
Machines de mutations pour l'enrichissement de test de protocoles

A. Kermarrec* — R. Groz** — B. Parreaux* — A. Petrenko***

* France Télécom R&D

2, avenue Pierre Marzin,

22307 Lannion Cedex, France

{audrey.kermarrec, benoit.parreaux}@francetelecom.com

** INPG-ENSIMAG, Lab. LSR

F-38402 St Martin d'Hères, France

roland.groz@imag.fr

*** CRIM, Centre de Recherche en Informatique de Montréal,

550, rue Sherbrooke Ouest, bureau 100

Montréal (Québec), Canada H3A 1B9

petrenko@crim.ca

RÉSUMÉ. Dans cet article, nous étudions l'amélioration de la qualité des suites de test de protocoles. Nous visons à augmenter la capacité de détection d'erreurs, en particulier dans la gestion des variables internes d'un protocole, lorsque ces erreurs n'ont pas été manifestées au cours du test.

Nous introduisons la notion de machine de mutations pour les EFSM. Les machines de mutations permettent de représenter de façon compacte un ensemble de machines d'implémentation possibles (et potentiellement erronées) d'une machine de spécification donnée, en fonction d'hypothèses de fautes. Nous identifions cinq grandes familles d'opérateurs de mutations élémentaires pour construire une machine de mutation. Enfin, nous illustrons la faisabilité et l'intérêt de notre approche en réalisant une expérimentation sur le protocole INRES

ABSTRACT. In this paper, we address the problem of improving the quality of a protocol test by targeting implementation errors that are missed by the test.

We introduce the notion of a mutation machine for a specification EFSM to represent in a compact way a set of the possibly erroneous implementations of the specification which reflect test assumptions of the test designer. We identify five families of elementary mutation operators that are used to construct a mutation machine. The mutation machine allows the test designer to generate a test postamble to enhance a given test. The feasibility and potentials of the approach we develop are illustrated on the INRES protocol.

MOTS-CLÉS: EFSM, Modèle de fautes, Tests.

KEYWORDS: EFSM, Fault Models, Testing.

1. Introduction

Le travail présenté ici vise à étendre les méthodes de génération de test pour les protocoles décrits par des automates étendus avec des variables. À l'issue de tests de conformité, qui ont pu s'assurer que les sorties produites par l'implantation sont bien conformes à la spécification d'un protocole, il n'est pas sûr que la configuration réelle atteinte par une implantation de protocole ne comporte pas des fautes non manifestées par les sorties observables dans la durée limitée de chaque test.

Afin d'augmenter les capacités de détection de fautes de configurations internes des machines, nous cherchons à définir ici des méthodes pour :

- représenter ces fautes
- en déduire des séquences de test permettant de les mettre en évidence.

Si l'on part d'une suite de tests de conformité qui auraient été produits par une approche visant à couvrir les transitions et les sorties observables d'un protocole, notre méthode permet de l'étendre facilement. Chaque test sera complété par une séquence adaptée à la détection des fautes résiduelles non détectées par le test.

Plus précisément, connaissant la configuration atteinte dans la spécification EFSM, appelée configuration attendue, en réponse à un cas de test appliqué à partir d'une configuration initiale, la méthode proposée par [PET 04] permet de déterminer une séquence d'entrées paramétrées qui produit une séquence de sorties différente de celles produites dans l'ensemble ou dans un sous-ensemble des configurations erronées. Nous appellerons de telles séquences, séquences de discrimination. L'objectif n'est pas de distinguer la configuration correcte de toute autre configuration mais uniquement d'un ensemble de configurations constituant une "liste noire". [PET 04] laisse le soin à l'expert en test de la déterminer. Le présent article ne présente que la partie « amont », nouvelle, de la méthode, qui permet de produire par calcul des configurations suspectes. La méthode de calcul des séquences de distinction, complément indispensable, a déjà été présentée dans [PET 04].

Nous abordons ici la génération automatique de "listes noires" pour les spécifications EFSM. Nous proposons d'étendre le concept de machines de mutations sur les FSM, proposé dans [PET 92], aux EFSM. Nous en déduisons comment, à partir d'une machine de mutations et de cas de test, déterminer les configurations correspondant à des fautes résiduelles au sein des spécifications EFSM. Enfin, nous définissons les fautes pouvant être introduites au sein des modèles EFSM.

La suite du document est organisée comme suit. Dans la section 2, nous présentons le modèle EFSM que nous utilisons, ainsi que la définition des configurations pour les EFSM. Nous rappelons ensuite brièvement la méthode de calcul de séquences de discrimination. Le modèle de fautes pour les FSM, basé sur les machines de mutations, sera présenté dans la section 3 et étendu aux EFSM. Ce modèle de fautes permettra de représenter de façon compacte l'ensemble des

machines d'implémentation pouvant être dérivées d'une spécification EFSM. Dans la section 4, nous esquissons la méthode permettant de déterminer, à partir d'une machine de mutations et de cas de test, les configurations correspondant à des fautes résiduelles au sein des implémentations. Dans la section 5, nous définissons cinq familles d'opérateurs de mutation sur les EFSM, et nous comparons brièvement le pouvoir d'expressivité des machines de mutations par rapport aux opérateurs de mutation rencontrés dans le cadre du test par mutations. Au long de cet article, nous illustrons l'application de la méthode sur l'exemple classique du protocole *INRES*, et nous présentons en section 6 les résultats.

2. Définitions

2.1. Modèle EFSM

Une machine à états finis étendue M (EFSM) est une paire (S, T) composée d'un ensemble fini d'états S et d'un ensemble fini de transitions T tel que chaque transition $t \in T$ est un 7-uplet (s, x, P, op, y, up, s') où :

- $s, s' \in S$ sont, respectivement, les états de départ et d'arrivée de la transition.
- $x \in X$ est une entrée, X est l'ensemble des entrées. A chaque entrée x est associé un domaine D_x de valeurs pour les paramètres associés à son entrée. Les éléments de D_x , notés \mathbf{p}_x , sont des vecteurs de valeurs élémentaires. Chaque élément de ce vecteur d'entrées correspond à un paramètre d'entrée associé à x .
- $y \in Y$ est une sortie, Y est l'ensemble des sorties. A chaque sortie y est associé un domaine D_y de valeurs pour les paramètres associés à sa sortie. Les éléments de D_y , notés \mathbf{p}_y , sont des vecteurs de valeurs élémentaires. Chaque élément de ce vecteur de sorties correspond à un paramètre de sortie associé à y .
- P, op et up sont des fonctions définies sur les paramètres d'entrée et les variables de contexte V telles que :
 - $P : D_x \times D_V \rightarrow \{\text{True}, \text{False}\}$ est un prédicat, où D_V est l'ensemble des vecteurs de contexte \mathbf{v} .
 - $op : D_x \times D_V \rightarrow D_y$ est une fonction qui permet de calculer les paramètres de sortie.
 - $up : D_x \times D_V \rightarrow D_V$ est la fonction de mise à jour du contexte.

Un vecteur de contexte $\mathbf{v} \in D_V$ est appelé *contexte* de M . Une *configuration* de M est une paire constituée d'un état s et d'un contexte \mathbf{v} . Dans le cas où l'ensemble des variables de contexte est vide, nous ne distinguons pas la configuration de l'état.

Une machine M est généralement initialisée dans une certaine configuration, que l'on appellera *configuration initiale*. Une paire constituée d'une EFSM M et d'une configuration initiale est appelée *EFSM initialisée*.

L'ensemble de transitions T définit le comportement (propriétés dynamiques) de l'EFSM. Une EFSM reçoit une entrée paramétrée [PET 04] et calcule les prédicats qui sont satisfaits pour la configuration courante. Une transition étiquetée avec cette entrée et qui satisfait le prédicat, appelée transition *tirable*, est alors tirée et l'EFSM change d'état (les transitions sont atomiques et ne peuvent être interrompues). Lors du passage de la transition, cette EFSM produit une sortie paramétrée et modifie la valeur de ses variables selon la fonction de mise à jour.

La transition $t \in T$ sera notée de la façon suivante : $(s - x, P / op, y, up \rightarrow s')$. Si, dans t , le prédicat P a toujours pour valeur True alors, P peut être supprimé de la notation. De la même façon, lorsque la transition ne modifie pas la valeur des

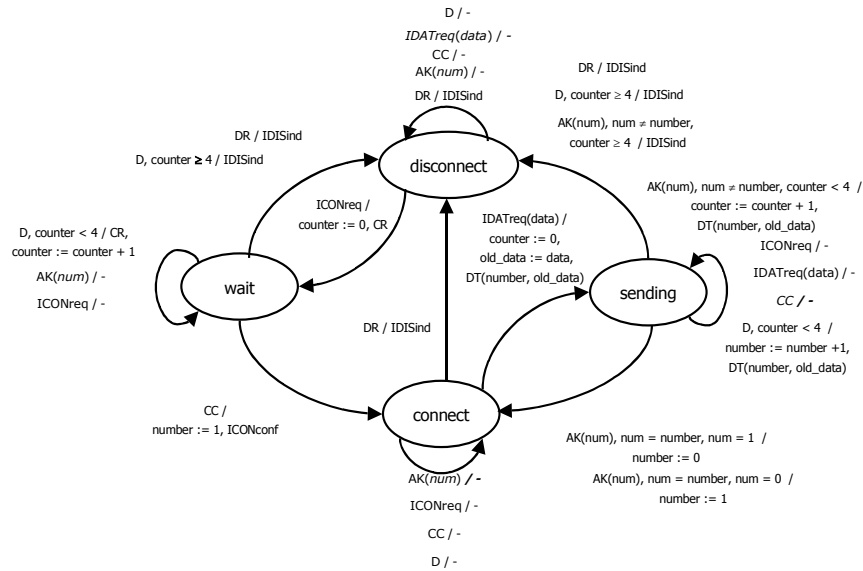


Figure 1. Exemple de spécification EFSM : processus appelant du protocole INRES.

variables, on peut omettre la fonction de mise à jour. Enfin, la fonction op peut être supprimée si la sortie y n'a pas de paramètres de sortie. Les notations $(s - x, P / y \rightarrow s')$, $(s - x / y, up \rightarrow s')$ et $(s - x / y \rightarrow s')$ sont des exemples de notations utilisées pour de telles situations.

Dans la suite du document, nous utilisons le protocole de communication INRES (INItiator-RESpnder) pour illustrer notre discours. Ce protocole, proposé dans [HOG 95], est un cas d'étude destiné à illustrer quelques-uns des mécanismes mis en œuvre dans beaucoup de protocoles, et il sert de pierre de touche pour les méthodes de génération de test. Ce protocole montre un établissement (simplifié) de connexion suivi d'un transfert de données entre un processus *appelant* (*initiator*) et un processus *appelé* (*responder*). La spécification EFSM du processus *appelant*, décrite sur la figure 1, possède 4 états et 28 transitions. Ces transitions sont étiquetées par :

- 6 entrées : DR , CC , $ICONreq$, AK et $IDATreq$, et D qui représente l'expiration du délai de garde associé à chaque émission d'une sortie DT ou CR .
- 5 sorties : DT , CR , $ICONconf$, $IDISind$ et $NULL$ (noté "-" sur la figure 1),
- 13 prédicats différents de True,
- 8 mises à jour de variables et,
- 3 variables de contexte : $counter$, $number$ et old_data .

Les entrées AK et $IDATreq$ sont, respectivement, paramétrées par un entier et par une donnée. La sortie DT est paramétrée par deux paramètres de sortie : un entier et une donnée.

Une *séquence d'entrées-sorties paramétrées* de M , notée α_{io} , est une séquence d'entrées paramétrées entrecoupée par des sorties paramétrées produite par cette EFSM suite à l'exécution de la séquence de transitions tirables à partir d'une certaine configuration en réponse à ces entrées.

Nous introduisons plusieurs propriétés sur les EFSM afin de définir les différents types de spécifications qui peuvent être prises en compte.

Une EFSM M est dite :

- *complète vis-à-vis des prédicats* si pour chaque transition t , tous les éléments de $D_x \times D_y$ évaluent au moins un prédicat à True parmi l'ensemble de toutes les transitions ayant même état de départ et même entrée que t ;
- *complète vis-à-vis des entrées* si pour chaque paire $(s, x) \in S \times X$, il existe au moins une transition, sortant de l'état s avec l'entrée x ;
- *déterministe* si pour toute paire de transitions sortant d'un même état avec une même entrée, leurs prédicats s'excluent mutuellement ;
- *observable* si pour chaque état et chaque sortie, toutes les transitions sortantes avec la même entrée possède une sortie distincte par le nom ou par la valeur de ses paramètres.

Nous introduisons également les notions de sur-machine et de sous-machine d'une EFSM qui nous permettront de décrire ultérieurement un modèle de fautes représentant de façon compacte l'ensemble des machines d'implémentation erronées.

Soient deux EFSM $M = (S, T)$ et $N = (S', T')$ définies sur les mêmes ensemble d'entrées, ensemble de sorties, et sur les mêmes paramètres d'entrées, paramètres de sorties et variables de contexte, i.e. $D_{xM} = D'_{xN}$, $D_{yM} = D'_{yN}$, et $D_{VM} = D'_{VN}$, la machine $N = (S', T')$ est une *sur-machine* de M si $S \subseteq S'$ et $T \subseteq T'$ (resp. la machine N est une *sous-machine* de M si $S' \subseteq S$ et $T' \subseteq T$). L'ensemble de toutes les sur-machines de M (resp. l'ensemble des sous-machines) est noté $Sup(M)$ (resp. $Sub(M)$).

Dans la suite du document, nous considérons les spécifications décrites dans le modèle EFSM comme étant complètes vis-à-vis des prédicats, complètes vis-à-vis des entrées, déterministes et observables. La complétude vis-à-vis des entrées peut être facilement obtenue en rajoutant à la spécification des transitions implicites pour

consommer en silence (sans production de sortie ni modification des variables) les entrées non attendues. Les transitions en italique sur la figure 1 sont des exemples de transitions implicites et peuvent être omises dans la spécification.

2.2. Séquences de discrimination

L'identification de la configuration d'arrivée au sein du modèle EFSM consiste à s'assurer que le test conduit la machine d'implémentation dans la configuration attendue dans la spécification. Il est donc nécessaire, d'une part, de savoir distinguer les configurations d'EFSM et, d'autre part, de définir l'équivalence de configurations.

Soient deux EFSM M et N , deux entrées (resp. deux sorties) de M et N sont dites *compatibles* si leurs noms coïncident et que les valeurs des paramètres des deux entrées (resp. des deux sorties) sont deux à deux égales. Deux séquences d'entrées-sorties, $z_1 \dots z_k$ de M et $z'_1 \dots z'_k$ de N , sont *compatibles* si pour tout $i = 1 \dots k$, z_i et z'_i sont compatibles.

A partir de cette définition, nous pouvons caractériser le fait que des configurations sont distinguables.

Soit une séquence d'entrées paramétrées α , une configuration c de M et une configuration c' de N sont distinguables par α si la séquence de sorties paramétrées, produite par (M, c) en réponse à α , n'est compatible à aucune séquence de sorties paramétrées qui peut être produite par (N, c') en réponse à α . α est dite *séquence distinguant* c de c' , encore appelée *séquence de discrimination*.

Les deux configurations, c et c' , d'EFSM complètes et déterministes, définies sur les mêmes ensemble d'entrées, ensemble de sorties et sur les mêmes paramètres d'entrées, paramètres de sorties et variables de contexte, non distinguables sont aussi référencées comme *configurations équivalentes*, notée $c \cong c'$. Deux EFSM complètes et déterministes, respectivement initialisées dans les configurations c_0 et c'_0 , sont équivalentes si c_0 et c'_0 sont équivalentes.

3. Modèles de fautes

Dans le cadre du test par mutations classique [BUD 81], [MIL 88], l'ensemble des fautes possibles d'une implémentation, pour une spécification donnée, est décrit par un certain ensemble de mutants, appelé domaine de fautes. Ce domaine de fautes est déterminé par un modèle de fautes.

L'ensemble des fautes possibles de la spécification est modélisé par un ensemble de machines d'implémentation. [PET 92], [KOU 99] ont proposé de représenter cet ensemble des machines d'implémentations pour une FSM par une machine de mutations et ont ainsi défini un modèle de fautes plus compact. Dans les paragraphes suivants, nous présentons le modèle de fautes pour les FSM et nous étendons cette proposition aux EFSM.

3.1. Modèle de fautes pour les FSM

Soit $A = (S, X, Y, h, s_0)$ une FSM complètement spécifiée et déterministe, appelée *machine de spécification*. Nous supposons que toutes les machines d'implémentation possibles de cette machine de spécification sont représentées par l'ensemble de toutes les sous-machines déterministes d'une FSM $\tilde{A} = (U, X, Y, F, u_0)$. \tilde{A} est une sur-machine de A , appelée *machine de mutations de la machine de spécification* [KOU 99]. Une sous-machine déterministe B de \tilde{A} est appelée *implémentation non-conforme* si elle n'est pas équivalente à A , sinon elle est appelée *machine conforme*. En d'autres termes, nous considérons le modèle de fautes $\langle A, \cong, Sub(\tilde{A}) \rangle$.

Chaque ensemble fini $E \subset X^*$ de séquences d'entrées est une *suite de test* pour A . La suite de test E *détecte* une implémentation non-conforme $B \in Sub(\tilde{A})$ si elle contient une séquence d'entrées $\alpha \in E$ discriminant les machines A et B .

Nous proposons d'étendre la définition de machine de mutations au modèle EFSM afin de représenter l'ensemble des fautes pouvant être commises sur cette dernière de façon compacte contrairement au test par mutation pour lequel chaque faute conduit à engendrer un mutant.

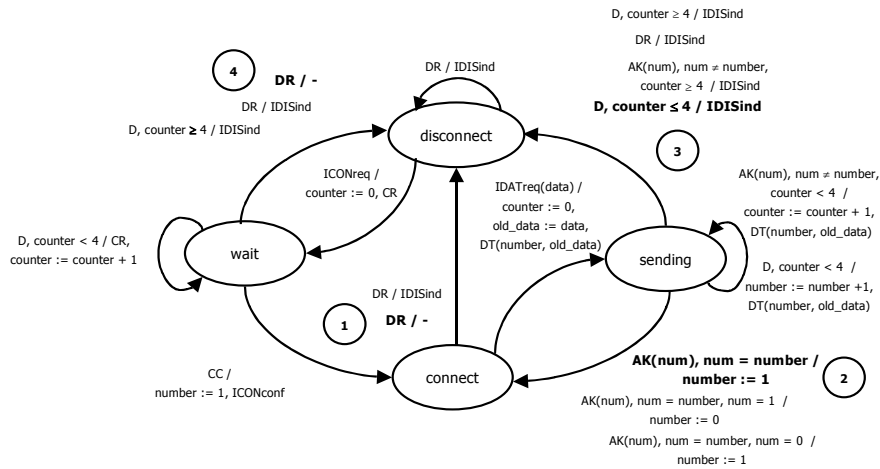


Figure 2. Exemple de machine de mutations \tilde{M} de l'EFSM présentée sur la Figure 1

3.2. Modèle de fautes pour les EFSM

Soit $M = (S, T)$ une EFSM déterministe, complète vis-à-vis des prédicats et des entrées, et observable appelée *EFSM de spécification*. Nous supposons que toutes les machines d'implémentation possibles de cette EFSM de spécification sont représentées par l'ensemble de toutes les sous-machines déterministes d'une EFSM \tilde{M}

(S', T') , où \tilde{M} est une sur-machine de M , appelée *machine de mutations* (ou *EFSM de mutations*) de l'EFSM de spécification. Une sous-machine N de \tilde{M} est appelée *implémentation non-conforme* si elle n'est pas équivalente à M , sinon elle est appelée *machine conforme*. En d'autres termes, nous considérons le modèle de fautes $\langle M, \cong, Sub(\tilde{M}) \rangle$.

Tout ensemble fini de séquences d'entrées-sorties paramétrées est une suite de test pour A . Une suite de test *détecte* une implémentation non-conforme N , où $N \in Sub(\tilde{M})$, si elle contient une séquence d'entrées paramétrée α_i qui distingue la configuration initiale de M de la configuration initiale de N .

La figure 2 présente une machine de mutations \tilde{M} particulière de l'EFSM M de la figure 1. La machine de mutations \tilde{M} est une sur-machine indéterministe de la machine de spécification M . Son ensemble de transitions inclut l'ensemble des transitions de l'EFSM M et quatre transitions additionnelles modélisant quatre mutations, numérotées de 1 à 4 sur la figure 2.

4. Fautes résiduelles à l'issue d'un test

4.1. Ensemble des configurations suspectes

Soient une machine de mutations \tilde{M} complète et indéterministe modélisant l'ensemble des fautes potentielles de l'implémentation, et un cas de test (séquence d'entrées-sorties paramétrées) α , l'exécution correcte du cas de test α conduit la machine de mutations \tilde{M} dans un ensemble de configurations C_α , appelé *ensemble de configurations suspectes résultant de α* .

On notera que les configurations atteintes suite à une exécution du cas de test manifestant, au cours du test, un comportement erroné de la machine de mutations seront exclues de la liste des configurations suspectes et il ne sera pas nécessaire de calculer une séquence de discrimination pour ces dernières.

Soit $c = (disconnect, (0, 0, NULL))$ la configuration initiale de la machine de spécification M (cf. figure 1) et de la machine de mutations \tilde{M} (cf. figure 2), où :

- $s = disconnect$, état de l'EFSM et,
- $\mathbf{v} = (0, 0, NULL)$, vecteur des variables de contexte (*counter*, *number*, *old_data*).

Soit α le cas de test suivant : $\alpha = (ICONreq / CR, CC / ICONconf, IDATreq(data) / DT(1,data_1), AK(1) / -)$. Les configurations atteintes, suite à l'exécution correcte du cas de test α , sur les machines EFSM initialisées (M, c) et (\tilde{M}, c) sont indiquées sur la figure 3. On peut remarquer que la machine de spécification M étant déterministe la trace d'exécution du cas de test est unique et mène la machine de spécification M dans la configuration attendue $c_A = (connect, (0, 0, data_1))$. L'exécution correcte de α sur la machine de mutations \tilde{M} indéterministe, produit

deux traces d'exécution identiques et mène \tilde{M} dans deux configurations distinctes $c_{S1} = (\text{connect}, (0, 0, \text{data}_1))$ et $c_{S2} = (\text{connect}, (0, 1, \text{data}_1))$.

L'ensemble des configurations suspectes résultant de α pour l'EFSM (M, c) se résume à $C_\alpha = \{c_{S2}\}$, car $c_{S1} = c_A$.

4.2. Le calcul des séquences de discrimination

L'exécution du cas de test conduit la machine de mutations dans des configurations suspectes. Notre objectif est de calculer des séquences de discrimination permettant de débusquer les configurations inatteignables par la machine de spécification.

Le problème que nous souhaitons résoudre peut s'énoncer de la façon suivante. Etant donné une spécification M , une machine de mutations \tilde{M} , un cas de test α et un ensemble de configurations suspectes C_α résultant du cas de test α , comment calculer une séquence de discrimination de longueur minimale qui distingue la configuration c attendue par la spécification, suite à l'exécution du test, des configurations suspectes, distinguables, c' appartenant à C_α . La longueur de la séquence de discrimination étant bornée, on ne distinguera pas toutes les configurations suspectes de l'ensemble C_α de la configuration attendue mais un maximum de configurations suspectes.

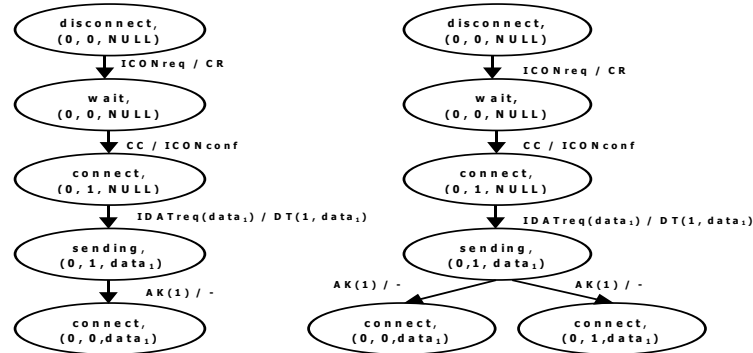


Figure 3. Configurations atteintes par M et \tilde{M} suite à l'exécution du cas de test.

Le calcul des séquences de discrimination a été présenté dans [PET 04]. Il s'appuie sur la notion de machine de distinction, une forme de produit synchrone de deux machines permettant de caractériser l'ensemble des séquences d'entrées distinguant deux configurations données. La machine de distinction de $(M, (s, \mathbf{v}))$ pour $(N, (q, \mathbf{u}))$ est une EFSM qui a pour ensemble d'état $S \times (Q \cup \{\text{fail}\})$, où S et Q sont, respectivement, les ensembles d'états de M et N , $\text{fail} \notin Q$ est un état.

Dans notre cas, nous construisons la machine de distinction de (M, c) pour (M, c') , c étant une configuration attendue et c' une configuration suspecte.

Une séquence d'entrées est une séquence de distinction pour les configurations (s, \mathbf{v}) de la machine M et (q, \mathbf{u}) de la machine N si et seulement si, à partir de la configuration initiale, elle mène la machine de distinction $[(M, (s, \mathbf{v})) - (N, (q, \mathbf{u}))]$ dans l'état *fail*.

La longueur des séquences de discrimination étant bornée pour réduire la complexité du test et du calcul du test, la notion de *l-fragment* de la machine de distinction est introduite. Étant donnée une EFSM K , le *l-fragment* de K est défini comme étant une sous-machine de K obtenue à partir du graphe de K en supprimant les nœuds, dont la distance à partir du nœud initial excède la longueur l , ainsi que leurs transitions adjacentes.

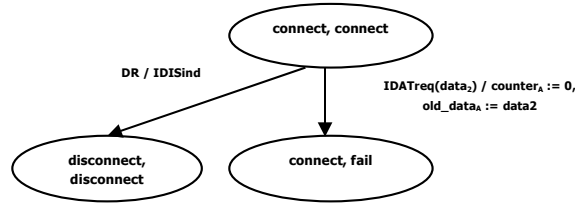


Figure 4. 1-fragment de la machine de distinction $[(M, (connect, (0, 0, data_1))) - (M, (connect, (0, 1, data_1)))]$.

Nous cherchons à déterminer une séquence (si elle existe) permettant de distinguer la configuration $c_A = (connect, (0, 0, data_1))$ de la configuration $c_{S2} = (connect, (0, 1, data_1))$. La longueur de la séquence ne doit pas excéder 1, i.e. $l = 1$. La figure 4 présente le 1-fragment de la machine de distinction $[(M, c_A) - (M, c_{S2})]$. Les variables de contexte de (M, c_A) (resp. (M, c_{S2})) sont $counter_A$ et old_data_A (resp. $counter_S$ et old_data_S). La transition $\alpha_{SD} = (IDATreq(data_2))$ permet de conduire la machine de distinction dans l'état *fail*.

Les configurations suspectes correspondent à des fautes résiduelles au niveau des implémentations. Dans la méthode proposée, les configurations suspectes sont atteintes suite à l'exécution correcte d'un cas de test par la machine de mutations. Les fautes introduites dans la machine de mutations doivent donc correspondre à des fautes "pertinentes" des systèmes. C'est pourquoi nous nous intéressons aux types de fautes pouvant être commises sur des EFSM.

5. Expressivité des machines de mutations

La machine de mutations définie dans la section 3 permet de décrire l'ensemble des EFSM d'implémentation possibles dérivées d'une EFSM de spécification. Dans cette section, nous cherchons à déterminer les types de mutations pouvant être

introduites au sein des spécifications EFSM. Nous définissons pour cela une fonction de mutation, combinaison d'opérateurs élémentaires permettant de rajouter des transitions (ou des états) à la machine de spécification. Nous comparerons les opérateurs que nous proposons à ceux rencontrés dans le cadre du test par mutations classique.

5.1. Opérateurs sur les spécifications EFSM

Soit une machine de spécification $M = (S, T)$. Les ensembles S et T constituent les possibilités de fautes pouvant être rencontrées dans les machines d'implémentation. Comme les machines sont supposées complètes, les couples (s, x) identifient toutes les possibilités de transition, de sorte que nous ne considérerons que les mutations de P , op , y , up et s' pour les transitions. Nous n'avons pour l'instant pas inclus de mutation introduisant des états supplémentaires, mais ceci se ferait assez facilement sur les mêmes bases. Notons que le non-déterminisme inhérent aux machines de mutations introduit des comportements similaires à ceux qu'apporteraient des états supplémentaires.

Pour chacune de ces 5 composantes mutables, nous définissons une famille d'opérateurs. Chaque opérateur associe à chaque transition de la machine de spécification un ensemble de transitions mutées. Le champ des valeurs prises par les mutations sera en général restreint à des valeurs proches des domaines de valeurs de la spécification.

Nous notons de la façon suivante les cinq familles d'opérateurs de mutation pouvant être introduites sur le 7-uplet (s, x, P, op, y, up, s') .

- Λ_P : famille d'opérateurs de mutation du prédicat P ,
- Λ_{op} : famille d'opérateurs de mutation de la fonction de calcul des paramètres de sortie op ,
- Λ_y : famille d'opérateurs de mutation de la sortie y ,
- Λ_{up} : famille d'opérateurs de mutation de la fonction de mise à jour du contexte up ,
- $\Lambda_{s'}$: famille d'opérateurs de mutation de l'état d'arrivée s' .

Nous présentons ici deux exemples de ces familles d'opérateurs de mutation. Les autres sont similaires.

5.1.1. Famille d'opérateurs de mutation Λ_P

La famille d'opérateurs de mutation Λ_P du prédicat P peut se définir ainsi : tout opérateur $\Lambda_P \in \Lambda_P$ est défini par la donnée d'un ensemble $E'_P \supseteq E_P$, l'ensemble des prédicats apparaissant dans l'EFSM M .

Pour chaque transition $t = (s - x \wedge P / up, y(op) \rightarrow s') \in T$ avec le prédicat P ,

$$\Lambda_P(t) = \{ (s - x \wedge P' / up, y(op) \rightarrow s') \mid P' \in E'_P \}$$

Nous présentons quelques formes particulières de cet opérateur.

– Si $E'_P = E_P$, la mutation aura pour effet de remplacer les prédicats d'une transition par des prédicats venant d'autres transitions de la machine de spécification. On note qu'il sera possible d'élargir le nombre de remplaçants potentiels en incluant :

- les prédicats complémentaires de l'ensemble E_P . Deux prédicats sont dits complémentaires, si à chaque fois que l'un d'eux est évalué à True, l'autre est évalué à False.

- les prédicats obtenus par conjonction et/ou disjonction des prédicats appartenant à l'ensemble E_P .

– Si E'_P contient $P' = \text{True}$, la mutation aura pour effet de supprimer le prédicat et de rendre la transition toujours tirable à la réception de l'entrée x .

– Si E'_P contient $P' = \text{False}$, la mutation aura pour effet de supprimer cette transition.

– La mutation $P' = \text{not}(P)$ aura pour effet d'inverser les conditions d'activation de la transition.

5.1.2. Famille d'opérateurs de mutation Λ_{up}

La famille d'opérateurs de mutation Λ_{up} des fonctions de mise à jour du contexte up peut se définir ainsi :

Pour chaque transition $t = (s - x \wedge P / up, y(op) \rightarrow s') \in T$ avec les fonctions up ,
 $\Lambda_{up}(t) = \{ (s - x \wedge P / up', y(op) \rightarrow s') \mid up' : D_x \times D_V \rightarrow D_V \}$

Nous présentons quelques formes particulières de ce type de fautes.

– La mutation $up' = id_{up}$ aura pour effet de ne pas modifier la valeur des variables de contexte. id_{up} désigne l'identité pour la fonction up . Par définition, la fonction de mise à jour des variables id_{up} ne modifie pas la valeur des variables de contexte, i.e. : $\forall (\mathbf{p}_x, v) \in D_x \times D_V, id_{up}(\mathbf{p}_x, v) = v$.

– Si up' est une fonction constante $up'(\mathbf{v}) \in D_V$, ces mutations auront pour effet de remplacer les vecteurs de contexte par des vecteurs arbitraires faisant partie de l'ensemble D_V .

Les cinq opérateurs de mutations définis ci-dessus peuvent être combinés afin de réaliser des mutations plus complexes sur les transitions.

5.2. Comparaison avec les opérateurs de mutations classiques

Dans le cadre du test par mutation classique, des fautes simples sont introduites au sein de la spécification. Chaque faute ainsi introduite donne naissance à une nouvelle spécification, appelée mutant. De nombreuses études ont permis de définir des ensembles d'opérateurs de mutation pour les spécifications, notamment les

opérateurs définis dans le cadre du test des protocoles de communication [PRO 91] [BOU 00] [KOV 03] [AMM 99] [SOU 00] [SUG 04].

Dans l'approche que nous proposons, les machines de mutations possèdent, à la fois, le comportement de la spécification et des comportements additionnels potentiellement erronés. Ces comportements additionnels sont induits par l'ajout de transitions mutées au sein de la spécification.

Notons qu'il est possible d'exhiber, à partir d'une machine de mutations, les mutants qui seraient obtenus par des techniques de mutation classiques. En effet, l'extraction d'une sous-machine qui possède une transition mutée et dont les autres transitions sont correctes équivaut au mutant obtenu par un opérateur classique. De plus, la combinaison de mutations et de l'indéterminisme introduit dans la machine de mutations engendrent un comportement qui ne peut être obtenu par une mutation simple. Par exemple, la machine de mutation étant non-déterministe, une exécution pourrait dans un premier passage suivre une transition spécifiée, mais en deuxième passage au même état avec le même input activer une transition mutante. Par conséquent, les machines de mutations regroupent mutations classiques et mutations "furtives".

Afin de pouvoir calculer des séquences de discrimination, il nous faut cependant limiter le type de mutations et obtenir des critères de choix des mutations.

6. Étude de cas

Nous présentons, dans cette section, les résultats obtenus par la mise en œuvre de notre approche sur l'exemple du protocole *INRES* présenté dans la section 2 et nous esquissons, au travers de l'application sur un exemple industriel, les problèmes et limitations de notre approche ainsi que des pistes pour les contourner. Le protocole *INRES* est spécifié dans le langage *IF* (*Intermediate Format*) [BOZ 98] [BOZ 00].

Nous travaillons sur un prototype qui implémente les trois étapes de notre approche :

- Le calcul de la machine de mutations,
- Le calcul des configurations suspectes résultant de cas de test,
- Le calcul des séquences de discrimination.

Actuellement, nous avons obtenu un prototype écrit en C++. Nous utilisons les différentes bibliothèques mises à disposition par la boîte à outils *IF* afin de construire la machine de mutations et de réaliser le parcours de l'espace d'états du modèle. Les calculs des produits synchrones sont réalisés à la volée en utilisant un simulateur dédié.

Nous construisons une machine de mutations en parcourant l'arbre syntaxique de la spécification (cf. figure 1) et en appliquant un ou plusieurs types d'opérateurs de mutations (parmi les cinq familles d'opérateurs définies dans la section 5) sur des

transitions "cibles" (dans l'expérimentation du protocole *INRES*, toutes les transitions sont mutées).

Machine de mutations	Λ_p	Λ_{op}	Λ_y	Λ_{up}	$\Lambda_{s'}$	Nombre de transitions de mutations
\tilde{M}_1	×					36
\tilde{M}_2		×				36
\tilde{M}_3	×				×	120
\tilde{M}_4		×			×	120
\tilde{M}_5	×	×				52
\tilde{M}_6	×	×	×	×	×	276

Tableau 1. Nombre de transitions de la machine de mutations obtenues après application des opérateurs de mutations sur la spécification *INRES*.

Le tableau 1 présente le nombre de transitions mutées additionnelles de la machine de mutations en fonction des opérateurs choisis. L'opérateur Λ_p consiste à remplacer un prédicat par son complémentaire. Les opérateurs Λ_{up} et Λ_{op} remplacent un appel de fonction par une constante du type retourné. Enfin, l'opérateur Λ_y (resp. $\Lambda_{s'}$) échange une sortie (resp. un état) par une autre sortie (resp. un autre état) appartenant à l'ensemble Y (resp. à l'ensemble S).

L'application en parallèle des différents opérateurs de mutations, i.e. plusieurs mutations peuvent être introduites sur une même transition, engendre un nombre important de transitions. Cependant, le concept de machine de mutations permet de représenter de façon compacte ces dernières et évite, à ce stade de la méthode, une explosion du nombre de mutants.

Ensemble de configurations suspectes	C_1	C_2	C_3	C_4	C_5	C_6
Nombre	2	10	17	40	14	64

Tableau 2. Nombre de configurations suspectes calculées après exécution correcte de la suite de test sur chacune des machines de mutations.

À partir d'une suite de tests (10 cas de test dont la longueur varie de 1 à 5) obtenue par un outil de génération de tests, nous calculons, pour chacune des machines de mutations construites précédemment, l'ensemble des configurations suspectes. Les configurations suspectes sont les configurations atteintes par la machine de mutations suite à une exécution correcte du cas de test. Nous indiquons, dans le tableau 2, le nombre de configurations suspectes atteintes par les machines de mutations $\tilde{M}_1 \dots \tilde{M}_6$ (cf. tableau 1) suite à l'exécution de la suite de test.

Ce tableau met en évidence la corrélation existant entre le nombre de transitions introduites au sein de la machine de mutations et le nombre de configurations suspectes calculées. Dans cette version du prototype, nous réalisons un calcul explicite des valeurs des variables ce qui se traduit par un nombre important de configurations suspectes. Nous étudions différentes solutions afin de réaliser un parcours symbolique de l'espace d'états qui entraînerait la réduction du nombre de configurations suspectes obtenues.

Notre objectif est de compléter une suite de tests existante par le calcul de tests additionnels : les séquences de discrimination. Pour cela, nous calculons la machine de distinction de (M, c_A) pour (M, C_S) , avec c_A la configuration attendue suite à l'exécution du cas de test sur la spécification et C_S l'ensemble des configurations suspectes calculées précédemment. La mise en œuvre du calcul de la machine de distinction s'effectue en réalisant le produit synchrone de (M, c_A) avec (M, C_S) . Ce produit synchrone consiste à comparer les entrées-sorties de ces deux automates au cours de leur exécution.

Nous avons calculé les séquences de discrimination pour les ensembles de configurations suspectes présentés dans le tableau 2. L'analyse des séquences de discrimination obtenues en fonction des configurations suspectes définies nous donne les résultats suivants :

- Pour les machines de mutations comportant 36 transitions, nous obtenons une séquence de discrimination permettant de compléter cinq tests.
- Pour les machines de mutations comportant 52 et 120 transitions, nous obtenons une séquence de discrimination permettant de compléter sept tests.
- Les séquences de discrimination permettent de distinguer entre 100% et 70% des configurations suspectes.

Cette méthode permet de distinguer en moyenne 80% des configurations suspectes en déterminant des séquences courtes, ainsi que d'enrichir environ 50% des tests de la suite existante.

Cette expérimentation sur un exemple jouet montre la faisabilité de notre méthode et notamment qu'il est possible de déterminer des configurations suspectes à partir des machines de mutations. De plus, la technique que nous proposons pour déterminer les configurations suspectes nous permet d'utiliser l'approche décrite dans [PET 04] pour arriver à un calcul efficace des séquences de discrimination.

Nous expérimentons actuellement notre méthode sur un service vocal déployé dans le réseau de France Télécom. Notre choix a été dicté par l'existence d'une spécification et d'une suite de tests pour ce service vocal. De plus, la disponibilité du service nous permettra de valider notre démarche.

Les expérimentations ont confirmé qu'il était plus efficace de se concentrer sur les mutations portant sur les variables de contexte, mais il nous reste à identifier de bons mécanismes d'abstraction pour représenter automatiser ces choix sans une

expertise particulière du protocole et pour se relier à un calcul efficace des séquences de distinction.

7. Conclusion

Dans ce travail, nous avons atteint deux objectifs. D'abord, nous avons complété la méthode de génération de test proposée par [PET 04], en fournissant une façon de calculer des configurations suspectes à partir d'un modèle de fautes. Rappelons que dans l'approche initiale, les configurations suspectes étaient supposées données, et la façon de les produire était laissée pour étude ultérieure. Ensuite, nous avons proposé une méthode simple, compacte et bien adaptée au calcul de séquences de vérification pour représenter un modèle de fautes, en étendant aux EFSM la notion de machine de mutations proposée par [PET 92] pour les automates simples. En outre, nous avons indiqué comment les machines de mutation étaient suffisamment expressives pour représenter les modèles de fautes classiquement utilisés pour les EFSM. Nous avons illustré l'utilisation de cette technique par le calcul de suites de test enrichies pour l'exemple habituel, le protocole *INRES*.

Nos travaux se poursuivent actuellement dans deux directions. D'abord, nous appliquons cette méthode à un cas d'étude réel lié à des services vocaux, afin de valider dans un contexte plus global notre approche. Ensuite nous poursuivons le développement de notre prototype. Enfin, nous profitons du pouvoir d'expression et du cadre conceptuel offert par la notion de machine de mutation pour envisager des extensions aux algorithmes de calculs de séquences de discrimination, dans l'espoir d'aboutir à des tests ayant de meilleures capacités de détection de fautes.

8. Bibliographie

- [AMM 99] Ammann P., Black P., "A specification-based coverage metric to evaluate test sets", In proc. of Fourth IEEE International High-Assurance, Systems Engineering Symposium (HASE 99), IEEE Computer Society, pp. 239-248, 1999.
- [BOR 02] Boroday S., Groz R., Petrenko A., Quemener Y.-M., "Techniques for Abstracting SDL Specifications", *Proc. 3rd SAM (SDL And MSC) Workshop*, LNCS 2599, pp. 141-157, 2002.
- [BOU 00] Bousquet L.D., Ramangalahy S., Simon S., Hiho C., Belinfante A., Vries R.G., "Formal test automation : The conference protocol with TGV/TORX", IFIP 13th International Conference on Testing of Communicating Systems (TestCom 2000), Kluwer Academic Publishers, Ural, H. Probert, R.L. and v. Bochmann, G. editors, 2000.
- [BOZ 98] Bozga M., Graf S., Mounier L., Sifakis J., "The Intermediate Representation IF", In Verimag Technical Report, 1998.
- [BOZ 00] Bozga M., Fernandez J.-C., Ghirvu L., Graf S., Krimm J.P., Mounier L., "IF : a Validation Environment for Timed Asynchronous Systems", In *Springer-Verlag, Ed.*, proceedings of CAV'00, Chicago, USA, July 2000.

- [BUD 81] Budd T.A., "Mutation Analysis: Ideas, Examples, Problems and Prospects", *In Computer Program Testing*, pp. 129-148, 1981.
- [HOG 95] Hogrefe D., "Report on the Validation of the Inres System". Technical Report IAM-95-007, Universitat Bern, November 1995.
- [KOU 99] Koufareva I., Petrenko A., Yevtushenko N., "Test Generation Driven by User-defined Fault models", *In proc. of the 12th International Workshop on Testing of Communicating Systems (IWTCS'99)*, pp. 215-233, Budapest, Hungary, September 1999.
- [KOV 03] Kovács G., Pap Z., Le Viet D., Wu-Hen-Chang A., Csopaki G., "Applying Mutation Analysis to SDL Specifications", *In Springer, Ed., SDL 2003 : System Design. 11th International SDL Forum*, pp. 269-284, Stuttgart, Germany, July 2003.
- [MIL 88] DeMillo R.A., Guindi D., King K., McCracken W.M., Offutt A.J., "An Extended Overview of the Mothra Software Testing Environment", *In proc. Of the 2nd Workshop on Software Testing, Verification, and Analysis*, pp.142-151, July 1988.
- [PET 04] Petrenko A., Boroday S., Groz R., "Confirming Configurations in EFSM Testing", *IEEE Transactions on Software Engineering*, vol. 30, no. 1, pp. 29-42, January 2004.
- [PET 92] Petrenko A., Yevtushenko N., "Test Suite Generation for a Given Type of Implementation Errors", *Proc. IFIP XII International Conf. on Protocol Specification, Testing, and Verification*, pp. 229-243, 1992.
- [PRO 91] Probert R.L., Guo F., "Mutation testing of protocols : Principles and preliminary experimental results.", in proc. of the IFIP TC6 Third International Workshop on Protocol Test Systems, Elsevier Science Publishers B.V., North-Holland", pp. 57-76, 1991
- [SOU 00] Souza S.R.S., Maldonado J.C., Fabbri S.C.P.F., Lopes de Souza W., "Mutation testing applied to Estelle Specifications", *In Quality Software Journal*, volume 8(4), pp. 285-301, 2000.
- [SUG 04] Sugeta T., Maldonado J.C., Wong W.E., "Mutation testing applied to validate SDL Specifications", *In proc. of the 16th IFIP International Conference on Testing of Communicating Systems (TestCom 2004)*, pp. 193-208, Oxford, 2004.