UNIVERSITÉ DU QUÉBEC EN OUTAOUAIS

GÉNÉRATION AUTOMATIQUE DE TESTS À PARTIR DE LA SPÉCIFICATION DES BESOINS

MÉMOIRE
PRÉSENTÉ
COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN INFORMATIQUE

PAR
LUIS CARDENAS

AOÛT 2007

UNIVERSITÉ DU QUÉBEC EN OUTAOUAIS

Département d'informatique et ingénierie

Ce mémoire intitulé :

GÉNÉRATION AUTOMATIQUE DE TESTS À PARTIR DE LA SPÉCIFICATION DES BESOINS

présenté par
Luis Cardenas

Pour l'obtention du grade de maîtrise en science (M.Sc)

a été évalué par un jury composé des personnes suivantes :

Dr. Michal Iglewski …….Directeur de recherche
Dr. Luigi Logrippo ……Président du jury
Dr. Kamel Adi ……Membre du jury

Mémoire accepté le : 27 août 2007

Remerciements

Je tiens à dire combien le soutien quotidien de mon épouse, ma fille et mon enfant ont été important, je leur dois beaucoup.

Je tiens à remercier le professeur Michal IGLEWSKI pour sa grande disponibilité, et ses judicieux conseils prodigués tout au long de ce mémoire.

Je remercie également mes amis et toutes les personnes qui m'ont aidé par leurs conseils et par la révision de ce mémoire.

Table des matières

Liste des figures

Liste des tableaux

Résumé

Le processus pour tester un système contient les étapes suivantes : l'identification de la partie du système à évaluer, la construction des scénarios pour tester l'application, la détermination du résultat attendu pour chaque scénario, l'obtention du résultat réel après avoir exécuté le scénario par l'application, la comparaison entre la value attendue et la valeur réelle et déterminer quand finir le processus de test. Chaque fois, lorsqu'il y a un changement dans la spécification ou dans le code du système, le testeur doit répéter le processus de test. Souvent, le testeur doit travailler avec des spécifications informelles ou avec des systèmes complexes. La construction des scénarios dans ces cas n'est pas évidente.

Dans ce mémoire nous présentons un nouvel algorithme de génération des scénarios à partir d'une spécification formelle SCR (Software Cost Reduction). Avant de générer les scénarios, la spécification SCR est convertie en une structure de graphe EFSM (Extended Finite State Machine), ensuite l'algorithme construit les scénarios et détermine quelles sont les exigences qui sont atteignables et quelles sont les exigences qui ne le sont pas.

Nous présentons aussi un nouvel algorithme qui minimise un ensemble de scénarios donné et garantit que chaque exigence du système sera testée au moins une fois par un scénario. L'algorithme de minimisation utilise un nouveau critère basé sur le nombre minimal des événements contenus dans l'ensemble des scénarios.

Abstract

The system test process has the following steps : identify the part system to test, build the scenarios to test the application, determine the expected result for every scenario, obtain the real result after running the scenario by the application, compare the expected and the real values and determine when the test process ends. Every time when there is a change in the specification or in the code, the tester has to repeat the test process. Often, the tester has to work with informal specifications or with complex systems. The construction of scenarios for these cases is not so evident.

In this thesis we present a new algorithm to generate scenarios from a formal specification SCR (Software Cost Reduction). Before generating scenarios, the SCR specification is converted in a graph structure EFSM (Extended Finite State Machine), and then the algorithm builds the scenarios and determines which are the reachable requirements and which ones are not.

Also, we present a new algorithm that minimises a given set of scenarios and guarantees that every system requirement will be tested at least once by a scenario. The minimization algorithm uses a new criterion based on the minimal number of events contained in the set of scenarios.

Chapitre 1

Introduction

Le processus de test des applications est un sujet important pour garantir la qualité du système final. Il y a plusieurs approches de test d'un système telles que des méthodes de boîte noire ou des méthodes de boîte blanche. Une méthode de boîte noire consiste à tester le comportement extérieur de l'application sans connaître l'implémentation des composantes et une méthode de boîte blanche est basée sur nos connaissances du code des composantes. Le processus de test est une tâche complexe. Le testeur doit comprendre la spécification des besoins du système qui souvent est décrite de manière informelle, comprendre le fonctionnement du logiciel, déterminer quelle partie du système doit être testée, créer et exécuter les scénarios, comparer les résultats produits par le logiciel avec les résultats attendus calculés sur la base de la spécification des besoins du système et déterminer quand arrêter le processus de test.

Dans notre travail de recherche, nous commençons par la définition de la spécification des besoins ou des exigences du système, (termes équivalents), ensuite nous proposons une méthode pour générer des scénarios à partir des exigences d'un système et un algorithme pour la minimisation des scénarios.

Nous utilisons des outils SCR (Software Cost Reduction) [10] pour définir la spécification de besoins d'un système de manière formelle. La méthode SCR a été créée par le NRL (Naval Research Laboratory), elle est surtout appliquée dans le domaine du développement de systèmes en temps réel et d'applications critiques tels que des systèmes d'armes, des programmes de vol et des systèmes de commande pour des réacteurs nucléaires.

Une fois que nous avons une spécification SCR, l'étape suivante est la génération des scénarios. Chaque scénario est composé d'un ou de plusieurs couples. Chaque couple décrit un événement d'entrée : une variable d'entrée et sa nouvelle valeur. L'étape finale sera de minimiser l'ensemble de scénarios.

Les objectifs généraux que nous nous sommes fixés sont de :

1. Concevoir une méthode et créer un prototype pour générer automatiquement des scénarios de test à partir de la spécification d'un système utilisant la méthode SCR.
2. Trouver une stratégie pour minimiser le coût de tests générés pour une spécification SCR donnée.

L'algorithme pour la génération automatique de scénarios de test doit inclure les étapes suivantes :

1. convertir une spécification SCR dans une machine d'états pour représenter les besoins du système ou exigences de la spécification SCR dans une même structure;
2. parcourir la machine d'états en utilisant un critère de couverture pour générer les scénarios du système, où l'ensemble des scénarios doit couvrir au moins une fois chaque exigence du système;
3. déterminer le taux de couverture de test de la spécification et identifier les états inaccessibles;
4. minimiser l'ensemble de scénarios en fonction du nombre total d'événements et possiblement, en fonction du nombre de scénarios;
5. analyser et juger l'accessibilité d'états dans le contexte du critère de couverture choisi.

Dans ce document nous présenterons d'abord la définition du problème dans le chapitre 1, ensuite une revue de la littérature sur la méthode SCR et la génération des scénarios dans le chapitre 2, puis le chapitre 3 est consacré à trouver une représentation commune des éléments qui composent une spécification SCR, après ça le chapitre 4 décrit l'algorithme de la génération des scénarios, ensuite le chapitre 5 contient la description de l'algorithme de minimisation des scénarios et finalement les conclusions se trouvent dans le chapitre 6.

À la suite du chapitre 1, nous présenterons le contexte du problème, le problème à résoudre et les bénéfices de trouver une solution.

1.1 Le contexte du problème

La vérification d'un logiciel consiste à déterminer si son implémentation répond aux exigences de la spécification. Elle commence par la génération de scénarios (séquences d'événements) à partir de la spécification, ensuite un oracle analyse chaque scénario et calcule la sortie attendue. L'implémentation du système (le logiciel) est exécutée avec le même scénario pour capturer la sortie réelle et la comparer à la sortie attendue. Advenant des différences entre les sorties, cela indique une violation de la spécification ou une erreur dans l'implémentation du système. S'il n'y a pas de différences, le scénario satisfait la spécification et le processus de vérification doit se répéter avec le scénario suivant (Voir FIG. 1).

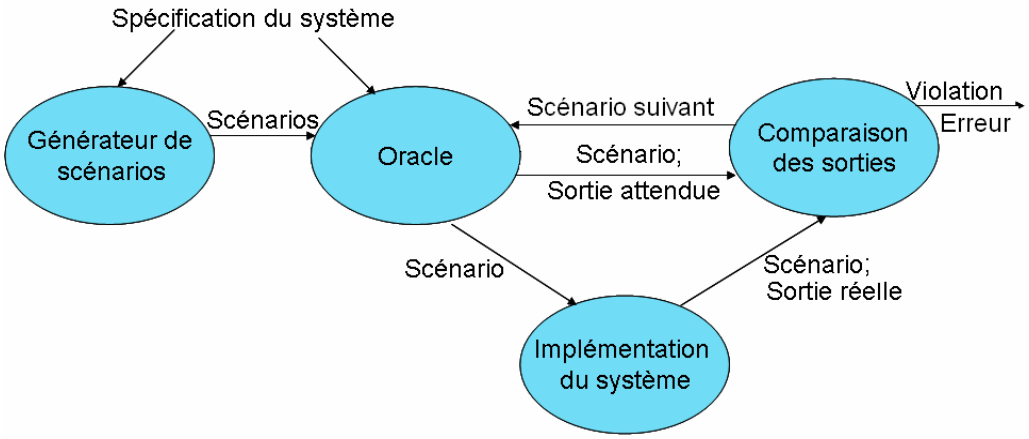


FIG. 1 - La vérification d'un logiciel

Le processus de vérification des applications contient plusieurs étapes complexes qui impliquent des efforts humains significatifs.

1.2 Le problème à résoudre

Notre projet est divisé en deux volets :

1. trouver une méthodologie pour générer automatiquement des scénarios de test à partir de la spécification d'un système; la méthodologie de génération de scénarios est présentée dans le chapitre 4.
2. trouver une stratégie pour minimiser l'ensemble de scénarios qui couvre « la majorité des cas ». Chaque scénario est composé d'un ou de plusieurs

événements, alors il s'agit de trouver le nombre minimal d'événements contenus dans l'ensemble des scénarios couvrant les exigences du système. La stratégie de minimisation de scénarios est détaillée dans le chapitre 5.

Avant de commencer la génération des scénarios, le générateur et l'oracle ont besoin d'identifier la représentation du comportement du système à travers une spécification. Parfois, cela n'est pas évident parce que dans la majorité des cas, la spécification se fait d'une manière informelle, ce qui complique le test d'un système; pour cette raison, nous avons choisi la méthode SCR (Software Cost Reduction) qui nous aidera à élaborer la définition formelle de la spécification d'un système.

1.3 Les bénéfices de trouver une solution à notre problème

La génération automatique de scénarios, à partir d'une spécification SCR, et ensuite leur exécution, donnent les bénéfices suivants :

1. l'obtention de manière automatique d'un ensemble de scénarios à partir de la spécification du système;
2. la réduction de l'effort humain et du temps de génération de scénarios;
3. la détermination du taux de couverture de test de la spécification;
   l'identification des violations du système à travers des contre-exemples.

Chapitre 2
État de l'art

2.1 Introduction

Dans ce chapitre, nous abordons quelques approches présentées dans la littérature qui font référence à la formalisation des spécifications en utilisant SCR et la génération des scénarios. Dans les chapitres trois et quatre, nous expliquerons les éléments que nous avons pris de l'état de l'art pour développer notre travail de recherche.

Dans le but de pouvoir les analyser, nous commençons par une description plus détaillée de la méthode SCR qui constituera le point de référence pour notre recherche. Pour mieux comprendre, nous expliquerons les concepts de base SCR avec la spécification SIS (Safety Injection System).

Ensuite, nous présenterons la génération des scénarios en utilisant les vérificateurs de modèles spin et SMV. Le résultat final est la comparaison de la génération des scénarios à l'aide de spin et SMV pour quatre spécifications. Une autre approche de génération de scénarios à partir d'une spécification SCR est T-VEC un outil commercial qui construit les scénarios à partir des valeurs possibles des variables contrôlées jusqu'aux variables surveillées.

La génération des scénarios est basée sur la représentation des exigences du système dans une machine d'états étendus, l'avantage de cette machine est d'appliquer un algorithme qui a comme résultat un ensemble de scénarios qui couvre au moins un fois chaque exigence du système.

2.2 La méthode SCR

La méthode SCR a été créée par le NRL (Naval Research Laboratory), elle est surtout appliquée dans le domaine du développement de système en temps réel, à grande échelle. Tout a commencé lorsqu'un groupe de chercheurs a décidé d'évaluer l'utilité et l'étendue des principes d'ingénierie de logiciel en appliquant les principes de reconstruction de software sur un système réel, le programme de vol opérationnel OFP (Operational Flight Program) pour l'avion A7. Le résultat de cette recherche a produit un ensemble de techniques pour la conception de systèmes de logiciel tels que la spécification des besoins, la conception de systèmes, les interfaces de modules, ainsi que le modèle de quatre variables pour la définition du comportement du système.

La méthode SCR est supportée par différents outils pour la modélisation du comportement d'un système, dès la définition jusqu'à la vérification de la spécification [10]. Les outils de SCR avec ses principales fonctions sont :

1. l'éditeur de spécifications : sert à créer et modifier la spécification du système;
2. le graphe de dépendance : sert à visualiser la dépendance des variables dans la spécification;
3. le vérificateur de consistance : sert à détecter les erreurs de la spécification;
4. le simulateur : sert à valider la spécification. Il calcule les valeurs de variables de sortie à partir des variables d'entrée;
5. les outils de vérification : servent à identifier les violations de propriétés de système telles que les propriétés de sécurité. Les outils sont: Spin, TAME (Time Automata Modeling Environment) et Salsa.

Le comportement du système est représenté à travers quatre types de variables: surveillées, contrôlées, entrées et sorties, et quatre relations : NAT, REQ, IN et OUT [11].

Le système est spécifié en deux étapes : d'abord, le comportement « idéal » du système est spécifié, c'est-à-dire NAT et REQ sont définis comme si le système capturait la valeur exacte de la variable surveillée et calculait la valeur exacte de la variable contrôlée. La deuxième partie utilise les relations IN et OUT pour

représenter la tolérance d'inexactitude (la marge d'erreur) quand la variable surveillée est capturée en utilisant un dispositif d'entrée et la valeur calculée est assignée à la variable contrôlée à travers un dispositif de sortie.

La relation NAT décrit les suppositions que sont les contraintes imposées aux variables surveillées et contrôlées par les lois physiques et l'environnement du système. REQ décrit tous les aspects que le système doit contrôler.

Maintenant, nous décrivons la *spécification des besoins* qui est une technique pour la conception des systèmes en tenant compte du comportement idéal du système.

D'abord, les éléments de base pour représenter la spécification des besoins sont: les types de variables, les variables elles-mêmes, les classes de mode, les assertions, les suppositions, les conditions, les événements et les fonctions.

Pour mieux comprendre, nous expliquerons les concepts de base avec la spécification SIS (Safety Injection System) [9]. Elle est composée d' un système de trois détecteurs de pression d'eau qui permet au liquide de refroidissement d'atteindre le cœur du réacteur quand la pression tombe en dessous d'une certaine limite. L'opérateur du système peut bloquer l'injection de refroidissement «Safety_Injection» en activant le bouton «Block» et il peut aussi réactiver le système en appuyant sur le bouton «Reset».

2.2.1 Types de variables

Les spécifications des besoins utilisent principalement deux variables : les surveillées et les contrôlées. En plus, elles utilisent des variables auxiliaires (termes) pour simplifier la définition de la spécification. La description des variables est la suivante :

1. surveillées : les entités observées par le système;
2. contrôlées : les entités contrôlées par le système;
3. termes : les variables auxiliaires pour définir le comportement du système.

Exemple SIS : les variables surveillées sont : la pression de l'eau «WaterPress», «Block» et «Reset» puis la variable contrôlée est «Safety_Injection» et le terme est «Overridden» qui sera défini plus tard comme fonction.

2.2.2 Classes de mode

La logique de la spécification est représentée en utilisant une machine d'états, où une classe de mode correspond à une relation d'équivalence et un mode est un ensemble d'états équivalents selon cette relation. Il est à noter que nous pouvons être intéressés par différents aspects du système et par conséquent, nous définirons différentes relations d'équivalence menant à différentes classes de mode. Exemple SIS : l'aspect qui nous intéresse dans le système est la pression de l'eau dans un des trois intervalles : l'intervalle bas, l'intervalle permis et l'intervalle haut. La classe de mode s'appelle «M_Pressure» et les modes sont : «TooLow», «Permitted », «High».

2.2.3 Assertions

Propriétés que le système est supposé respecter en tout temps.

2.2.4 Suppositions

Contraintes externes qui sont imposés aux variables surveillées et contrôlées par les lois physiques et l'environnement du système.

2.2.5 Conditions

Prédicats définis sur une ou plusieurs entités (variables surveillées et contrôlées, termes et mode courant).

2.2.6 Événements

Changements observables de variables surveillées. Nous utiliserons la notation suivante pour décrire les événements :

$$@T(c) \text{ WHEN } d \overset{def}{=} \neg'c \wedge c' \wedge d,$$

$$@F(c) \text{ WHEN } d \overset{def}{=} 'c \wedge \neg c' \wedge d,$$

c et d sont des conditions.

'c : l'ancienne valeur.

c' : la nouvelle valeur.

Exemple SIS : l'événement @T (Block = On) WHEN Reset = OFF est défini par ¬'Block=ON ∧ Block'=ON ∧ 'Reset = OFF, c'est-à-dire, le résultat sera vrai quand la variable «Block» change de «OFF» à «ON» quand Reset=OFF.

2.2.7 Les fonctions

Le comportement du système est représenté principalement en utilisant le concept de machine d'états et de fonctions de transition de mode, d'événement et de condition. Chaque fonction a la forme d'une table.

1. La fonction de transition de mode : génère comme résultat un nouveau mode en fonction d'un événement et des valeurs de variables dans l'ancien mode. La représentation d'une fonction de transition de mode sous forme d'une table est la suivante :

Ancien mode	Événement	Nouveau mode
M_1	E_1	O_1
. . .	. . .	. . .
m_n	E_n	O_n

TAB. 1 - La fonction de transition de mode

La définition de la fonction de transition de mode est la suivante :

$$f(e_1, \ldots, e_p, m) \overset{def}{=} O_i \textbf{ SI } m=m_i \wedge E_i (e_1, \ldots, e_p, m)$$

f : la fonction définissant la transition de mode

$e_1, \ldots, e_p$: les entités (variables surveillées, contrôlées, termes)

$m_1, \ldots, m_p$: les anciens modes

O_1 , ... , O_p : les nouveaux modes

m : le mode courant

E_1 , ... , E_p : les événements définis en fonction des entités et du mode courant.

Exemple SIS : la fonction de transition de mode (FIG. 2) fait référence à la classe de mode «M_pressure», laquelle contient les modes «TooLow», «Permitted», et «High». Cette fonction représente une machine d'états où chaque état est un mode et la transition est associée à l'événement. Pour passer de «TooLow» à «Permitted», le prédicat @T(WaterPres>=low) doit être vrai. «low» et «permit» sont des constantes, 900 et 1000 respectivement.

Ancien mode	Événement	Nouveau mode
TooLow	@T(WaterPres>=low)	Permitted
Permitted	@T(WaterPres<low)	TooLow
Permitted	@T(WaterPres>=permit)	High
High	@T(WaterPres<permit)	Permitted

FIG. 2 - La fonction de transition de mode (M_Pressure)

2. La fonction d'événement : définit une variable (terme ou variable contrôlée) dans le nouveau mode en fonction d'un événement et des valeurs de variables dans l'ancien mode.

Modes	Événements		
m_1	E_{11}	. . .	E_{1k}
. . .	. . .	. . .	. . .
m_n	E_{n1}	. . .	E_{nk}
Var	r_1	. . .	r_k

TAB. 2 - La fonction d'événement

La définition de la fonction d'événement est la suivante :

$$f(e_1 , \dots , e_p, m) \overset{def}{=} r_j \ \textbf{SI}\ m=m_i \wedge E_{ij} (e_1 , \dots , e_p, m)$$

f : la fonction définissant l'événement.

e_1,…,e_p : les entités (variables surveillées, contrôlées ou termes)

m_1 , ... , m_p : les modes

m : le mode courant

E_{11} , ... , E_{nk} : les événements définis en fonction des entités et du mode courant

r_1 , ... , r_k : les valeurs possibles de la fonction.

Modes	**Événements**	
High	Never	@T(Inmode)
TooLow, Permitted	@T(Block=ON) WHEN (Reset=OFF)	@T(Reset=ON)
Overriden	TRUE	FALSE

FIG. 3 - La fonction d'événement (Overriden)

La notation @T(Inmode) utilisée dans cet exemple est équivalente à l'expression :

'M_pressure <> High Λ M_Pressure' = High

3. La fonction de condition : définit une variable (Var) en fonction des valeurs d'autres variables pour chaque mode. La représentation d'une condition sous forme d'une table est la suivante :

Modes	**Conditions (Expressions booléennes)**		
M_1	C_{11}	. . .	C_{1k}
. . .	. . .	. . .	. . .
m_n	C_{n1}	. . .	C_{nk}
Var	r_1	. . .	r_k

TAB. 3 - La fonction de condition

La définition de la fonction de condition est la suivante :

$$f(e_1 , \dots , e_p, m) \overset{def}{=} r_j \ \textbf{SI}\ m=m_i \wedge C_{ij}\ (e_1 , \dots , e_p, m)$$

f : la fonction définissant la condition.

$e_1,\dots,e_p$: les entités (variables surveillées, contrôlées ou termes)

m_1 , ... , m_p : les modes

m: le mode courant

C_{11} , ... , C_{nk} : les conditions définies en fonction des entités, des modes et du mode courant.

r_1 , ... , r_k : les valeurs possibles de la fonction.

Exemple SIS : La fonction de condition fait référence à la classe de mode «M_Pressure», laquelle contient les modes «TooLow», «Permitted» et «High». La fonction «Safety_Injection», est une variable contrôlée et peut prendre la valeur «ON» ou «OFF» en fonction du mode courant et de la valeur du prédicat associé.

Modes	**Conditions**	
High, Permitted	FALSE	TRUE
TooLow .	NOT OVERRIDDEN	OVERRIDDEN
Satety_injection	ON	OFF

FIG. 4 - La fonction de condition (Safety_injection_system)

2.3 La génération de scénarios

Dans le travail [7], la spécification du système est faite en SCR et la génération des scénarios est faite à l'aide des vérificateurs de modèles spin et SMV. L'article compare les deux vérificateurs en utilisant quatre spécifications de taille différente : une spécification relativement simple de SIS (Safety Injection System), une version enrichie de SIS, Cruise Control System et WCP1.

Un outil, supporté par SCR, peut traduire la spécification en langage d'un vérificateur de modèle (spin ou SMV) et pour cette raison, nous pouvons utiliser le vérificateur pour construire les scénarios, calculer les sorties attendues et détecter les états qui ne sont pas atteignables.

Il y a deux types de génération de scénarios : de propriété et de spécification opérationnelle.

La première méthode est :

2.3.1 La génération de scénarios de propriété.

Dans la génération de scénarios de propriété, le vérificateur de modèle doit déterminer si l'ensemble des propriétés défini par l'usager est satisfait. Le principal problème est qu'un ensemble de propriétés ne peut pas garantir la couverture « totale » du système, c'est-à-dire, la spécification du système est exprimée en plusieurs propriétés, alors une propriété ou un sous-ensemble de propriétés ne

couvrent pas la totalité de la spécification du système. Par exemple, si nous avons la propriété suivante : (@T(WaterPres<900) WHEN (Block=ON AND Reset=OFF AND Pressure=TooLow)) implique (SI'=OFF), un exemple de scénario de longueur 6 qui satisfait cette propriété est composé par les événements suivants: (Reset=OFF, WaterPres=500, WaterPres=800, WaterPres=1000, Block=ON, WaterPres=800); c'est-à-dire, que après de déclenche une série des événements la propriété devient vrai, voir le tableau (TAB. 4).

Le scénario qui satisfait cette propriété est généré de la manière suivante :

1. Le pas 0 est l'état initial du système avec WaterPres=200, Block=OFF, Reset=ON, Safety_injection=ON, Pressure= TooLow.
2. Le pas 1 déclenche l'événement Reset = OFF, alors dans la propriété Reset=OFF est valide. Nous remplaçons l'événement Reset = OFF pour la valeur vrai. La propriété devient :
   (@T(WaterPres<900) WHEN (Block=ON AND TRUE AND Pressure= TooLow)) implique (SI'=OFF)
3. Le pas 2 change WaterPres=500.
4. Le pas 3 change WaterPres=800.
5. Le pas 4 déclenche WaterPres=1000, alors l'événement @T(WaterPres<900 est vrai, qui implique SI=OFF et l'état Pressure est changé de TooLow à Permitted. La propriété devient :
   (TRUE)WHEN(Block=ON AND TRUE AND Pressure=TooLow)) implique (TRUE)
6. Le pas 5 déclenche Block= ON, alors la propriété devient :
   (TRUE) WHEN (TRUE AND TRUE AND Pressure=TooLow) implique (TRUE)
7. Le pas 6 déclenche WaterPressure=800, pour changer Pressure de Permitted à TooLow. La propriété devient :
   (TRUE) WHEN (TRUE AND TRUE AND TRUE) implique (TRUE)

No. Pas	Variables surveillées	Variables contrôlées	Mode ou état
0	WaterPres=200 Block=OFF Reset=ON	Safety_Injection=ON	Pressure=TooLow
1 2 3 4 5 6	Reset=OFF WaterPres=500 WaterPres=800 WaterPres=1000 Block=ON WaterPres=800	Safety_Injection=OFF	Pressure=Permitted Pressure=TooLow

TAB. 4 - La génération de scénarios de propriété

La deuxième méthode est :

2.3.2 Spécification opérationnelle

La génération de scénarios pour une spécification opérationnelle utilise une méthode ne dépendant pas d'un ensemble de propriétés. La méthode fait automatiquement la conversion d'une spécification opérationnelle en spin ou SMV, à partir des tables de transition de modes, des tables d'événements et des tables de conditions.

La spécification en spin ou SMV est représentée sous forme d'une machine d'états où les transitions correspondent aux exigences du système en forme de prédicats. La génération des scénarios se fera en utilisant le critère de couverture de branches qui consiste à parcourir les transitions de la machine d'états.

Dans le cas de la méthode de SCR, le critère de couverture de branches peut être défini comme suit :

1. pour chaque table de condition, chaque condition qui n'équivaut pas à faux est testée au moins une fois;
2. pour chaque table d'événement ou transition de mode, tous les événements sont testés au moins une fois;
3. pour chaque table d'événement ou transition de mode, dans chaque mode, on teste tous les cas où il n'y a pas de changement au moins une fois.

L'outil de génération automatique de scénarios d'une spécification opérationnelle doit suivre les étapes suivantes :

1. La conversion de la spécification SCR dans un langage spin ou SMV;
2. l'identification des différents cas (branches) résultant de l'application du critère de couverture de branches.

L'exemple suivant illustre l'application de ces opérations à la table de transition de mode du système SIS qui a été présenté dans la FIG. 2.

```
if □ Pressure=TooLow
    If □ @T(WaterPres >= Low) -> Pressure' = Permitted     C1
      □ (else) -> Pressure' = Pressure                     C2
    Fi
  □ Pressure=Permitted
    If  □ @T(WaterPres >= Permit) -> Pressure' = High      C3
        □ @T(WaterPres ] <= Low) -> Pressure' = TooLow     C4
        □ (else) -> Pressure' = Pressure                   C5
    Fi
  □ Pressure=High
    If  □ @T(WaterPres < Permit) -> Pressure' = Permitted  C6
        □ (else) -> Pressure' = Pressure                   C7
    fi
fi
```

FIG. 5 - L'identification de prédicats dans une fonction de transition de mode

Chaque cas est traduit en spin ou SMV. La traduction en spin pour C1 et C2 est :

```
if  : : (Pressure=TooLow) ->
        If  : : (WaterPresP >= Low) &&! WaterPres >= Low
              -> PressureP = Permitted;          CasePressure = 1;
            : : else                             CasePressure = 2;
        fi
. . .
fi
```

FIG. 6 - La traduction des prédicats C1 et C2 en spin

où WaterPresP représente la nouvelle valeur de la pression et WaterPres représente son ancienne valeur.

De plus, on ajoute assert(CasePressure!=1) (« trap property ») pour identifier un scénario satisfaisant C1.

3. l'exécution du vérificateur de modèle pour chaque cas identifié dans l'étape 2 pour générer un scénario et vérifier si le nouveau scénario est contenu dans les scénarios déjà trouvés. Si ce n'est pas le cas;
4. enregistrer chaque scénario dans un fichier, sinon écarter le nouveau scénario.

Le résultat d'exécution du générateur de scénarios obtenu, en utilisant les vérificateurs de modèle spin et SMV sur quatre différentes spécifications, a été évalué en utilisant les critères suivants : le nombre de variables surveillées et contrôlées, le nombre de prédicats, le total de scénarios générés, l'ensemble minimal

de scénarios, le temps d'exécution et le total de variables surveillées contenues dans tous les scénarios.

Les résultats de cette comparaison sont :

Spéc.	No. de Var	No. de Préd.	Total scénarios		Ensemble minimal de scénarios		Temps d'exéc		Total de var. surveillées	
			Spin	SMV	Spin	SMV	Spin	SMV	Spin	SMV
Small SIS	6	33	7	14	5	11	73s	3.7s	280	62
Large SIS	6	33	12	14	3	11	165s	4099s	10,529	778
Cruise Cont.	5	27	19	23	7	15	100s	5.1s	245	66
WCP1	55	50	15	--	10	--	500s	∞	4795	--

TAB. 5 - Le résultat de la génération des scénarios obtenu en utilisant spin et SMV

Les conclusions obtenues après l'analyse des résultats sont :

1. spin génère moins de scénarios que SMV, mais la longueur (total des variables surveillées) moyenne d'un scénario en spin est plus grande que celle générée par SMV;
2. pour les spécifications d'envergure, plus grandes en nombre des variables surveillées, contrôlées et des prédicats comme WCP1, SMV dans la colonne Temps d'exécution, consomme toutes les ressources de l'ordinateur quand il est en train de faire la génération automatique des scénarios.

Dans le deuxième tableau, nous présentons la longueur de chaque scénario qui correspond au total des variables surveillées par chaque scénario. Voir le tableau (TAB. 6).

La notation *m(n)* est utilisée pour indiquer qu'il y a *n* scénarios de longueur *m*.

Un jeu de tests efficace satisfait les deux critères (opposés) suivants [6] :

1. avoir le nombre minimum de scénarios et le nombre minimum de données de test (le nombre de variables surveillées);
2. avoir la couverture maximale de la spécification pour identifier les erreurs possibles du système.

Spécifications	Scénarios générés par Spin			Scénarios générés par SMV		
	Ensemble minimal de scénarios	**Prédicats qui ne peuvent pas être déclenchés**	**Total de variables surveillées par chaque scénario**	**Ensemble minimal de scénarios**	**Prédicats qui ne peuvent pas être déclenchés**	**Total des variables surveillées par chaque scénario**
Small SIS	5	1	1, 20, 54, 99, 106	11	1	2, 3(2), 5(2), 6(2), 8(4)
Large SIS	3	1	3082, 3908, 3539	11	1	2, 3(2), 91(2), 101(4)
Cruise Cont.	7	7	31(2), 35, 37(4)	15	7	2(3), 3(3), 5(3), 6(6)
WCP1	10	2?	2(2), 3, 30, 72, 104, 895, 1086, 1292, 1309	--	--	--

TAB. 6 - La longueur de chaque scénario généré par spin et SMV

Nous constatons, après l'analyse des données de la table, que le principal problème de spin est la longueur de scénarios et celui de SMV est la consommation de toutes les ressources de l'ordinateur quand il est en train d'exécuter l'algorithme de génération des scénarios. Ce type de problème est nommé dans la littérature [11] comme une explosion d'états. En conséquence, aucune des deux méthodes ne satisfait la propriété d'un jeu de tests efficace.

2.4 Représentation de la machine d'états finis étendue.

Nous présentons une autre méthode qui est utilisée pour la génération des scénarios. Ce travail [4] comprend la génération automatique de vecteurs fonctionnels en utilisant la représentation de la machine d'états finis étendue (EFSM Extended Finite State Machine) pour circuits séquentiels où l' EFSM est une représentation compacte du modèle traditionnel d'une machine d'états étendue de registres de données (variables de données locales). La méthode de génération de vecteurs garantit la couverture de chaque branche du graphe du système au moins une fois et les vecteurs sont utilisés pour la vérification de la conception, les tests de circuits et l'estimation de la puissance.

Nous commençons par donner quelques définitions de base :

2.4.1 EFSM

Dans cette section, nous présentons les définitions de la machine d'état finis étendue (EFSM), le concept de stabilisation pour éviter l'exposition d'états et l'algorithme pour la génération de scénarios. Dans le chapitre 4 nous proposons un autre algorithme pour éviter l'explosion d'états et la génération des scénarios en utilisant un graphe EFSM.

L' EFSM M est définie comme $M = \{S,I,O,D,T\}$ où S est l'ensemble des états, I est l'ensemble des entrées, O est l'ensemble des sorties, D est l'espace n-dimensionnel $D1x\ldots xDn$ où $D1,\ldots,Dn$ représentant les valeurs des variables locales $R1,\ldots, Rn$ associées à M, appelées registres de données et T est une relation de transition, $T : S \times D \times I \rightarrow S \times D \times O$.

L'EFSM peut être vue comme une représentation compacte d'une machine où les registres de données ne forment pas une partie explicite de l'espace d'état. Les opérations sur les registres de données sont spécifiées dans les transitions d'état.

Nous utilisons un graphe pour représenter l'EFSM. Les sommets correspondent aux états et les arcs dirigés sont représentés par des couples (prédicat, entrée)/ (action, sortie). Si les conditions du prédicat et de l'entrée sont vraies, l'action sera calculée pour définir les nouvelles valeurs de registres de données et de la variable de sortie.

Pour comprendre les concepts du graphe EFSM nous présentons l'exemple suivant : Les états sont S_0, S_1, les variables d'entrée i_1, i_2, la variable de sortie *out*, le registre de données r_1 et les transitions suivantes :

T1 : $S_0 \rightarrow S_1$ (true, i_1=1) /(r_1=i_2, out=0)

T2 : $S_0 \rightarrow S_0$ (true, i_1=0) /(NOP, out=0)

T3 : $S_1 \rightarrow S_1$ (r1<=7, i_1=1) /(r_1+=4,out=1)

T4 : $S_1 \rightarrow S_0$ (r1>=8, i_1=1) /(NOP, out=r_1)

T5 : $S_1 \rightarrow S_1$ (true, i_1=0) /(NOP, out=1)

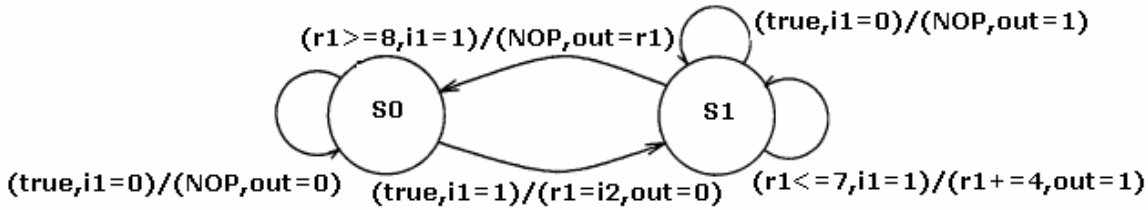


FIG. 7 - Un graphe EFSM

La notation ($r_1 \geq 8$, i_1=1)/(NOP, out=r_1) liée à la transition de S_1 à S_0 veut dire que si $r_1 \geq 8$ et la variable d'entrée i_1=1, aucun changement n'a lieu et la sortie r_1 est produite.

2.4.2 Configuration

Une *configuration* est un couple <s,x> où *s* est un état et $x \in D$.

2.4.3 Bloc

Un *bloc* est un ensemble de configurations qui partagent un même état.

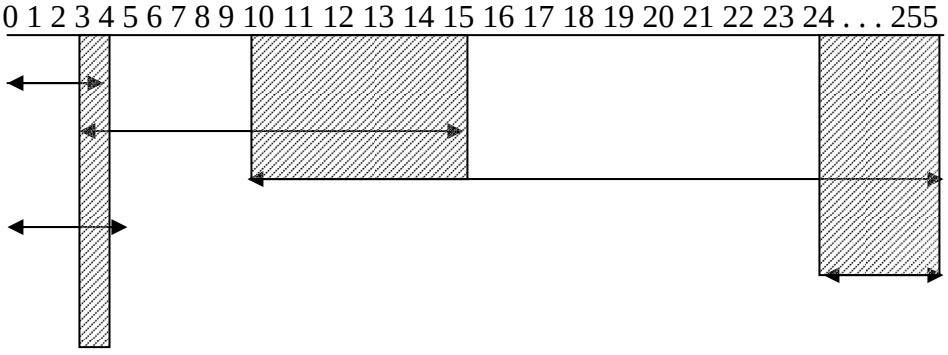
2.4.4 Partition

Une *partition* est un ensemble de blocs disjoints dont l'union est *S x D.*

2.4.5 Orthogonalisation

Le *processus d'orthogonalisation* consiste à trouver tous les intervalles qui sont utilisés pour tester une variable et diviser l'intervalle maximum (la limite inférieure et la limite supérieure) de cette variable dans des intervalles disjoints.

Exemple : La variable x peut prendre les valeurs de [0..255] et la spécification du système utilise les intervalles suivants : {x<5, 2<x≤15, x>9, x<6, x≥24}. Le résultat, après l'application du processus d'orthogonalisation, est le suivant :



{0≤x≤2, 3≤x≤4, x=5, 6≤x≤9, 10≤x≤15, 16≤x≤23, 24≤x≤255}
FIG. 8 - Le résultat du processus d'orthogonalisation

2.4.6 Graphe de transition de blocs

Un graphe de transition de blocs est une EFSM où les sommets sont les blocs d'une partition et les arcs, nommés *transitions de bloc*, sont composés d'un bloc,

d'une entrée et d'un ensemble de couples (bloc suivant, action). Une transition de bloc a la forme suivante :

<$Bloc_1$, entrée, {($Bloc_2$,$Action_2$),…, ($Bloc_n$,$Action_n$)}>

2.4.7 Transition de bloc stable

Une *transition de bloc est stable* s'il y a seulement un bloc dans l'ensemble des blocs suivants. Autrement, elle est instable.

Exemple : un graphe de transition de blocs est composé des blocs B_0 et B_1, et des transitions : BT_1=< B_0, i_1=0, {(B_0, NOP)}> , BT_2 =< B_0, i_1 =1,{(B_1, r_1= i_2)}>, BT_3 =< B_1, i_1=0, {(B_1,NOP)}>, BT_4 =< B_1, i_1=1,{ (B_0, NOP),(B_1, r_1 = r_1+4)}>.

La transition de bloc BT_4 est instable parce qu'il y a deux blocs dans son ensemble des blocs suivants, B_0 et B_1. Voir la figure (FIG. 8).

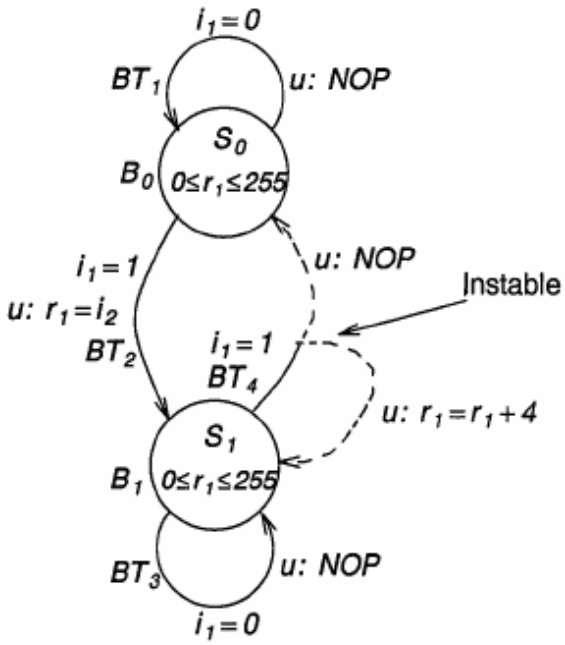


FIG. 9 - La transition de blocs instable

Pour stabiliser une transition, nous devons couper les blocs et les variables d'entrée à l'aide du processus d'orthogonalisation [3].

Exemple : nous pouvons stabiliser le BT_4 en coupant le bloc B_1 en

B_{10}={ S_1,0<= r_1<=7} et

B_{11}={ S_1,8<= r_1<=255}.

Voir la figure (FIG. 9)

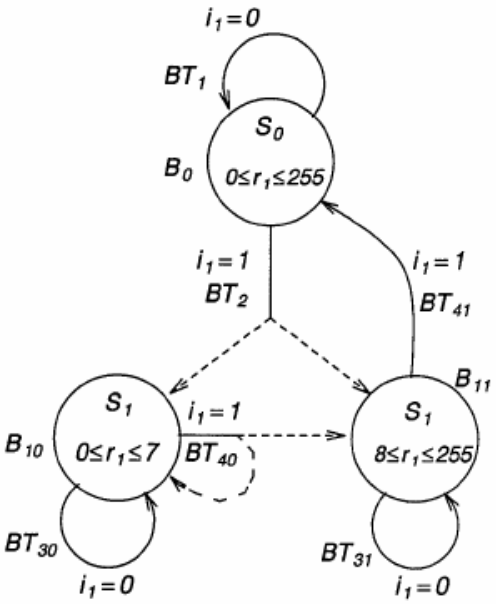


FIG. 10 - La stabilisation d'une transition en coupant les blocs

Après de continuer la stabilisation du graphe EFSM, le résultat final est le suivant:

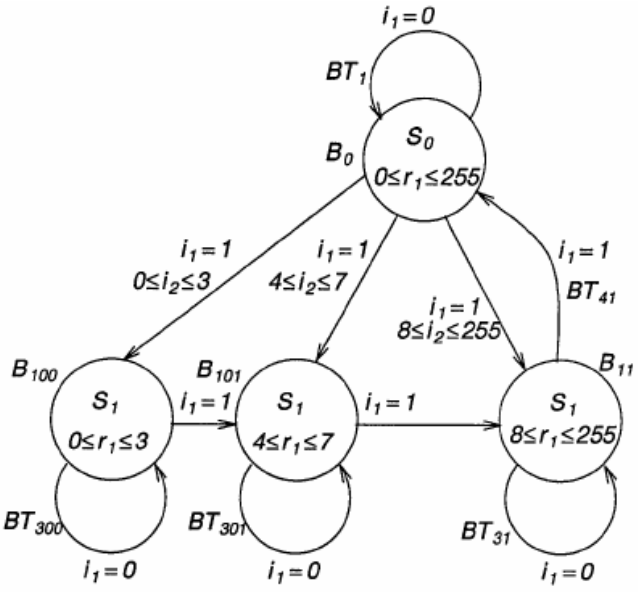


FIG. 11 - Le graphe EFSM stable

2.4.8 Transition semi-stable

Une transition est *semi-stable* si elle est reliée à deux blocs où le bloc courant appartient à l'ensemble des blocs suivants et l'action associée à la transition est une augmentation ou une diminution d'une seule variable. Exemple :

T1 : $S_1 \rightarrow S_1$ (count<9, i_1=0) /(count++)

T2 : $S_1 \rightarrow S_0$ (count=9, i_1=0) /(count=0)

BT1=< S_1, i_1=0, {(S_1, count++), (S_0,count=0)}>

Si nous essayons de stabiliser un graphe avec une transition semi-stable, le résultat est une explosion d'états associés au nombre de fois que nous devons augmenter ou diminuer la variable. Pour éliminer cette explosion, nous arrêtons le processus de stabilisation au moment de trouver une transition semi-stable. Les transitions semi-stables surviennent souvent dans la pratique et le fait que nous les admettons dans le graphe final, peut apporter des gains significatifs au niveau du temps et de la mémoire nécessaire pour représenter le graphe.

2.4.9 Algorithme de génération des scénarios

La génération automatique de vecteurs fonctionnels à partir d'un graphe ESFM stabilisé comporte les étapes suivantes :

1. identifier le bloc contenant l'état initial;
2. exécuter une recherche «breadth-first» pour trouver une transition qui n'a pas encore été visitée et qui est la plus proche du bloc actuel;
3. une fois une telle transition trouvée, on doit ajouter le chemin de recherche à l'ensemble des scénarios et ensuite reprendre la recherche mentionnée dans le paragraphe précédent jusqu'à ce que toutes les transitions soient visitées au moins une fois.

Les résultats de la génération des scénarios en utilisant la représentation d'un graphe EFSM sont : la gestion de l'explosion d'état, l'utilisation de l'orthogonalisation pour stabiliser le graphe et la génération des scénarios en utilisant l'algorithme «breadth-first».

En conclusion, nous pouvons adapter cette approche à la génération de scénarios à partir de la spécification SCR, en profitant des bénéfices de la représentation de l'EFSM selon les aspects suivants :

1. nous pouvons traduire une spécification SCR dans un graphe EFSM, parce que les deux représentent le comportement du système en utilisant les principes de base d'une machine d'états finis;
2. la représentation du comportement du système dans une EFSM nous aidera à réduire le problème d'explosion d'états;
3. la génération de vecteurs, à partir de l'EFSM, nous aidera à trouver la manière de parcourir chaque branche du graphe du système au moins une fois pour la création de scénarios.

2.5 T-VEC

Une autre méthode qui est utilisée pour la génération des scénarios est T-VEC. Cet article [2] fait l'intégration de la spécification des besoins en style SCR avec T-VEC (Test VECtors), le générateur automatique de vecteurs de test. L'outil T-VEC a été utilisé dans le développement de deux systèmes d'avions, lesquels ont été certifiés par la FAA (Federal Aviation Administration). Le générateur de vecteurs détermine les entrées à évaluer, les sorties attendues, et aussi il garantit que chaque exigence de la spécification du système ait au moins une correspondance dans un vecteur de test [1].

Dans une spécification, T-VEC, l'ancien état est caractérisé par une pré-condition et le nouvel état est caractérisé par une post-condition. Les éléments qui constituent cette spécification sont les suivants :

1. des entrées;
2. des sorties;
3. des FR (Functional Relationships) : la description de la sortie attendue dans le nouvel état;
4. des RP (Revelance Predicate) : la représentation des contraintes associées aux entrées dans l'ancien état;
5. des LS (Logic Structure) : l'ensemble des expressions booléennes qui définissent les contraintes utilisées dans un RP.

Nous ferons référence à l'exemple de la spécification SIS (Safety Injection Control System) qui est défini dans la section : méthode SCR, chapitre 2. L'exemple suivant démontre comment la spécification SIS est traduite en une spécification T-VEC. Le comportement du système est défini dans les tableaux suivants : le dictionnaire de types, de constantes, de variables surveillées, de variables contrôlées, de classes de modes, de termes et des fonctions de transition de mode, d'événements et de conditions. Dans les dictionnaires N/A indique Non Applicable. Les tableaux qui contiennent la représentation du comportement du système SIS en utilisant SCR sont les suivants :

Le dictionnaire des types

Nom	Type	Unité	Valeur	Commentaire
Pressure	Integer	psi	[0,2000]	
Switch	Enumerated	N/A	ON,OFF	

Le dictionnaire des constants

Nom	Type	Valeur	Commentaire
Low	Pressure	900	
Permit	Pressure	1000	

Le dictionnaire des variables surveillées

Nom	Type	Valeur Initiale	Exactitude	Interprétation
Block	Switch	OFF	N/A	
Reset	Switch	ON	N/A	
WaterPres	Pressure	14	0.05%	

La déclaration des variables contrôlées

Nom	Type	Value Initiale	Exactitude	Interprétation
Safety_Injection	Switch	OFF	N/A	

Le dictionnaire des classes de mode

Nom	Modes	État initial	Commentaire
M_Pressure	TooLow, Permited, High	TooLow	

Le dictionnaire des terms

Nom	Type	Value Initiale	Exactitude	Commentaire
Overridden	Boolean	TRUE	N/A	

La fonction de transition de mode M_Pressure

État précédent	Événement	État suivant
TooLow	@T(WaterPres >= Low)	Permitted
Permitted	@T(WaterPres <= Low)	TooLow
Permitted	@T(WaterPres >= Permit)	High
High	@T(WaterPres <= Permit)	Permitted

La fonction d'événements Overridden

Nom : Overridden	**Classe de Mode :** M_Pressure	
État	**Événement**	
High	Never	@T(Inmode)
TooLow, Permitted	@T(Block=ON) WHEN Reset = OFF	@T(Inmode) OR @T(Reset=ON)
Overridden =	TRUE	FALSE

La fonction de condition Safety Injection

Nom : Safety_Inj.	**Classe de Mode :** M_Pressure	
État	**Événement**	
High, Permitted	TRUE	FALSE
TooLow	Overridden	Not Overridden
Safety_Injection =	OFF	ON

TAB. 7 - La specification SIS représenté en SCR

Les tableaux suivants sont la traduction que T-VEC produit à partir de la spécification SCR.

2.5.1 Types et constantes

Les types et les constantes sont traduits comme suit :

```
SUBSYSTEM safety_injection1 (WaterPres1, WaterPres2, Reset1, Reset2, Block1,
Block2, Overridden1, Overridden2, Pressure1, Pressure2)
TYPE Switch              ISA ENUMERATION RANGE {OFF=0, ON=1};
TYPE Pressure            ISA INTEGER RANGE {0..2000};
TYPE M_Pressure_Mode     ISA ENUMERATION RANGE {TooLow=0, Permitted=1,
                         High=2};
CONSTANT Low             ISA Pressure VALUE 900;
CONSTANT Permit          ISA Pressure VALUE 1000;
```

TAB. 8 - La traduction des types et des constantes SIS en T-VEC

2.5.2 Variables

Pour chaque variable *v* dans SCR (sauf variables contrôlées), T-VEC déclare les variables v_1 et v_2 correspondant aux valeurs de *v* dans l'ancien et le nouvel état. Voir tableau (TAB. 9)

VARIABLE Safety_Injection	ISA Switch;
VARIABLE WP1	ISA Pressure;
VARIABLE WP2	ISA Pressure;
VARIABLE Reset1	ISA Switch;
VARIABLE Reset2	ISA Switch;
VARIABLE Block1	ISA Switch;
VARIABLE Block2	ISA Switch;
VARIABLE Overridden1	ISA BOOLEAN;
VARIABLE Overridden2	ISA BOOLEAN;
VARIABLE Pressure1	ISA M_Pressure_Mode;
VARIABLE Pressure2	ISA M_Pressure_Mode;

TAB. 9 - La représentation des variables SIS en T-VEC

2.5.3 Expressions LS

Les événements tels les prédicats @T(c), @F(c), @T(c) WHEN d, @F(c) WHEN d, ainsi que le tableau de transition de modes et les termes sont représentés sous forme d'expressions LS. Voir tableau (TAB. 10).

2.5.4 Expressions FR

La variable contrôlée Safety_Injection est spécifiée à l'aide d'une FR. Les prédicats RP.1.* et RP.2.* regroupent les contraintes associées aux deux valeurs possibles de Safety_Injection, On et OFF. Voir le tableau (TAB. 11).

2.5.5 Génération des scénarios

Une fois que la spécification T-VEC a été créée, l'étape suivante est la génération automatique de vecteurs de test. Ce processus contient les étapes suivantes :

1. on normalise les prédicats LS (Logic Structure); la normalisation des prédicats consiste à transformer chaque prédicat sous forme disjonctive :

$$(C_{1,1} \wedge \ldots \wedge C_{1,n}) \vee \ldots \vee (C_{m,1} \wedge \ldots \wedge C_{m,n})$$

où chaque expression $C_{i,j}$ a possiblement une négation et chaque conjonction $C_{i,1} \wedge \ldots \wedge C_{i,n}$ est considérée comme un événement séparé.

2. on associe chaque FR à un ensemble de disjonctions de conjonctions des LS normalisés; chaque disjonction est appelée DCP (Domain Converge Path);

LS1	LOGIC STRUCTURE At_T_Reset_TooLow (Pressure1:M_Pressure_Mode, Reset1:Switch, Reset2:Switch)
	CONSTRAINT(Pressure1=TooLow and Reset1=OFF and Reset2=ON);
LS2	LOGIC STRUCTURE At_T_Reset_Permitted
	(Pressure1:M_Pressure_Mode, Reset1:Switch, Reset2:Switch)
	CONSTRAINT(Pressure1=Permitted and Reset1=OFF and Reset2=ON);
LS3	LOGIC STRUCTURE At_T_Inmode_High (Pressure1:M_Pressure_Mode, Pressure2:M_Pressure_Mode)
	CONSTRAINT (Pressure2 = High and Pressure1 != High);
LS4	LOGIC STRUCTURE At_T_Inmode_TooLow_Permitted
	(Pressure1:M_Pressure_Mode, Pressure2:M_Pressure_Mode)
	CONSTRAINT ((Pressure2 = Permitted or Pressure2 = TooLow) and
	(Pressure1 != Permitted and Pressure1 != TooLow));
LS5	LOGIC STRUCTURE At_T_Block_On (Reset1:Switch, Reset2:Switch, Block1:Switch, Block2:Switch)
	CONSTRAINT (Block2 = ON and Block1 = OFF and Reset1 = OFF and Reset2 = OFF);
LS6	LOGIC STRUCTURE Overridden_Term (Overridden2, Pressure1, Pressure2, Reset1, Reset2, Block1, Block2)
	CONSTRAINT
LS6.1	Overridden2 = false and
LS6.1.1	(At_T_Reset_TooLow(Pressure1, Reset1, Reset2)
LS6.1.2	OR At_T_Reset_Permitted(Pressure1, Reset1, Reset2)
LS6.1.3	OR At_T_Inmode_High(Pressure1, Pressure2)
LS6.1.4	OR At_T_Inmode_TooLow_Permitted(Pressure1, Pressure2))
LS6.2	OR Overridden2 = true and
LS6.2.1	(Pressure1 = TooLow and At_T_Block_On(Reset1, Reset2, Block1, Block2)
LS6.2.2	OR (Pressure1 = Permitted and At_T_Block_On(Reset1, Reset2, Block1, Block2)));
LS7	LOGIC STRUCTURE M_Pressure (WaterPres1, WaterPres2, Pressure2, Pressure1)
	CONSTRAINT
LS7.1	Pressure2 = TooLow and Pressure1 = Permitted
	and WaterPres2 < Low and WaterPres1 >= Low and WaterPres1 < Permit
LS7.2	OR Pressure2 = High and Pressure1 = Permitted
	and WaterPres2 >= Permit and WaterPres1 < Permit and WaterPres1 >= Low
LS7.3	OR Pressure2 = Permitted and
	(Pressure1 = TooLow and WaterPres2 >= Low and WaterPres2 < Permit and WaterPres1 < Low
	OR Pressure1 = High and WaterPres2 < Permit and WaterPres2 >= Low and WaterPres1 >= Permit)
LS7.4	OR Pressure2 = Pressure1 and
	((Pressure1 = TooLow and WaterPres2 < Low and WaterPres1 < Low)
	OR (Pressure1 = Permitted and WaterPres2 < Permit and WaterPres1 < Permit
	and WaterPres2 >= Low and WaterPres1 >= Low)
	OR (Pressure1 = High and WaterPres2 >= Permit and WaterPres1 >= Permit));

TAB. 10 - Les expressions LS (Logic Structure) en T-VEC

3. on sélectionne les données de test à partir des contraintes d'un DCP. Ce processus se fait en identifiant les sous-domaines du DCP. Les limites de chaque sous-domaine sont choisies comme données de test. Si une variable d'entrée n'est pas restreinte, on choisit soit les limites dans le cas des variables numériques, soit toutes les valeurs possibles dans le cas d'une variable de type énuméré;
4. on choisit le mode de génération de vecteurs *entrée multiple* (toutes les combinaisons des limites), *limité* (toutes les combinaisons entre les limites supérieures et les limites inférieures et les combinaisons des limites supérieures) ou *simple* (les combinaisons des limites supérieures).

```
FR.0    RELATIONSHIP produce Safety_Injection;
        RELEVANCE PREDICATE {
RP.0    DISJUNCTION {M_Pressure};}
        LEVEL {
FR.1    RELATIONSHIP Safety_Injection = ON;
        RELEVANCE PREDICATE {
RP.1.1  DISJUNCTION {Pressure2 = TooLow, Overridden2 = false,
        Overridden_Term};
RP.1.2  DISJUNCTION {Pressure2 = TooLow, Overridden2 = false, Overridden1 =
        false, NOT:At_T_Reset_TooLow, NOT:At_T_Reset_Permitted,
        NOT:At_T_Inmode_High, NOT:At_T_Inmode_TooLow_Permitted,
        NOT:At_T_Block_On};
        }
FR.2    RELATIONSHIP Safety_Injection = OFF;
        RELEVANCE PREDICATE {
RP.2.1  DISJUNCTION {Pressure2 = High};
RP.2.2  DISJUNCTION {Pressure2 = Permitted};
RP.2.3  DISJUNCTION {Pressure2 = TooLow, Overridden2 = true,
        Overridden_Term};
RP.2.4  DISJUNCTION {Pressure2 = TooLow, Overridden2 = true, Overridden1 =
        true, NOT:At_T_Reset_TooLow, NOT:At_T_Reset_Permitted,
        NOT:At_T_Inmode_High, NOT:At_T_Inmode_TooLow_Permitted,
        NOT:At_T_Block_On};
        } } } }
```

TAB. 11 - La représentation des variables contrôlées en T-VEC

L'approche décrite ci-dessus ne garantit pas qu'au moins un vecteur soit généré pour chaque DCP. Pour cette raison, une analyse de couverture doit être exécutée à la fin du processus de génération de vecteurs.

Le tableau (TAB. 12) contient l'ensemble de vecteurs généré par T-VEC pour la spécification de Safety Injection System.

Les conclusions de l'approche T-VEC sont les suivantes :

1. la génération des vecteurs se fait en arrière à partir des variables de sortie jusqu'aux variables d'entrée. Elle est basée directement sur l'ensemble des prédicats associés aux variables de sortie, donc la création du vecteur commence en choisissant une valeur attendue de la variable de sortie;
2. l'article présenté ne mentionne pas en détail le taux de couverture du graphe de système et la minimisation du jeu de vecteurs de test générés.

Vector #	FR	RP	Mode de Convergence	S_I	Over2	Pressure1	Pressure2	WP1	WP2	Rt1	Rt2	Bk1	Bk2
1	1	RP_1« 1 »	Low Bound	ON	FALSE	TooLow	TooLow	0	0	OFF	ON	-	-
2	1	RP_1« 2 »	Low Bound	ON	FALSE	Permitted	TooLow	900	0	OFF	ON	-	-
3	1	RP_1« 1 »	Hi Bound	ON	FALSE	TooLow	TooLow	899	899	OFF	ON	-	-
4	1	RP_1« 2 »	Hi Bound	ON	FALSE	Permitted	TooLow	999	899	OFF	ON	-	-
5	2	RP_2« 1 »	Low Bound	OFF	FALSE	Permitted	High	900	1000	OFF	OFF	OFF	OFF
6	2	RP_2« 1 »	Low Bound	OFF	FALSE	High	High	1000	1000	OFF	OFF	OFF	OFF
7	2	RP_2« 2 »	Low Bound	OFF	FALSE	TooLow	Permitted	0	900	OFF	OFF	OFF	OFF
8	2	RP_2« 2 »	Low Bound	OFF	FALSE	High	Permitted	1000	900	OFF	OFF	OFF	OFF
9	2	RP_2« 2 »	Low Bound	OFF	FALSE	Permitted	Permitted	900	900	OFF	OFF	OFF	OFF
10	2	RP_2« 3 »	Low Bound	OFF	TRUE	TooLow	TooLow	0	0	OFF	OFF	OFF	ON
11	2	RP_2« 4 »	Low Bound	OFF	TRUE	Permitted	TooLow	900	0	OFF	OFF	OFF	ON
12	2	RP_2« 1 »	Hi Bound	OFF	FALSE	Permitted	High	999	2000	OFF	OFF	OFF	OFF
13	2	RP_2« 1 »	Hi Bound	OFF	FALSE	High	High	2000	2000	OFF	OFF	OFF	OFF
14	2	RP_2« 2 »	Hi Bound	OFF	FALSE	TooLow	Permitted	899	999	OFF	OFF	OFF	OFF
15	2	RP_2« 2 »	Hi Bound	OFF	FALSE	High	Permitted	2000	999	OFF	OFF	OFF	OFF
16	2	RP_2« 2 »	Hi Bound	OFF	FALSE	Permitted	Permitted	999	999	OFF	OFF	OFF	OFF
17	2	RP_2« 3 »	Hi Bound	OFF	TRUE	TooLow	TooLow	899	899	OFF	OFF	OFF	ON
18	2	RP_2« 4 »	Hi Bound	OFF	TRUE	Permitted	TooLow	999	899	OFF	OFF	OFF	ON

TAB. 12 - La génération des vecteurs de SIS

Chapitre 3

Génération de jeux de test pour une spécification SCR

3.1 Introduction

Dans ce chapitre, nous avons comme objectif d'établir une méthode pour la génération de scénarios à partir d'une spécification SCR.

Nous commençons par identifier les dépendances des variables d'une spécification SCR, en suite nous traduisons la spécification SCR en EFSM. La traduction se fait en représentant chaque fonction de la spécification SCR dans un graphe étendu; après nous présentons la génération des scénarios en utilisant l'algorithme « breadth-first » qui parcourt le graphe par branches.

À la fin, nous faisons une analyse du graphe pour maximiser la couverture des scénarios par rapport aux exigences de la spécification.

Avant de définir notre approche, nous partons des suppositions suivantes :

1. nous avons à notre disposition un ensemble de spécifications SCR;
2. l'ensemble contient des spécifications de taille différente;
3. les spécifications sont correctes, c'est-à-dire les vérifications suivantes ont déjà été effectuées : la validation de types, les disjonctions entre les transitions, la couverture du domaine et les assertions ont été prouvées.

Nous utilisons quatre spécifications satisfaisant ces critères de vérification : une version simple de SIS (Safety Injection System), une version enrichie de SIS, Cruise Control System et WCP1 (Weapon Control Panel). Ce choix nous permettra de comparer les résultats de notre travail avec ceux obtenus par Gargantini et Heitmeyer [7].

Notre approche est constituée des étapes suivantes :

1. la construction du graphe de dépendance à partir de la spécification SCR;
2. la traduction du modèle SCR en EFSM;
3. la normalisation de chaque prédicat de l'EFSM;
4. la stabilisation du graphe EFSM;
5. la génération des scénarios;
6. la maximisation de la couverture.

Nous présentons notre approche en deux étapes. D'abord, en décrivant de façon détaillée notre approche tout en utilisant la version simple de SIS pour finalement présenter les résultats que nous aimerions obtenir avec notre prototype.

3.2 Construction du graphe de dépendance

La construction du graphe de dépendance se fait à partir de la spécification SCR. Le graphe représente les dépendances entre les variables d'une spécification SCR. Pour chaque fonction de transition de mode, d'événement et de condition nous identifions les dépendances entre les variables surveillées, les mêmes fonctions, les variables contrôlées et les termes. La construction du graphe de dépendance est faite à l'aide du logiciel SCRTool, option navigateur de graphe de dépendances.

Exemple : Le graphe de dépendance du système SIS qui a été défini dans le chapitre 2 section méthode SCR, est le suivant.

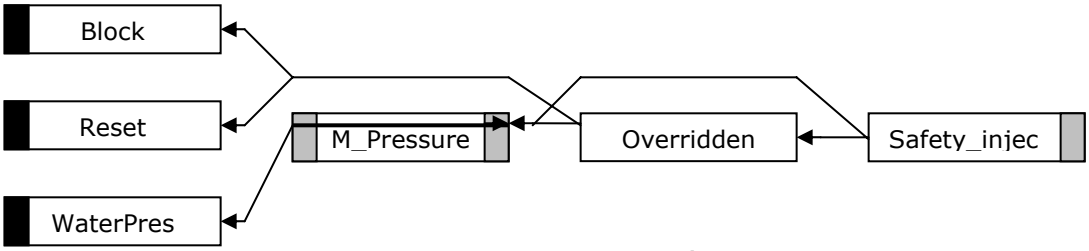


FIG. 12 – Le graphe de dépendance

3.3 Traduction du modèle SCR en EFSM

La méthode SCR utilise une machine d'états finis pour modéliser le comportement d'un système et EFSM utilise ainsi les principes de base de la machine d'états finis. En conséquence, la traduction consiste à trouver un équivalent de chaque élément SCR en EFSM.

Nous définissons l'EFSM M, comme suit M = {S,I,O,D,T} où

S est le produit cartésien de classes des modes SCR.

I est le produit cartésien de types des variables surveillées SCR.

O est le produit cartésien de types des variables contrôlées SCR.

D est le produit cartésien de types de variables termes SCR.

T est une relation de transition, T : $S \times D \times I \rightarrow S \times D \times O$.

Chaque transition est définie par un couple prédicat/action. Les transitions correspondant aux différentes fonctions de SCR, elles sont définies comme suit :

- soit f_v une fonction de condition pour la variable *v* (voir la méthode SCR) :

  $f_v(e_1, \ldots, e_p, m) \overset{def}{=} r_j$ **si** $m=m_i \wedge C_{ij}(e_1, \ldots, e_p, m)$

  Le résultat de la traduction est une série de transitions sur l'état m_i de la forme (prédicat)/(action), où

  le prédicat : $C_{ij}(e_1, \ldots, e_p, m)$;

  l'action : $v = r_j$.

- soit f_v une fonction d'événement pour la variable *v* (voir la méthode SCR):

  $f_v(e_1, \ldots, e_p, m) \overset{def}{=} r_j$ **si** $m=m_i \wedge E_{ij}(e_1, \ldots, e_p, m)$

  Le résultat de la traduction est une série de transitions sur l'état m_i de la forme (prédicat)/(action), où

  le prédicat : $E_{ij}(e_1, \ldots, e_p, m)$;

  l'action : $v = r_j$.

- soit f une fonction de transition de mode (voir la méthode SCR) :

  $f(e_1, \ldots, e_p, m) \overset{def}{=} O_i$ **si** $m=m_i \wedge E_i(e_1, \ldots, e_p, m)$

Le résultat de la traduction est une série de transitions de l'état m_i à l'état O_i ; où le prédicat est : E_i (e_1 , ... , e_p, m) et l'action: aucune.

Une fois que nous avons trouvé l'équivalence de chaque élément SCR en EFSM, nous pouvons simplifier les transitions en utilisant le graphe de dépendance. La transition simplifiée montre la relation entre une variable surveillée et une variable contrôlée, par conséquent, la transition simplifiée est de la forme suivante : (prédicat/(action,sortie)) où le prédicat doit contenir un événement et la sortie une variable contrôlée.

Le processus de simplification consiste à répéter les étapes suivantes :

- choisir un *chemin de dépendance (cd)* complet qui contient les dépendances entre une variable d'entrée et une variable de sortie à partir du graphe de dépendance. Le chemin cd est sous la forme suivante :
  $\{(p_1/a_1),\ldots (p_n/a_n)\}$ où p_i est le prédicat et a_i l'action;
- fractionner le chemin de dépendance *cd* en transitions $t_{1,\ \ldots,}\ t_{n,}$ où chaque transition t_i contient seulement un événement associé à une variable d'entrée;
- simplifier la séquence de transitions $t_{1,\ \ldots,}\ t_n$. Si t_i est le résultat d'unifier (p_k/a_k) et (p_l/a_l) la transition simplifiée sera :
  $(p_k \wedge p_l) / ((a_{k,}\ a_l)$, sortie), où la sortie est associée à une action soit $a_{k,}$ ou a_l qui définit la valeur de la variable contrôlée du chemin *cd*.

Le graphe EFSM sera composé des transitions entre des états différents et des transitions dans le même état. Les transitions entre états différents sont construites à partir de la fonction de transition de modes et les transitions dans le même état sont construites à partir des fonctions de condition et d'événement.

Exemple : pour illustrer le processus de traduction du modèle SCR en EFSM, nous partons de la spécification du système SIS en SCR. Le résultat de trouver l'équivalence de chaque élément SCR en EFSM sera détaillé dans un tableau sous forme d'état précédent, de prédicat, d'action et d'état suivant.

Le résultat de traduire la table de transition de mode, la table d'événement «Overridden» et la table de condition «Safety_Injection» en EFSM est le suivant :

- La table transition de mode :

État précédent	Prédicat	Action	État suivant
TooLow	@T (WP>=low)	--	Permitted
Permitted	@T(WP<low)	--	TooLow
Permitted	@T(WP>=permit)	--	High
High	@T(WP<permit)	--	Permitted

TAB. 13 - La fonction de transition de mode SCR en EFSM

- La table d'événement :

État précédent	Prédicat	Action	État suivant
High	Never	OV=TRUE	High
TooLow	@T(BK=ON) WHEN RT=OFF	OV=TRUE	TooLow
Permitted	@T(BK=ON) WHEN RT=OFF	OV=TRUE	Permitted
High	@T(Inmode)	OV=FALSE	High
TooLow	@T(RT=ON)	OV=FALSE	TooLow
Permitted	@T(RT=ON)	OV=FALSE	Permitted

TAB. 14 - La fonction d'événement SIS en EFSM

- La table de condition :

État précédent	Prédicat	Action	État suivant
High	TRUE	SI=OFF	High
Permitted	TRUE	SI=OFF	Permitted
TooLow	Overridden	SI=OFF	TooLow
TooLow	Not Overridden	SI=ON	TooLow

TAB. 15 - La fonction de condition SIS en EFSM

Le résultat final de la traduction de la spécification du système SIS en EFSM après avoir effectué le processus de simplification où chaque transition doit être sous forme (prédicat/(action,sortie)), est donné par la table suivante :

Tr	État précédent	Prédicat	Action	Sortie	État suivant
T1	TooLow	@T(BK=ON) WHEN RT=OFF	OV=TRUE	SI=OFF	TooLow
T2	TooLow	@T(RT=ON)	OV=FALSE	SI=ON	TooLow
T3	TooLow	@T(WP>=low)	--	SI=OFF	Permitted
T4	Permitted	@T(BK=ON) WHEN RT=OFF	OV=TRUE	SI=OFF	Permitted
T5	Permitted	@T(RT=ON)	OV=FALSE	SI=OFF	Permitted
T6	Permitted	@T(WP>=permit)	--	SI=OFF	High
T7	High	@T(WP<permit)	--	SI=OFF	Permitted
T8	Permitted	@T(WP<low) $\wedge$ OV	--	SI=OFF	TooLow
T9	Permitted	@T(WP<low) $\wedge$ not OV	--	SI=ON	TooLow

TAB. 16 - La génération de transitions SIS en EFSM

3.4 Normalisation

Voir la définition dans la section 2.5.5 du chapitre 2.

3.5 Stabilisation du graphe EFSM

La stabilisation a comme objectif d'identifier les transitions instables dans un graphe EFSM sous forme de transitions de blocs. Une transition est instable quand elle est associée à deux ou plusieurs blocs différents. À l'aide du processus d'orthogonalisation (voir la section 2.4.5 du chapitre 2) qui consiste en diviser les variables d'un bloc en intervalles disjoints, le graphe EFSM devient stable. Le processus d'orthogonalisation est utilisé pour stabiliser le graphe EFSM (voir exemple dans la section 2.4.7 du chapitre 2);

3.6 Génération des scénarios

Une fois que le graphe EFSM est stable, nous appliquons l'algorithme «breadth-first» pour générer les scénarios en utilisant le critère de couverture par branches dans le graphe. Nous observons que nous pouvons obtenir de cette manière des scénarios qui ne sont pas valides.

Exemple : si nous exécutons les transitions T_3,T_4,T_9, la valeur de «Overridden» devient «TRUE» dans T_4, mais au moment d'exécuter T_9, l'évaluation du prédicat sera «FALSE» et par conséquent, le scénario {@T(WP>=low), @T(BK=ON), @T(WP<low)} devient invalide.

Pour résoudre ce problème, nous considérerons deux approches :

- l'identification de dépendances entre les transitions
- la génération de séquence par propriété, c'est-à-dire l'utilisation d'un modèle de vérification (spin ou SMV). Voir la section 2.3.1 du chapitre 2.

Exemple : Les transitions dépendantes de la variable «Overridden» sont :

Td_1 : T_4,T_8

Td_2 : T_5,T_9

Td_3 : T_1,T_3,T_8

Td_4 : T_2,T_3,T_9

Les transitions T_i, les transitions associées à chaque scénarios Ts_i et les événements associés à chaque scénario S_i générés en appliquant l'algorithme «breadth-first» sont :

Ts_1 : {T_1,T_3,T_8},
S_1 : {@T(BK=ON), @T(WP>=low), @T(WP<low)}
Ts_2 : {T_2,T_3,T_9}.
S_2 : {@T(RT=ON), @T(WP>=low), @T(WP<low)}
Ts_3 : {T_3,T_4,T_8}.
S_3 : {@T(WP>=low), @T(BK=ON), @T(WP<low)}
Ts_4 : {T_3,T_5,T_9}.
S_4 : {@T(WP>=low), @T(RT=ON), @T(WP<low)}
Ts_5 : {T_3,T_6,T_7,T_9}.
S_5 : {@T(WP>=low), @T(WP>=permit), @T(WP<permit), @T(WP<low) }

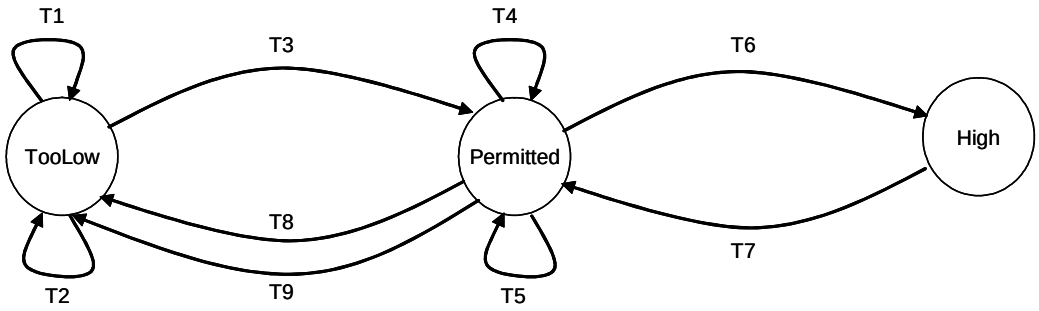


FIG. 13 - Le graphe SIS pour la génération des scénarios

3.7 Maximisation de la couverture

Dans l'EFSM chaque prédicat est associé à un ou plusieurs états, nous disons que la couverture totale du graphe EFSM se fait quand l'ensemble des scénarios couvre chaque prédicat avec ses états associés. Le taux de couverture sera calculé en pourcentage du nombre de prédicats par état dans l'ensemble des scénarios par rapport au nombre de prédicats par état contenus dans la spécification. Dans l'analyse de couverture, nous identifierons les prédicats qui ne sont pas contenus dans l'ensemble des scénarios.

La solution à notre problème de génération automatique de test à partir d'une spécification en SCR utilisera l'architecture suivante :

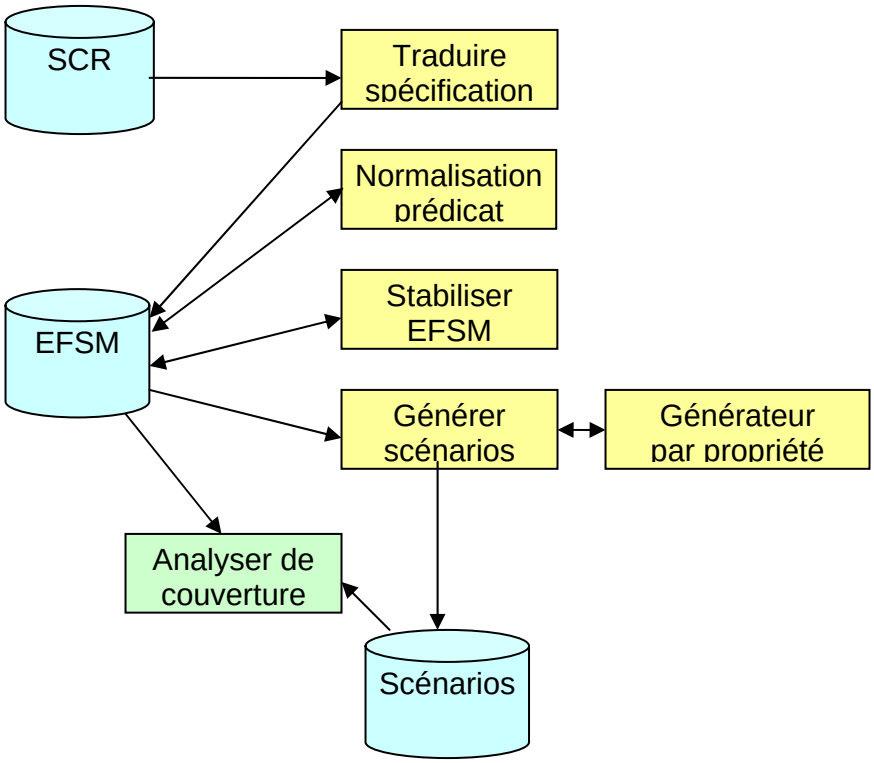


FIG. 14 - Les processus de la génération automatique de scénarios

En conclusion, nous proposons une architecture pour la génération automatique de scénarios à partir d'une spécification SCR. Dans le chapitre suivant nous proposons des améliorations ou des solutions aux inconvénients suivants que nous avons identifié, comme :

1. la génération de scénarios qui ne sont pas valides après d'exécuter l'algorithme « breadth-first »;
2. la réduction de la complexité qui est associée au processus de stabilisation d'un graphe EFSM et
3. l'addition des événements complémentaires pour augmenter la maximisation de la couverture des scénarios par rapport aux transitions de la spécification.

Chapitre 4
Algorithme de génération de jeux de test

4.1 Introduction

Dans ce chapitre, nous présentons l'algorithme de génération de scénarios à partir d'une spécification SCR. Les améliorations apportées à l'algorithme ont été d'ajouter les événements complémentaires, d'utiliser une structure d'arbre pour la génération des scenarios en contrôlant l'explosion d'états et de prouver quelles sont les transitions qui ne sont pas atteignables.

Nous commençons par identifier les événements, les événements complémentaires, les conditions et les chemins de dépendances des variables qui ont la spécification SCR, en suite nous générons les scénarios en utilisant une structure d'arbre par niveaux où le nœud initial (niveau zéro) contient les valeurs initiales de la spécification. Les niveaux suivants seront créés, une fois que nous aurons identifié les événements à déclencher et les valeurs attendues après d'avoir évalué les prédicats associés aux événements.

À la fin, nous faisons un parcours en profondeur dans l'arbre pour générer les scénarios et nous analysons la couverture des scénarios par rapport aux transitions de la spécification en prouvant quelles sont les transitions que le système ne peut pas déclencher.

Nous présentons notre algorithme en décrivant de façon détaillée les étapes à suivre tout en utilisant la version simple de SIS qui a été présentée dans la section 3.3 traduction du modèle SCR en EFSM du chapitre 3 :

4.2 Identifier les événements et les conditions

Identifier les événements et les conditions à partir de la spécification. Exemple :
Les événements identifiés sont :

	Événement
E1	@T(WP>=low)
E2	@T(WP<low)
E3	@T(WP>=permit)
E4	@T(WP<permit)
E5	@T(BK=ON) WHEN RT=OFF
E6	@T(Inmode)
E7	@T(RT=ON)

TAB. 17 - La liste des événements

Les conditions identifiées sont :

	Condition
C1	Overridden
C2	Not Overridden

TAB. 18 - La liste des conditions

4.3 Déterminer les événements complémentaires

Les événements complémentaires correspondent à la négation des événements qui sont dans la spécification du système. Exemple :

	Négation de l'événement	**Événements complémentaires**
EE1	@T(NOT(WP>=low))	@T(WP<low)
EE2	@T(NOT(WP<low))	@T(WP>=low)
EE3	@T(NOT(WP>=permit))	@T(WP<permit)
EE4	@T(NOT(WP<permit))	@T(WP>=permit)
EE5	@T(NOT(BK=ON))	@T(BK<>ON)
EE6	@T(NOT(Inmode))	@T(NOT(Inmode))
EE7	@T(NOT(RT=ON))	@T(RT<>ON)

TAB. 19 - La liste des événements complémentaires

4.4 Identifier les événements équivalents

Une fois que nous avons l'ensemble des événements de la spécification et l'ensemble des événements complémentaires, l'intersection entre les deux ensembles correspond aux événements équivalents. Exemple :

Événements équivalents	
L'intersection de l'ensemble des événements	**L'intersection de l'ensemble des événements complémentaires**
E1	EE2
E2	EE1
E3	EE4
E4	EE3

TAB. 20 - La liste des événements équivalents

L''identification des événements équivalents sera utilisée au moment de générer les scénarios pour éviter l'explosion d'états équivalents.

4.5 Normaliser

Trouver la forme normale des événements et des conditions; Voir la définition de normalisation dans la section 2.5.5 du chapitre 2.

4.6 Déterminer les chemins de dépendance

Déterminer les chemins possibles de dépendance entre les variables surveillées et contrôlées à partir du graphe de dépendance. Exemple :

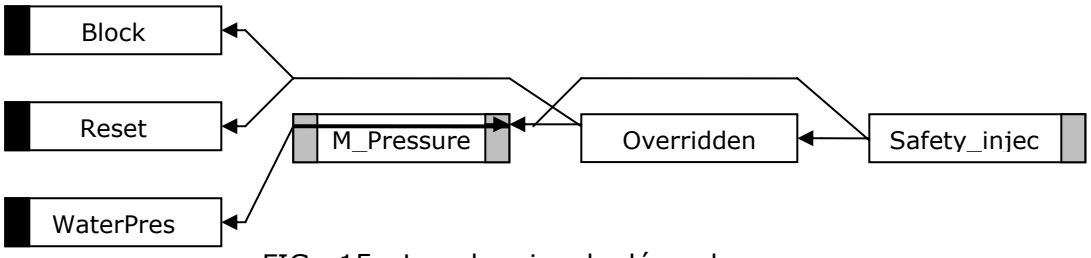


FIG. 15 - Les chemins de dépendance

Une fois qu'un événement arrive, il y a deux fonctions possibles qui peuvent se déclencher M_Pressure ou Overridden, après M_Pressure est reliée à Overridden ou Safety_injection et pour Overridden l'unique option est Safety_Injection, alors les

chemins possibles de ce graphe de dépendance sont : M_Pressure – Overridden - Safety_injection, M_Pressure - Safety_injection, et Overridden - Safety_injection. Voir la figure (FIG. 15).

4.7 Identifier les valeurs initiales

Identifier le mode et les valeurs initiales des variables du système.

Exemple :

M_P = TooLow, SI= OFF, BK=OFF, RT=ON, WP=14 et OV=TRUE.

4.8 Traduire la spécification SCR en EFSM

L'objectif de cette étape est de convertir la spécification SCR qui est définie pour la fonction de condition, d'événements et de transition de modes, dans un table de transitions EFSM où chaque transition doit être sous forme (prédicat/(action,sortie)).

Le processus de traduction a été précisé dans le chapitre précédent section 3.3. Le résultat de traduire la spécification SIS (Safety Injection System) de SCR à EFSM est présenté dans le tableau suivant.

Trans	État précédent	Prédicat	Action	Sortie	État suivant
T1	TooLow	E5	OV=TRUE	SI=OFF	TooLow
T2	TooLow	E7	OV=FALSE	SI=ON	TooLow
T3	TooLow	E1	--	SI=OFF	Permitted
T4	Permitted	E5	OV=TRUE	SI=OFF	Permitted
T5	Permitted	E7	OV=FALSE	SI=OFF	Permitted
T6	Permitted	E3	--	SI=OFF	High
T7	High	E4	--	SI=OFF	Permitted
T8	Permitted	E2 $\wedge$ C1	--	SI=OFF	TooLow
T9	Permitted	E2 $\wedge$ C2	--	SI=ON	TooLow

TAB. 21 - La traduction de la spécification SIS de SCR à EFSM

4.9 Générer les scénarios

Une fois que nous avons la spécification d'un système dans un tableau de transitions EFSM, l'idée sera de générer les scénarios à partir de l'état initial en identifiant les transitions que nous pouvons déclencher.

Un des éléments à considérer quand nous pouvons déclencher une transition sera d'évaluer le prédicat avec les valeurs initiales des variables pour avoir un résultat des variables qui sera produit de passer d'un état précédent à un état suivant. Pour cette raison, nous créerons une structure d'arbre pour construire les scénarios par niveau à partir du nœud racine (état initial de la spécification), lequel contient le mode et les valeurs initiales des variables.

Cet arbre sera représenté par des nœuds et des arêtes. Chaque nœud est placé dans un niveau et interconnecté avec un autre nœud à travers d'une arête. Un nœud est associé aux valeurs initiales. Une arête est associée à un prédicat (événement et/ou condition), qui génère des résultats pour un nœud différent ou le même. L'arbre est construit par niveaux (en appliquant les deux étapes qui sont décrites ci-dessous) jusqu'à ce que l'arbre contienne tous les événements de la spécification du système. Cet arbre est représenté en utilisant le tableau de prédicats EFSM, en plus nous avons ajouté la colonne Valeurs initiales. Le tableau final a la structure suivante :

Niveau-Arête	Noeud précédent	Valeurs initiales	Prédicat	Noeud suivant	Résultat

TAB. 22 - Les scénarios représentés dans un tableau de prédicats

À partir du nœud racine avec les valeurs initiales et le système spécifié dans le tableau de prédicats, le processus pour générer les chemins possibles commence par l'analyse de trois types de groupes d'événements : les événements qui sont reliés directement au nœud courant dans le tableau de prédicats, lesquels on peut déclencher, les événements qui ne sont pas reliés directement au nœud courant dans la table de prédicats, mais lesquels on peut déclencher et les événements complémentaires en éliminant les événements équivalents. Par la suite, nous présentons le détail de la génération des scénarios à partir des différents groupes d'événements :

- Les événements qui sont reliés directement au nœud courant dans le tableau de prédicats : à partir de chaque chemin identifié dans le graphe de dépendance, nous commençons à construire l'arbre par niveaux. Chaque chemin doit contenir un seul événement, l'état précédent, l'état suivant et le type de fonction SCR.

Exemple : les chemins possibles détectés dans le graphe de dépendance sont MP-SI, MP-OV-SI et OV-SI, alors la représentation de ces chemins dans l'arbre pour le niveau 0 est dans le tableau (TAB. 23).

Niveau-Arête	**Nœud précédent**	**Valeurs initiales**	**Prédicat**	**Nœud suivant**	**Résultat**
0-1 (MP-SI)	TooLow	SI= OFF BK=OFF RT=ON WP=14 OV=TRUE	E1 ∧ TRUE	Permitted	SI= OFF BK=OFF RT=ON WP>=low OV=TRUE
0-2 (MP-OV-SI)	TooLow	SI= OFF BK=OFF RT=ON WP=14 OV=TRUE	E1	Permitted	SI= OFF BK=OFF RT=ON WP>=low OV=TRUE
0-3 (OV-SI)	TooLow	SI= OFF BK=OFF RT=ON WP=14 OV=TRUE	E5 ∧ OV	TooLow	SI=OFF OV=TRUE
0-4 (OV-SI)	TooLow	SI= OFF BK=OFF RT=ON WP=14 OV=TRUE	(E6 v E7) ∧ NOT OV	TooLow	SI=ON OV=TRUE

TAB. 23 - Les événements qui sont reliés directement au nœud courant

- Les événements qui ne sont pas reliés directement au nœud courant dans le tableau de prédicats correspondent aux événements qui ne sont pas inclus dans le niveau présent de l'arbre et ils sont : E2, E3 et E4. Voir le tableau (TAB. 24).

Niveau - Arête	**Nœud précédent**	**Valeurs initiales**	**Prédicat**	**Nœud suivant**	**Résultat**
0-5	TooLow	SI=OFF, BK=OFF RT=ON,WP=14 OV=TRUE	E2	TooLow	
0-6	TooLow	SI=OFF, BK=OFF RT=ON, WP=14 OV=TRUE	E3	TooLow	
0-7	TooLow	SI=OFF, BK=OFF RT=ON, WP=14 OV=TRUE	E4	TooLow	

TAB. 24 - Les événements qui ne sont pas reliés directement au nœud courant

- L'ensemble des événements complémentaires est {EE1, EE2, EE3, EE4, EE5, EE6, EE6, EE7} moins l'ensemble des événements équivalents {EE1, EE2, EE3, EE4}, le résultat final est l'ensemble {EE5, EE6, EE7}.

Niveau-Arête	Nœud précédent	Valeurs initiales	Préd.	Nœud suivant	Résultat
0-8	TooLow	SI= OFF, BK=OFF, RT=ON, WP=14, OV=TRUE	EE5	TooLow	
0-9	TooLow	SI= OFF, BK=OFF RT=ON, WP=14 OV=TRUE	EE6	TooLow	
0-10	TooLow	SI= OFF, BK=OFF RT=ON, WP=14 OV=TRUE	EE7	TooLow	

TAB. 25 - Les événements complémentaires du nœud courant

4.9.1 Calculer les valeurs attendues

Une fois que nous avons les événements à déclencher à partir du nœud courant, nous évaluons chaque prédicat associé à chaque arête, pour calculer les nouvelles valeurs du nœud suivant. Les résultats possibles obtenus de l'évaluation du prédicat pour chaque groupe d'événements peuvent être les suivants :

- Pour les événements qui sont reliés directement au nœud courant dans le tableau de prédicats, si le résultat est faux après avoir évalué le prédicat, nous considérons que le prédicat est en blocage dans le nœud courant.
  Exemple : E5 est @T(BK=ON) WHEN RT=OFF. L'événement @T(BK=ON) peut être déclenché parce que BK=OFF, mais RT=ON fait que le prédicat soit FALSE, par conséquent le prédicat est en blocage. La solution à ce problème est de déclencher l'événement @T(RT=OFF) avant E5.

Niveau-Arête	Nœud précédent	Valeurs initiales	Prédicat	Nœud suivant	Résultat
0-3 (OV-SI)	TooLow	SI=OFF BK=OFF RT=ON WP=14 OV=TRUE	E5 $\wedge$ OV	TooLow	FALSE

TAB. 26 - L'évaluation des prédicats

La solution au blocage est la suivante :

Niveau-Arête	Nœud précédent	Valeurs initiales	Prédicat	Nœud suivant	Résultat
0-3 (OV-SI)	TooLow	SI= OFF BK=OFF RT=ON WP=14 OV=TRUE	EE7	TooLow	SI= OFF BK=OFF RT=OFF WP=14 OV=TRUE
0-3.1-1 (OV-SI)	TooLow	SI= OFF BK=OFF RT=OFF WP=14 OV=TRUE	E5 $\wedge$ OV	TooLow	SI= OFF BK=ON RT=OFF WP=14 OV=TRUE

TAB. 27 - Solution au déblocage d'un prédicat

- Pour les événements qui ne sont pas reliés directement au nœud courant dans le tableau de prédicats, si le résultat d'évaluer le prédicat est vrai et les modes des nœuds courant et suivant sont différents, nous considérons ce cas comme une anomalie parce que le prédicat n'est pas défini dans le tableau de prédicats pour le nœud courant.

Niv. Arêt	Nœud préc.	Valeurs initiales	Préd	Nœud suivant	Résultat
0-6	TooLow	SI= OFF, BK=OFF, RT=ON, WP=14 OV=TRUE	E3	Permitted	SI= OFF, BK=OFF, RT=ON, WP>=permit OV=TRUE

TAB. 28 - La détection des anomalies des prédicats

La solution est de modifier le prédicat E1 pour E1$\wedge$NOT(E3) dans la fonction de transition de modes M_Pressure.

- Pour les événements complémentaires moins les événements équivalents : si le résultat d'évaluer le prédicat est vrai et les modes des nœuds courant et suivant sont différents, nous considérons ce cas comme une anomalie.

Niv. Arêt	Nœud préc.	Valeurs initiales	Préd	Nœud suivant	Résultat
0-8	TooLow	SI= OFF, BK=OFF RT=ON, WP=14 OV=TRUE	EE5	TooLow	SI= OFF, BK=OFF RT=ON, WP=14 OV=TRUE

TAB. 29 - La détection des inconsistances des prédicats pour les événements complémentaires

La solution est de modifier la spécification du système en ajoutant la négation de l'événement.

4.10 Réduire l'explosion d'états

Nous évitons le traitement des états qui ont déjà été considérés quand l'algorithme est en train de construire les scénarios. Une fois que les valeurs attendues ont été calculées, nous vérifions si les valeurs initiales pour cet état (nœud) ont déjà été traitées pour arrêter le processus de génération des scénarios reliés à ses valeurs initiales.

4.11 Finaliser la construction des scénarios

Le processus de construction de l'arbre se termine quand il y a au moins une arête de l'arbre associée à chaque prédicat spécifiée dans le tableau de prédicats ou quand on ne peut pas générer un nouveau nœud dans l'arbre après avoir essayé de déclencher les trois groupes d'événements.

4.12 Identifier les transitions qui ne sont pas atteignables

La génération des scénarios est obtenue en parcourant l'arbre en profondeur à partir du nœud racine jusqu'aux feuilles. Le scénario est construit en ajoutant l'événement associé à chaque niveau.

Parfois, le processus de génération des scénarios ne peut pas parcourir les transitions de la spécification parce qu'il y a des transitions qui ne sont pas atteignables.

Une solution à ce problème sera de prouver que la transition n'est pas atteignable.

Ensuite, nous présentons cette preuve. Soit @T(C) WHEN D un événement donné. Son événement complémentaire est @T(NOT C). Le tableau (TAB. 18) présente les événements qui sont reliés directement au nœud courant dans le

tableau de prédicats du système SIS. Si nous considérons les variables surveillées BK et RT, les événements reliés sont :

- @T(BK=ON) WHEN RT=OFF.
- @T(RT=OFF).

Les événements complémentaires sont :

- @T(NOT(BK=ON)) WHEN RT=OFF.
- @T(NOT(RT=OFF)).

Dans le cas où il existe encore des transitions à parcourir, nous pouvons essayer de prouver que la transition donnée n'est pas atteignable. Considérons la transition suivante :

@T(c) $\stackrel{def}{=}$ ¬'c Λ c' Λ 'M=m où la variable M représente le mode courant.

Si nous prouvons qu'il est impossible de satisfaire la condition ¬'c dans le mode 'M, la transition ne pourra jamais être déclenchée et, par conséquent, elle ne sera pas atteignable. Observons que

¬(¬'c Λ 'M=m) ⇔ 'c V 'M≠m ⇔ 'M=m => 'c.

Si nous prouvons 'M=m => 'c, la transition n'est pas atteignable. Si le mode m n'est pas atteignable, l'implication sera vraie.

Nous utiliserons les outils suivants pour faire la preuve :

- SCR fait la preuve à travers la création d'une assertion et en utilisant l'option de preuve. L'outil va nous indiquer s'il est capable de prouver l'assertion.

- Les vérificateurs de modèle SPIN ou SMV font la preuve à travers la création d'une assertion pour vérifier si elle est vraie ou fausse.

Chapitre 5

Minimisation de jeux de test

5.1 Introduction

Dans ce chapitre, nous présentons l'algorithme de minimisation de jeux de test (scénarios) qui a pour objectif de réduire les redondances entre les scénarios et de sélectionner un ensemble minimal des scénarios basé dans la cardinalité des scénarios ou la cardinalité des transitions.

Nous commençons par une revue de la littérature des heuristiques traditionnelles Greedy comme : Classique Greedy, HGS (Harrold, Gupta et Soffa) et Delayed Greedy qui est l'heuristique plus performante. Ensuite, nous proposons des modifications et des améliorations à l'algorithme Delayed Greedy en créant les heuristiques : Cardenas Greedy, Merge Greedy et Delayed Greedy version 2.

Après avoir développé les heuristiques Greedy et les différents exemples d'ensemble des scénarios, nous évaluons la performance des heuristiques en déterminant le pourcentage de réduction, un indicateur de redondance des transitions et le temps d'exécution.

Pour mieux comprendre la problématique de la minimisation des scénarios nous ferons référence aux transitions de la spécification SIS et à son ensemble de scénarios en forme de tableau.

La spécification du système SIS est présentée sous forme d'un graphe où les nœuds correspondent à des états et les arêtes correspondent à des transitions associées à des événements. Considérons l'exemple du chapitre 4.

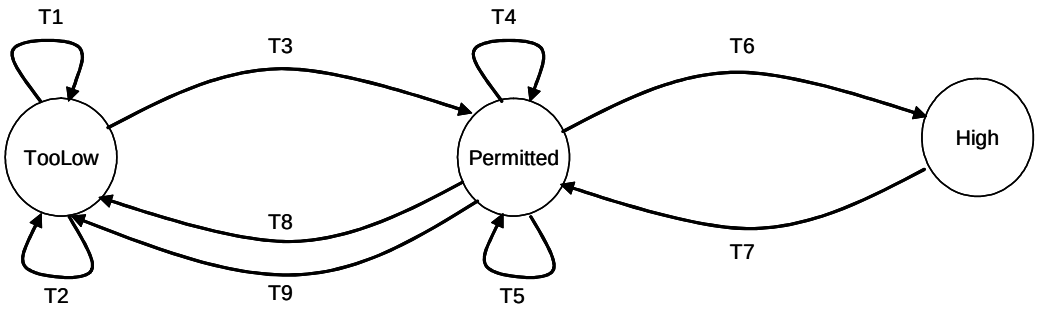


FIG. 16 - La représentation du système SIS dans un graphe

Tr.	État précédent	Prédicat	Action	Sortie	État suivant
T1	TooLow	@T(BK=ON) WHEN RT=OFF	OV=TRUE	SI=OFF	TooLow
T2	TooLow	@T(RT=ON)	OV=FALSE	SI=ON	TooLow
T3	TooLow	@T(WP>=low)	NOP	SI=OFF	Permitted
T4	Permitted	@T(BK=ON) WHEN RT=OFF	OV=TRUE	SI=OFF	Permitted
T5	Permitted	@T(RT=ON)	OV=FALSE	SI=OFF	Permitted
T6	Permitted	@T(WP>=permit)	--	SI=OFF	High
T7	High	@T(WP<permit)	--	SI=OFF	Permitted
T8	Permitted	@T(WP<low) ^ OV	--	SI=OFF	TooLow
T9	Permitted	@T(WP<low) ^ not OV	--	SI=ON	TooLow

TAB. 30 - La représentation du système SIS dans un tableau de prédicats

Les scénarios S_i obtenus suite à l'application de l'algorithme de la génération de scénarios du chapitre 4 à la spécification SIS sont les suivants :

S1 : T1.T3.T5.T6.T7.T9

S2 : T1.T3.T6.T7.T9

S3 : T1.T2.T3.T6.T7.T9

S4 : T3.T4.T5.T8

S5 : T1.T3.T8.T2

À partir de la spécification des besoins, nous utilisons le modèle SCR pour formaliser la spécification, après nous faisons la conversion de la spécification SCR dans un tableau des prédicats EFSM où chaque événement du système SCR est représenté dans un graphe EFSM de transitions. Pour cette raison, chaque scénario est composé par une ou plusieurs transitions et chaque transition est composée par un événement.

La couverture des différentes transitions par les cinq scénarios est présentée dans le tableau ci-dessous. Les lignes correspondent aux scénarios et les colonnes correspondent aux transitions.

	T1	T2	T3	T4	T5	T6	T7	T8	T9
S1	X		X		X	X	X		X
S2	X		X			X	X		X
S3	X	X	X			X	X		X
S4			X	X	X			X	
S5	X	X	X					X	

TAB. 31 - Les scénarios par transitions

La redondance est un aspect prédominant qu'on trouve dans ce tableau. Par exemple, le scénario S2 est redondant parce que S2 est contenu dans S1. Par conséquent, le testeur prendra plus du temps pour tester l'ensemble de scénarios par rapport à l'ensemble minimal des scénarios.

Pour limiter la redondance, nous devons trouver un ensemble minimal de scénarios couvrant toutes les transitions d'un ensemble donné. La spécification de besoins est représentée dans un graphe EFSM de transitions. Après avoir exécuté l'algorithme de génération des scénarios en utilisant le critère de couverture par transitions, nous avons comme résultat un ensemble de scénarios qui couvre au moins une fois chaque transition de la spécification des besoins. L'objectif de l'algorithme de minimisation de scénarios est de réduire les redondances au niveau des scénarios ou de transitions en respectant le critère de couverture par transitions, c'est-à-dire, que l'ensemble minimal des scénarios doit couvrir au moins une fois chaque transition de la spécification des besoins.

Observons aussi que l'existence d'un outil permettant de générer automatiquement des jeux de tests après chaque modification de la spécification met en évidence l'importance de calculer un jeu de test minimal et de le trouver rapidement. Nous analyserons ces deux aspects dans nos tentatives de trouver un algorithme plus efficace que ceux présentés dans la littérature.

Dans la pratique, les modifications à la spécification de besoins ont lieu continuellement. Si un nouveau jeu de tests est créé, l'ancien jeu n'est pas éliminé complètement. On garde les tests qui ont échoué pour les vérifier de nouveau.

Un ensemble minimal de scénarios peut être défini

- en fonction du nombre minimal de scénarios, ou
- en fonction du nombre minimal de transitions qui sont contenues dans les scénarios.

Exemple : Si nous avons la spécification suivante avec quatre (4) scénarios et six (6) transitions, l'ensemble minimal de scénarios en fonction du nombre minimal de scénarios est *{S1,S3}* et en fonction du nombre minimal des transitions est *{S1,S2,S4}*.

	T1	T2	T3	T4	T5	T6
S1	X	X	X			
S2				X	X	
S3			X	X	X	X
S4						X

TAB. 32 - La représentation des scénarios par transitions

L'importance d'utiliser le critère de minimisation de scénarios en fonction du nombre minimal de transitions est de réduire le coût et le temps pour exécuter un test. Par exemple, le coût pour tester un système de vol, un mécanisme de freinage d'un train ou un réacteur nucléaire est trop élevé. Le coût et le temps pour arrêter le système, exécuter le test, valider les résultats et démarrer le système est trop élevé. Pour cette raison, nous avons développé dans notre recherche, des algorithmes qui utilisent le critère de minimisation des scénarios basé sur le nombre de transitions.

Par la suite, nous présentons les différents heuristiques Greedy comme : classique Greedy, HGS et Delayed Greedy pour résoudre la minimisation de scénarios en fonction du nombre minimal de scénarios et nous avons développé les

algorithmes Cardenas Greedy et Merge Greedy qui utilisent le critère de minimisation des scénarios en fonction du nombre minimal de transitions.

5.2 Heuristique traditionnelle Greedy.

La minimisation de scénarios est un problème NP-complet, par conséquent l'utilisation d'une heuristique pour trouver une solution à ce problème devient importante.

5.2.1 L'heuristique classique Greedy.

L'algorithme classique Greedy [5] minimise les scénarios de la manière suivante: il commence par choisir le scénario qui couvre le plus de transitions du système, ensuite le processus se répète en choisissant le scénario qui couvre le plus de transitions non couvertes pour le moment. Quand il y a plus d'un scénario à choisir, l'algorithme choisit le scénario d'une manière arbitraire. L'algorithme finit quand l'ensemble de scénarios couvre toutes les transitions du système.

L'heuristique classique Greedy n'est pas toujours la meilleure option parce qu'elle cherche le scénario le plus long pour couvrir le plus de transitions. Exemple:

	T1	T2	T3	T4	T5	T6
S1	X	X	X			
S2	X			X		
S3		X			X	
S4			X			X
S5					X	

TAB. 33 - L'exemple de scénarios pour classique Greedy

Les colonnes représentent les transitions et les lignes les scénarios. La couverture d'une transition par un scénario donné est indiquée par un 'X' dans une colonne qui est associée à la transition. L'heuristique classique Greedy dans cet exemple sélectionne le scénario le plus long (*S1*) qui couvre *T1, T2* et *T3*. Ensuite, les scénarios qu'il est possible de choisir soit *S2, S3* ou *S4* de longueur 2 pour couvrir les transitions *T4, T5* ou *T6*. On peut dire que l'algorithme choisit de manière arbitraire *S2* pour couvrir la transition *T4*, ensuite *S3* et *S4*. Le résultat de

l'application de l'heuristique classique Greedy est *{S1, S2, S3, S4}* au lieu de *{S2, S3, S4}* qui représente l'ensemble minimal de scénarios.

5.2.2 HGS

Cet algorithme a pour objectif de minimiser les scénarios, il a été créé par Harrold, Gupta et Soffa [8]. L'heuristique contient les étapes suivantes :

- Former des sous-ensembles *SEi* : Chaque *SEi* contient tous les scénarios *Sj* qui couvrent la transition *Ti*. Par exemple :

	T1	T2	T3	T4	T5
S1	X	X			
S2	X		X		X
S3		X	X	X	
S4			X	X	
S5				X	
S6					X
S7					X

TAB. 34 - L'exemple de scénarios pour HGS

La transition *T1* est contenue dans les scénarios *S1* et *S2* qui forment le sous-ensemble *SE1*

SE1 = {S1,S2}, SE2 = {S1,S3}, SE3 = {S2,S3,S4},
SE4 = {S3,S4,S5} et SE5 = {S2,S6,S7}

- Calculer la cardinalité de chaque *SEi*. Par exemple :

  La cardinalité de SE1 = {S1,S2} et SE2 = {S1,S3} est 2.
  La cardinalité de SE3 = {S2,S3,S4}, SE4 = {S3,S4,S5} et
  SE5 = {S2,S6,S7} est 3

- Choisir les scénarios qui ne sont pas marqués à partir des sous-ensembles de cardinalité *n :* l'algorithme commence à choisir les sous-ensembles de cardinalité la plus petite jusqu'à la cardinalité la plus grande. Une fois qu'un sous-ensemble a été choisi, l'algorithme sélectionne de manière aléatoire un scénario qui n'a pas été marqué, ensuite il parcourt tous les *SEi* en marquant le scénario choisi comme visité. L'algorithme se termine quand toutes les transitions sont contenues dans un ensemble minimal de scénarios. Dans notre exemple, il n'y a pas de sous-ensembles avec cardinalité 1, alors l'algorithme choisit *SE1* et *SE2* avec cardinalité 2. Les scénarios possibles à choisir sont *S1*, *S2* et *S3*, la sélection est *S1*; après, *SE3*, *SE4* et *SE5* sont

choisis avec la cardinalité 3. Les scénarios candidats sont : *S2*, *S3* et *S4*, *S5*, *S6* et *S7*, la sélection est *S2*. L'unique transition qui manque pour couvrir est *T4*, les tests susceptibles d'être choisis dans le sous-ensemble *SE4* sont : *S3*, *S4 et S5* tandis que la sélection est *S3*. Le résultat après avoir exécuté HGS sont les scénarios suivants : *{S1*, *S2* et *S3}*, mais il y a des transitions du scénario *S1* qui sont déjà contenues dans les scénarios *S2* et *S3.*

Les résultats de l'ensemble minimal de scénarios, en utilisant les heuristiques classiques Greedy et HGS, présentent des redondances comme nous l'avons constaté dans les deux exemples précédents, où les transitions du scénario *S1* sont contenues dans l'ensemble minimal de scénarios générés par chaque algorithme.

Maintenant, nous présentons une autre approche nommée Delayed Greedy qui est plus performante que les heuristiques classiques Greedy et HGS.

5.3 Delayed Greedy

L'algorithme Delayed Greedy détermine les relations entre les scénarios et entre les transitions du système avant de choisir le scénario le plus adéquat. Pour faire ce type d'analyse, l'algorithme utilise plusieurs stratégies de réduction par scénario et par transitions [12]. Pour mieux comprendre l'algorithme Delayed Greedy nous utiliserons l'exemple suivant :

	T1	T2	T3	T4	T5	T6
S1	X	X	X			
S2	X			X		
S3		X			X	
S4			X			X
S5					X	

TAB. 35 - L'exemple de scénarios pour Delayed Greedy

Les stratégies de réduction sont les suivantes :

5.3.1 Réduction par scénario

La réduction par scénario consiste à trouver deux scénarios pour exemple *S3* et *S5* tels que le scénario *S5* est contenu complètement dans le scénario *S3*. Le résultat

consiste à éliminer le scénario *S5*. Après la réduction du scénario *S5* le résultat est le suivant :

	T1	T2	T3	T4	T5	T6
S1	X	X	X			
S2	X			X		
S3		X			X	
S4			X			X

TAB. 36 – La réduction par scénario

5.3.2 Réduction par transition

La réduction par transition est une réduction par colonnes, le but de cette réduction est de trouver deux colonnes, par exemple *T6* et *T3* telles que la colonne *T6* est contenue dans la colonne *T3*. Le résultat éliminera la colonne *T3*.

	T1	T2	T4	T5	T6
S1	X	X			
S2	X		X		
S3		X		X	
S4					X

TAB. 37 – La réduction par transition

5.3.3 Réduction de propriétaire

La réduction de propriétaire se fait quand la cardinalité de chaque transition est égale à 1, par conséquent l'ensemble actuel de scénarios devient l'ensemble minimal des scénarios. Après avoir fait la réduction de propriétaire, le tableau sera vide et le processus de réduction arrêtera. Dans notre exemple, la réduction de propriétaire commence quand la cardinalité des transitions T2, T4, T5, T6 est égale à 1, alors l'ensemble minimal des scénarios est : S2, S3 et S4.

	T2	T4	T5	T6
S2		X		
S3	X		X	
S4				X

TAB. 38 – La réduction de propriétaire

Le résultat final de l'ensemble minimal des scénarios est le suivant :

	T1	T2	T3	T4	T5	T6
S2	X			X		
S3		X			X	
S4			X			X

TAB. 39 - L'ensemble minimal pour Delayed Greedy

5.3.4 Interférence

Une interférence est détectée dans le cas où l'algorithme ne peut pas faire aucune réduction par scénario, par transition ou par propriétaire. L'interférence sera résolue en utilisant l'heuristique classique Greedy.

L'algorithme Delayed Greedy contient les étapes suivantes dans un bloc répétitif qui fait :

- La réduction par scénario
- La réduction par transition
- La réduction de propriétaire
- S'il y a une interférence, elle sera résolue en appliquant l'heuristique classique Greedy.

L'algorithme Delayed Greedy garantit de produire un ensemble de scénarios qui ont la même taille ou une taille plus petite que celles produites par l'algorithme classique Greedy. Dans les expériences réalisées, l'ensemble minimal de Delayed Greedy est toujours plus petit ou égal en nombre aux scénarios comparés à l'heuristique Greedy.

Par la suite, nous présentons deux algorithmes qui font la minimisation de scénarios en trouvant le nombre minimal de transitions contenues dans le jeu de test.

Une des stratégies pour minimiser un jeu de test est de trouver le plus petit nombre de scénarios qui couvrent toutes les transitions du système, mais cette stratégie n'est pas toujours la meilleure solution, parce qu'un scénario est composé de transitions. Nous pouvons trouver des redondances au niveau des transitions dans l'ensemble minimal de scénarios, c'est-à-dire qu'au moment d'exécuter l'ensemble minimal de scénarios, il est possible d'exécuter plusieurs fois une même transition. Cette redondance est générée quand l'algorithme de minimisation de scénarios utilise

la réduction de scénarios en faisant la fusion des scénarios qui ont des transitions communes.

Nous redéfinirons le concept d'ensemble minimal de scénarios qui consiste à minimiser le nombre total de transitions qui sont contenues dans un ensemble de scénarios et qui sont reliées au moins une fois à chaque transition du système.

Pour identifier le nouvel ensemble minimal de scénarios, les algorithmes se basent sur la minimisation du nombre total d'événements reliés aux transitions du système et sont nommés Cardenas Greedy et Merge Greedy.

5.4 Cardenas Greedy

Cardenas Greedy est composé des étapes suivantes :

5.4.1 Cardinalité par transition

À partir du tableau de scénarios par transitions, l'algorithme calcule le nombre de transitions pour chaque colonne. Ensuite, il choisit les cardinalités les plus petites du total par transition.

5.4.2 Ensemble de scénario-cible

Nous utilisons le concept de scénario-cible pour identifier les scénarios candidats et construire l'ensemble minimal des scénarios. La création du scénario-cible commence par l'identification des scénarios qui sont associés aux transitions avec la cardinalité la plus petite. Les scénarios qui ont la cardinalité égale à un (1) seront fusionnés pour avoir comme résultat le scénario cible; dans le cas contraire, le scénario cible sera sélectionné entre les scénarios qui contiennent la cardinalité par transitions la plus petite en utilisant le critère de la meilleure fusion, sans ou avec redondance.

5.4.3 Meilleure fusion sans redondance

À partir du scénario-cible, l'algorithme identifie le nombre de transitions nouvelles et communes qui apportent un scénario, au scénario-cible. Le critère pour choisir le meilleur scénario sera donné par le nombre maximal de transitions nouvelles et zéro transition commune qui apportent le scénario, au scénario-cible. Une fois identifié, le meilleur scénario sera ajouté à l'ensemble minimal de scénarios.

Si, pendant le processus de fusion, nous trouvons plusieurs scénarios candidats à fusionner, le critère de sélection sera de choisir le scénario de manière aléatoire.

5.4.4 Meilleure fusion avec redondance

Si, après avoir identifié toutes les fusions sans redondance, le scénario-cible ne couvre pas toutes les transitions, nous procéderons à identifier la meilleure fusion qui contient le nombre minimal de transitions communes entre les scénarios. Une fois identifiées, les meilleurs scénarios seront ajoutés à l'ensemble minimal de scénarios.

Si, pendant le processus de fusion nous trouvons plusieurs scénarios candidats à fusionner, le critère de sélection se fera de manière aléatoire.

Exemple : le total par transition :

	T1	T2	T3	T4	T5	T6	T7	T8
S1	X		X		X	X		
S2			X					X
S3			X	X			X	X
S4					X		X	X
S5	X	X						
S6				X		X		
S7							X	X
	2	1	3	2	2	2	3	4

TAB. 40 - Le nombre total par transitions contenues dans les scénarios

- La transition la plus petite par fréquence est : *T2*.
- Le scénario cible : *S5*.
- La meilleure fusion sans redondance à partir du scénario cible *S5* est *S3* parce qu'il apporte quatre transitions nouvelles.

L'ensemble des scénarios cibles est *{S5, S3}*. Le tableau de scénarios par transition est :

	T1	T2	T3	T4	T5	T6	T7	T8
S1	X		X		X	X		
S2			X					X
S4					X		X	X
S5.S3	X	X	X	X			X	X
S7							X	X
	2	1	3	2	2	2	3	4

TAB. 41 - La réduction des scénarios

- La meilleure fusion avec redondance : à partir du scénario- cible *S5.S3*, le scénario candidat est *S1* qui a deux nouvelles transitions et deux transitions communes. Le résultat final de l'ensemble minimal de scénarios est *{S5.S3.S1}* qui a une somme totale de 10 transitions.

5.5 Merge Greedy

L'algorithme contient les étapes suivantes :

- Calculer la cardinalité par transition (le nombre de transitions par colonnes).
- Trier les cardinalités par transition de la plus petite à la plus grande.
- Choisir la cardinalité la plus petite entre les transitions disponibles. Les scénarios qui contiennent les dites transitions seront ajoutés à l'ensemble minimal des scénarios.
- Compter le nombre de transitions nouvelles et communes par chaque scénario disponible par rapport aux transitions contenues dans l'ensemble minimal des scénarios.
- Choisir le scénario avec la plus grande différence entre les transitions nouvelles et communes. S'il y a plusieurs scénarios à choisir, l'algorithme utilisera le concept de meilleure fusion sans redondance et meilleure fusion avec redondance pour choisir un scénario et l'ajouter à l'ensemble minimal des scénarios.
- Une fois le scénario choisi, le scénario et les transitions seront marqués comme non disponibles.
- L'algorithme se termine quand il n'y a plus de transitions disponibles.

5.6 Implémentation des algorithmes Greedy

Le but de cette étape est d'implémenter les algorithmes classiques Greedy, HGS, Cardenas Greedy, Merge Greedy et Delayed Greedy, de comparer les résultats de minimisation des scénarios faits par chaque heuristique et de faire l'analyse des résultats pour en tirer des conclusions.

5.6.1 Implémenter les algorithmes

Ils ont été développés en java en utilisant la suite NetBeans IDE 5.0. Présentement, il y a sept (7) exemples d'ensemble de scénarios desquels six (6) exemples sont des ensembles de scénarios statiques et le septième est généré de manière aléatoire. L'exemple 7 consiste à générer un tableau avec *s* scénarios et *t* transitions. Nous définirons le nombre de transitions qui seront distribuées dans le tableau *s x t* pour avoir l'ensemble de scénarios définitifs.

L'heuristique HGS est une version améliorée de la version originale. La modification apportée à l'algorithme est la manière de résoudre les interférences. Quand il y a plusieurs scénarios à choisir avec la même cardinalité, il fait l'analyse récursive des cardinalités à un niveau supérieur jusqu'au moment de choisir la cardinalité la plus petite pour éliminer l'interférence et choisir un scénario.

Nous présentons la visualisation des résultats dans deux étapes. La première phase de l'analyse est basée sur les six exemples qui contiennent des ensembles de scénarios statiques et la deuxième phase analyse des ensembles de scénarios qui sont générés de manière aléatoire contenus dans l'exemple 7.

5.6.2 Jeux de test statique

Les données que nous considérons sont : l'ensemble de scénarios, l'ensemble minimal de scénarios et les indicateurs pour évaluer la performance. Le détail des données est le suivant :

- L'ensemble de scénarios est caractérisé par le nombre de scénarios (No. S), le nombre de transitions (No. T) et le total de transitions contenues dans l'ensemble de scénarios (Total T).
- L'ensemble minimal de scénarios : est composée par l'ensemble minimal de scénarios (Scénario), le nombre de scénarios dans l'ensemble minimal (No. S) et le nombre total de transitions contenues dans l'ensemble minimal de scénarios (Total T).
- Les indicateurs : le ratio de réduction de scénarios consiste à calculer le pourcentage de réductions entre le nombre de scénarios contenus dans l'ensemble minimal de scénarios et le nombre de scénarios dans l'ensemble de scénarios. Le ratio de redondance des transitions est le résultat de diviser le nombre de transitions contenues dans l'ensemble de scénarios par le nombre total de transitions qui sont contenues dans l'ensemble minimal de scénarios. Cet indicateur est bon quand il est plus proche de 1. Les résultats obtenus dans les premiers six (6) exemples par heuristique qui ont été pris de la littérature [12] et des exemples que nous avons créé, sont présentés dans le tableau (TAB. 42).

La figure (FIG. 17) visualise la distribution de l'indicateur de réduction de scénarios pour chacun des exemples et heuristiques.

L'analyse pour l'indicateur de réduction de scénarios est le suivant :

- Les heuristiques Greedy font une réduction de scénarios entre 17% et 71%. Le classique Greedy est l'heuristique qui a rapporté le plus petit pourcentage de réduction de scénarios. HGS, Cardenas Greedy et Delayed Greedy présentent des résultats similaires entre l'exemple 1 et 5, à l'exception de l'exemple 6 où Delayed Greedy est plus performant.
- Le meilleur résultat obtenu dans la réduction des transitions a été Delayed Greedy.

La figure (FIG. 18) visualisant la distribution de l'indicateur de redondance des transitions. Les résultats obtenus après avoir analysé cet indicateur sont les suivants:

- Le classique Greedy est l'heuristique qui a rapporté le plus grand facteur de redondance des transitions. HGS, Cardenas Greedy et Delayed Greedy présentent des résultats similaires.
- Merge Greedy obtient les meilleurs résultats dans tous les exemples.

5.6.3 Jeu de tests généré de manière aléatoire.

La deuxième phase est basée sur l'exemple 7. Les données que nous ajoutons à l'analyse sont : le pourcentage de remplissage et le temps d'exécution en millisecondes. Le pourcentage de remplissage détermine le nombre de transitions qui seront distribuées dans le jeu de tests de manière aléatoire. Dans l'analyse de jeu de test nous avons ajouté le temps d'exécution pour trouver une relation avec les différents pourcentages de remplissage pour chaque jeu de test. Le tableau qui contient les résultats est le (TAB. 38).

Pour mieux comprendre les résultats des indicateurs, nous présentons des graphiques pour représenter l'indicateur de réduction de tests (FIG. 19), la redondance des transitions (FIG. 20) et le temps d'exécution (FIG.21). Après ça, nous faisons une analyse de chaque indicateur.

L'analyse pour l'indicateur de réduction de scénarios (FIG. 19) est la suivante :

- Le pourcentage de réduction décroît quand le taux de remplissage est bas.
- Les heuristiques qui préservent une tendance entre 100% et 85% pendant les différents pourcentages de remplissage de l'ensemble de scénarios sont : HGS, Merge Greedy, Deladey Greedy et Delayed Greedy v2, au contraire de Cardenas Greedy et Classique Greedy.
- Les heuristiques qui ont eu la meilleure performance ont été HGS et Delayed Greedy v2.

L'heuristique Delayed Greedy a été modifiée pour avoir une version 2 améliorée. Cette version a eu pour objectif de réduire le nombre de transitions contenus dans l'ensemble minimal de scénarios et de diminuer le temps d'exécution. La modification apportée dans Delayed Greedy v2 a été de résoudre les interférences en identifiant le scénario qui apporte le plus de transitions nouvelles à l'ensemble minimal de scénarios. (Ce critère est défini dans l'heuristique Cardenas Greedy).

Exemple	Nom de l'heuristique	Scénarios			Ensemble minimal de scénarios			Indicateurs	
		No. S	No. T	Total T	Scénario	No. S	Total T	Réduction de scénario	Redondance des transitions
1	Classique Greedy	5	6	10	1.2.3.4	4	9	20%	1.50
	HGS				2.4.3	3	6	40%	1.00
	Cardenas Greedy				2.4.3	3	6	40%	1.00
	Merge Greedy				2.4.3	3	6	40%	1.00
	Delayed Greedy				2.3.4	3	6	40%	1.00
2	Classique Greedy	7	5	13	1.2.3	3	8	57%	1.60
	HGS				2.3	2	6	71%	1.20
	Cardenas Greedy				2 3	2	6	71%	1.20
	Merge Greedy				1.4.6	3	5	57%	1.00
	Delayed Greedy				3.2	2	6	71%	1.20
3	Classique Greedy	7	8	19	1.2.3.4.5	5	15	29%	1.88
	HGS				5.3.1	3	10	57%	1.25
	Cardenas Greedy				5.3.1	3	10	57%	1.25
	Merge Greedy				5.6.4.2	4	9	43%	1.13
	Delayed Greedy				5.1.3	3	10	57%	1.25
4	Classique Greedy	4	5	12	1.2.3	3	9	25%	1.80
	HGS				1.3	2	6	50%	1.20
	Cardenas Greedy				1.3	2	6	50%	1.20
	Merge Greedy				1.3	2	6	50%	1.20
	Delayed Greedy				1.3	2	6	50%	1.20
5	Classique Greedy	3	3	5	1.2	2	4	33%	1.33
	HGS				1.2	2	4	33%	1.33
	Cardenas Greedy				1.3	2	3	33%	1.00
	Merge Greedy				1.3	2	3	33%	1.00
	Delayed Greedy				1.2	2	4	33%	1.33
6	Classique Greedy	6	6	12	1.2.3.4.5	5	10	17%	1.67
	HGS				5.2.1.4	4	8	33%	1.33
	Cardenas Greedy				5 2 1 4	4	8	33%	1.33
	Merge Greedy				5.3.4	3	6	50%	1.00
	Delayed Greedy				5.3.4	3	6	50%	1.00

TAB. 42 - Le résultat des algorithmes de minimisation de scénarios sur les ensembles de scénarios statiques

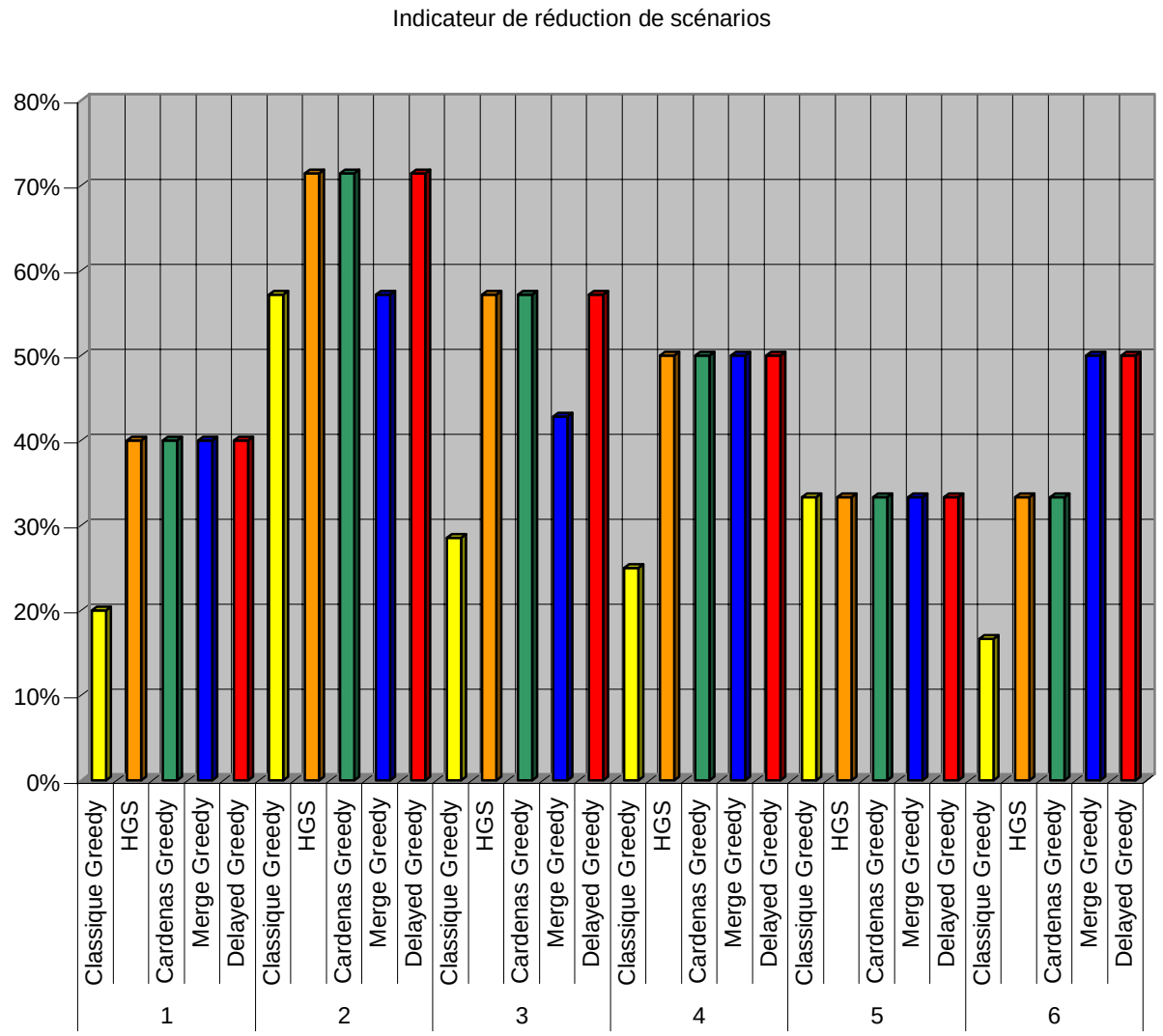


FIG. 17 - L'indicateur de réduction de scénarios sur les ensembles de scénarios statiques

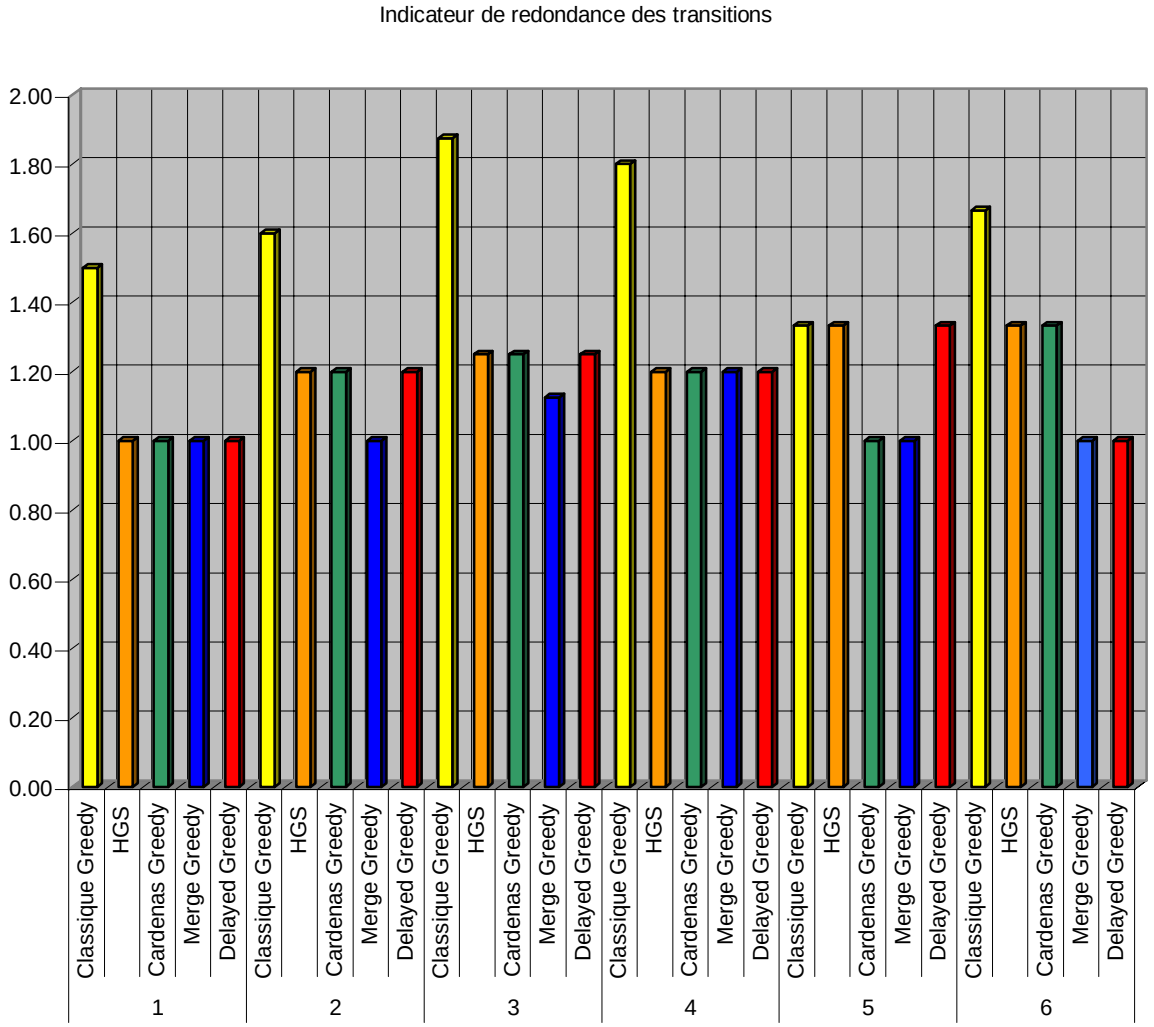
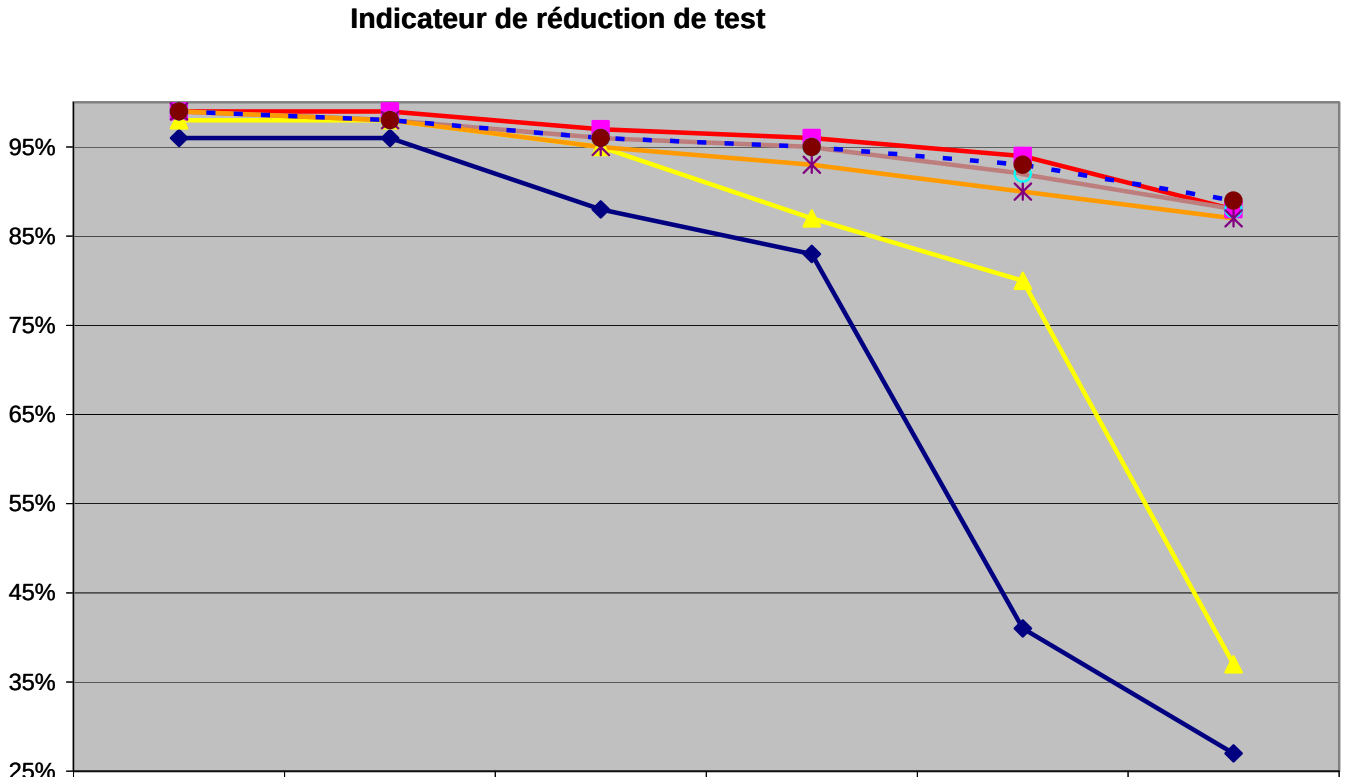


FIG. 18 - L'indicateur de redondance de transitions sur l'ensemble de scénarios statiques

Ex.	Nom de l'heuristique	Ensemble de scénarios				Ensemble minimal		Indicateurs		Temps d'éxec (milliseg)
		No. S	No. T	Total T	% de remplissage	No. S	Total T	Réduc. de scénarios	Redond. des transitions	
7	Classique Greedy	500	160	40000	50%	20	1254	96%	7.84	0
	HGS					5	355	99%	2.22	50
	Cardenas Greedy					11	606	98%	3.79	10
	Merge Greedy					6	385	99%	2.41	30
	Delayed Greedy					7	447	99%	2.79	190
	Delayed Greedy v2					7	359	99%	2.24	200
7	Classique Greedy	500	160	24000	30%	20	845	96%	5.28	0
	HGS					7	329	99%	2.06	50
	Cardenas Greedy					10	374	98%	2.34	20
	Merge Greedy					10	388	98%	2.43	30
	Delayed Greedy					11	459	98%	2.87	561
	Delayed Greedy v2					8	336	98%	2.10	330
7	Classique Greedy	500	160	8000	10%	61	967	88%	6.04	0
	HGS					14	288	97%	1.80	100
	Cardenas Greedy					24	292	95%	1.83	10
	Merge Greedy					19	313	96%	1.96	30
	Delayed Greedy					27	437	95%	2.73	1082
	Delayed Greedy v2					18	284	96%	1.78	651
7	Classique Greedy	500	160	4800	6%	87	852	83%	5.33	0
	HGS					19	252	96%	1.58	140
	Cardenas Greedy					65	399	87%	2.49	30
	Merge Greedy					27	284	95%	1.78	40
	Delayed Greedy					36	374	93%	2.34	1743
	Delayed Greedy v2					23	266	95%	1.66	861
7	Classique Greedy	500	160	2400	3%	296	1483	41%	9.27	0
	HGS					31	249	94%	1.56	260
	Cardenas Greedy					101	310	80%	1.94	40
	Merge Greedy					38	234	92%	1.46	41
	Delayed Greedy					50	307	90%	1.92	2043
	Delayed Greedy v2					33	211	93%	1.32	1111
7	Classique Greedy	500	160	800	1%	366	699	27%	4.37	0
	HGS					62	209	88%	1.31	1392
	Cardenas Greedy					316	401	37%	2.51	140
	Merge Greedy					61	197	88%	1.23	81
	Delayed Greedy					65	210	87%	1.31	1041
	Delayed Greedy v2					56	189	89%	1.18	751

TAB. 43 - Le résultat des algorithmes de génération de scénarios sur l'ensemble de scénarios aléatoires



	R 50%	R 30%	R 10%	R 6%	R 3%	R 1%
Classique Greedy	96%	96%	88%	83%	41%	27%
HGS	99%	99%	97%	96%	94%	88%
Cardenas Greedy	98%	98%	95%	87%	80%	37%
Merge Greedy	99%	98%	96%	95%	92%	88%
Delayed Greedy	99%	98%	95%	93%	90%	87%
Delayed Greedy v2	99%	98%	96%	95%	93%	89%

FIG. 19 - L'indicateur de réduction de scénarios sur les scénarios aléatoires

L'analyse pour l'indicateur de redondance des transitions (FIG. 20) est la suivante :

- Les résultats obtenus par Classique Greedy sont les plus éloignés de 1, qui indique un taux élevé de répondants.
- L'indicateur de redondance diminue quand le taux de remplissage diminue aussi.
- Les heuristiques Delayed Greedy v2 et HGS ont été les plus performantes.

L'indicateur du temps d'exécution (FIG. 21) a comme objectif de trouver une relation entre le temps d'exécution et le pourcentage de distribution des transitions pour chaque ensemble de scénarios.

L'analyse pour l'indicateur du temps d'exécution (FIG. 21) est la suivante :

- L'heuristique Delayed Greedy v2 a généré 45% de réduction de temps d'exécution approximatif à l'égard de Delayed Greedy.
- Le temps d'exécution de HGS augmente de manière exponentielle quand le taux de remplissage est petit.
- Les heuristiques qui ont obtenu les meilleurs temps d'exécution ont été Classique Greedy, Merge Greedy et Cardenas Greedy.

Si nous considérons la redéfinition de l'ensemble minimal de scénarios comme le nombre minimal de transitions, la meilleure heuristique parmi les premiers six (6) exemples est Merge Greedy et pour le 7ie exemple est Delayed Greedy v2, principalement à cause du nombre minimal de transitions et du temps d'exécution. Les résultats de HGS ont été similaires à ceux de Delayed Greedy v2, mais le temps d'exécution a été sa faiblesse.

Nous sommes conscients que les tableaux que nous avons créés en déterminant le nombre de tests, de transitions et le taux de remplissage peuvent varier par rapport à des spécifications réelles. Pour cette raison, un des travaux futurs pour continuer cette recherche sera d'utiliser des outils pour vérifier si les résultats sur les données réelles confirment les résultats obtenus dans les exemples.

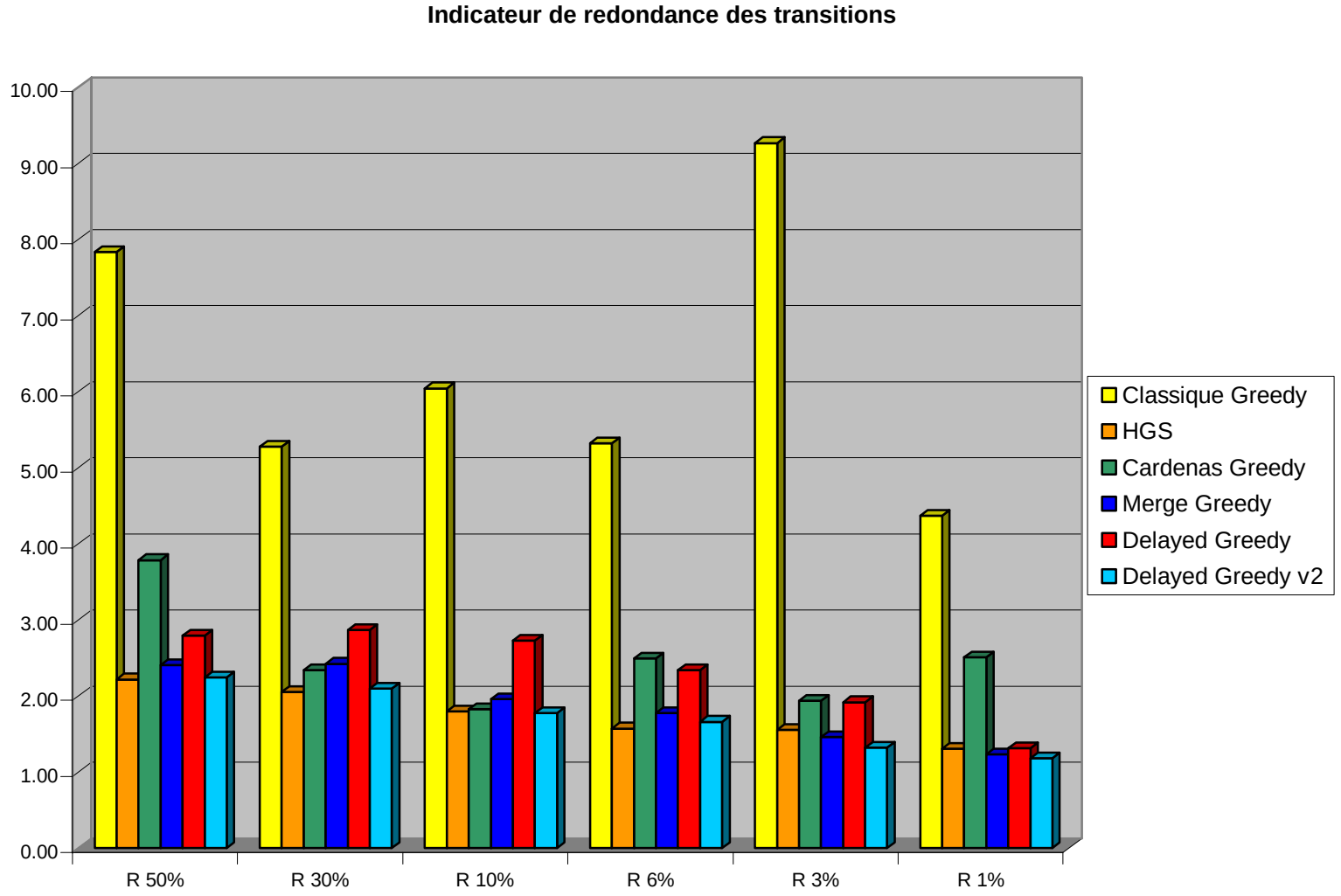


FIG. 20 - L'indicateur de redondances des transitions (Ensemble de scénarios aléatoires)

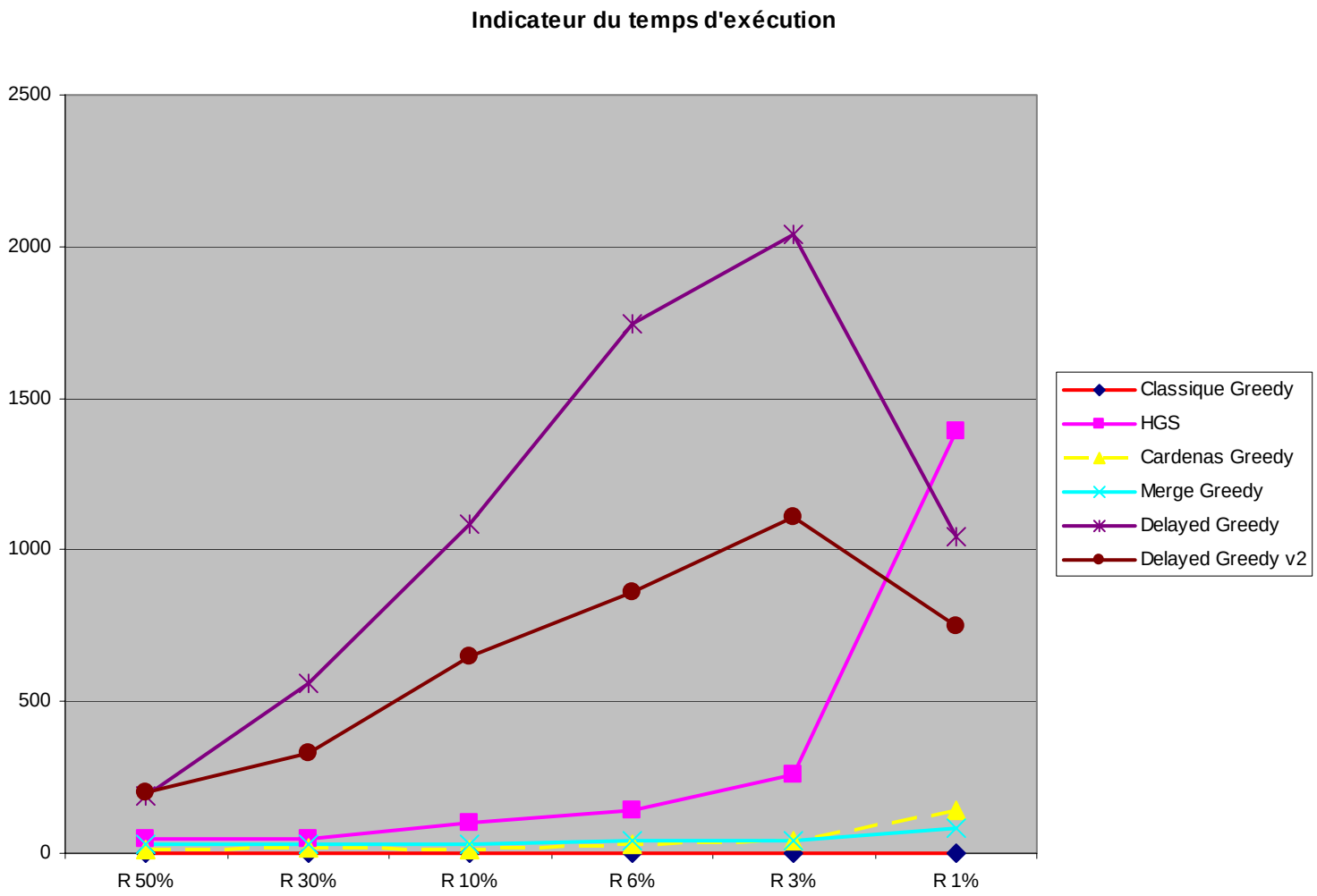


FIG. 21 - L'indicateur du temps d'exécution sur l'ensemble de scénarios aléatoires

Chapitre 6

Conclusions

Dans ce document, nous avons présenté un nouvel algorithme de génération automatique des scénarios à partir d'une spécification SCR et une nouvelle stratégie pour minimiser un ensemble de scénarios. Le résultat après l'exécution des algorithmes de génération et de minimisation de scénarios est d'avoir un ensemble minimal des scénarios qui couvrent au moins une fois, les transitions de la spécification.

Notre algorithme de génération de scénarios est basé sur le critère de couverture par transition. Il donne comme résultats :

- un ensemble de scénarios visant le critère de couverture par transition;
- les transitions qui sont atteignables;
- les transitions qui ne sont pas atteignables après avoir évalué les événements complémentaires;
- les transitions qui ne sont pas atteignables après avoir exécuté un vérificateur de modèle (spin ou SMV);
- l'indicateur de couverture.

De plus, notre technique évite l'explosion d'états en mémorisant les états déjà traités par l'algorithme.

Une fois, l'algorithme de génération des scénarios définie, nous avons développé un prototype. Pour évaluer le prototype, nous avons pris la spécification SIS (Safety Injection System) comme référence dans notre recherche.

Les résultats du prototype de génération de scénarios sont : un ensemble de scenarios où chaque scénario est composé d'un ensemble de transitions reliées aux exigences de la spécification, un indicateur de couverture qui fait la relation entre les transitions atteignables et non atteignables, une liste de transitions qui sont dans l'ensemble de scénarios et une liste des transitions qui ne sont pas dans l'ensemble de scénarios. Les résultats de la génération des scénarios pour la spécification SIS sont les suivants :

- Nombre des scénarios (*Si*) générés : 7.
- Total des transitions (*Ti*) contenues dans l'ensemble de scénarios : 34
- Ensemble de scénarios :
  S1=T0.T1.T2.T5.T6.T8
  S2=T0.T2.T4
  S3=T0.T2.T5.T6.T4
  S4=T0.T2.T5.T6.T7.T1
  S5=T1.T2.T5.T6.T8.T9
  S6=T2.T3
- Taux de couverture : 100%
- Liste de transitions atteignables : T0,T1,T2,T3,T4,T5,T6,T7,T8 et T9
- Liste de transitions non atteignables : aucune.

Notre algorithme génère moins ou une qualité égale de scénarios que l'algorithme de génération de scénarios à l'aide des vérificateurs de modèles spin et SMV [7].

Le nombre des scénarios générés pour notre algorithme en exécutant la spécification SIS est de sept scénarios. L'algorithme de génération des scénarios fait à l'aide des vérificateurs de modèles spin et SMV [7] (voir numéral 2.3 chapitre 2) ont obtenu sept scénarios de spin et 14 de SMV.

Nous avons présenté un algorithme de minimisation de scénarios basé sur le nombre minimal de transitions contenues dans l'ensemble des scénarios. Selon nos expériences, Delayed Greedy v2 donne une meilleure performance que les autres heuristiques présentées dans la littérature.

Notre algorithme de minimisation d'ensemble de scénarios réduit plus ou une quantité égale que l'algorithme de minimisation qui utilise les vérificateurs de modèles spin et SMV. De plus, notre ensemble minimal de scénarios contient moins de variables surveillées (transitions) que l'algorithme de minimisation qui utilise spin ou SMV.

Le résultat obtenu après l'exécution de notre algorithme de minimisation pour la spécification SIS est un ensemble minimal de 4 scénarios (S7, S3, S6 et S5). L'algorithme de minimisation basé en spin réduit l'ensemble de scénarios à 5 et SMV à 11.

Le nombre total des variables surveillées de l'ensemble minimal de notre algorithme pour la spécification SIS est de 19 par rapport à 280 variables surveillées qui sont contenues dans l'ensemble minimal de spin et 62 variables surveillées de SMV.

Les travaux futurs que nous considérons pour la génération automatique de test sont les suivants :

- L'intégration des prototypes que nous avons développés avec le logiciel « l'outil de test pour les spécifications SCR » qui a été développé par M. Iglewski;
- La validation des prototypes de génération et de minimisation avec d'autres spécifications réelles, parce que nous avons validé nos prototypes seulement avec la spécification SIS;
- notre recherche est la base pour faire la vérification du logiciel (voir FIG. 1). Cette recherche a eu comme objectif de générer l'ensemble de scénarios, la minimisation des scénarios et le développement d'un oracle. La recherche future sera d'analyser et de développer le module « comparaison de sorties » qui fait la comparaison de la sortie attendue (la valeur calculée par l'oracle) et la sortie réelle (la valeur calculée par l'implémentation du système).

Références

[1] Blackburn Mark R. and Busser Robert D. T-VEC a Tool for Developping Critical System. In *Compass'96 : Eleventh Annual Conference on Computer Assurance*, Gaithersburg, Maryland. National Institute of Standard and Technology.

[2] Blackburn Mark R., Busser Robert D. and Fontaine Joseph S.. Automatic Generation of Test Vectors for SCR-Style Specifications. *Proceeding of the 12th Annual Conference on Computer Assurance*, Gaithersburg, Maryland, pg. 5467, juin 1997.

[3] Brace, K. S., Rudell, R. L., and Bryant, R.E. Efficient Implementation of BDD Package. In *Proceedings of the 27th Design Automation Conference*, Orlando FL, juin 1990, pg. 40 -45.

[4] Cheng Kwang-Ting and Krishnakumar A. S. Automatic Generation of Functional Vectors Using the Extended Finite State Machine Model. *Proceedings of the 30th International Conference on Design Automation*, pg. 86-91, 1993.

[5] V. Chvatal. A Greedy Heuristique for the Set-Covering Probleme. *Mathematics of Operations Research* 4(3), Août 1979

[6] Fujiwara, S., Bochmann, G., Khendek, F., Amalou, M., Ghedamsi, A., Test Selection Based on Finite State Models. *IEEE Trans. On Software Eng.* 17(6), juin 1991.

[7] Gargantini, Angelo and Heitmeyer, Constance, Using Model Checking to Generate Test from Requirements Specifications. *7th European software engineering conference*, octobre 1999.

[8] M.J Harrold, R. Gupta et M.L Soffa, A Methodology for Controlling the Size of a Test Suite. *ACM Transactions on Software Engineering and Methodology* 2(3) :270-285, juillet 1993

[9] Heitmeyer, Constance L., Jeffords, Ralph D., Labaw, Bruce G., Automated Consistency Checking of Requirements Specifications. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Volume 5, pg. 231-261, 1996.

[10] Heitmeyer, Constance L., Jeffords Ralph D., Labaw, Bruce G., Tools for Analyzing SCR-style Requirements Specification : A formal foundation. Tech. Rep., *Naval Research Laboratory*.

[11] Parnas, David L. and Madey, Jan. Functional documentation for computer systems. *Science of Computer Programming*, 25(1) :41-61, octobre 1995.

[12] Tallam Sriraman, Gupta Neelam, A Concept Analysis Greedy Algorithm for Test Suite Minimization, septembre 2005.