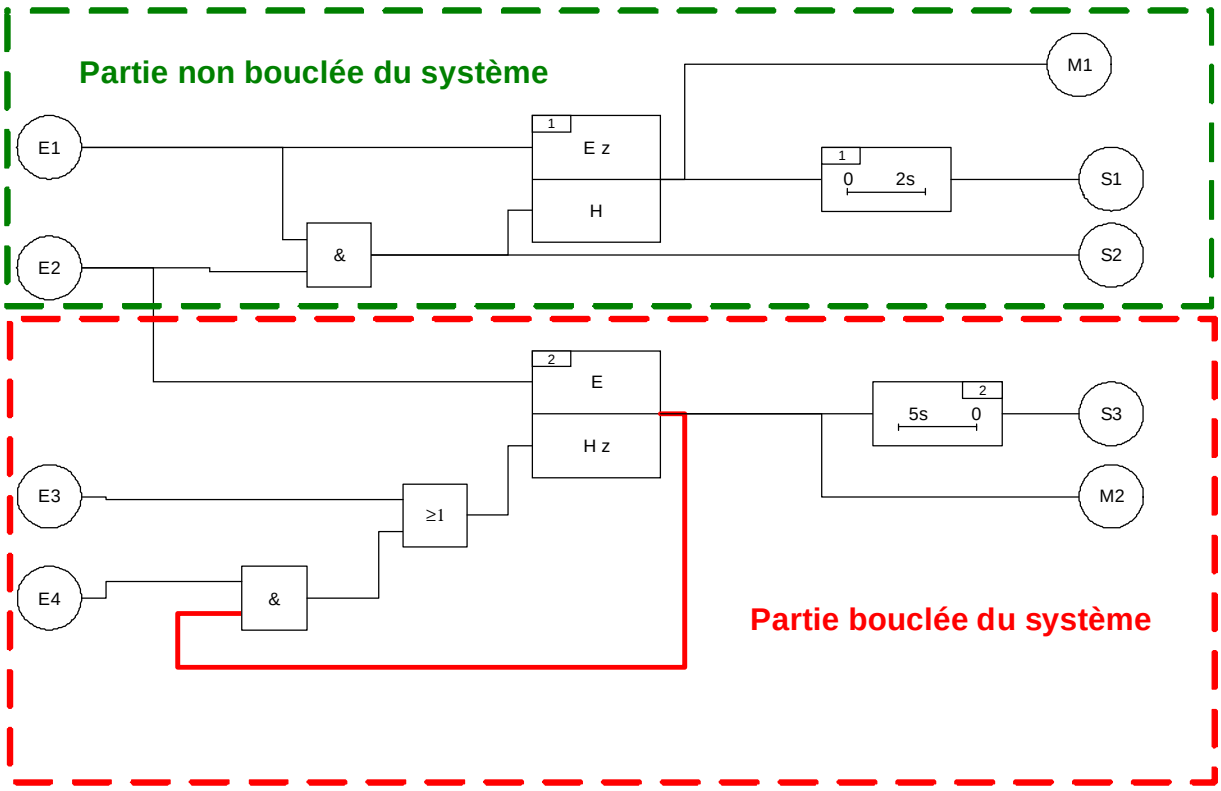


Figure 3 : Exemple de l'isolement de la partie bouclée d'un système bouclé

L'outil générera alors un message informant l'utilisateur qu'une partie du système est bouclée et que, de ce fait, les tests devront être générés classiquement à la main pour cette partie, tandis que la partie non bouclée du système peut être testée automatiquement par l'outil.

La détection des boucles est réalisée sortie par sortie. Dans le schéma précédent, on part des sorties M1, S1 puis S2, pour remonter jusqu'aux entrées sans avoir à repasser par le même bloc, donc ce sont des sorties sans boucle. Pour les sorties S3 puis M2, le même type de parcours conduit à revenir à la mémoire 2, donc il y une boucle. Si on modifie le schéma pour utiliser la sortie de la mémoire 2 en entrée du « ET » de la partie verte, par exemple, on crée une boucle pour toutes les sorties et il n'y aurait plus de sortie sans boucle.

La présence de boucles est détectée lors des vérifications en boîte blanche ([6] vérification n°4). Cette vérification sera donc modifiée en conséquence. Désormais, « si une boucle ou un bloc non autorisé existe dans le calcul des variables mémorisées, la partie du circuit bouclée ou faisant intervenir des

[8] L'exemple est illustratif, en réalité une boucle peut être moins évidente quand les spécifications utilisent plusieurs pages.

blocs non autorisés est isolée et déclarée non testable par l'outil Testminator, la partie restante si elle existe est testée par l'outil ». Testminator générera un message informant le testeur que certaines sorties n'ont pu être testées et qu'elles devront l'être manuellement.

4. Méthodologie proposée pour le test progressif par parties de systèmes logiques programmés

4.1. Améliorations apportées par le test progressif par parties

L'objectif de l'intégration du test progressif par parties (TPPP) est de diminuer le nombre de tests nécessaires pour couvrir l'objectif de test assuré par Testminator : ne pas avoir d'intérêt à ajouter de séquences de test au vu des vérifications réalisées (test complet). L'évolution, par rapport au premier développement, concerne les applications dont une sortie au moins dépend d'une ou de plusieurs autres sorties. Ces sorties que l'on peut qualifier « d'intermédiaires » seront testées par Testminator par ailleurs. La sortie qui en dépend peut donc être testée en fonction d'un objectif de test des valeurs de ces sorties intermédiaires et non plus d'un critère de couverture de l'ensemble des valeurs des entrées.

Exemple :

Considérons un système purement combinatoire.

Soit S une sortie de ce système dépendant de 10 entrées et S1 et S2 deux sorties, dont S dépend, qui dépendent respectivement de A1, A2, A3 et A8, A9, A10.

S = f (A1, A2, …, A10) S1 = f1 (A1, A2, A3) S2 = f2 (A8, A9, A10)

On ne peut pas réduire l'objectif de test de S avec une couverture des « entrées » S1, A4, A5, A6, A7, S2 car A1 intervient aussi directement dans le calcul de S.

Dans ce cas :

Les vérifications minimales seront faites sur S pour prouver S = f (A1, S1, A4, A5, A6, A7, S2) & S1 = f1 (A1, A2, A3) & S2 = f2 (A8, A9, A10)

La couverture des « entrées » dans le test de S sera : A1, S1, A4, A5, A6, A7, S2.

Cette technique correspond à tester le système sortie par sortie, de la sortie la plus en amont du système à la sortie la plus en aval. Elle s'appliquera pour des systèmes pour lesquels une sortie dépend d'un nombre d'entrées important (15 entrées et plus). Dans ce cas, l'outil utilisera le TPPP pour tester cette sortie. Si une sortie a moins de 15 entrées, les tests pourront être générés par les algorithmes du premier développement sur demande de l'utilisateur. Une autre option devrait permettre de générer des tests en utilisant le TPPP pour l'ensemble des sorties.

Le Test Progressif Par Parties sera intéressant, donc automatiquement choisi, pour des sorties qui dépendent de plus de 15 entrées.

Le TPPP est d'un niveau de complexité au moins équivalent aux premiers algorithmes du premier prototype Testminator. Son intégration s'accompagne de l'optimisation décrite au §3.

On vise, par l'utilisation du TPPP, une réduction drastique du nombre de variables de test retenues. Afin de ne pas ajouter à la complexité de cet algorithme, on renonce à l'utilisation d'autres techniques de réduction du problème (par exemple : répartition des entrées en classes d'équivalence en combinatoire, tests d'une mémoire en laissant inchangés les états des mémoires en amont en séquentiel, séparation du test d'un système avec mise à zéro de toutes les temporisations du test

avec les temporisations à leur valeur réelle).

4.2. Définitions

4.2.1. Entrées directes/indirectes

Parmi les entrées dont dépend une sortie ***S***, on peut distinguer les entrées indirectes des entrées directes.

On appellera **entrée indirecte** e_i une entrée dont dépend la sortie ***S*** via au moins une sortie intermédiaire.

Toute autre entrée dont dépend cette sortie ***S*** sera appelée **entrée directe** e_d. En particulier, une entrée dont dépend la sortie ***S*** qui est à la fois directe et indirecte est considérée comme entrée directe.

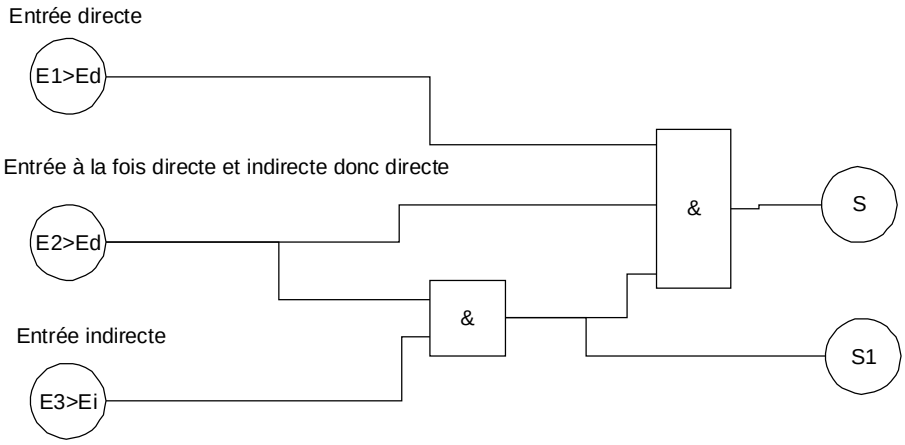


Figure 4 : entrée directe et entrée indirecte dans le TPPP

Sur cet exemple, on constate que :

- S dépend directement de E1, E1 sera donc considérée comme une entrée directe pour le test de S.
- S dépend à la fois directement et indirectement (par l'intermédiaire de la sortie S1) de E2, E2 sera donc considérée comme une entrée directe pour le test de S.
- S dépend indirectement de E3 (par l'intermédiaire de la sortie S1), E3 sera donc considérée comme une entrée indirecte pour le test de S.

On appelle **sortie intermédiaire** vis-à-vis d'une sortie ***S*,** une sortie dont dépend cette sortie ***S***.

4.2.2. Sorties directes/indirectes

Parmi les sorties intermédiaires dont dépend une sortie ***S***, on peut distinguer les sorties intermédiaires indirectes des sorties intermédiaires directes.

On appellera **sortie intermédiaire indirecte S_i** une sortie dont dépend la sortie ***S*** uniquement via une autre sortie intermédiaire.

Toute autre sortie intermédiaire dont dépend cette sortie ***S*** sera appelée **sortie intermédiaire directe S_d**. En particulier, une sortie intermédiaire dont dépend directement la sortie ***S*** mais aussi via une autre sortie intermédiaire sera une sortie intermédiaire directe.

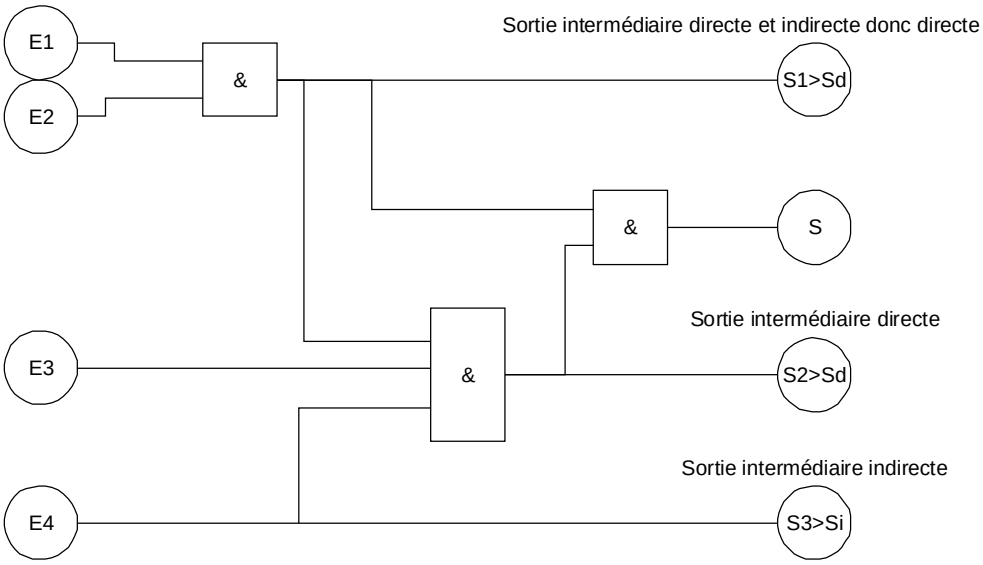


Figure 5 : sortie intermédiaire introduit dans le TPPP.

Sur cet exemple, on constate que :

- S dépend à la fois directement et indirectement de S1, S1 sera donc considérée comme une sortie intermédiaire directe pour le test de S.
- S dépend directement de S2, S2 sera donc considérée comme une sortie intermédiaire directe pour le test de S.
- S dépend indirectement de S3 (par l'intermédiaire de S2), S3 sera donc considérée comme une sortie intermédiaire indirecte pour le test de S.

4.2.3. Mémoires directes/indirectes

La différence entre les sorties d'instrumentation des mémoires et les sorties précédemment évoquées tient à leur situation particulière (systématiquement en sortie d'une mémoire). Elles peuvent être considérées comme des sorties particulières. C'est pourquoi tout comme pour les « sorties intermédiaires » dont dépend une sortie ***S*** à tester, on distinguera des « mémoires directes » ; M_d, et des « mémoires indirectes » ; M_i.

On appelle **mémoire indirecte** M_i, une mémoire dont dépend la sortie ***S*** via une autre sortie intermédiaire et **mémoire directe** M_d, une mémoire dont dépend directement la sortie ***S*** sans l'intermédiaire d'une autre sortie (sortie d'instrumentation ou sortie intermédiaire telles que définies plus haut). Tout comme pour les « sorties intermédiaires », notons qu'une mémoire à la fois directe et indirecte en amont de ***S*** est à considérer comme directe.

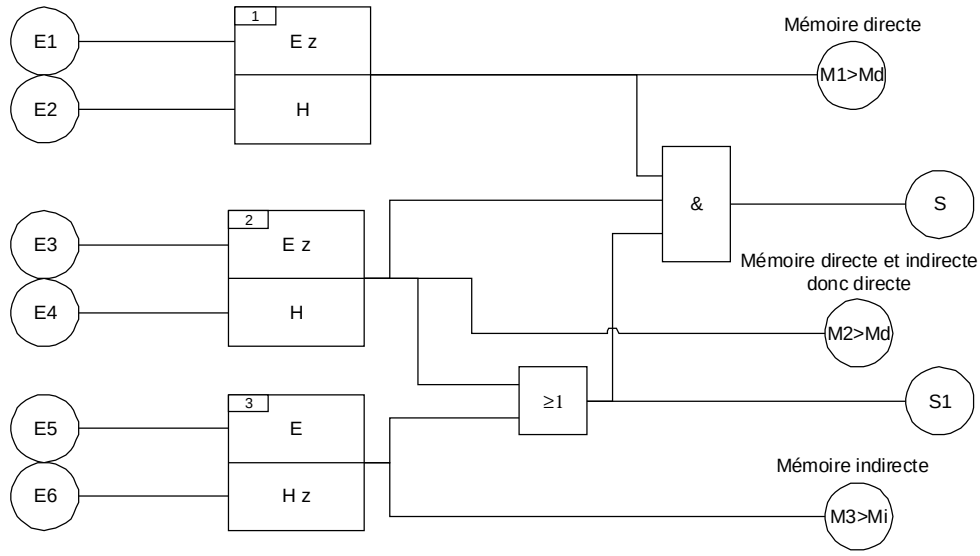


Figure 6 : mémoire directe et mémoire indirecte introduit dans le TPPP

Sur cet exemple, on constate que :

- S dépend directement de M1, M1 sera donc considérée comme une mémoire directe pour le test de S.
- S dépend à la fois directement et indirectement (par l'intermédiaire de S1) de M2, M2 sera donc considérée comme une mémoire directe pour le test de S.
- S dépend indirectement de M3 (par l'intermédiaire de S1), M3 sera donc considérée comme une mémoire indirecte pour le test de S.

4.2.4. Temporisations directes/indirectes

Parmi les temporisations dont dépend une sortie ***S***, on peut distinguer les temporisations indirectes des temporisations directes.

On appellera **temporisation indirecte T_i** une temporisation dont dépend la sortie ***S*** via au moins une sortie intermédiaire ou une mémoire.

Toute autre temporisation ***S*** sera appelée **temporisation directe T_d**. En particulier une temporisation dont dépend directement la sortie ***S*** mais aussi via au moins une sortie intermédiaire (ou une mémoire) sera considérée comme une temporisation directe.

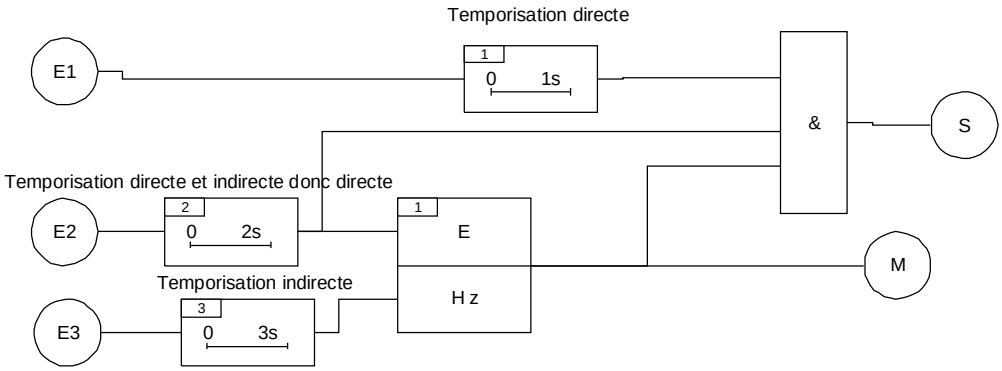


Figure 7 : temporisation directe et temporisation indirecte introduit dans le TPPP

Sur cet exemple, on constate que :

- S dépend directement de T1, T1 sera donc considérée comme une temporisation directe pour le test de S.
- S dépend à la fois directement et indirectement (par l'intermédiaire de M) de T2, T2 sera donc considérée comme une temporisation directe pour le test de S.
- S dépend indirectement de T3 (par l'intermédiaire de M), T3 sera donc considérée comme une temporisation indirecte pour le test de S.

4.3. Spécification des vérifications en boîte blanche pour le TPPP

Dans le premier développement (version B/BT, A/AT des algorithmes), un certain nombre de contrôles en boîte blanche sont réalisés afin de s'assurer que la méthode de génération de jeux de tests est applicable. Tous ces contrôles sont réalisés automatiquement par l'outil.

Le TPPP prévoit la modification des conséquences d'une des vérifications de la première version de Testminator comme évoqué au §3 (vérification n°4).

Ces six vérifications réalisées sortie par sortie dans le premier développement sont :

Vérification n°1 : La présence de variables du type mémoire et de temporisations est conforme entre la spécification et le code du système [E110] ;

Vérification n°2 : Les dépendances entre les entrées, les sorties et les variables mémorisées du système sont conformes à celles de la spécification [E120] ;

Vérification n°3 : Chaque variable mémorisée du système et de la spécification est aussi une sortie [E130] ;

Vérification n°4 : Il n'existe pas de boucle dans le calcul des variables mémorisées [E140] ;

Vérification n°5 : Les dépendances entre les entrées du système et les entrées des blocs temporisation sont conformes à la spécification [E150] ;

Vérification n°6 : Chaque sortie (donc en particulier les variables internes mémorisées du système) est reliée aux mêmes entrées par l'intermédiaire de temporisations du même type sur la spécification et sur le système programmé [E160] ;

On propose de faire l'ensemble des vérifications nécessaires au test progressif par parties en une seule passe. Pour le TPPP, il est nécessaire de vérifier des relations de dépendances entrées / sorties qui ne sont pas vérifiées dans le premier développement (c'est-à-dire des dépendances entre sorties, mais également des dépendances entre des sorties et des mémoires par l'intermédiaire de temporisations). On profite du fait qu'à chaque mémoire est associée une sortie. La vérification du premier développement parcourt depuis une sortie les spécifications et le code jusqu'à toutes les entrées dont elle dépend, sans examen des sorties intermédiaires. Ici on introduit des vérifications plus complexes en ne parcourant pas jusqu' à l'ensemble des entrées dont dépend une sortie mais en se limitant aux entrées directes et sorties directes.

La vérification introduite dans le TPPP est :

Vérification n°7 : Chaque sortie (donc en particulier les variables internes mémorisées du système) est reliée aux mêmes entrées par l'intermédiaire de sorties du même type sur la spécification et sur le système programmé, via des temporisations similaires [E170]

Sur l'application, on vérifiera donc pour le TPPP : **S** = f (***M*****$_d$**, ***T*****$_d$** (***M*****$_d$**, ***T*****$_d$** , ***e*****$_d$**, ***S*****$_d$**), ***e*****$_d$**, ***S*****$_d$**).

(la notation est simplifiée car si S est une mémoire on vérifiera S = f (M (,)) et la vérification est récursive pour chaque temporisation ***T*****$_d$** : si S ne dépend que d'une entrée E par l'intermédiaire de trois temporisation T1, T2 puis T3, on vérifiera S = f (T3 (T2 (T1 (E)))))

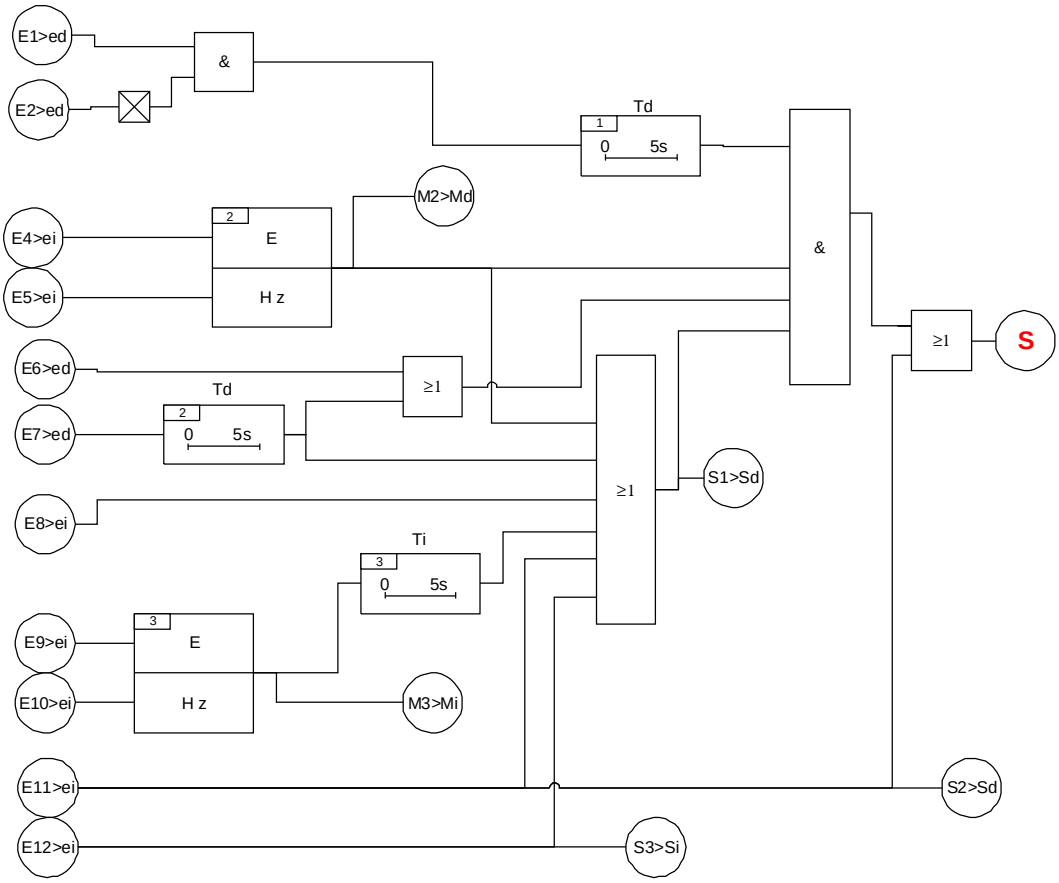


Figure 8 : Illustration de la formule sur un exemple

Dans cet exemple, les vérifications à réaliser sont :

Pour le test et la vérification de S : S = f(T1(E1, E2), M2, E6, T2(E7), S1, S2) ;

Pour le test et la vérification de M2 : M2 = f(M(E4, E5)) ;

Pour le test et la vérification de S1 : S1 = f(M2, T2(E7), E8, T3(M3), S2, S3) ;

Pour le test et la vérification de M3 : M3 = f(M(E9, E10)) ;

Pour le test et la vérification de S2 : S2 = f(E11) ;

Pour le test et la vérification de S3 : S3 = f(E12).

4.4. Algorithme de positionnement du système vers un état stabilisé (BTP)

4.4.1. Objectifs

Le premier prototype Testminator [6] a été l'occasion de développer un algorithme de positionnement de chaque état stabilisé d'un système purement séquentiel, appelé B, et un autre pour les systèmes temporisés, appelé BT. Ces algorithmes permettent de déterminer l'ensemble des états stabilisés (c'est-à-dire après échéance des temporisations dans le cas des systèmes temporisés) atteignables d'un système, ainsi que la séquence des vecteurs d'entrée qui permet d'atteindre chaque état. L'algorithme BT calcule également le temps d'atteinte de chacun des états atteignables du système.

Pour le Test Progressif Par Parties, on réutilise l'algorithme de type BT en le renommant simplement BTP. La suite de ce chapitre 4.4 rappelle la spécification de cet algorithme BT / BTP, en utilisant comme exemple illustratif une spécification fonctionnelle qui servira aussi à la spécification de l'algorithme de génération d'un test progressif par parties (algorithme ATP décrit au chapitre 4.5 suivant).

Pour une application temporisée, l'état ***E*** considéré par l'algorithme BT / BTP dépend des valeurs des états des mémoires et des temporisations, soit ***E*** = f (***M, T***). (Etat de toutes les mémoires M et de toutes les temporisations T participant au calcul de la sortie S à tester)

L'algorithme BTP construit la séquence qui mène à chaque état stabilisé ***E***, notée ***seq*** (***E***).

4.4.2. Algorithme BTP

On initialise toutes les mémoires **M** et temporisations **T** du système à l'état indéterminé I. Cet état global est noté $\mathbf{E_0}$. Soit n le nombre d'entrées dont dépend S. On simule avec l'oracle[9] les conséquences de l'application de chaque vecteur d'entrée possible (il y en a 2^n avec n le nombre d'entrées de la sortie testée du système) en notant **E=(M,T)** l'état des mémoires et temporisations directes du système.

On obtient alors un ensemble ENS_1 de k états (stabilisés) du système spécifié ($E_1,\dots E_k$) et on associe pour chacun de ces états E_i de ENS_1 un vecteur d'entrée V_i permettant de l'atteindre. Pour un état E_i quelconque de ENS_1, plusieurs vecteurs d'entrée peuvent être candidats (un système peut atteindre le même état par des séquences d'entrées différentes). Il s'agit de choisir ceux dont le temps d'atteinte est minimum, afin de minimiser les temps d'exécution des tests lors du positionnement des états internes. De plus, l'état E_i peut être partiellement indéterminé (des mémoires et des temporisations sont dans un état indéterminé)[10].

Pour chaque état E_i de ENS_1, on positionne l'oracle dans l'état des mémoires et temporisations de E_i et on simule par l'oracle les conséquences de l'application de chaque vecteur d'entrée. Trois cas sont à envisager :

- Soit l'application du vecteur V_j conduit à un nouvel état du système E_j[11] en un temps total T_{Ej} ;
- Soit l'application d'un vecteur V_j conduit à un état E_j déjà atteint du système, et le temps d'atteinte de cet état E_j (temps total d'atteinte depuis l'état indéterminé E_0) est inférieur à celui déjà obtenu, on note que la séquence V_i puis V_j, soit le vecteur V_j depuis l'état E_i conduit à l'état E_j en lieu et place de la séquence d'atteinte déjà calculée de E_j ;
- Dans les autres cas, aucune séquence d'atteinte n'est retenue.

On ajoute tous les nouveaux états E_j à ENS_1 pour construire un nouvel état ENS_2. On continue le traitement pour chaque état de ENS_2, puis ENS_g afin de construire un ensemble ENS_f d'états tel qu'à partir de chaque état E_u de ENS_f on retrouve un état de ENS_f suite au positionnement dans l'oracle de E_u et à l'application d'un vecteur d'entrée quelconque. Cette propriété constitue le point fixe de l'algorithme ; invariant existant puisque le nombre d'états stables du système est fini.

Après constitution de la table d'états atteignables et temporisés, on vérifie que chacune des temporisations est ACTIVABLE et DESACTIVABLE à terme (i.e. après stabilisation de l'état du système). Dans le cas contraire, une erreur bloquante est générée. En effet, des temporisations qui ne sont ni activables, ni désactivables, ou seulement de façon fugitive, sont de nature pathologique et

---

[9] L'oracle est la formalisation mathématique exécutable de la spécification de l'application logique, obtenue par transformation du Diagramme Fonctionnel Logique (DFL).

[10] Si l'état Ei est indéterminé, au moins une mémoire est dans l'état indéterminé. Si toutes les mémoires sont à 0 ou 1 alors forcément l'état stable des temporisations est déterminé.

[11] $\forall i \neq j$, Ei=(Ti,Mi) ≠Ej=(Tj,Mj) (i.e. Ti ≠ Tj ou Mi ≠ Mj)..

a priori signe d'une mauvaise spécification du système, qui pourrait alors introduire des comportements cachés.

On vérifie aussi que chaque mémoire est associée à au moins un état de ENS_f qui la positionne à 0 et au moins à un état qui la positionne à 1. Si ce n'est pas le cas, l'outil indique une erreur car cette mémoire est non utilisable et non testable (voir [6] pour le cas d'une mémoire unique).

Première remarque :

Afin de limiter le nombre de simulations, il peut être utile de considérer une table contenant autant d'éléments que le nombre de vecteurs d'entrée à jouer (soit 2^n). Cette table est remplie durant la première passe des 2^n vecteurs de l'algorithme (i.e. depuis l'état E_0 totalement indéterminé du système) et indique le caractère déterministe ou non de chaque vecteur : V_i est déterministe si et seulement si l'application de V_i depuis l'état E_0 conduit à un état entièrement déterminé du système (toutes les mémoires, temporisations et sorties directes considérées sont à 0 ou 1). Lors de la construction des états ENS_g puis ENS_f, seuls seront examinés à nouveau les vecteurs d'entrée V_j non déterministes.

En effet, en l'absence de bouclage, si l'application d'un vecteur V depuis l'état E_0 totalement indéterminé du système conduit à un état E' complètement déterminé, alors l'application du même vecteur V depuis n'importe quel état du système conduit au même état E'. De plus, dans le cadre de l'algorithme BTP dit de positionnement, il n'est pas restrictif de ne plus considérer l'action du vecteur V depuis les états trouvés puisque, d'une part, cela ne conduit pas à la découverte de nouvel état (en vertu de la propriété précédente) et que, d'autre part, le temps d'atteinte d'une séquence V' puis V est forcément supérieur ou égal au temps d'atteinte d'une séquence V.

En conclusion, au prix essentiellement d'un coût mémoire, le nombre de simulations peut être utilement réduit notamment dans le cas où la plupart des vecteurs contraignent de façon déterministe le système. Bien entendu cela peut varier d'un système à l'autre; certains étant plus ou moins favorables à ce type d'optimisation.

Seconde remarque :

Il est inutile de rejouer un vecteur V indéterministe depuis l'état qu'il a permis de trouver lors de la passe précédente (même s'il est indéterminé), puisqu'il conduira de nouveau à cet état.

4.4.3. Exemple

Considérons le système suivant :

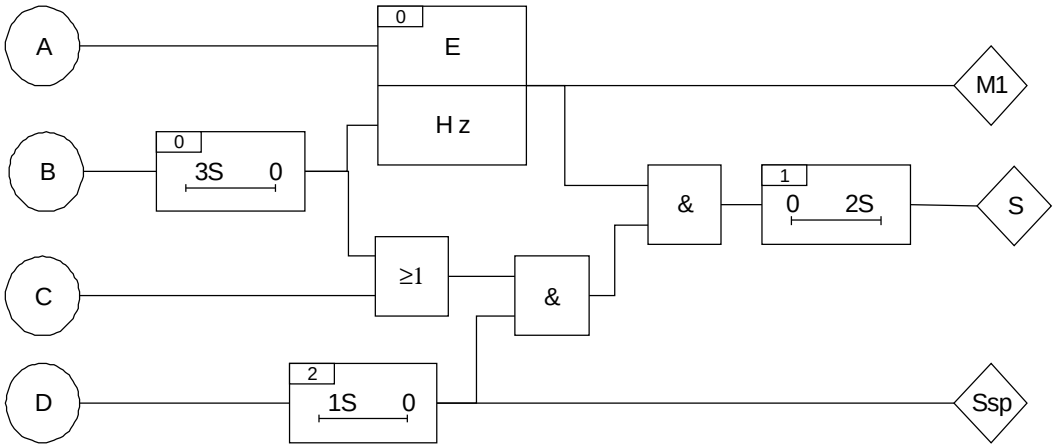


Figure 9 : Illustration de l'algorithme sur un exemple

Dans cet exemple :

- S est la sortie à tester.
- M1 est une mémoire directe (M_d).
- S_{sp} une sortie intermédiaire directe (S_d).
- A et D sont des entrées indirectes (e_i).
- B et C sont des entrées directes (e_d).
- T0 et T1 sont des temporisations directes (T_d).
- T2 est une temporisation indirecte (T_i).

On initialise toutes les temporisations, mémoires et sorties intermédiaires à l'état indéterminé.

On simule avec l'oracle les conséquences de l'application de chaque vecteur d'entrées possible (il y en a 2^n) sur le système. On obtient :

Vecteur	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16
A	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
B	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
C	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
D	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
M1	I	I	I	I	0	0	0	0	1	1	1	1	0	0	0	0
T0	D	D	D	D	A	A	A	A	D	D	D	D	A	A	A	A
T2	D	A	D	A	D	A	D	A	D	A	D	A	D	A	D	A
T1	A	A	A	I	A	A	A	A	A	A	A	D	A	A	A	A
Verdict[12]	S	S	NS	S	S	S	NS	NS	S	S	NS	S	NS	NS	NS	NS
Vecteurs d'atteinte	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Temps d'atteinte	2 s	2 s	2 s	1 s	3 s	5 s	3 s	5 s	2 s	2 s	2 s	1 s	3 s	5 s	3 s	5 s
Etat	E1	E2	E1	E3	E4	E5	E4	E5	E6	E7	E6	E8	E4	E5	E4	E5

ENS_1 = (E1, E2, E3, E4, E5, E6, E7, E8)

Les vecteurs coloriés en vert sont ceux qui conduisent à un état indéterminé. Les états coloriés en orange correspondent aux nouveaux états stables et déterminés calculés par l'algorithme.

On sélectionne l'ensemble des états entièrement déterminés (E4, E5, E6, E7, E8) atteints lors de cette première passe.

On rejoue ensuite, chacun des vecteurs V1, V2, V3, V4 qui conduisent à un état indéterminé (E1, E2, E3) depuis tous les états atteints, y compris ceux indéterminés, à l'exception de l'état qu'il a permis d'atteindre lors de la première passe. En effet, rejouer le vecteur V1 depuis l'état E1 conduira à nouveau à l'état E1.

On rejoue le vecteur V1 (0000) depuis tous les états atteints (sauf E1).

Depuis l'état	E2	E3	E4	E5	E6	E7	E8
M1	I	I	0	0	1	1	1
T0	D	D	D	D	D	D	D
T2	D	D	D	D	D	D	D
T1	A	A	A	A	A	A	A
Temps d'atteinte	2 s	3 s	3 s	5 s	2 s	2 s	3 s
Etats déjà atteints	E1	E1			E6	E6	E6
Nouveaux états			E9	E9			

[12] la séquence de positionnement de l'état Ei est retenue pour le test de la sortie S
NS : la séquence de positionnement de l'état Ei n'est pas retenue pour le test de la sortie S

Un nouvel état trouvé en jouant le vecteur V1 depuis l'état E4. L'état E9 est atteint en jouant successivement les vecteurs V5 et V1 en 3+0 secondes.

On rejoue le vecteur V2 (0001) depuis tous les états atteints (sauf E2).

Depuis l'état	E1	E3	E4	E5	E6	E7	E8
M1	I	I	0	0	1	1	1
T0	D	D	D	D	D	D	D
T2	A	A	A	A	A	A	A
T1	A	A	A	A	A	A	A
Temps d'atteinte	3 s	3 s	4 s	5 s	3 s	2 s	3 s
Etats déjà atteints	E2	E2			E7	E7	E7
Nouveaux états			E10	E10			

Un nouvel état trouvé en jouant le vecteur V2 depuis l'état E4. L'état E10 est atteint en jouant successivement les vecteurs V5 et V2 en 3+1 secondes.

On rejoue le vecteur V3 (0010) depuis tous les états atteints (sauf E1).

Depuis l'état	E2	E3	E4	E5	E6	E7	E8
M1	I	I	0	0	1	1	1
T0	D	D	D	D	D	D	D
T2	D	D	D	D	D	D	D
T1	A	A	A	A	A	A	A
Temps d'atteinte	2 s	3 s	3 s	5 s	2 s	2 s	1 s
Etats déjà atteints	E1	E1			E6	E6	E6
Nouveaux états			E9	E9			

On atteint à nouveau l'état E9 en 3 secondes. Ce temps n'étant pas inférieur au temps d'atteinte de E9 précédemment retenu, cette nouvelle séquence d'atteinte n'est pas retenue.

On rejoue le vecteur V4 (0011) depuis tous les états atteints (sauf E3).

Depuis l'état	E1	E2	E4	E5	E6	E7	E8
M1	I	I	0	0	1	1	1
T0	D	D	D	D	D	D	D
T2	A	A	A	A	A	A	A
T1	I	I	A	A	D	D	D
Temps d'atteinte	3 s	2 s	4 s	5 s	3 s	2 s	1 s
Etats déjà atteints	E3	E3			E8	E8	E8
Nouveaux états			E10	E10			

On atteint à nouveau l'état E10 en 4 secondes. Ce temps n'étant pas inférieur au temps d'atteinte de E10 précédemment retenu, cette nouvelle séquence d'atteinte n'est pas retenue.

L'ensemble des nouveaux états déterminés (E9, E10) trouvés lors de cette passe est ajouté à ENS_1 pour constituer l'ensemble ENS_2.= (E1, E2, E3, E4, E5, E6, E7, E8, E9, E10). E9 est atteint en 3 secondes par application des vecteurs V5 puis V1. E10 est atteint en 4 secondes par application des vecteurs V5 puis V2.

On rejoue ensuite l'ensemble des vecteurs conduisant à des états non entièrement déterminés depuis les nouveaux états trouvés.

Remarquons que, si ces vecteurs sont les derniers vecteurs de la séquence d'atteinte des nouveaux états trouvés, il est inutile de les rejouer puisqu'ils laisseront l'état inchangé.

Par exemple, l'état E9 est atteint en jouant successivement les vecteurs V5 puis V1. Si on rejoue le vecteur V1 (qui mène effectivement à un état non entièrement déterminé à partir d'un état entièrement indéterminé) depuis l'état E9, on reste dans l'état E9.

On rejoue donc le vecteur V1 (0000) uniquement depuis l'état E10 :

Depuis l'état	E10
M1	0
T0	D
T2	D
T1	A
Temps d'atteinte	4 s
Etat déjà atteint	E9

On rejoue le vecteur V2 (0001) uniquement depuis l'état E9 :

Depuis l'état	E9
M1	0
T0	D
T2	A
T1	A
Temps d'atteinte	4 s
Etat déjà atteint	E10

On rejoue le vecteur V3 (0010) uniquement depuis l'état E10 :

Depuis l'état	E10
M1	0
T0	D
T2	D
T1	A
Temps d'atteinte	4 s
Etat déjà atteint	E9

On rejoue le vecteur V4 (0011) uniquement depuis l'état E9 :

Depuis l'état	E9
M1	0
T0	D
T2	A
T1	A
Temps d'atteinte	4 s
Etat déjà atteint	E10

On constate que cette passe ne révèle aucun nouvel état.

Voici, récapitulé l'ensemble des états atteignables de ce système ainsi que les séquences d'atteinte associées :

Etat	M1	T0	T2	T1	Séquence d'atteinte		Temps
E1	I	D	D	A	0000		2 s
E2	I	D	A	A	0001		2 s
E3	I	D	A	I	0011		1 s
E4	0	A	D	A	0100		3 s
E5	0	A	A	A	0101		5 s
E6	1	D	D	A	1000		2 s
E7	1	D	A	A	1001		2 s
E8	1	D	A	D	1011		1 s
E9	0	D	D	A	0100	0000	3 s
E10	0	D	A	A	0100	0001	4 s

4.5. Algorithme de génération des séquences de tests (ATP)

4.5.1. Objectif et principe

Dans le premier développement, il existe un algorithme de sélection de séquences de test pour les applications purement combinatoires et purement séquentielles, appelé A et un autre pour les applications ou systèmes temporisés, appelé AT.

Dans le Test Progressif Par Parties, on appelle son équivalent ATP (algorithme de génération des séquences de test). Cet algorithme n'a plus pour but de tester une temporisation au regard d'une sortie mais de tester une sortie en suivant son évolution temporelle s'il y a lieu. Il consiste à tester successivement les sorties (qu'elles soient des sorties d'instrumentation des mémoires ou d'autres sorties) de la plus en amont à la plus en aval en se servant des résultats du test des sorties en amont pour le test d'une sortie en aval.

Pour une application temporisée, l'algorithme AT considère un vecteur d'entrée candidat $\boldsymbol{V_{candidat}}$ ($\boldsymbol{e}$).

Pour le TPPP, l'algorithme ATP considère un vecteur d'entrée candidat $\boldsymbol{V_{candidat}}$ ($\boldsymbol{e_i}$, $\boldsymbol{e_d}$) (avec $\boldsymbol{ei}$, l'ensemble des entrées indirectes de la sortie à tester et $\boldsymbol{e_d}$ l'ensemble des entrées directes de la sortie à tester). On rappelle qu'une entrée à la fois directe et indirecte (dont S dépend par l'intermédiaire d'une mémoire ou d'une sortie intermédiaire) est considérée comme étant une entrée directe.

On considère pour le critère de couverture du test d'une sortie par l'algorithme ATP, non pas un état composé de l'état global **E=(M,T)** de positionnement interne du système, état de toutes les mémoires M et de toutes les temporisations T participant au calcul de la sortie S à tester, plus les valeurs possibles des sorties intermédiaires directes de la partie à tester $\mathbf{S_d}$, mais un état partiel $\mathbf{E_p=(M_d,T_d,S_d)}$ représentatif des valeurs des variables à considérer de la partie à tester, en plus de la valeur de la sortie S. Cet état partiel est composé des seules mémoires, temporisations et sorties directes de la sortie testée. On rappelle qu'une mémoire, temporisation ou sortie à la fois directe et indirecte pour une sortie S est considérée comme directe. Parmi les états déterminés par l'algorithme BTP, il n'est pas exclu, qu'à 2 états globaux E différents correspondent des mémoires directes M_d et des temporisations directes T_d des états partiels E_p identiques.

On ne considère pour l'algorithme ATP que les états globaux E complètement déterminés calculés par BTP. L'algorithme ATP consiste à simuler un positionnement d'un état global E à partir de sa séquence d'atteinte, calculé à partir de l'algorithme BTP, et à retenir, ou non, un vecteur de test correspondant à l'ensemble des valeurs d'entrées, suivant un objectif de test qui porte sur un état partiel et des évolutions des temporisations de cet état partiel plus la valeur de la sortie à tester.

Notation : L'hypothèse est faite que chaque mémoire de la spécification est instrumentée, c'est-à-dire reliée à une sortie (vérification en « boîte blanche » n° 3 du chapitre 4.3). Si ce n'est pas le cas sur une partie des spécifications, le testeur devra ajouter une sortie à chaque mémoire non-instrumentée de cette partie pour utiliser l'outil. Les mémoires directes M_d et sorties directes S_d de l'état partiel de la sortie à tester sont finalement les sorties intermédiaires de la partie à tester à considérer pour son état partiel, qui peuvent être notées S_p. L'état partiel $E_p=(M_d,T_d,S_d)$ de l'objectif de test peut donc aussi être noté $E_p=(S_p,T_d)$.

Objectif de test : L'algorithme ATP cherche une séquence de test avec un objectif de test portant sur ($\boldsymbol{T_d}$, $\boldsymbol{e_d}$, $\boldsymbol{E_p^1}$,$\boldsymbol{S^1}$, $\boldsymbol{T_d^2}$,$\boldsymbol{S^2}$,…, $\boldsymbol{T_d^k}$, $\boldsymbol{S^k}$). $\boldsymbol{T_d}$ représente l'état des temporisations directes de la partie à tester dans l'état global E. Cet objectif, qui porte sur un nombre réduit de variables du système à tester, n'empêche pas, qu'en simulation, on soit obligé de considérer toutes les possibilités d'évolution de chaque nuplet ($\boldsymbol{E}$, $\boldsymbol{e_d}$, $\boldsymbol{E^1}$,…, $\boldsymbol{E^k}$) pour traiter la couverture complète du critère précédent. En français, l'objectif de test porte sur les états initiaux des temporisations directes de la partie à tester, sur les valeurs des entrées directes d'un vecteur de test candidat, sur les valeurs des sorties intermédiaires directes et des temporisations directes obtenues immédiatement après l'utilisation de ce vecteur de test ($\boldsymbol{E_p^1}$), plus de la valeur de la sortie à tester, puis sur toutes les valeurs successives possibles des temporisations directes et de la sortie à tester.

Dans l'état partiel E_p^1, une mémoire est à 0 ou à 1, une temporisation est dans un des trois états DESACTIVE, ACTIVE ou TRANSITOIRE. La valeur de la sortie S est observable dès le

positionnement de E avec les entrées correspondant au dernier vecteur de la séquence de positionnement, elle est aussi observable, par conséquence, dans l'obtention de l'état partiel E_p^1, lors de l'utilisation en entrée du vecteur de test candidat. Pour chaque ensemble successif des nouvelles valeurs de T^i, S^i (i supérieur ou égal à 2) on note son temps d'atteinte.

La durée de la séquence de test est égale à celle de la séquence ***seq*** (***E***) de positionnement de l'état ***E*** augmentée de la durée de stabilisation du système par application du vecteur candidat $\boldsymbol{V_{candidat}}$ (***e***) via les états globaux successifs ($\boldsymbol{E^1,\ldots,E^k}$) correspondant à une évolution des variables de l'objectif de test $\boldsymbol{E_p^1,S^1}$, $\boldsymbol{T_d^2,S^2,\ldots,T_d^k,S^k}$

1re étape

On positionne l'application dans un état ***E*** grâce à la séquence calculée par l'algorithme BTP. (cf. Annexe 3 pour le choix de l'état positionné)

2e étape

On applique un vecteur d'entrée candidat $\boldsymbol{V_{candidat}}$ ($\boldsymbol{e_i}$, $\boldsymbol{e_d}$) et on simule jusqu'à la stabilisation du système (lorsque toutes les temporisations directes et indirectes sont arrivées à échéance). On obtient une succession d'états intermédiaires jusqu'à la stabilisation, ***E*** évolue en $\boldsymbol{E^1}$, puis $\boldsymbol{E^2,\ldots,E^k}$ suite à l'application en entrée du vecteur candidat, soit ($\boldsymbol{E,E^1,\ldots,E^k}$) et les variables de l'objectif de test évoluent suivant une séquence $\boldsymbol{E_p^1,S^1}$, $\boldsymbol{T_d^2,S^2,\ldots,T_d^k,S^k}$ [13].

3e étape

Si le nuplet ($\boldsymbol{T_d}$, $\boldsymbol{e_d}$, $\boldsymbol{E_p^1,S^1}$, $\boldsymbol{T_d^2,S^2,\ldots,T_d^k,S^k}$) qui caractérise l'objectif de la séquence de test est déjà connu et que la séquence obtenue est plus longue que celle déjà sélectionnée, la séquence est éliminée.

Sinon si le nuplet ($\boldsymbol{T_d}$, $\boldsymbol{e_d}$, $\boldsymbol{E_p^1,S^1}$, $\boldsymbol{T_d^2,S^2,\ldots,T_d^k,S^k}$) qui caractérise l'objectif de la séquence de test n'est pas déjà connu ou que la séquence obtenue est plus courte que celle déjà sélectionnée, la séquence est sélectionnée et efface, le cas échéant, sa concurrente.

[13] L'attente de l'échéance de toutes les temporisations directes ou non du schéma est nécessaire en simulation car elles peuvent influer sur les variables finales de l'objectif de test $\boldsymbol{T_d^k,S^k}$. Le temps retenu pour le test sera celui de l'atteinte de ces valeurs des variables finales de l'objectif de test $\boldsymbol{T_d^k,S^k}$ (cas où l'atteinte de l'échéance de certaines temporisations indirectes ne modifie plus le sous-état, soit un temps supplémentaire inutile pour la partie testée).

4.5.2. Algorithme ATP

Soit **S** une sortie du système à tester ;

Pour chaque état **E** déterministe associé à **S** dans la table BTP ou au moins une fois s'il n'existe pas d'état E à positionner[14]

{

On positionne l'oracle dans l'état **E** ou on ne fait rien s'il n'existe pas d'état E

{

Pour chaque vecteur $\mathbf{V_{candidat}=(e_d, e_i)}$ parmi les 2^n vecteurs (n étant le nombre d'entrées de S)

{

On simule en positionnant les entrées avec $\mathbf{V_{candidat}=(e_d, e_i)}$

Les variables de l'objectif de test évoluent alors suivant une succession d'états **Succ** = $(T_d, e_d, E_p^1, S^1, T_d^2, S^2, \ldots, T_d^k, S^k)$ avant stabilisation en un temps $\mathbf{T_{Succ}}$

S'il existe déjà un vecteur $\mathbf{V_{candidat}}$**' retenu** ayant les mêmes entrées directes $\mathbf{e_d}$ que $\mathbf{V_{candidat}}$ et faisant évoluer **Succ** de la même manière que $\mathbf{V_{candidat}}$, c'est-à-dire si les variables de la partie testée passent successivement par les mêmes états, $(T_d, e_d, E_p^1, S^1, T_d^2, S^2, \ldots, T_d^k, S^k)$ en un temps total $\mathbf{T_{Succ}}$'$\mathbf{<=T_{Succ}}$ lors de l'application de $\mathbf{V_{candidat}}$'.

{

On ne retient pas $\mathbf{V_{candidat}}$ pour le test de **S**

}

Sinon

{

On retient $\mathbf{V_{candidat}}$ pour une séquence de test qui consiste à positionner **E** et à tester la valeur de la sortie S après positionnement stabilisé de **E** puis[15] de façon continue lors de l'application de $\mathbf{V_{candidat}}$.

On efface un vecteur $\mathbf{V_{candidat}}$**' retenu** ayant les mêmes entrées directes $\mathbf{e_d}$ que $\mathbf{V_{candidat}}$ et faisant évoluer les variables de l'objectif de test de la partie testée de la même manière que $\mathbf{V_{candidat}}$, c'est-à-dire si ces variables passent successivement par les mêmes états, $(T_d, e_d, E_p^1, S^1, T_d^2, S^2, \ldots, T_d^k, S^k)$ en un temps total $\mathbf{T_{Succ}}$'$\mathbf{>T_{Succ}}$ lors de l'application de $\mathbf{V_{candidat}}$'.

}

}

}

}

Pour tout état **E** et vecteur $\mathbf{V_{candidat}}$ retenu :

Dans le cas où $\mathbf{V_{candidat}}$ correspond au dernier vecteur d'atteinte de l'état **E**, la séquence de test consiste à positionner **E**, et à tester la valeur de la sortie **S** après positionnement stabilisé de **E**

---

[14] C'est le cas, par exemple, si la sortie S à tester n'a ni temporisation ni mémoire la reliant à ses entrées, même si la sortie S se trouve elle-même être une mémoire

[15] Note : La valeur de S est observable dès le positionnement de E avec les entrées correspondant au dernier vecteur de la séquence de positionnement

4.5.3. Exemple

Les états entièrement déterminés trouvés précédemment et leur séquence d'atteinte sont :

Etat	M1	T0	T2	T1	Séquence d'atteinte		Temps
E4	0	A	D	A	0100		3 s
E5	0	A	A	A	0101		5 s
E6	1	D	D	A	1000		2 s
E7	1	D	A	A	1001		2 s
E8	1	D	A	D	1011		1 s
E9	0	D	D	A	0100	0000	3 s
E10	0	D	A	A	0100	0001	4 s

E1, E2 et E3 ne sont pas considérés car non entièrement déterminés.

On positionne le premier état E4 par application du vecteur V5 (0100) puis on relève l'évolution des variables de l'objectif de test ($T0,T1,B,C,T0^1,T1^1,M1,S_{sp},S^1,T0^2,T1^2,S^2,\ldots,T0^k,T1^k,S^k$) ainsi que les temps d'atteinte des différentes valeurs successives des variables de l'objectif de test T0, T1 et de la valeur de la sortie à tester S pour l'ensemble des 2^n vecteurs d'entrée possibles (A, B, C, D).

On détermine par ailleurs, à quel état atteignable du système déterminé par l'algorithme BTP correspond l'état final du système atteint après application du vecteur V. On précise au §4.6 comment la connaissance de cet état permet d'optimiser le jeu des séquences de test.

On réalise ensuite le même exercice pour l'ensemble des états entièrement déterminés (E5, E6, E7, E8, E9, E10) que l'algorithme BTP a permis de déterminer.

On obtient alors les résultats suivants[16]:

Dans ce qui suit :

[16] Les séquences de test sélectionnées pour le test de la sortie S sont marquées « s » et les séquences qui ne le sont pas sont marquées « ns » dans des cases de différentes couleurs qui correspondent à des vecteurs de test qui couvrent le même objectif de test. Pour les vecteurs d'entrée, les valeurs des entrées indirectes, A et D, sont notées en noir et les valeurs des entrées directes, B et C, sont notées en rouge.

E=(M,T) **M = M1, T = (T0, T1, T2)** **➔ E = (M1, T0, T1, T2)**

E_p=(M_d,T_d,S_d) **M_d = M1, T_d = (T0, T1), S_d = Ssp** **➔ E_p = (M1, T0, Ssp, T1)**

V=(e_d,e_i) **e_d = (B, C), e_i = (A, D)** **➔ V = (A, B, C, D)**

Etat de départ	Vecteur V	T_d / S	E_p[1] / S	T_d[2] (t) / S	Verdict	Etat d'arrivée	Type
E4 [M1,T0,T1,T2] = [0,A,A,D]	0000	0A0A / 0	0D0A / 0		s	E9	R
	0001	0A0A / 0	0D0A / 0	0D1A (1 s) / 0	ns	E10	
	0010	0A0A / 0	0D0A / 0		s	E9	R
	0011	0A0A / 0	0D0A / 0	0D1A (1 s) / 0	ns	E10	
	0100	0A0A / 0	~~0A0A / 0~~		s	E4	R
	0101	0A0A / 0	0A0A / 0	0A1A (1 s) / 0	ns	E5	
	0110	0A0A / 0	0A0A / 0		s	E4	R
	0111	0A0A / 0	0A0A / 0	0A1A (1 s) / 0	ns	E5	
	1000	0A0A / 0	1D0A / 0		s	E6	R
	1001	0A0A / 0	1D0A / 0	1D1A (1 s) / 0	ns	E7	
	1010	0A0A / 0	1D0A / 0		s	E6	R
	1011	0A0A / 0	1D0A / 0	1D1D (1 s) / 1	s	E8	S
	1100	0A0A / 0	0A0A / 0		ns	E4	
	1101	0A0A / 0	0A0A / 0	0A1A (1 s) / 0	ns	E5	
	1110	0A0A / 0	0A0A / 0		ns	E4	
	1111	0A0A / 0	0A0A / 0	0A1A (1 s) / 0	ns	E5	

Etat de départ	Vecteur V	T_d / S	E_p^1 / S		Verdict	Etat d'arrivée	Type
E5 [M1,T0,T1,T2] = [0,A,A,A]	0000	0A1A / 0	0D0A / 0		ns	E9	
	0001	0A1A / 0	0D1A / 0		s	E10	R
	0010	0A1A / 0	0D0A / 0		ns	E9	
	0011	0A1A / 0	0D1A / 0		s	E10	R
	0100	0A1A / 0	0A0A / 0		ns	E4	
	0101	0A1A / 0	~~0A1A / 0~~		s	E5	R
	0110	0A1A / 0	0A0A / 0		ns	E4	
	0111	0A1A / 0	0A1A / 0		s	E5	R
	1000	0A1A / 0	1D0A / 0		ns	E6	
	1001	0A1A / 0	1D1A / 0		s	E7	R
	1010	0A1A / 0	1D0A / 0		ns	E6	
	1011	0A1A / 0	1D1D / 1		s	E8	S
	1100	0A1A / 0	0A0A / 0		ns	E4	
	1101	0A1A / 0	0A1A / 0		ns	E5	
	1110	0A1A / 0	0A0A / 0		ns	E4	
	1111	0A1A / 0	0A1A / 0		ns	E5	

Etat de départ	Vecteur V	T_d / S	E_p^1 / S	T_d^2 (t) / S	T_d^3 (t) / S	E_p^4 (t) / S	Verdict	Etat d'arrivée	Type
E6 [M1,T0,T1,T2] = [1,D,A,D]	0000	1D0A / 0	1D0A / 0				ns	E6	
	0001	1D0A / 0	1D0A / 0	1D1A (1 s) / 0			ns	E7	
	0010	1D0A / 0	1D0A / 0				s	E6	R
	0011	1D0A / 0	1D0A / 0	1D1D (1 s) / 1			ns	E8	
	0100	1D0A / 0	1T0A / 0	0A0A (3 s) / 0			s	E4	R
	0101	1D0A / 0	1T0A / 0	1T1A (1 s) / 0	0A1A (2 s) / 0		ns	E5	
	0110	1D0A / 0	1T0A / 0	0A0A (3 s) / 0			s	E4	R
	0111	1D0A / 0	1T0A / 0	1T1D (1 s) / 1	0A1T (2 s) / 1	0A1A (2 s) / 0	s	E5	R
	1000	1D0A / 0	~~1D0A / 0~~				s	E6	R
	1001	1D0A / 0	1D0A / 0	1D1A (1 s) / 0			ns	E7	
	1010	1D0A / 0	1D0A / 0				ns	E6	
	1011	1D0A / 0	1D0A / 0	1D1D (1 s) / 1			s	E8	
	1100	1D0A / 0	1T0A / 0	0A0A (3 s) / 0			ns	E4	
	1101	1D0A / 0	1T0A / 0	1T1A (1 s) / 0	0A1A (2 s) / 0		ns	E5	
	1110	1D0A / 0	1T0A / 0	0A0A (3 s) / 0			ns	E4	
	1111	1D0A / 0	1T0A / 0	1T1D (1 s) / 1	0A1T (2 s) / 1	0A1A (2 s) / 0	ns	E5	

Etat de départ	Vecteur V	T_d / S	E_p^1 / S	T_d^2 (t) / S	T_d^3 (t) / S	Verdict	Etat d'arrivée	Type
E7 [M1,T0,T1,T2] = [1,D,A,A]	0000	1D1A / 0	1D0A / 0			ns	E6	.
	0001	1D1A / 0	1D1A / 0			ns	E7	
	0010	1D1A / 0	1D0A / 0			ns	E6	.
	0011	1D1A / 0	1D1D / 1			s	E8	S
	0100	1D1A / 0	1T0A / 0	0A0A (3 s) / 0		ns	E4	
	0101	1D1A / 0	1T1A / 0	0A1A (3 s) / 0		s	E5	R
	0110	1D1A / 0	1T0A / 0	0A0A (3 s) / 0		ns	E4	
	0111	1D1A / 0	1T1D / 1	0A1T (3 s) / 1	0A1A (2 s) / 0	s	E5	R
	1000	1D1A / 0	1D0A / 0			ns	E6	
	1001	1D1A / 0	~~1D1A / 0~~			s	E7	R
	1010	1D1A / 0	1D0A / 0			ns	E6	
	1011	1D1A / 0	1D1D / 1			ns	E8	
	1100	1D1A / 0	1T0A / 0	0A0A (3 s) / 0		ns	E4	
	1101	1D1A / 0	1T1A / 0	0A1A (3 s) / 0		ns	E5	
	1110	1D1A / 0	1T0A / 0	0A0A (3 s) / 0		ns	E4	
	1111	1D1A / 0	1T1D / 1	0A1T (3 s) / 1	0A1A (2 s) / 0	ns	E5	

Etat de départ	Vecteur V	T_d / S	E_p[1] / S	T_d[2] (t) / S	T_d[3] (t) / S	Verdict	Etat d'arrivée	Type
E8 [M1,T0,T1,T2] = [1,D,D,A]	0000	1D1D / 1	1D0T / 1	1D0A (2 s) / 0		s	E6	R
	0001	1D1D / 1	1D1T / 1	1D1A (2 s) / 0		s	E7	R
	0010	1D1D / 1	1D0T / 1	1D0A (2 s) / 0		s	E6	R
	0011	1D1D / 1	1D1D / 1			ns	E8	
	0100	1D1D / 1	1T0T / 1	1T0A (2 s) / 0	0A0A (1 s) / 0	s	E4	R
	0101	1D1D / 1	1T1T / 1	1T1A (2 s) / 0	0A1A (1 s) / 0	s	E5	R
	0110	1D1D / 1	1T0T / 1	1T0A (2 s) / 0	0A0A (1 s) / 0	s	E4	R
	0111	1D1D / 1	1T1D / 1	0A1T (3 s) / 1	0A1A (2 s) / 0	s	E5	R
	1000	1D1D / 1	1D0T / 1	1D0A (2 s) / 0		ns	E6	
	1001	1D1D / 1	1D1T / 1	1D1A (2 s) / 0		ns	E7	
	1010	1D1D / 1	1D0T / 1	1D0A (2 s) / 0		ns	E6	
	1011	1D1D / 1	~~1D1D / 1~~			s	E8	S
	1100	1D1D / 1	1T0T / 1	1T0A (2 s) / 0	0A0A (1 s) / 0	ns	E4	
	1101	1D1D / 1	1T1T / 1	1T1A (2 s) / 0	0A1A (1 s) / 0	ns	E5	
	1110	1D1D / 1	1T0T / 1	1T0A (2 s) / 0	0A0A (1 s) / 0	ns	E4	
	1111	1D1D / 1	1T1D / 1	0A1T (3 s) / 1	0A1A (2 s) / 0	ns	E5	

Etat de départ	Vecteur V	T_d / S	E_p[1] / S	T_d[2] (t) / S	T_d[3] (t) / S	Verdict	Etat d'arrivée	Type
E9 [M1,T0,T1,T2] = [0,D,A,D]	0000	0D0A / 0	~~0D0A / 0~~			s	E9	R
	0001	0D0A / 0	0D0A / 0	0D1A (1 s) / 0		ns	E10	
	0010	0D0A / 0	0D0A / 0			s	E9	R
	0011	0D0A / 0	0D0A / 0	0D1A (1 s) / 0		ns	E10	
	0100	0D0A / 0	0T0A / 0	0A0A (3 s) / 0		s	E4	R
	0101	0D0A / 0	0T0A / 0	0T1A (1 s) / 0	0A1A (2 s) / 0	ns	E5	
	0110	0D0A / 0	0T0A / 0	0A0A (3 s) / 0		s	E4	R
	0111	0D0A / 0	0T0A / 0	0T1A (1 s) / 0	0A1A (2 s) / 0	ns	E5	
	1000	0D0A / 0	1D0A / 0			ns	E6	
	1001	0D0A / 0	1D0A / 0	1D1A (1 s) / 0		ns	E7	
	1010	0D0A / 0	1D0A / 0			ns	E6	
	1011	0D0A / 0	1D0A / 0	1D1D (1 s) / 1		ns	E8	
	1100	0D0A / 0	0T0A / 0	0A0A (3 s) / 0		ns	E4	
	1101	0D0A / 0	0T0A / 0	0T1A (1 s) / 0	0A1A (2 s) / 0	ns	E5	
	1110	0D0A / 0	0T0A / 0	0A0A (3 s) / 0		ns	E4	
	1111	0D0A / 0	0T0A / 0	0T1A (1 s) / 0	0A1A (2 s) / 0	ns	E5	

Etat de départ	Vecteur V	T_d / S	E_p[1] / S	T_d[2] (t) / S	Verdict	Etat d'arrivée	Type
E10 [M1,T0,T1,T2] = [0,D,A,A]	0000	0D1A / 0	0D0A / 0		ns	E9	
	0001	0D1A / 0	~~0D1A / 0~~		s	E10	R
	0010	0D1A / 0	0D0A / 0		ns	E9	
	0011	0D1A / 0	0D1A / 0		s	E10	R
	0100	0D1A / 0	0T0A / 0	0A0A (3 s) / 0	ns	E4	R
	0101	0D1A / 0	0T1A / 0	0A1A (3 s) / 0	s	E5	R
	0110	0D1A / 0	0T0A / 0	0A0A (3 s) / 0	ns	E4	
	0111	0D1A / 0	0T1A / 0	0A1A (3 s) / 0	s	E5	R
	1000	0D1A / 0	1D0A / 0		ns	E6	
	1001	0D1A / 0	1D1A / 0		ns	E7	
	1010	0D1A / 0	1D0A / 0		ns	E6	
	1011	0D1A / 0	1D1D / 1		ns	E8	
	1100	0D1A / 0	0T0A / 0	0A0A (3 s) / 0	ns	E4	
	1101	0D1A / 0	0T1A / 0	0A1A (3 s) / 0	ns	E5	
	1110	0D1A / 0	0T0A / 0	0A0A (3 s) / 0	ns	E4	
	1111	0D1A / 0	0T1A / 0	0A1A (3 s) / 0	ns	E5	

L'application de l'algorithme ATP permet de déterminer39 séquences de test de la sortie S : 4 de type S et 35 de type R.

Une première solution pour l'enchaînement des séquences de test consiste à prendre dans l'ordre de leur production les séquences déterminées par ATP. On positionne ainsi le premier état déterministe E4, puis on joue l'ensemble des vecteurs $V_{candidat}$ retenus pour le test de la sortie S.

On obtiendrait ainsi la séquence de test suivante depuis l'état E4 :

Etat de départ	Vecteur V	T_d / S	E_p^1 (t) / S	T_d^2 (t) / S	Verdict	Valeur observée en sortie
E4 [M1,T0,T1,T2] = [0,A,A,D]	0000	0A0A / 0	0D0A / 0		s	0
	0001	0A0A / 0	0D0A / 0	0D1A (1 s) / 0	s	0 (1 s)
	0010	0A0A / 0	0D0A / 0		s	0
	0011	0A0A / 0	0D0A / 0	0D1A (1 s) / 0	s	0 (1 s)
	0100	0A0A / 0	0A0A / 0		s	0
	0101	0A0A / 0	0A1A (1 s) / 0		s	0
	0110	0A0A / 0	0A0A / 0		s	0
	0111	0A0A / 0	0A1A (1 s) / 0		s	0
	1000	0A0A / 0	1D0A / 0		s	0
	1001	0A0A / 0	1D0A / 0	1D1A (1 s) / 0	s	0
	1010	0A0A / 0	1D0A / 0		s	0
	1011	0A0A / 0	1D0A / 0	1D1D (1 s) / 1	s	1 (1 s)

Lors de l'enchaînement de ces séquences de test (positionnement de l'état E4 puis jeu d'un vecteur candidat), les vecteurs candidats, hormis 1011, laissent la sortie inchangée à 0. Le jeu du vecteur 1011 fait passer la sortie de 0 à 1 au bout d'une seconde.

Lors du test, la valeur de la sortie peut être testée lors d'un positionnement de l'état global stable de départ E_k (la sortie S vaut 0 lors du positionnement de E4 pour la couverture de l'état partiel 0A0A dans le tableau de l'exemple précédent). La valeur de la sortie S doit être testée suite à l'application du vecteur d'entrée V de test (la sortie S vaut 0 suite à l'atteinte de l'état partiel 1D0A lors de l'application en entrée du vecteur de test 1011 à partir de l'état E4, dernière ligne du tableau précédent). La valeur de la sortie S doit ensuite être testée de façon continue jusqu'à l'atteinte d'un état stable (pour la dernière ligne du tableau précédent, la valeur de la sortie S sera testée en permanence à 0 pendant 1 seconde, puis on testera son passage à 1 au bout d'une seconde).

On procède ensuite de même pour chacun des 7 états déterministes identifiés par l'algorithme BTP.

Cette technique de test s'avère cependant coûteuse en temps, il faut en effet systématiquement repositionner l'état de départ, si le jeu du vecteur candidat précédent n'a pas permis de laisser l'état de départ inchangé.

Le chapitre suivant décrit une façon d'optimiser l'enchaînement de ces séquences de test afin de réduire le coût induit par un repositionnement systématique d'un état initial.

4.6. Optimisation du parcours des séquences de test

4.6.1. Algorithme TPPP optimisé

L'application de n'importe quel vecteur d'entrée à partir d'un état stable entièrement déterminé conduit le système dans un nouvel état stable entièrement déterminé.

L'algorithme BTP permet de calculer l'ensemble des états stables entièrement déterminés du système (à échéance des temporisations) ainsi que les vecteurs ou séquences de vecteurs permettant d'atteindre ces états.

L'application d'une séquence de test déterminée par ATP (positionnement de l'état E et jeu d'un vecteur candidat) conduit donc le système dans un état stable E_k entièrement déterminé (forcément compris dans l'ensemble des états atteignables du système déterminé par l'algorithme BTP).

Cette propriété sert à l'optimisation de l'enchaînement des séquences de test déterminées par l'algorithme ATP. L'idée est de prendre l'état obtenu E_k comme état de départ E d'une autre séquence de test, sans avoir à le positionner.

Les vecteurs candidats joués et retenus par l'algorithme ATP peuvent être de 3 types :

(1) Vecteurs faisant passer la sortie à 1 depuis un état de celle-ci à 0. On parle de vecteur Set (Vecteur S),

(2) Vecteurs faisant passer la sortie à 0 depuis un état de celle-ci à 1. On parle de vecteur Reset (Vecteur R),

(3) Vecteurs laissant la sortie inchangée à 0 depuis un état où celle-ci est à 0 et laissant la sortie inchangée à 1 depuis un état où celle-ci est à 1. De tels vecteurs ne se rencontrent que si la sortie sous test est une sortie mémoire (les deux entrées de la mémoire sont à 0), on parle de vecteur inchangé (Vecteur I).

La valeur de S considérée est celle correspondant à l'état stabilisé E_k.

On ne qualifie pas un vecteur R, S ou I à partir d'une lecture d'une ligne d'évolution des états partiels et de la valeur correspondante de la sortie à tester. Seul le dernier sous-état stable d'une ligne des tableaux de test précédents est considéré pour le test d'une sortie S correspondant à une mémoire : soit les entrées (Set, Reset) de la mémoire à tester valent (0,0) et il s'agit d'un vecteur de type I, soit il s'agit d'autres valeurs et c'est un vecteur R ou S (exemple (1,0) = S ; (0,1) = R ; (1,1) = S si mémoire à Set prioritaire et (1,1)=R si mémoire à Reset prioritaire). Si la sortie n'est pas une mémoire mais une sortie simple, il n'y a pas de vecteur de type I car on transforme la sortie simple à tester en mémoire pour pouvoir réutiliser l'ensemble de l'algorithme du test d'une mémoire sans avoir une version spécifique du test d'une sortie simple à développer. La transformation de la sortie simple S en une mémoire S à Set prioritaire dont le Reset est toujours à 1 fait qu'on ne peut rencontrer le cas (0,0) = I. Il n'existe donc pas de vecteur de type I inchangé pour une sortie simple.

On organise ensuite la séquence de test RII…IISII…RSRSR :

On part d'un état E, positionné par BTP, qui correspond à un vecteur R.

Tant qu'il existe des vecteurs non encore testés :

Dans le choix d'un nouveau vecteur pour la construction de cette séquence, quel que soit le type recherché (R, S ou I) :

A. On choisit un vecteur laissant inchangé l'état E et on ajoute sa séquence au scénario de test

B. S'il n'y en a pas, on choisit un vecteur dont l'état de départ E est l'état correspondant à l'état d'arrivée E_k du vecteur de test précédent et on ajoute sa séquence au scénario de test

C. S'il n'y en a pas, on choisit un vecteur dont l'état de départ E est différent de l'état correspondant à l'état d'arrivée E_k du vecteur de test précédent et on ajoute la séquence de positionnement de l'état E puis la séquence de test du vecteur choisi au scénario de test

Dans les tableaux des évolutions des vecteurs de test de l'algorithme ATP, la troisième colonne correspond à l'état des temporisations directes de la sortie à tester $\mathbf{T_d}$ telles que fixées par l'état global E qui a été positionné. La valeur de la sortie S est celle obtenue avec les valeurs des entrées directes du dernier vecteur de positionnement de E dans BTP. La valeur de la sortie S est donc complètement déterminée et peut être testée (troisième possibilité C dans les choix A, B et C précédents).

Pour chaque vecteur de test, on teste de façon permanente la valeur de la sortie jusqu'à l'atteinte d'un état stable.

Les choix A et B précédents permettent d'éviter de repositionner à chaque fois l'état global E_k de

départ. Dans ce cas où on part d'un état d'arrivée E en utilisant l'état d'arrivée du vecteur de test précédent (donc sans repositionnement de E, première et seconde possibilités A et B dans les choix précédents) la valeur de S n'est pas forcément celle obtenue suite à la séquence de positionnement de cet état E de l'algorithme BTP (les valeurs des entrées directes sont celles du vecteur de test précédent, qui peuvent être différentes des valeurs des entrées directes du dernier vecteur de positionnement de l'état E dans BTP). Dans ce cas, la valeur de la sortie S de la troisième colonne n'est donc pas significative et la sortie n'est pas à tester avec cette valeur de positionnement. Le test de S commence avec la valeur obtenue à partir de cet état E suite à l'application du nouveau vecteur de test V en entrée (effacement du test de S en $\mathbf{T_d}$ / S et début du test de S à la colonne $\mathbf{E_p^1}$ (t) / S dans les tableaux précédents).

De plus, lors du positionnement des états globaux E par l'algorithme BTP, la sortie S n'est pas forcément significative tant que la séquence de positionnement n'est pas terminée (on note i comme indéfini pour la valeur attendue de la sortie S dans les scénarios de test).

4.6.2. Exemple

Dans l'exemple suivant, l'organisation du test en une séquence RII…SII…RSRSRS… n'est pas considérée car il n'y a pas de vecteurs de type I.

Séquence de tests suite à l'application de l'algorithme TPPP optimisé sur la sortie S

Etat de départ E6 (initialisation)

Vecteur appliqué

A:1 B:0 C:0 D:0

Etats réduits atteints (T:2000MS)

PAGE_1__TF1_1:j PAGE_1__TN1_0:D SSP:0 M1:1 Sortie=i T: 0 ms

PAGE_1__TF1_1:A PAGE_1__TN1_0:D SSP:0 M1:1 Sortie=0 T: 2000 ms

Vecteur appliqué

A:1 B:0 C:1 D:1

Etats réduits atteints (T:1000MS)

PAGE_1__TF1_1:A PAGE_1__TN1_0:D SSP:0 M1:1 Sortie=0 T: 0 ms

PAGE_1__TF1_1:D PAGE_1__TN1_0:D SSP:1 M1:1 Sortie=1 T: 1000 ms (S on est passé dans E8)

Vecteur appliqué

A:0 B:0 C:0 D:0

Etats réduits atteints (T:2000MS)

PAGE_1__TF1_1:T PAGE_1__TN1_0:D SSP:0 M1:1 Sortie=1 T: 0 ms

PAGE_1__TF1_1:A PAGE_1__TN1_0:D SSP:0 M1:1 Sortie=0 T: 2000 ms (R on revient dans E6)

Reprise 1 : Etat de départ E4

Vecteur appliqué

A:0 B:1 C:0 D:0

Etats réduits atteints (T:3000MS)

PAGE_1__TF1_1:j PAGE_1__TN1_0:j SSP:0 M1:i Sortie=i T: 0 ms

PAGE_1__TF1_1:A PAGE_1__TN1_0:j SSP:0 M1:i Sortie=0 T: 2000 ms

PAGE_1__TF1_1:A PAGE_1__TN1_0:A SSP:0 M1:0 Sortie=0 T: 3000 ms (on a positionné E4 avec son vecteur de positionnement 0100 pour pouvoir jouer le vecteur suivant à partir de cet état E4 pour lequel il a été retenu)

Vecteur appliqué

A:1 B:0 C:1 D:1

Etats réduits atteints (T:1000MS)

PAGE_1__TF1_1:A PAGE_1__TN1_0:D SSP:0 M1:1 Sortie=0 T: 0 ms

PAGE_1__TF1_1:D PAGE_1__TN1_0:D SSP:1 M1:1 Sortie=1 T: 1000 ms (rejeu du seul vecteur S de E4)

Etc.

…

4.7. Observabilité des temporisations

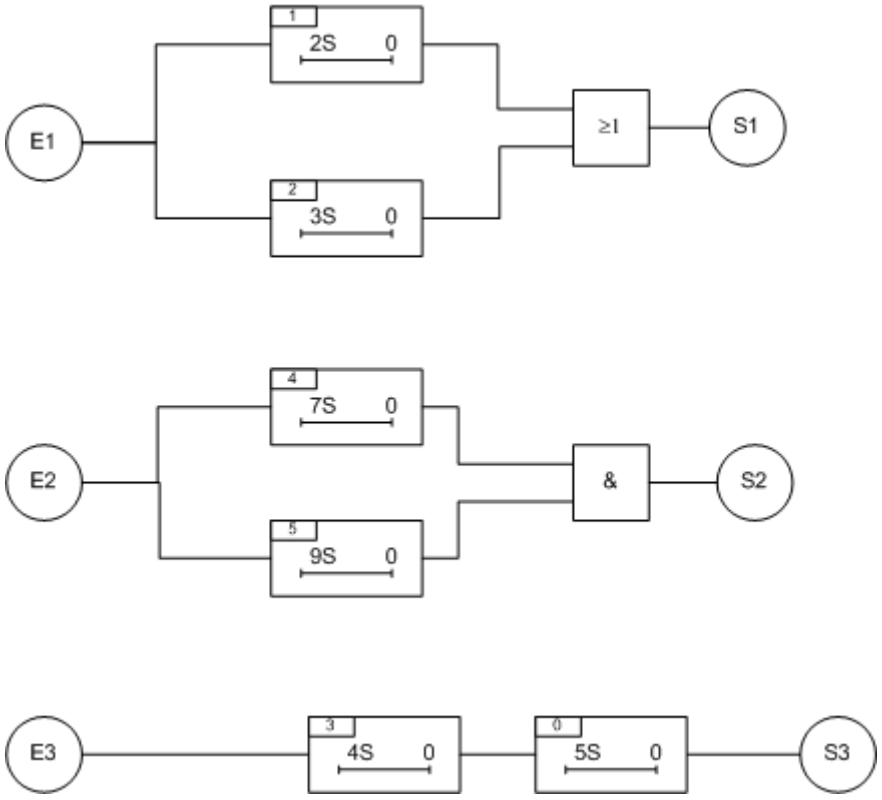
Le critère retenu pour avertir de la non-observabilité des temporisations dans le projet VACSIM est plus simple que celui utilisé dans le premier développement Testminator.

Une temporisation T est déclarée observable en regard d'une sortie S, si au cours de la stabilisation du système simulée par l'oracle :

1. T passe de l'état TRANSITOIRE à l'état ACTIVE en un instant $t_{activation}$ et c'est la seule temporisation en cet instant à passer à l'état ACTIVE ;
2. En cet instant $t_{activation}$ la sortie S change de valeur.

Si une temporisation n'est observable en regard d'aucune sortie, un message d'erreur non bloquant est généré.

Dans les schémas suivants, les temporisations 2, 3 et 4 sont non observables au sens du critère précédent.



5. Conclusion

Ce document propose un algorithme de test progressif par parties de systèmes logiques non bouclés.

Il correspond au livrable L1.1 du projet VACSIM.

Le développement de cet algorithme dans l'environnement ControlBuild fait l'objet du livrable L1.2.

Les exemples utilisés dans cette note n'ont que pour but d'illustrer les algorithmes proposés. Ils ont été produits à partir d'une spécification qui emploie un nombre très réduit d'entrées (quatre) et pour laquelle il existe un très faible écart entre l'état global et l'état partiel de la partie à tester (uniquement une temporisation T2). De ce fait, ils ne permettent pas de mettre en évidence la réduction du problème combinatoire pour les séquences de test retenues, que l'on espère drastique par rapport à la version précédente de Testminator pour des spécifications disposant de nombreuses entrées et de sorties intermédiaires. Cette réduction sera étudiée à l'occasion des travaux du livrable L1.3 du projet (« Evaluation sur études de cas industriels d'un algorithme de test progressif par parties de systèmes logiques non bouclés dans l'environnement ControlBuild »).

6. Documents de référence

[1] H-P1A-2009-00557-FR, Projet MACCOPA 2 : Algorithmes de génération et d'exécution du test des systèmes logiques, séquentiels et temporisés non bouclés pour le projet ANR TESTEC, F.Chériaux, D. Trognon, L. Picci, T. Gazet, EDF R&D/STEP/P1A, 20/01/2010

[2] H-11/04/018/A, Projet VD3-900 R&D : Cahier des charges et proposition d'algorithmes pour un outil de test des systèmes logiques programmés en regard de spécifications fonctionnelles, F. Chériaux, D. Trognon, 03/2004

[3] H-11/04/018/B, Projet VD3-900 R&D : Cahier des charges et proposition d'algorithmes pour un outil de test des systèmes logiques programmés en regard de spécifications fonctionnelles –

version mise à jour du prototype 1, F.Chériaux, D. Trognon, L. Picci, T. Gazet, 09/2009

[4] Dossier de spécification du logiciel de test de systèmes logiques programmés, L. Besson, D. Van Eeckhaute, 05/2004

[5] Dossier de conception du logiciel de test de systèmes logiques programmés, L. Besson, D. Van Eeckhaute, 2004

[6] Projet ANR TESTEC (TEst des Systèmes Temps réel Embarqués Critiques - TLOG07-022), Sous-Projet SP5, « Maîtrise de l'explosion combinatoire et complétude du test pour les systèmes logiques critiques », Livrable L5.1, « Algorithmes de génération et d'exécution du test des systèmes logiques, séquentiels et temporisés non bouclés », mai 2011

[7] Norme internationale NF EN CEI 61131-3 « Automates programmables - Partie 3 : Langages de programmation » 2003

[8] "Prospects for model-based testing of discrete systems", 1st IFAC Workshop Dependable Control of Discrete Systems (DCDS'07), P. Salaun, F. Chériaux, D. Trognon, June 13-15, 2007, Cachan (France)

[9] "Model-based Testing of Safety Logics" F. Chériaux, D. Trognon, P. Salaün, 18th Annual Joint ISA POWID/EPRI Controls and Instrumentation Conference and the 51st ISA POWID Division Symposium, Phoenix (USA), June 2008

[10] "Functional Test of Control Systems Ensuring a High Coverage Rate". F. Chériaux, P. Salaün, F. Daumas. Sixth American Nuclear Society International Topical Meeting on Nuclear Plant Instrumentation, Control, and Human-Machine Interface Technologies, NPIC&HMIT 2009, Knoxville (USA), April 2009.

[11] "Conformance test of logic controllers of critical systems from industrial specifications", F. Chériaux, L. Picci (EDF R&D) J. Provost, J.M. Faure (ENS LURPA). Proceedings of ESREL 2010, Rhodes, Greece, Ale, Papazoglou & Zio (eds), Taylor & Francis, ISBN 978-0-415-60427-7, pp. 1569-1576.

7. Glossaire

Abréviation	Signification
A	Algorithme de sélection de séquences de test pour les applications non temporisées
AT	Algorithme de sélection de séquences de test pour les applications Temporisées
ATP	Algorithme de sélection de séquences de test pour les applications Temporisées dédié au test progressif par Parties
B	Algorithme de positionnement vers un état stabilisé pour les applications non temporisées
BT	Algorithme de positionnement vers un état stabilisé pour les applications Temporisées
BTP	Algorithme de positionnement vers un état stabilisé pour les applications Temporisées dédié au test progressif par Parties
DFL	Diagramme Fonctionnel Logique
EP	Mémoire à En Prioritaire
HP	Mémoire à Hors Prioritaire
TON	Temporisation à l'excitation (ON)
TOF	Temporisation à la désexcitation (OFF)
TOR	Variable Tout Ou Rien
TPPP	Test Progressif Par Parties

8. Annexes

8.1. Annexe 1 : Exemple d'état stable que Testminator ne permet pas de détecter

Sur l'exemple suivant, on réalise un ET logique entre deux créneaux (le premier de 5 s et le second de 3 s décalé de 7 s par la temporisation TON 1), en entrée E d'une mémoire à EN prioritaire.

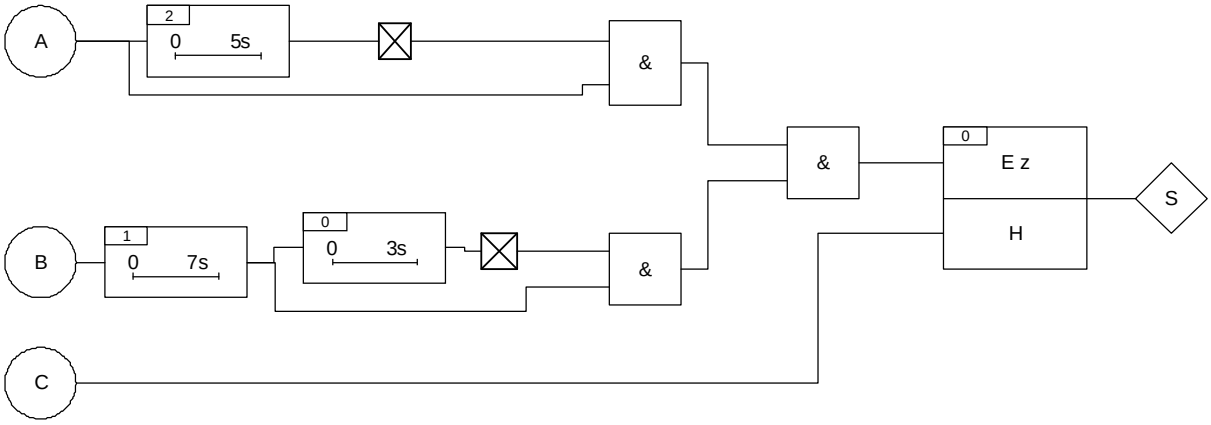


Figure 6 : Exemple de système problématique

Voici un extrait de fichier .MESSAGES généré par le Testminator…

Génération du jeu de tests non temporisé...

ERREUR NON BLOQUANTE : La mémoire en amont 1_0SR n'a pas de vecteurs Set. Mémoire non testée !

Génération du jeu de tests non temporisé...[OK]

Nombre de vecteurs non temporisés : 0

ERREUR NON BLOQUANTE : Détection de 3 porte(s) inactive(s). Possible masquage des structures.

ERREUR NON BLOQUANTE : Branche inactive : ET(ET(TON(1_1TN),NON(TON(1_0TN))),ET(A,NON(TON(1_2TN)))). Possible masquage des structures en amont de la (des) sortie(s) - S.

ERREUR NON BLOQUANTE : Branche inactive : ET(TON(1_1TN),NON(TON(1_0TN))). Possible masquage des structures en amont de la (des) sortie(s) - S.

ERREUR NON BLOQUANTE : Branche inactive : ET(A,NON(TON(1_2TN))). Possible masquage des structures en amont de la (des) sortie(s) - S.

…ainsi que le plan de test généré par Testminator

Test de la temporisation 1_1TN au regard de la sortie S (Algorithme AT version 2 : 1 séquence(s) de test)

A B C S

Cas coché [X,VE] = [1_0SR:0 , B:1] Q=0

0 0 1 0

0 0 0 0

0 1 0 T:10000MS;0MS:0;10000MS:0

Test de la temporisation 1_0TN au regard de la sortie S (Algorithme AT version 2 : 1 séquence(s) de test)

A B C S

Cas coché [X,VE] = [1_0SR:0 , B:1] Q=0

0 0 1 0

0 0 0 0

0 1 0 T:10000MS;0MS:0;10000MS:0

Test de la temporisation 1_2TN au regard de la sortie S (Algorithme AT version 2 : 1 séquence(s) de test)

A B C S

Cas coché [X,VE] = [1_0SR:0 , A:1] Q=0

0 0 1 0

0 0 0 0

1 0 0 T:5000MS;0MS:0;5000MS:0

Testminator n'a pas identifié de vecteur Set pour la mémoire mais avertit néanmoins de la présence possible de masquages de structure. Testminator considère en effet que la commutation des entrées se fait instantanément depuis un état stable.

Cependant on observe que l'enclenchement de la mémoire depuis un état où elle est à 0 est possible. Si l'on joue à l'instant T=0s le vecteur [ABC] = [010] puis à T=6s le vecteur [110], les deux créneaux se rejoignent pendant 3s à T=7s, permettant ainsi l'enclenchement de la mémoire.

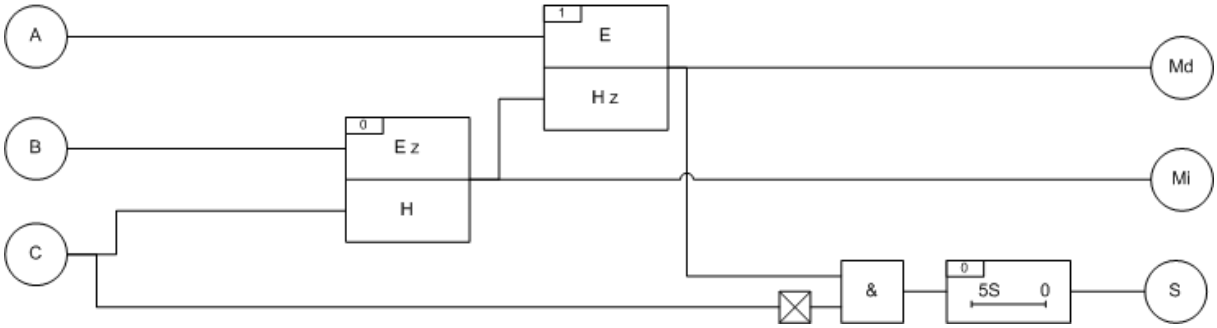
8.2. Annexe 2 : Justification du choix fait de positionner E et non E_p dans l'algorithme ATP

Dans les travaux, la question s'est posé de savoir s'il était nécessaire de simuler le système à tester à partir de tous les états globaux ou seulement de tous les états partiels.

Le problème est qu'à partir d'un même état partiel, avec un même vecteur en entrée, le système peut évoluer différemment pour des états globaux différents. Il est donc nécessaire de simuler à partir de tous les états globaux du système, mais les états partiels restent les seuls à considérer pour juger de la couverture du test.

Cet exemple présente deux états E ayant les mêmes états réduits E_p pour une sortie S à tester, mais qui par application des mêmes vecteurs d'entrée conduisent à des états réduits différents.

Soit l'application suivante :



La table des états atteignables du système et les séquences de vecteurs associées à chacun d'entre eux, obtenus par application de l'algorithme BTP, sont :

Etat atteint	Md	Mi	T0	S	Séquence d'atteinte	
E3	0	1	D	0	010.	
E4	1	0	D	0	101.	
E5	1	0	A	1	101.	000.
E7	0	0	D	0	010.	001.

Les deux états E3 et E7 ont les mêmes états réduits $E_p=(M_d, T_0)= (0,D)$.

Lors de la construction des tables par l'algorithme ATP, le jeu en entrée du vecteur 100 (A=1, B=0, C=0) conduit, à partir de l'état E3, à l'état partiel $E_p=(0,D)$ tandis que le jeu de ce même vecteur, depuis l'état E7, conduit à l'état partiel $E_p=(1,A)$ en 5 secondes.

Cet exemple justifie le choix fait dans l'algorithme ATP de positionner non pas l'état E_p mais l'état E, lors de la détermination des séquences de test. En effet le positionnement uniquement des états Ep, qui conduit à une réduction des états de départ considérés (par exemple ici, on pourrait ne considérer que 3 (E3, E4, E5) des 4 états atteignables (E3 et E7 ayant les mêmes états réduits) déterminés par l'algorithme BTP), pourrait faire passer à coté de certaines séquences de tests.

8.3. Annexe 3 : Optimisation de l'enchaînement des vecteurs de test des séquences de test RISI…RSRS..

Les états stables du système ainsi que leur séquence de positionnement sont obtenus par un algorithme BTP. Cet algorithme permet notamment de « typer » l'ensemble des vecteurs d'entrée en fonction de l'influence qu'ils ont sur la sortie à tester : on appelle vecteur Reset un vecteur faisant passer la sortie à tester à 0 depuis un état de la sortie à 1, vecteur Set un vecteur faisant passer la sortie à tester à 1 depuis un état de la sortie à 0, vecteur Inchangé un vecteur qui laisse inchangée une sortie à 0 depuis un état à 0 et laisse cette même sortie inchangée à 1 depuis un état à 1. La valeur de S considérée est celle correspondant à l'état stabilisé E_k.

Dès la première version du prototype Testminator, les algorithmes développés permettent d'abord de tester une mémoire, qui est une sortie particulière car toutes les mémoires sont instrumentées. Pour tester une sortie simple qui n'est pas une mémoire, ces algorithmes sont réutilisés en transformant la sortie simple à tester en une mémoire à enclenchement prioritaire dont le déclenchement est toujours à 1 [6] [4]. Dans cette transformation d'une sortie simple en mémoire, soit on a 0 en entrée de l'enclenchement prioritaire et le déclenchement à 1 met la sortie à 0, soit on a 1 en entrée de l'enclenchement prioritaire qui met la sortie à 1. Il n'existe pas de cas dans cette transformation où la mémoire remplaçant la sortie simple a en entrée 0 à la fois sur l'enclenchement et sur le déclenchement (vu que le déclenchement est toujours à 1, donc il n'existe pas de vecteur de type I pour une sortie autre qu'une mémoire).

Pour une vraie mémoire à tester (donc aussi une sortie mais qui n'appartient pas au cas précédent), qu'elle soit à enclenchement prioritaire ou au déclenchement prioritaire, on peut trouver des cas où on applique 0 en enclenchement et 0 en déclenchement. Dans ce cas, la mémoire conserve son état : depuis un état où la mémoire est à 0 le vecteur utilisé en entrée laisse inchangée à 0 la mémoire (donc sa sortie) et depuis un état où la mémoire est à 1 le vecteur utilisé en entrée laisse inchangée à 1 la mémoire, ces deux possibilités n'étant bien sûr pas simultanées (soit la mémoire est à 0 soit elle est à 1)

Dans le premier développement, la séquence de test d'une sortie consiste à enchaîner les vecteurs de test de la façon suivante :

- Jeu d'un vecteur Reset (passage de 1 à 0 de la sortie à tester), depuis un état initial de la sortie à 1,
- Jeu d'un vecteur Inchangé (la sortie reste à 0),
- Jeu d'un vecteur Set (passage de 0 à 1 de la sortie à tester), depuis un état initial de la sortie à 0,
- Jeu du même vecteur Inchangé (la sortie reste à 1),
- Jeu d'un nouveau vecteur Reset…
- Etc.

On se trouve ici dans le cas où il existe plusieurs vecteurs Set, Reset et Inchangé pour la sortie à tester. On se reportera à [6], pour les cas particuliers.

On obtient des séquences RISIRIS..RSRSRS.

Une des optimisations proposée pour le nouveau développement consiste à réarranger les vecteurs de test de façon à tester successivement tous les vecteurs laissant inchangée la sortie (à 0 puis à 1). On obtient alors des séquences de test de type RII..IISII..IIRSRSR permettant de réduire la taille du jeu de tests.

Prenons pour exemple une sortie pour laquelle ont été identifié un très grand nombre de vecteurs la laissant inchangée et seulement quelques vecteurs Set et Reset. La nouvelle séquence de test

permet de jouer successivement tous les vecteurs Inchangés depuis un état de la sortie à 0, puis après passage d'un vecteur Set, de jouer tous les vecteurs Inchangés depuis un état de la sortie à 1. Cette optimisation évite à l'oracle de repositionner la sortie tantôt à 0, tantôt à 1 avec des vecteurs Set et Reset déjà testés pour le test successif de chacun des différents vecteurs Inchangés.

Organisation des tests en choisissant comme état de départ l'état d'arrivée du test précédent

Lors de la construction de la séquence de test, il est possible d'avoir un choix entre plusieurs vecteurs de test, quels que soient leurs types (S, R, I). Dans ce cas, il est utile de choisir un vecteur de test pour lequel l'état global E à positionner correspond à l'état final E^k de l'étape de test précédente s'il existe. Ceci permet d'économiser l'étape de positionnement de l'état E pour la séquence de test associée au nouveau vecteur.