Associer des techniques de preuve et de résolution de contraintes pour la construction d'abstractions

Pierre-Christophe Bué*, Jacques Julliand*, Pierre-Alain Masson*, Fabrice Bouquet*

*LIFC, Université de Franche-Comté, 16 Route de Gray F-25030 Besançon Cedex
{bouquet, bue, julliand, masson}@lifc.univ-fcomte.fr,
http://lifc.univ-fcomte.fr

Résumé. Nous présentons un algorithme de génération d'abstractions de modèles B à partir d'un ensemble de prédicats d'abstraction. Nous donnons des éléments de preuve de cet algorithme établis en utilisant B4free sur une modélisation B. Nous évaluons expérimentalement sur quatre exemples, trois mises en œuvre de cet algorithme, l'une par preuve en utilisant GeneSyst basé sur le prouveur de l'atelier B, l'autre par résolution de contraintes en utilisant le solveur CLPS-B, et l'association des deux afin d'obtenir des abstractions plus précises. Enfin, nous présentons l'application de la méthode de calcul d'abstraction pour la génération de tests. Les contributions de ce papier consistent à décliner un algorithme de calcul d'abstraction à partir de prédicats sur des modèles B et à montrer sa correction. Nous montrons également qu'il est implantable avec toute technologie fournissant deux primitives : un calcul de plus faible précondition et une procédure SAT. Nous présentons une solution d'utilisation adaptée aux limites, aux avantages et aux inconvénients des techniques de preuve et de résolution de contraintes, en associant les deux technologies.

Mots-clés : Abstraction, Résolution de contraintes, Preuve, Génération de tests à partir de modèles

1 Motivations

Les techniques formelles appliquées au développement de programmes permettent de fiabiliser ces derniers. Ces techniques reposent en général sur l'utilisation d'un modèle formel du système à implanter. On peut par exemple procéder à la vérification par model-checking de ce modèle avant son implantation.

Dans une approche de test à partir de modèles (*Model-Based Testing*) Beizer (1995); Broy et al. (2005); Utting et Legeard (2006), un modèle est rédigé séparément d'une implantation, à partir du document initial de spécification. Des tests sont calculés à partir du modèle, en utilisant des critères de couverture de celui-ci. En confrontant l'implantation aux tests issus du modèle, on établit un verdict de conformité de l'implantation par rapport au modèle.

Mais la mise en œuvre de ces techniques de vérification ou de validation reste difficile pour des systèmes de taille industrielle, en raison de la très grande taille de leur espace d'états. On peut alors calculer et utiliser des abstractions de ces modèles pour faire face à ce problème.

Nous considérons dans ce papier des modèles formels rédigés en utilisant la notation logico-ensembliste B Abrial (1996a), sous la forme de systèmes d'événements B Abrial (1996b). Des outils tels que l'Atelier B ou B4free sont disponibles pour cette notation afin de réaliser des preuves de correction des algorithmes modélisés. Nous calculons des abstractions de ces modèles B, sous la forme de systèmes de transitions symboliques. Notre intention est de réduire la taille du modèle à partir duquel sont générés les tests. L'objectif est de maîtriser l'explosion combinatoire du nombre d'états, de transitions ou de chemins qui sont les critères de couverture habituels utilisés sur ce type de modèle.

Nous présentons un algorithme de calcul d'abstraction de modèles B qui prend en entrée, outre le modèle à abstraire, un ensemble de prédicats d'abstraction Graf et Saïdi (1997); Ball (2005). L'algorithme calcule les états symboliques atteignables et évalue des conditions visant à établir l'existence de transitions entre ces états.

Cet algorithme a déjà été mis en œuvre par l'outil *GeneSyst* Bert et al. (2005); Stouls (2007). Nous le présentons ici en montrant qu'il repose sur l'utilisation de deux fonctions primitives : un calcul de plus faible précondition et une évaluation de satisfiabilité (par un solveur SAT). De ce fait, l'algorithme est applicable à tout langage de spécification pour lequel est défini le calcul de plus faible précondition. En B, celui-ci est défini par le calcul des substitutions. Nous avons modélisé l'algorithme lui-même en B, ce qui nous a permis d'en établir la correction grâce à l'environnement de preuve associé au langage. Nous décrivons dans ce papier quelques éléments de preuve de l'algorithme réalisés avec B4free.

Les abstractions calculées par l'algorithme sont des sur-approximations du modèle source, c'est à dire qu'elles peuvent définir plus (mais pas moins) de chemins d'exécution que n'en définit le modèle concret. Nous présentons deux techniques permettant d'implanter cet algorithme, par preuve et par résolution de contraintes, et nous en décrivons trois mises en œuvre que nous comparons de manière expérimentale.

La première mise en œuvre, par preuve uniquement, est celle réalisée par l'outil *GeneSyst*. Elle utilise le calcul de plus faible précondition et les techniques de preuve de l'atelier B pour évaluer la satisfiabilité. L'existence de transitions entre les états symboliques est démontrée au moyen d'un ensemble d'obligations de preuve que l'atelier B tente de résoudre automatiquement. Nous avons développé la seconde en utilisant les techniques à contraintes de CLPS-B Bouquet et al. (2004). L'existence des transitions se démontre par la résolution d'un ensemble de contraintes représentant le problème de satisfiabilité. Le calcul de plus faible précondition est effectué par résolution de contraintes en utilisant le prédicat avant-après d'un événement. La troisième mise en œuvre consiste à associer ces deux techniques de preuve et de résolution de contraintes pour améliorer la précision de l'abstraction, c'est à dire pour en réduire la sur-approximation. En effet, chaque technique est incomplète dans le sens où les preuves et les systèmes de contraintes peuvent échouer à être résolus de manière automatique. Quand c'est possible, nous résolvons par contraintes l'existence des transitions lorsque celle-ci n'a pu être établie par preuve automatique.

Nous appliquons notre calcul d'abstraction à un processus de génération de tests à partir de modèles. Les sur-approximations que nous calculons sont bien adaptées pour vérifier des propriétés de sûreté car, si celles-ci sont vérifiées par l'abstraction, elles le sont par le système concret. Pour l'application à la génération de tests, il serait préférable d'utiliser des sous-approximations de telle sorte que tous les tests abstraits symboliques soient instanciables dans le modèle concret. Cette approche est proposée dans Ball et al. (2005) pour l'abstrac-

tion de programmes C. Mais, nous avons constaté sur les exemples traités que l'application de cette approche à des modèles conduisait à des sous-approximations trop imprécises ne permettant pas d'engendrer un nombre suffisant de tests. C'est pourquoi nous proposons d'utiliser une sur-approximation. Mais, avec cette approche, les tests générés extraits de l'abstraction, doivent être instanciés sur le modèle concret. Cette fois, les inconvénients sont le coût et le risque d'échec de l'instanciation. Dans cet article, nous mesurons les temps et les taux d'instanciation avec ou sans association des deux techniques d'implantation.

Les notions formelles de système d'événements B et de système de transitions symbolique sur lesquelles se fondent ce travail sont présentées dans la section 2. Nous décrivons pour illustrer notre propos l'exemple d'un système d'alimentation électrique en section 3. L'algorithme de calcul d'abstractions est présenté dans la section 4, accompagné d'éléments de preuve de sa correction. Nous décrivons dans la section 5 un processus de génération de tests à partir de modèles, auquel s'applique ce calcul d'abstraction. Les trois mises en œuvre de l'algorithme, ainsi que les résultats expérimentaux de leur utilisation (de manière séparée, puis conjointe) sont présentés dans la section 6. Enfin, nous concluons notre travail en section 7.

2 Préliminaires

L'algorithme d'abstraction que nous présentons dans cet article est basé sur deux fonctions primitives : le calcul de *plus faible précondition* d'une action a pour atteindre un état défini par un prédicat P, dénotée $wp(a, P)$, et l'évaluation de la *satisfiabilité d'un prédicat* P dénotée $SAT(P)$. Les différentes implantations utilisent deux mises en œuvre différentes de ces fonctions primitives, l'une par preuve, l'autre par résolution de contraintes. L'abstraction est effectuée à partir d'un ensemble fini de prédicats et aboutit à un modèle à nombre d'états fini.

Nous utilisons la notation B Abrial (1996a) pour décrire les modèles formels sources. Notons que les travaux présentés sont facilement généralisables à n'importe quelle modélisation par des systèmes pré-post tels qu'ils sont définis dans Namjoshi et Kurshan (2000) par exemple. Pour appliquer la méthode à la génération de tests, les modèles traités sont des modélisations fermées de systèmes afin de pouvoir les animer automatiquement, ce sont donc des systèmes d'événements B Abrial (1996b) (voir la définition 1). L'abstraction est définie par un système de transitions étiquetées. Ce choix nous permet de baser nos résultats sur la théorie logico-ensembliste B. En particulier, la notion de plus faible précondition est définie par le calcul des substitutions généralisées B. L'évaluation de $SAT(P)$ peut être effectuée par résolution d'une obligation de preuve existentielle $\exists X.P$. Expérimentalement, nous avons utilisé GeneSyst Bert et al. (2005) qui est basé sur le prouveur de l'atelier B. Nous aurions pu utiliser un SMT-solver comme Z3 de Moura et Bjørner (2008). Dans le cas de systèmes à nombre fini d'états, elle peut également être évaluée par résolution de contraintes Bouquet et al. (2004) en mettant P sous la forme d'un ensemble de contraintes.

Dans la suite du papier, on utilise les notations suivantes : x, y, z sont des variables, X, Y, Z sont des ensembles de variables et E et F sont des expressions B. $Pred_X$ est l'ensemble des prédicats B exprimables sur l'ensemble de variables X. I ($\in Pred_X$) est un invariant et P, P_1 et P_2 dénotent d'autres prédicats. Les modifications des variables sont appelées *substitutions* en B. Nous notons S, S_1, S_2 et S_i les substitutions généralisées B. Le terme de *substitution* fait référence à la règle de la logique de Hoare Hoare (1969) définissant par substitution la plus faible précondition d'une affectation $x := E$ pour qu'elle aboutisse au prédicat P par

$[x := E]P$ où $[x := E]P$ dénote la substitution de toutes les occurrences libres de x dans P par E. En B, cette règle est généralisée à toutes les structures de contrôle comme nous le définissons pour les plus courantes d'entre elles ci-dessous.

Étant donné une substitution S et une post condition P, il est possible de calculer la plus faible précondition ainsi :

IF P_1 THEN S_1 ELSE S_2 END$]P$	$\widehat{=}$	$(P_1 \Rightarrow [S_1]P) \wedge (\neg P_1 \Rightarrow [S_2]P)$
[SELECT P_1 THEN S END$]P$	$\widehat{=}$	$(P_1 \Rightarrow [S]P)$
[CHOICE S_1 OR S_2 END$]P$	$\widehat{=}$	$[S_1]P \wedge [S_2]P$
[ANY z WHERE P_1 THEN S END$]P$	$\widehat{=}$	$\forall z.(P_1 \Rightarrow [S]P)$ si z n'est pas libre dans P

Définition 1 (Système d'événements B correct) *Etant donné un ensemble fini de noms d'événements NE, un système d'événements B est un quadruplet $\langle X, I, Init, Ev\rangle$ où :*

- *X est un ensemble fini de variables d'états,*
- *I ($\in Pred_X$) est un prédicat* invariant *sur les variables de X,*
- *Init est une substitution d'*initialisation*,*
- *Ev est un ensemble de définitions de chaque événement e de NE par une équation $e \widehat{=} S$.*

Ce système d'événements est correct si :

- *l'initialisation établit l'invariant : $wp(Init, I) \widehat{=} [Init]I$ est un prédicat valide,*
- *chaque événement maintient l'invariant : $I \Rightarrow wp(S, I) \widehat{=} I \Rightarrow [S]I$ est un prédicat valide.*

L'abstraction d'un système d'événements est un système de transitions symbolique (STS). Celui-ci est calculé à partir d'un ensemble de n prédicats élémentaires d'abstraction $PA = \{p_1, p_2, ..., p_n\}$ calculé à partir du modèle à abstraire. Il est par exemple constitué de tous les prédicats élémentaires portant sur les variables d'états collectés dans chacun des événements du modèle. Dans l'exemple Fig. 2, l'ensemble de prédicats d'abstraction PA est le suivant : $\{H = tic, \mathrm{card}(Bat \rhd \{ok\}) > 1\}$. Notons que le prédicat $H = tac$ n'est pas dans PA car c'est la négation du prédicat $H = tic$ étant donné que H est une variable à deux états. L'ensemble d'états Q est alors défini comme le sous-ensemble des états atteignables du produit cartésien de $\{p_1, \neg p_1\} \times \{p_2, \neg p_2\} \times ... \times \{p_n, \neg p_n\}$. Un état symbolique est un élément de ce produit cartésien constitué d'un n-uplet de prédicats élémentaires. Un STS est formalisé dans Def. 2. Un état q est défini par le n-uplet des prédicats élémentaires d'abstraction satisfaits dans cet état. Notons que le nombre d'états de l'abstraction est borné par 2^n.

Définition 2 (Système de Transitions Symbolique (STS)) *Etant donné un ensemble fini de noms d'événements NE, un ensemble de n prédicats élémentaires PA $= \{p_1, p_2, ..., p_n\}$ définissant un ensemble de 2^n états abstraits $A = \{p_1, \neg p_1\} \times \{p_2, \neg p_2\} \times ... \times \{p_n, \neg p_n\}$, un STS est défini par un triplet $\langle Q, Q_0, \Delta\rangle$ainsi :*

- *$Q \subseteq A$ est un ensemble fini d'états,*
- *$Q_0 (\subseteq Q)$ est un ensemble d'états initiaux,*
- *$\Delta (\in Q \times NE \times Q)$ est une relation de transitions étiquetées.*

Pour simplifier les notations, nous notons $q \xrightarrow{e} q'$ le prédicat $(q, e, q') \in \Delta$. Un fragment d'exécution de longueur n d'un STS est une séquence de couples $(e_j, q_j), (e_{j+1}, q_{j+1}), \ldots, (e_n, q_n)$ telle que $q_i \xrightarrow{e_{i+1}} q_{i+1}$ pour i appartenant à l'intervalle $j..n-1$. Nous notons $q \xrightarrow{*} q'$ quand il existe un fragment d'exécution de longueur $n \geq 0$ entre q et q'.

Par exemple, la Fig. 5 représente le STS qui abstrait le système modélisé dans la Fig. 2 à partir de l'ensemble de prédicats d'abstraction $PA \widehat{=} \{H = tic, \mathrm{card}(Bat \rhd \{ok\}) > 1\}$.

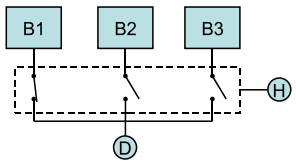

FIG. 1 – *Schéma du Système Electrique*

3 Exemple du système d'alimentation électrique

Nous présentons un exemple de système qui sert à illustrer nos propos tout au long de l'article. Un appareil D est alimenté par l'une des trois batteries B_1, B_2, B_3 comme le montre Fig. 1. Un interrupteur connecte (ou non) une batterie B_i à l'appareil D. Une horloge H envoie périodiquement un signal de commutation d'interrupteur, c'est à dire un ordre de changement de la batterie alimentant D. Le fonctionnement du système doit satisfaire les trois exigences suivantes :

- Req_1 : il n'y a pas de court-circuit, autrement dit, un seul interrupteur est fermé,
- Req_2 : l'appareil est constamment alimenté, autrement dit, il y a toujours un interrupteur fermé,
- Req_3 : lorsque l'horloge envoie une commande de commutation, l'interrupteur fermé est modifié.

Mais les batteries peuvent tomber en panne. Quand la batterie qui alimente D tombe en panne, le système effectue une commutation exceptionnelle pour satisfaire l'exigence Req_2. Les batteries en panne sont remplacées par un service de maintenance. Nous supposons que ce service travaille suffisamment rapidement pour qu'il n'y ait jamais trois batteries en panne simultanément. Quand deux batteries sont en panne, l'exigence Req_3 est relaxée et les ordres de commutation de l'horloge ne sont plus pris en compte.

$$
\begin{array}{lcl}
X & \widehat{=} & \{H,\ Sw,\ Bat\} \\
I & \widehat{=} & H \in \{tic,\ tac\} \land Sw \in 1..3 \land (Bat \in 1..3 \rightarrow \{ok,\ ko\}) \land Bat(Sw) = ok \\
Init & \widehat{=} & H,\ Sw,\ Bat := tac, 1,\ \{1 \mapsto ok,\ 2 \mapsto ok,\ 3 \mapsto ok\} \\
Tic & \widehat{=} & \textsf{SELECT } H = tac \textsf{ THEN } H := tic \textsf{ END} \\
Com & \widehat{=} & \textsf{SELECT } H = tic \textsf{ THEN} \\
 & & \quad \textsf{ANY } ns \textsf{ WHERE } ns \in 1..3 \land Bat(ns) = ok \land ns \neq Sw \\
 & & \quad \textsf{THEN } H, Sw := tac, ns \textsf{ END} \\
 & & \textsf{END} \\
Fail & \widehat{=} & \textsf{ANY } nb \textsf{ WHERE } nb \in 1..3 \land nb \neq Sw \land Bat(nb) = ok \\
 & & \textsf{THEN} \\
 & & \quad \textsf{CHOICE } Sw, Bat(Sw) := nb, ko \\
 & & \quad \textsf{OR } Bat(nb) := ko \textsf{ END} \\
 & & \textsf{END} \\
Rep & \widehat{=} & \textsf{ANY } nb \textsf{ WHERE } nb \in 1..3 \land nb \in \mathrm{dom}(Bat \rhd \{ko\}) \textsf{ THEN } Bat(nb) := ok \textsf{ END}
\end{array}
$$

FIG. 2 – *Spécification B du système électrique*

Le système est modélisé dans Fig. 2 au moyen de trois variables. H modélise l'horloge et prend deux valeurs : tic pour demander une commutation et tac quand la commutation a été prise en compte. Sw modélise l'état des trois interrupteurs par un entier dans l'intervalle $1..3$: $Sw = i$ indique que l'interrupteur i est fermé alors que les deux autres sont ouverts. Ainsi, ce modèle satisfait l'exigence Req_1. Bat modélise les pannes de batteries par une fonction

totale. La valeur ko indique que la batterie est en panne. En plus du typage des variables, le prédicat $Bat(Sw) = ok$ de l'invariant exprime que la batterie connectée par l'interrupteur fermé est en fonctionnement. Cette contrainte permet de respecter l'hypothèse sur les pannes et l'exigence Req_2. Notons que l'exigence Req_3 est une propriété dynamique qui n'est pas formalisée dans I. L'état initial est défini par $Init$ dans Fig. 2. Les changements d'états du système sont décrits par quatre événements :

- **Tic** envoie un ordre de commutation,
- **Com**[1] effectue une commutation en changeant d'interrupteur fermé,
- **Fail** modélise une panne électrique de l'une des batteries,
- **Rep** modélise une intervention du système de maintenance en remplaçant une batterie en panne par une en état de marche.

4 Algorithme de calcul d'abstractions

Nous nous plaçons dans le contexte du calcul d'une abstraction AM d'un modèle formel M à partir d'un ensemble de prédicats PA. Cette méthode est connue sous l'appelation d'*abstraction à partir de prédicats* (Predicate abstraction) Graf et Saïdi (1997); Ball (2005); Namjoshi et Kurshan (2000). Dans Bouquet et al. (2009), nous avons défini une méthode pour extraire un ensemble de prédicats d'abstraction PA à partir d'un objectif de test dans le contexte d'un processus de génération de tests à partir de modèles. Dans Ball (2005), l'ensemble PA est formé de tous les prédicats élémentaires des structures de contrôle d'un programme C. Dans Namjoshi et Kurshan (2000), cet ensemble de prédicats est raffiné itérativement afin de calculer une bisimulation quand celle-ci existe. Nous définissons l'algorithme et sa spécification, puis nous donnons des éléments de preuve.

4.1 Spécification et Algorithme

On suppose disposer d'un ensemble PA de n prédicats élémentaires d'abstraction. Cet ensemble permet de définir un ensemble d'états abstraits A comme dans Def. 2. Un état abstrait q est un n-uplet de prédicats élémentaires $(p_1, \ldots, p_n)$. Nous notons également q le prédicat définissant cet état abstrait, $q = \bigwedge_{i=1}^{n} p_i$. Une valuation qui satisfait q est un état concret. Par abus de langage, nous dirons qu'il appartient à q. Le prédicat $q \wedge I$ définit le sous-ensemble des états concrets de q qui satisfont l'invariant I. La spécification et l'algorithme présentés dans cette section reposent sur deux fonctions primitives wp et SAT. Pour simplifier nous notons $wp(S, q')$ le prédicat $[S](q' \wedge I) \wedge I$ qui définit le plus grand ensemble d'états satisfaisant I et permettant d'atteindre l'un des états de q' satisfaisant I par l'action S. Nous notons $SAT(wp(S, q') \wedge q)$ le prédicat qui est vrai s'il existe au moins un état concret de q et de q' qui sont en relation par l'action $e \mathrel{\widehat{\in}} S$. Autrement dit, il existe une transition $q \xrightarrow{e} q'$.

La propriété (1) en Fig. 3 signifie que tout état abstrait q est un état initial si il existe au moins un état concret de q qui est atteint par l'initilisation. La propriété (2) définit Δ et Q en indiquant qu'une transition $q \xrightarrow{e} q'$ appartient à Δ si il existe des états concrets de q et q' pour lesquels la transition existe et si q est atteignable depuis un état initial.

[1] Une expression $r \rhd E$ denote une relation dont le co-domaine est restreint par l'ensemble E. par exemple, $\{1 \mapsto ok,\ 2 \mapsto ko,\ 3 \mapsto ok\} \rhd \{ok\} = \{1 \mapsto ok,\ 3 \mapsto ok\}$.

Données A : ensemble d'états abstraits
$M \widehat{=} \langle X, I, Init, Ev \rangle$
système d'événements correct tel que pour tout $e \in NE$
il y a une définition $e \widehat{=} S$ dans Ev
Résultats $AM \widehat{=} \langle Q, Q_0, \Delta \rangle$: Système de transitions symbolique
Post condition

$$\forall \mathsf{q}.(\mathsf{q} \in \mathsf{A} \wedge \mathsf{SAT}(\mathsf{wp}(\mathsf{Init}, \mathsf{q})) \Rightarrow \mathsf{q} \in \mathsf{Q}_0) \quad (1)$$

$$\forall (\mathsf{q}, \mathsf{q}', \mathsf{e}).(\mathsf{q} \in \mathsf{A} \wedge \mathsf{q}' \in \mathsf{A} \wedge \mathsf{e} \in \mathsf{NE} \wedge \mathsf{SAT}(\mathsf{wp}(\mathsf{S}, \mathsf{q}') \wedge \mathsf{q}) \wedge \exists q_0.(q_0 \in Q_0 \wedge q_0 \xrightarrow{*} q) \Rightarrow q \xrightarrow{e} q' \wedge q \in Q \wedge q' \in Q) \quad (2)$$

FIG. 3 – *Spécification du calcul d'abstraction*

Variables QR, QS : ensembles d'états ;
q, q' : état ; e : nom d'événement
Début
/* calcul des états initiaux */
(1) $QR := A$; /* QR : ens. d'états à examiner */
(2) **Tantque** $QR \neq \varnothing$ **faire**
(3) Choisir q dans QR ; $QR := QR - \{q\}$;
(4) **Si** $SAT(wp(Init, q))$ **alors** $Q_0 := Q_0 \cup \{q\}$ **finsi**
(5) **fait ;**
/* calcul des états Q et des transitions Δ */
(6) $\Delta := \varnothing$; $Q := \varnothing$; $QS := Q_0$;
(7) **Tantque** $QS \neq \varnothing$ **faire**
(8) Choisir q dans QS ; $QS := QS - \{q\}$; $Q := Q \cup \{q\}$
(9) **PourTout** $q' \in A$ **faire**
(10) **PourTout** $e \in NE$ **faire**
(11) **Si** $SAT(wp(S, q') \wedge q)$ **alors**
(12) $\Delta := \Delta \cup \{q \xrightarrow{e} q'\}$;
(13) **Si** $q' \notin Q$ **alors** $QS := QS \cup \{q'\}$ **finsi**
(14) **finsi**
(15) **fait**
(16) **fait**
(17) **fait**
fin

FIG. 4 – *Algorithme de calcul d'une abstraction*

L'algorithme de calcul de l'abstraction décrit dans Fig. 4 procède en deux étapes, le calcul des états initiaux Q_0 (lignes 1-5) et le calcul de l'ensemble d'états atteignables Q et de transitions Δ (lignes 6-17). Pour calculer l'ensemble d'états atteignables Q, il utilise la variable QS qui contient les états sources dont il faudra calculer les successeurs, soit parce qu'ils sont initiaux, soit parce qu'ils sont cibles d'une transition de Δ. Pour calculer les successeurs d'un état source q (lignes 8-16), on enumère tous les états cibles q' (ligne 9) possibles et tous les événements $e \widehat{=} S$ (ligne 10) qui pourraient conduire à q'. La transition $q \xrightarrow{e} q'$ est ajoutée à Δ si la condition $SAT(wp(S, q') \wedge q)$ est vraie (lignes 11-12), c'est à dire si il existe au moins un état concret de q' atteignable à partir d'un état concret de q par l'action S (ligne 11). Si l'état cible q' n'est pas encore un état atteignable ($q' \notin Q$), il est ajouté aux états sources dont il faudra calculer les successeurs (ligne 13).

4.2 Eléments de preuve

Dans cette section, nous donnons quelques éléments de preuve de terminaison et des propriétés de correction (1) et (2).

L'algorithme termine car A et NE sont des ensembles finis. Ceci garantit la terminaison de l'itération de calcul des états initiaux et des deux itérations internes du calcul des transitions. La finitude de A garantit également la terminaison de l'itération externe (ligne 7) puisqu'un état q' n'est ajouté dans QS que s'il n'appartient pas à Q. Donc, dans le pire des cas, l'algorithme termine avec $Q = A$.

Il est évident que la partie de calcul des états initiaux satisfait la propriété (1). Elle a été prouvée avec B4free[2] en modélisant l'algorithme décrit Fig. 4. La modélisation fait abstraction des deux fonctions primitives SAT et wp en les modélisant par deux fonctions constantes :

- $SAT\_Init \in A \rightarrow BOOL$; $SAT\_Init(q)$ est vrai si et seulement si $SAT(wp(Init, q))$ est vrai,
- $SAT\_S \in A \times NE \times A \rightarrow BOOL$; $SAT\_S(q, e, q')$ est vrai si et seulement si $SAT(wp(S, q') \wedge q)$ est vrai, e étant défini par la substitution S dans Ev.

La preuve a été effectuée en introduisant l'invariant I_1 suivant pour cette itération : $\forall q.(q \in A - QR \wedge SAT\_Init(q) \Rightarrow q \in Q_0)$. L'itération termine avec $QR = \varnothing$ et alors $I_1 \wedge QR = \varnothing \Rightarrow (1)$ est valide.

L'algorithme satisfait également la propriété (2). En effet, QS ne contient que des états atteignables, au début les états initiaux, puis au fur et à mesure du calcul les états atteignables depuis les états de QS. Δ contient toutes les transitions déclenchables à partir des états atteignables. Ceci a été prouvé interactivement sur la modélisation B dans laquelle on a introduit les invariants de l'itération externe suivants :

$$Q \cap QS = \varnothing \wedge Q_0 \subseteq (Q \cup QS) \tag{3}$$

$$\begin{aligned} &\forall (q, q', e).(q \in Q \wedge q' \in A \wedge e \in NE \wedge SAT\_S(q, e, q') \wedge \\ &(q \in Q_0 \vee \exists (qc, ec).(qc \in Q \wedge ec \in NE \wedge qc \xrightarrow{ec} q)) \Rightarrow q \xrightarrow{e} q' \wedge (q' \in Q \vee q' \in QS)) \end{aligned} \tag{4}$$

$$\forall (q, q', e).(q \in Q \wedge q' \in A - (QS \cup Q) \wedge e \in NE \Rightarrow \neg SAT\_S(q, e, q')) \tag{5}$$

La propriété (3) (Fig. 3) signifie qu'un état atteignable n'est pas parmi les états dont on doit calculer les successeurs et que tout état initial est soit atteignable, soit à introduire dans les états atteignables après avoir calculé ses successeurs. La propriété (4) spécifie la correction du calcul de l'ensemble d'états atteignables Q. Toutes les transitions déclenchables à partir des états atteignables de Q sont calculées. La propriété (5) spécifie la complétude du calcul de l'ensemble d'états atteignables Q. Il n'existe pas de transitions d'un état atteignable vers des états abstraits de A qui n'est pas soit atteignable, soit à introduire dans les états atteignables. Quand l'algorithme termine avec $QS = \varnothing$, la propriété (5) signifie qu'il n'y a pas d'état q' de $A - Q$ qui soit atteignable à partir d'un état de Q. Des trois propriétés $Q_0 \subseteq Q$, (4) et (5), on déduit que la propriété (2) est valide.

B 4free engendre 473 obligations de preuve dont 145 sont à résoudre interactivement. 143 obligations ont été résolues avec l'outil essentiellement en sélectionnant les hypothèses utiles, en les instanciant et en faisant prouver quelques lemmes intermédiaires. Deux obligations qui vérifient l'invariant de la boucle externe dans le cas de sortie de la boucle de la ligne 9 dans la Fig. 4 n'ont pas été prouvées avec l'outil, mais ont été vérifiées manuellement.

L'application de cet algorithme sur le système électrique Fig. 2 aboutit au STS Fig. 5.

[2] http ://www.b4free.com/

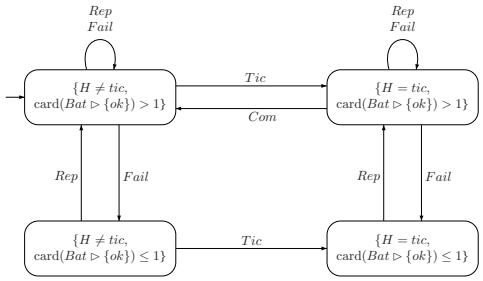

FIG. 5 – *Système Electrique Abstrait*

5 Application du calcul d'abstractions à la génération de tests

Notre intention est d'utiliser les abstractions produites selon l'algorithme décrit dans la section 4 dans un processus de génération de tests. Nous avons décrit un tel processus dans Bouquet et al. (2010), et nous en résumons ici les grandes lignes. Les tests sont extraits de l'abstraction AM selon un critère de sélection dynamique, que décrit un objectif de test OT. Nous avons défini dans Julliand et al. (2008) un langage basé sur les expressions régulières pour exprimer des objectifs de test. Sur l'exemple, l'objectif de test est d'observer les comportements du système lorsque les batteries tombent en panne. On veut notamment observer le comportement lorsque la batterie qui alimente le système tombe en panne et lorsqu'il n'y a plus qu'une seule batterie en fonctionnement. L'ensemble de prédicats d'abstraction est constitué de $H = tic$ qui déclenche une commutation et $card(Bat \rhd \{ok\}) \leq 1$ qui spécifie le cas où il n'y a plus qu'une seule batterie en fonctionnement.

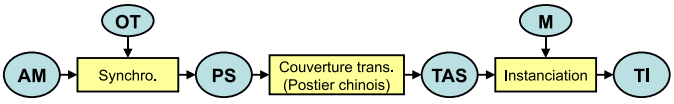

FIG. 6 – *Génération de tests à partir d'une abstraction et d'un objectif de test*

5.1 Processus de génération de tests

Ce processus est présenté en Fig. 6. L'abstraction AM est un système de transitions (symbolique) et la sémantique de l'objectif de test OT est un automate. On calcule le produit synchronisé PS entre AM et OT. C'est un système de transitions (symbolique) dont les traces sont les exécutions de AM qui sont conformes à OT. On assure la couverture des transitions de PS au moyen d'une implantation Thimbleby (2003) de l'algorithme du postier chinois. Cela produit TAS, un ensemble de tests abstraits symboliques. Ces tests sont ensuite instanciés sur le modèle concret M en un ensemble TI de tests instanciés. Ces tests ont alors le même niveau d'abstraction que ceux produits, par exemple par des critères de couverture structurels, directement à partir de M. Ils peuvent bénéficier pour être exécutés du même environnement de concrétisation et d'exécution que celui mis en place pour exécuter les tests structurels Bouquet et al. (2010).

L'abstraction AM en entrée du processus décrit Fig. 6 est calculée à partir du modèle concret M et d'un ensemble PA de prédicats d'abstractions portant sur les variables de M, en appliquant l'algorithme de la section 4. L'abstraction calculée par cet algorithme est une sur-approximation de M : certains fragments d'exécution de AM peuvent ne pas être des fragments d'exécution de M. Considérons en effet un fragment $(e, q), (e', q'), (e'', q'')$ d'exécution de AM. L'état concret de q' atteint depuis un état concret de q ne permet pas forcément d'atteindre un état concret de q''.

Nous disposons de deux implantations de cet algorithme. La première utilise des techniques de preuve et est fournie par l'outil *GeneSyst* Bert et al. (2005); Stouls (2007). Nous avons développé la seconde en utilisant l'outil CLPS-B Bouquet et al. (2004) de résolution de contraintes. Ces deux techniques sont incomplètes, et peuvent produire des abstractions moins précises que l'abstraction théorique.

En effet, *GeneSyst* produit des obligations de preuve de l'existence d'une transition entre deux états symboliques. Plus précisément, pour chaque paire d'états symboliques q et q' et pour chaque action $e = S$, *GeneSyst* produit des obligations de preuve pour garantir $SAT(wp(S, q') \wedge q)$, et tente de les résoudre automatiquement. Les transitions dont l'existence est prouvée sont ajoutées à Δ. Les transitions non prouvées, c'est à dire dont l'absence ou l'existence n'a pas été prouvée automatiquement, sont elles aussi ajoutées à Δ. Elles sont donc conservées dans l'abstraction produite.

La technique utilisant CLPS-B est elle aussi incomplète dans le sens où un *timeout* (i.e. un dépassement de délai) est susceptible d'interrompre la résolution des contraintes visant à établir $SAT(wp(S, q') \wedge q)$, lorsque celle-ci prend trop de temps. Là aussi les transitions dont l'absence ou l'existence n'a pas été résolue sont conservées dans l'abstraction produite.

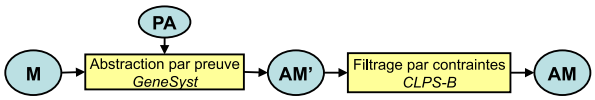

FIG. 7 – *Association des techniques de preuve et de contraintes pour le calcul d'abstractions*

Nous proposons d'associer les deux techniques afin de réduire les imprécisions liées à l'incomplétude des outils d'abstraction. Notre processus est en deux temps comme indiqué Fig. 7. Une première abstraction AM' est calculée par preuve avec *GeneSyst*. Les transitions non prouvées de AM' sont collectées, et CLPS-B tente de les résoudre par contraintes. L'abstraction AM ainsi produite ne contient alors comme transitions non prouvées que celles qui n'ont pu ni être prouvées, ni être résolues par contraintes.

5.2 Instanciation des tests

Les tests abstraits symboliques issus de AM sont ensuite instanciés sur le modèle concret M (voir Fig. 6). Comme AM est une sur-approximation de M, certains tests issus de AM peuvent ne pas être instanciables sur M. Nous effectuons une animation symbolique des tests sur le modèle concret M. Lorsqu'une action e de changement d'état du test abstrait n'est pas déclenchable sur M, nous intercalons l'exécution d'une ou plusieurs transitions réflexives de PS jusqu'à ce que e devienne déclenchable ou que l'animation aboutisse à un blocage. Les transitions réflexives peuvent conduire à une autre instance du même état symbolique à partir de

laquelle la transition devient déclenchable. Mais, cette méthode est incomplète car il n'est pas suffisant d'essayer d'intercaler des sous-séquences de transitions réflexives, il faudrait également essayer tous les cycles de longueur supérieure à un. Cette méthode étant explosive, nous ne l'avons pas appliquée. Enfin, notons qu'en augmentant la précision de l'abstraction, on réduit le nombre de tests non instanciables. Les taux de réussite de l'instanciation des tests abstraits sont mesurés de manière expérimentale et discutés dans la section 6.

6 Implantation et résultats expérimentaux

Nous présentons dans cette section les trois implantations de l'algorithme de calcul d'abstraction présentés en Section 4. Nous montrons par des résultats expérimentaux que la génération d'abstraction par une technique de résolution de contraintes n'est pas applicable sur des exemples de grande taille. Nous montrons également que la génération d'abstraction par une technique de preuve est sujette à des défauts de preuve posant problème pour l'instanciation des tests. Pour pallier à ce problème, nous utilisons une technique de résolution de contraintes pour résoudre ces défauts de preuve quand c'est possible, et améliorer l'instanciation des tests.

Les expérimentations ont été menées sur quatre exemples : un système électrique présenté en Section 3, un annuaire téléphonique inversé appelé Qui-Donc, un système de transport de pièces appelé Robot, et DeMoney modélisant un porte-monnaie électronique. Pour chacun de ces exemples, deux objectifs de test, et donc deux ensembles de prédicats d'abstraction, ont été utilisés.

6.1 Implantations de l'algorithme de génération d'abstraction

Deux techniques d'implantations des primitives de l'algorithme de génération d'abstraction ont été utilisées pour nos expérimentations : preuve et résolution de contraintes.

L'outil *GeneSyst* Bert et al. (2005) s'appuie sur le calcul de plus faible précondition et sur le prouveur de l'atelier B pour réaliser les deux primitives wp et SAT. Le calcul de satisfiabilité revient à prouver qu'il existe une valuation de l'ensemble des variables permettant de satisfaire le prédicat considéré. La complexité de l'algorithme de preuve ne dépend pas de la taille des domaines des variables intervenant dans l'obligation de preuve.

Une autre implantation de cet algorithme a été réalisée en s'appuyant sur le solveur de contraintes CLPS-B Bouquet et al. (2004). Celui-ci est un solveur de contraintes logico-ensemblistes prenant en compte les entiers, la quasi totalité des structures de données B (ensembles, fonctions et relations) et des opérateurs associés. Le problème de calcul de la plus faible précondition $wp(S, q')$ est exprimé par un ensemble de contraintes C obtenu par application du calcul des substitutions tel qu'il est défini dans la section 2. Le problème de satisfiabilité $SAT(wp(S, q') \wedge q)$ est exprimé par l'ensemble de contraintes $C \cup q$ où q est l'ensemble de prédicats définissant l'état abstrait source q. Le solveur recherche une solution satisfaisant toutes les contraintes. Cette recherche de solution s'effectue en plusieurs étapes : les contraintes sont propagées par des algorithmes de consistance afin de réduire les domaines des variables, puis une énumération est effectuée sur ces domaines. L'avantage de cette technique réside dans sa rapidité de calcul pour des variables dont les domaines sont petits. Mais pour les variables de grande taille, l'énumération de toutes les valeurs pour tester la satisfiabilité peut s'avérer

coûteuse en temps et en mémoire. Sur l'exemple DeMoney on peut voir dans le tableau 1 que la génération d'abstraction a échoué par manque de ressources mémoire.

Notons que quelque soit la technique d'implantation, si elle ne parvient pas à décider si une transition existe ou non dans l'abstraction, alors cette transition est conservée.

Modèle	♯Var. énum.	♯Var. entières	♯Cpt.	♯ États symboliques	Abstraction GeneSyst α			Abstraction CLPS-B β			Comparaison
					♯ États	♯ Trans.	Temps	♯ États	♯ Trans.	Temps	
Elec.	3	0	5	4	4	15	27 s.	4	15	3 s.	$\beta = \alpha$
				2	2	6	9 s.	2	6	1 s.	$\beta = \alpha$
Qui-Donc	3	0	20	6	5	24	2 min.	5	22	5 s.	$\beta \subset \alpha$
				6	5	18	1 min. 30 s.	5	16	5 s.	$\beta \subset \alpha$
Robot	6	0	10	6	6	37	5 min.	6	36	7 s.	$\beta \subseteq \alpha$
				8	8	50	8 min.	8	50	12 s.	$\beta = \alpha$
DeMoney	3	6	42	3	3	148	35 min.	Échec du calcul			
				9	7	228	1h. 30 min.				

TAB. 1 – *Mesures sur les modèles et génération d'abstractions*

Le tableau 1 montre la taille des modèles utilisés en terme de nombre de variables à domaine énuméré (i.e. de petite taille), à domaine entier (i.e. de grande taille) et de nombre de chemins dans les graphes de flot de contrôle des opérations (colonne ♯ Cpt.). Le nombre d'états symboliques est le nombre d'éléments du produit cartésien de l'ensemble de prédicats d'abstraction satisfaisant l'invariant. Pour la comparaison des générations d'abstraction, nous nous appuyons sur le nombre d'états et de transitions de l'abstraction générée, ainsi que sur le temps nécessaire pour le calcul. La colonne "Comparaison" indique la relation entre les ensembles de transitions des deux abstractions.

Nous pouvons constater qu'à partir d'un ensemble d'états symboliques identiques, les deux implantations du calcul d'abstraction génèrent des abstractions comparables. Pour le Qui-Donc et le premier cas du Robot, les transitions qui existent dans l'abstraction calculée par *GeneSyst* mais pas dans l'abstraction calculée par CLPS-B sont dues à des défauts de preuve, et sont éliminées par résolution de contraintes.

Pour l'ensemble des exemples ne comportant pas de variables de grande taille, le calcul d'abstraction pas résolution de contraintes s'avère plus rapide (de 15 à 40 fois) que la preuve. Mais dès qu'une variable ayant un grand domaine et peu contrainte est utilisée, cette constatation est fausse. Elle conduit même dans notre exemple DeMoney à l'échec du calcul de l'abstraction. Cet échec est dû à des problèmes de ressources mémoire liés à la recherche d'une solution par le solveur de contraintes, qui s'effectue par énumération des valeurs des variables.

L'algorithme de calcul d'abstraction ne peut donc pas être implanté par résolution de contraintes, mais cette technique peut être associée à une technique de preuve pour aider à la résolution des défauts de preuve, comme nous le présentons dans la section suivante.

6.2 Association des techniques pour la génération de tests

Le tableau 2 présente les résultats de la génération de tests à partir d'abstractions calculées par l'outil *GeneSyst*. Ces résultats comparent les tests générés par la méthode décrite en section 5 en utilisant d'une part directement l'abstraction et d'autre part l'abstraction où les transitions en défaut de preuve ont été filtrées par une technique de résolution de contraintes. Nous comparons les ratios des nombres de tests instanciés par rapport au nombre de tests générés dans les deux cas.

Nous pouvons constater que le filtrage des transitions de l'abstraction apporte un meilleur taux d'instanciation (40 % d'instanciation sans filtrage à 60 % d'instanciation avec filtrage

en moyenne) des tests générés. Cela s'explique par le fait que les transitions filtrées sont des transitions pour lesquelles la preuve a échouée, et qui ont été conservées par défaut. Le ratio d'instanciation des tests sur l'abstraction filtrée augmente pour deux raisons, d'une part le nombre de tests générés diminue et d'autre part le nombre de tests instanciés augmente. Ayant supprimé des transitions dans l'abstraction le nombre de chemins pour couvrir toutes les transitions est plus faible. Par ailleurs, des chemins non instanciables utilisant ces transitions ont été remplacés par des chemins instanciables utilisant un autre fragment de chemin pour réaliser la transition.

La méthode échoue dans le premier cas de l'application DeMoney car la première transition de tous les tests abstraits n'est pas déclenchable. Elle est due à un défaut de preuve et n'est pas filtrée par CLPS-B. Par contre, dans le second cas, on obtient finalement 17 tests instanciés sur 18 car le filtrage a éliminé la transition non déclenchable qui fait échouer toute instanciation des tests issus du modèle sans filtrage.

Modèle	Instanciation des tests sans filtrage		Instanciation des tests avec filtrage				
	♯Tests instan. / ♯tests symb.	Temps d'instanciation	♯Trans. en défaut	♯Trans. filtrés	Temps de filtrage	♯Tests instan. / ♯tests symb.	Temps d'instanciation
Elec.	8/17 (47%)	1 s.	15	4	$\leq$ 1 s.	9/12 (75%)	1 s.
	3/3 (100%)	1 s.	6	0		3/3 (100%)	
Qui-Donc	4/10 (40%)	1 s.	16	2		6/10 (60%)	
	5/15 (33%)	1 s.	12	2		6/11 (54%)	
Robot	7/12 (58%)	3 min.	32	1		7/11 (64%)	2 min.
	8/23 (35%)	5 min.	37	0		8/23 (35%)	5 min.
DeMoney	0/32 (0%)	2 h.	133	49	1 h.	0/19 (0%)	2 h.
	0/42 (0%)	2 h.	188	56		17/18 (94%)	1 h.

TAB. 2 – *Génération de tests à partir d'abstractions*

7 Conclusion et perspectives

Nous avons présenté dans cet article un algorithme de génération d'abstraction à partir d'un ensemble de prédicats. À partir de l'ensemble d'états initiaux, cet algorithme calcule l'ensemble des états symboliques atteignables par l'ensemble des opérations présentes dans le système. Nous avons fourni des éléments de preuve permettant d'établir la correction de cet algorithme. Nous avons montré que cet algorithme pouvait être implanté avec toute technologie fournissant deux primitives, un calcul de plus faible précondition et une procédure SAT.

Nous avons expérimenté cet algorithme avec deux implantations, l'une utilisant des techniques de preuve, et l'autre utilisant des techniques de résolution de contraintes. Les résultats expérimentaux nous montrent que les techniques de résolution de contraintes sont plus efficaces en temps, mais sur des exemples utilisant des variables avec de grands domaines, la génération d'abstraction n'est plus réalisable. Pour l'implantation utilisant des techniques de preuve, les résultats expérimentaux nous montrent que la génération d'abstraction s'effectue dans des temps raisonnables, mais certaines obligations de preuve n'ont pas pu être résolues. Certaines de ces transitions en défaut de preuve ont un impact fort sur la génération de tests, car certaines fois elles conduisent à l'échec de l'instanciation des tests les contenant. Pour pallier à ce problème, nous proposons d'associer des techniques de résolution de contraintes afin de filtrer les transitions en défaut de preuve, et donc d'éliminer les transitions n'ayant pas lieu d'être dans l'abstraction. Notons qu'il serait nécessaire de comparer notre approche à un

calcul utilisant des SMT-solvers comme Z3 de Moura et Bjørner (2008). La difficulté est de définir les théories Déharbe (2010) permettant de prendre en compte des spécifications logico-ensemblistes (ensemble, fonction, relation, séquences) à la B.

D'autres travaux se sont intéressés à la génération de tests à partir d'abstractions. Les techniques définies dans Calamé et al. (2007); Jeannet et al. (2005) et implantées dans l'outil STG Clarke et al. (2002) se basent sur une abstraction fournie par l'utilisateur sous la forme d'un IOSTS. Un produit synchronisé entre un objectif de test (également donné sous la forme d'un IOSTS) et l'abstraction est calculé, puis, après une phase d'optimisation consistant en la suppression des états inatteignables par interprétation abstraite, des algorithmes de parcours sont appliqués pour générer des tests à partir du produit synchronisé. Deux différences existent entre nos travaux et les techniques implantées dans STG. Premièrement, les abstractions que nous utilisons sont calculées à partir d'un ensemble de prédicats défini à partir de l'objectif de test, alors que les abstractions utilisées par STG doivent être fournies par l'utilisateur. Deuxièmement, la suppression des états inatteignables de l'abstraction est effectuée par l'algorithme d'abstraction en utilisant des propriétés invariantes du modèle qui n'existent pas dans les IOSTS.

Les travaux présentés dans Lahiri et al. (2007) proposent une technique d'abstraction appliquée aux programmes C basée sur des prédicats. Une abstraction booléenne est calculée à partir de l'ensemble des prédicats atomiques présents dans le flot de contrôle du programme. Cette méthode est basée sur des calculs de plus faible précondition. Nous avons la même approche sur des modèles, mais nous bénéficions des propriétés invariantes exprimées sur ces modèles pour obtenir une abstraction plus précise. De plus, nous sommes capable de construire un oracle associé aux tests générés. Dans Ball (2005), Ball propose une technique pour calculer une sous-approximation du programme basée sur les Tri-Modal Transitions System, lui permettant de ne générer que des tests instanciables.

La suite de ce travail consiste en l'amélioration de l'algorithme d'abstraction afin d'obtenir une suite de tests directement instanciable en calculant une sous-approximation. En effet, la seule information que nous pouvons extraire des transitions présentes dans l'abstraction générée est qu'il existe un état concret dans l'état abstrait source permettant de franchir l'opération de la transition afin d'atteindre un état concret de l'état abstrait cible. Tout le problème actuel de l'instanciation réside dans la recherche d'une succession d'états concrets permettant de concrétiser le test symbolique. Pour pallier à ce problème, nous souhaitons classifier les transitions de l'abstraction en utilisant les Tri-modal Transitions System Ball (2005) permettant d'enchaîner certaines transitions en étant sûr de leur concrétisation. Le problème à résoudre par rapport aux travaux de Ball (2005) est de calculer des sous-approximations suffisamment précises pour engendrer des ensembles de tests suffisant.

Références

Abrial, J.-R. (1996a). *The B Book*. Cambridge Univ. Press.

Abrial, J.-R. (1996b). Extending B without changing it (for developing distributed systems). In *1st B Conference*, pp. 169–190.

Ball, T. (2005). A theory of predicate-complete test coverage and generation. In *FMCO'04*, Volume 3657 of *LNCS*, pp. 1–22.

Ball, T., O. Kupferman, et G. Yorsh (2005). Abstraction for falsification. In *CAV*, pp. 67–81.

Beizer, B. (1995). *Black-Box Testing : Techniques for Functional Testing of Software and Systems*. New York, USA : John Wiley & Sons.

Bert, D., M.-L. Potet, et N. Stouls (2005). Genesyst : a tool to reason about behavioral aspects of B event specifications. In *ZB'05*, Volume 3455 of *LNCS*, pp. 299–318.

Bouquet, F., P.-C. Bué, J. Julliand, et P.-A. Masson (2009). Génération de tests à partir de critères dynamiques de sélection et par abstraction. In *AFADL'09*, Toulouse, France, pp. 161–176.

Bouquet, F., P.-C. Bué, J. Julliand, et P.-A. Masson (2010). Test generation based on abstraction and test purposes to complement structural tests. In *A-MOST'10, 6th int. Workshop on Advances in Model Based Testing*, IEEE proceedings of ICST'2010, Paris, France.

Bouquet, F., B. Legeard, et F. Peureux (2004). CLPS-B : A constraint solver to animate a B specification. *Software Tools for Technology Transfer 6*(2), 143–157.

Broy, M., B. Jonsson, J.-P. Katoen, M. Leucker, et A. Pretschner (Eds.) (2005). *Model-Based Testing of Reactive Systems*, Volume 3472 of *LNCS*. Springer.

Calamé, J., N. Ioustinova, et J. van de Pol (2007). Automatic model-based generation of parameterized test cases using data abstraction. *ENTCS 191*, 25–48.

Clarke, D., T. Jéron, V. Rusu, et E. Zinovieva (2002). STG : A symbolic test generation tool. In *TACAS'02, Tools and Algorithms for the Construction and Analysis of Systems*, Volume 2280 of *LNCS*, pp. 151–173. Springer.

de Moura, L. et N. Bjørner (2008). An efficient smt solver. In *TACAS'08*, Volume 4963 of *LNCS*, pp. 337–340.

Déharbe, D. (2010). Automatic verification for a class of proof obligations with smt-solvers. In *ASM*, pp. 217–230.

Graf, S. et H. Saïdi (1997). Construction of abstract state graphs with pvs. In *CAV'97*, Volume 1254 of *LNCS*, pp. 72–83.

Hoare, C. (1969). An axiomatic basis for computer programming. *CACM 12*, 576–580.

Jeannet, B., T. Jéron, V. Rusu, et E. Zinovieva (2005). Symbolic test selection based on approximate analysis. In *TACAS'05*, Volume 3440 of *LNCS*.

Julliand, J., P.-A. Masson, et R. Tissot (2008). Generating security tests in addition to functional tests. In *AST'08*, pp. 41–44. ACM Press.

Lahiri, S., T. Ball, et B. Cook (2007). Predicate abstraction via symbolic decision procedures. *Logical Methods in Computer Science 3*, 1–20.

Namjoshi, K. S. et R. P. Kurshan (2000). Syntactic program transformations for automatic abstraction. In *CAV'00*, Volume 1855 of *LNCS*, pp. 435–449.

Stouls, N. (2007). *Systèmes de transitions symboliques et hiérarchiques pour la conception et la validation de modèles B raffinés*. PhD thesis, INP Grenoble.

Thimbleby, H. (2003). The directed chinese postman problem. *Software : Practice and Experience 33*(11), 1081–1096.

Utting, M. et B. Legeard (2006). *Practical Model-Based Testing*. Morgan Kaufmann.