Test de systèmes réactifs non réceptifs

Grégory Lestiennes — Marie-Claude Gaudel

Université de Paris-Sud,
L.R.I, Bâtiment 490,
F-91405 Orsay Cedex, France
lestienn@lri.fr, mcg@lri.fr

RÉSUMÉ. *Une hypothèse récurrente dans le domaine du test de systèmes à entrées-sorties est que ces systèmes sont réceptifs, i.e. doivent pouvoir accepter toute entrée à tout instant. Dans cet article, nous nous intéressons aux systèmes non réceptifs. Nous définissons un nouveau modèle afin de pouvoir traiter de tels systèmes, ainsi qu'une nouvelle relation de conformité : rioco. Nous donnons également un ensemble de tests, le verdict associé et les hypothèses de test utilisées. Enfin, la validité et le non biais de ce contexte vis à vis de la relation rioco sont prouvés.*

ABSTRACT. *A prevalent hypothesis in the area of testing systems with inputs and outputs is that those systems are input enabled, that is, they must accept any input in any state. In this paper, we consider non input enabled systems. We define a new kind of model to deal with such systems as well as a new conformance relation: rioco. A set of tests, its associated verdict and some test hypotheses are given. It is proven that they ensure validity and unbias with regard to rioco.*

MOTS-CLÉS : *I/O automates, conformité, test*

KEYWORDS: *I/O automata, conformity, test*

RS - JESA – 39/2005. MSR'05, pages 255 à 270

1. Introduction

Nous nous intéressons ici à la conformité dans le domaine des systèmes à entrées-sorties dont les systèmes réactifs sont un cas particulier. La relation la plus reconnue dans ce domaine est la relation $ioco$ définie par Tretmans dans [TRE 96]. L'une des conditions nécessaires de cette relation est que l'implémentation doit être réceptive, i.e. doit pouvoir accepter toute entrée dans tout état. Cependant, comme cela a déjà été mentionné par plusieurs auteurs [LYN 89, SEG 93, PET 02], cette condition n'est pas toujours remplie par les systèmes réels. Certains d'entre eux doivent refuser certaines entrées dans des situations particulières.

Une entrée peut être impossible pour deux raisons : tout d'abord, nous avons le cas où un utilisateur, ou un autre système, essaie d'effectuer une entrée correspondant à une action interdite dans l'état courant du système. Cette entrée sera ignorée, généralement en prévenant l'utilisateur, et n'aura aucun effet sur l'état du système. De telles impossibilités sont appelées *impossibilités logicielles*. Cliquer sur une option grisée dans un menu ou sauver des modifications dans un texte "en lecture seule" sont des exemples d'entrées logiciellement impossibles. Ces entrées sont physiquement acceptées par le système, mais les effets attendus ne peuvent avoir lieu car ils ne sont pas autorisés. Dans des modèles tels que les $IOLTS$ ($Input\ Output\ Labelled\ Transition\ System$) [TRE 96] ou les IOA ($Input\ Output\ Automata$) [LYN 89], cette notion d'impossibilité pourrait éventuellement être modélisée à l'aide de boucles étiquetées par les entrées considérées.

Mais les entrées d'un système peuvent également être impossibles pour des raisons $physiques$. On peut prendre l'exemple d'un distributeur automatique dont le monnayeur n'est pas utilisable tant qu'aucun article n'a été choisi, ou l'exemple d'un ATM dans lequel on ne doit pas pouvoir insérer de carte tant qu'une autre carte est à l'intérieur. Un dernier exemple d'impossibilité physique, est le blocage des touches d'un clavier en attendant que le système soit prêt à accepter des entrées. Ces aspects ne peuvent pas être spécifiés à l'aide d'$IOLTS$ et ne peuvent donc pas être testés par des méthodes basées sur la relation $ioco$.

Pourtant, si une entrée interdite est permise dans l'implémentation du système, et que rien n'empêche qu'elle soit effectuée, cela peut avoir des conséquences indésirables : dans le cas de l'ATM, la machine pourrait garder la carte insérée.

Notons que la distinction entre impossibilité logicielle et impossibilité physique est une distinction entre des choix d'implémentation. Elle disparait si on se place à un niveau d'abstraction suffisant et n'est pas toujours claire même au niveau de l'implémentation (cas des parties de menu grisées). Mais au niveau du modèle ce qui est essentiel, c'est que l'entrée est interdite.

Avec des modèles classiques tels que les LTS, qui ne distinguent pas entre actions d'entrée et actions de sortie, ces actions physiquement impossibles seraient simplement non spécifiées puisque n'étant pas sensées se produire. Par exemple, dans [PHI 87], Phillips propose une approche pour tester que les actions non spécifiées sont

impossibles. Le problème est qu'il n'y a pas dans cette approche de distinction entre actions interdites et actions non spécifiées : toute action non spécifiée est interdite.

Or en pratique il est important de différencier ces actions : on peut ainsi concentrer l'effort de test sur les actions réellement interdites et sur l'effet des actions autorisées, tout en laissant des choix d'implémentation pour ce qui est des actions non spécifiées. De plus, le fait qu'une action soit à l'initiative de l'environnement (entrée) ou du système (sortie) est également une distinction importante.

Nous proposons donc, le modèle $RIOLTS$ (pour *Restrictive Input/Output Labelled Transition System*). Ce modèle permet de spécifier aussi bien les entrées possibles qu'impossibles, en laissant les autres non spécifiées. Par rapport aux modèles existants, celui-ci combine les possibilités du test avec entrées-sorties [TRE 96] et le test de refus [PHI 87].

La suite de l'article se présente comme suit : la section 2 présente le modèle $RIOLTS$. La section 3 donne la définition de la relation de conformité $rioco$ (*Restrictive Input/Output COnformance*). La section 4 présente un ensemble exhaustif de tests $exhaustive_{rioco}(S)$ avec son verdict et ses hypothèses de test. Ces trois éléments permettent d'établir si une implémentation est conforme à sa spécification vis à vis de la relation $rioco$. La section 5 traite des différences de notre approche par rapport à des travaux similaires [HEE 98, HUO 04, PET 02, HEE 97]. La section 6 présente des remarques de conclusion.

2. Modèle

2.1. *Définitions*

Un $RIOLTS$ est un 6-uplet $(Q, q_0, \mathcal{I}, \mathcal{U}, T, T_\times)$ où Q est un ensemble fini d'états, $q_0 \in Q$ est l'état initial, $\mathcal{I}$ est l'ensemble des actions d'entrée observables, $\mathcal{U}$ est l'ensemble des actions de sortie observables, $T \subseteq Q \times (\mathcal{I} \cup \mathcal{U} \cup \{\tau\}) \times Q$ est l'ensemble des transitions spécifiées, et $T_\times \subseteq Q \times \mathcal{I}$ est l'ensemble des transitions d'entrée impossibles. De plus, $T_\times$ et T sont tels qu'il n'existe pas de tuple (q, q', i) tel que $(q, i, q') \in T$ et $(q, i) \in T_\times$. Les transitions appartenant à $T_\times$ concernent uniquement les actions d'entrée. Dans la suite, les actions d'entrée spécifiées seront également appelées actions possibles.

Nous adaptons ici, quelques notions classiques des systèmes de transitions avec entrées-sorties aux $RIOLTS$.

Une trace sur un ensemble d'actions $L{=}\mathcal{I} \cup \mathcal{U}$ avec $\mathcal{I} \cap \mathcal{U}{=}\emptyset$ est une séquence finie d'actions observables appartenant à L^*.

Un état est dit quiescent si aucune sortie ni aucune action interne ne peut être exécutée dans cet état. La quiescence sera comme toujours, modélisée par une boucle étiquetée par l'action fictive $\delta \notin \mathcal{I} \cup \mathcal{U}$, et détectée à l'aide de mécanismes de timeout.

Une Strace (Suspension trace), sur un ensemble d'actions $L{=}\mathcal{I} \cup \mathcal{U}$ est une trace sur l'ensemble $L \cup \{\delta\}$. Pour définir les Straces d'un $RIOLTS$, celui-ci est complété

par des boucles étiquetées par δ sur les états quiescents. Comme il a été montré dans [TRE 96], l'utilisation des Straces à la place des traces classiques permet de définir des relations de conformité plus discriminantes en particulier par rapport à la quiescence.

Soit une spécification S=($Q, q_0, \mathcal{I}, \mathcal{U}, T, T_\times$) :

$q \overset{a}{\not\longrightarrow}$ où $a \in L$, signifie que soit $(q, a) \in T_\times$ soit $\nexists\, q'$ tel que $(q, a, q') \in T$.

$q \overset{a}{\longrightarrow} \times$ où $a \in \mathcal{I}$, signifie que $(q, a) \in T_\times$. L'entrée a est dite $impossible$ dans l'état q.

$q \overset{a}{\longrightarrow}$ où $a \in L$, signifie qu'il existe un état q' tel que $(q, a, q') \in T$. Si $a \in \mathcal{I}$ alors a est une entrée dite $possible$.

$q \overset{\delta}{\longrightarrow}$ signifie que l'état q est quiescent i.e. $\nexists\, a \in \mathcal{U} \cup \{\tau\} \mid q \overset{a}{\longrightarrow}$.

$q \overset{\sigma}{\Longrightarrow} q'$ signifie qu'il existe une suite de transitions menant de l'état q à l'état q' et dont la Strace est σ.

La définition de l'ensemble d'états $P\ after\ \sigma$, où P est un ensemble d'états et σ est une Strace, est la suivante :

$$P\ after\ \sigma = \{q \in Q \mid \exists p \in P, p \overset{\sigma}{\Longrightarrow} q\ \}$$

Cet ensemble est l'ensemble de tous les états pouvant être atteints à partir des états de P en exécutant une suite d'actions dont la Strace est σ. Si σ ne peut être exécutée à partir d'aucun des états de P, alors l'ensemble $P\ after\ \sigma$ est vide.

Lorsque l'on utilise cette notation avec un $RIOLTS\ R$, l'ensemble $R\ after\ \sigma$ correspond à l'ensemble des états atteignables depuis l'état initial de R. Pour un $RIOLTS$ dont l'état initial est q_0, cette notation est équivalente à $\{q_0\}\ after\ \sigma$. Une trace σ est exécutable par un $RIOLTS$ uniquement si cet ensemble n'est pas vide.

2.2. *Exemple*

L'exemple suivant permet d'illustrer le modèle $RIOLTS$. Les actions impossibles sont représentées à l'aide de transitions aboutissant à une croix.

L'exemple est un distributeur dont le comportement est le suivant : avant de pouvoir insérer de l'argent, il faut choisir une boisson. Quand cela est fait, le choix peut être modifié, ou l'on peut mettre de l'argent. Une fois la boisson payée, on ne peut plus ni modifier son choix, ni payer à nouveau tant que la boisson n'a pas été servie.

3. Relation de conformité

Le possible non déterminisme de la spécification fait qu'une même trace peut mener à plusieurs états distincts dans lesquels les ensembles d'entrées possibles, impossibles et non spécifiés sont différents. Lorsque l'on observe une implémentation sous test, on ne connait d'elle que la trace observable qui est exécutée. On ne sait pas exactement dans quel état l'implémentation se trouve par rapport à la spécification. On ne peut que considérer l'ensemble des états de la spécification atteignables après cette trace pour définir quelles sont les actions autorisées. Pour savoir quelles sont

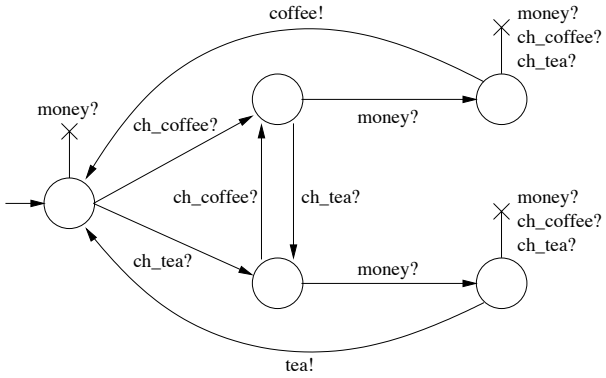

Figure 1. *RIOLTS d'un distributeur de boissons*

ces actions après une certaine trace, il nous faut donc définir ce que sont les actions possibles, impossibles et non spécifiées pour un ensemble d'états et non pour un seul état :

- Les entrées non spécifiées sont celles qui sont soit non spécifiées dans au moins un des états considérés, soit spécifiées dans un état et impossibles dans un autre.

- Un ensemble d'entrées est possible pour un ensemble d'états s'il contient, pour chaque état de cet ensemble, au moins une action possible.

- Une entrée est impossible pour un ensemble d'états si elle est impossible dans tous les états de cet ensemble.

Les actions de sortie quant à elles, sont gérées comme elles le sont habituellement [TRE 96], i.e. les sorties correctes sont celles spécifiées dans au moins un des états de l'ensemble considéré.

Nous avons également besoin de définir l'ensemble des entrées spécifiées pour un ensemble d'états. Il sera utilisé dans l'algorithme de génération de tests en section 4.5. Cet ensemble contient toutes les actions d'entrée possibles de chacun des états de l'ensemble considéré.

Formellement, ces concepts correspondent aux définitions suivantes :
Soient S=($Q, q_0, \mathcal{I}, \mathcal{U}, T, T_\times$) une spécification, p un état, et P un ensemble d'états.

Entrées non spécifiées :

$unspec(p) = \{\ a \in \mathcal{I} \mid \forall q \in S, (p,a,q) \notin T \text{ et } (p,a) \notin T_\times\ \}$

$unspec(P) = \bigcup_{p \in P} unspec(p) \cup \{a \in \mathcal{I} \mid \exists\, p \text{ et } p' \in P, p \neq p', p \xrightarrow{a}, p' \overset{a}{\not\rightarrow}\}$

Ensembles possibles d'entrées :
$poss(p) = \{A \subseteq \mathcal{I} \mid \exists\, a \in A, p \xrightarrow{a}\}$
$Poss(P) =$
$\{A \subseteq \mathcal{I} \mid \forall\, p \in P, \exists\, a \in A, p \xrightarrow{a}\}$

Sorties :
$out(p) = \{a \in \mathcal{U} \cup \{\delta\} \mid p \xrightarrow{a}\}$
$Out(P) = \bigcup_{p \in P} out(p)$

Entrées impossibles :
$impo(p) = \{A \subseteq \mathcal{I} \mid \exists\, a \in A, p \xrightarrow{a}\!\!\!\!\!\!/\ \}$
$Impo(P) = \bigcap_{p \in P} impo(p)$

Entrées spécifiées :
$spec(p) = \{\, a \in \mathcal{I} \mid p \xrightarrow{a}\}$
$Spec(P) = \bigcup_{p \in P} spec(p)$

L'ensemble $Poss(P)$ est l'ensemble de tous les ensembles d'actions d'entrée qui doivent être acceptés dans l'implémentation. Pour tout état p appartenant à P, ces ensembles contiennent au moins une action possible dans p. L'ensemble possible d'entrées est proche des $must\ sets$ utilisés pour le $test\ d'equivalence$ de De Nicola et Hennessy [NIC 84], et le $test\ de\ refus$ de Phillips [PHI 87] et dans la thèse de Tretmans [TRE 92]. La différence est qu'ici nous distinguons les actions d'entrée des actions de sortie.

L'ensemble $Poss(P)$ peut être réduit en supprimant les ensembles incluant un autre ensemble de $Poss(P)$, puisque si un ensemble d'entrées A est possible pour un ensemble d'états P, alors tout ensemble contenant A est également un ensemble possible pour P. Il suffit donc de vérifier que les plus 'petits' ensembles de $Poss(P)$ sont possibles dans l'implémentation. On peut réduire l'ensemble $Poss(P)$ en utilisant la fonction suivante :
Soient P un ensemble d'états, et $\mathcal{P}$ un ensemble d'ensembles d'états, on a :

$$min_{\subseteq}(\mathcal{P}) = \{P \in \mathcal{P} \mid \forall\, P' \in \mathcal{P}\ ,\ P' \not\subset P\}$$

L'ensemble minimal des ensembles possibles d'entrées peut alors être défini par :

$$Poss_{min}(P) = min_{\subseteq}(Poss(P))$$

Ainsi, l'ensemble $Poss_{min}(Poss(P))$ ne contient que les ensembles possibles d'entrées les plus restrictifs de l'ensemble $Poss(P)$.

Remarque 1 : l'ensemble résultant de la fonction $Poss$ appliquée à un ensemble d'états, ne peut contenir l'ensemble vide que si l'ensemble passé en argument est vide : supposons que P ne soit pas l'ensemble vide. D'après la définition de $Poss(P)$, si l'ensemble vide appartenait à l'ensemble résultat $Poss_{min}(P)$, alors il contiendrait, pour chaque état de P, au moins une action possible. Or c'est impossible. Supposons maintenant que P soit l'ensemble vide. Dans ce cas tout ensemble d'actions appartient à l'ensemble $Poss_{min}(P)$, y compris l'ensemble vide.

Remarque 2 : si une action a est possible dans tous les états d'un ensemble P, alors le singleton $\{a\}$ appartient à $Poss_{min}(P)$, et aucun autre ensemble de $Poss_{min}(P)$ ne contient a.

Remarque 3 : les implémentations sont modélisées par des $RIOLTS$ d'un genre particulier pour lesquels dans tout état, toute action est implémentée ou non. Pour tout état p, $unspec(p)$ est donc vide.

Donnons maintenant la définition de la relation de conformité $rioco$ prenant en compte à la fois l'aspect entrées et l'aspect sorties d'un système :

Définition

Soit I un $RIOLTS$ complètement spécifié et S un $RIOLTS$. I et S ont les mêmes ensembles d'actions. On a I $rioco$ S si et seulement si pour toute Strace de S, σ, exécutable par I, on a :

- $Out(I\ after\ \sigma) \subseteq Out(S\ after\ \sigma)$
- $Impo(S\ after\ \sigma) \subseteq Impo(I\ after\ \sigma)$
- $\forall\ A \in Poss_{min}(S\ after\ \sigma)$, $\exists\ B \in Poss_{min}(I\ after\ \sigma)$ tel que $B \subseteq A$

La partie de la définition concernant les sorties correspond à ce qui est fait dans la relation $ioco$: l'implémentation ne doit pas produire de sortie non spécifiée, et elle ne peut être quiescente que si la spécification peut l'être.

Le second point de la définition traite des entrées impossibles. Toute entrée impossible dans la spécification doit être impossible dans l'implémentation. L'ensemble des entrées impossibles de l'implémentation après une trace σ peut être plus grand que celui de la spécification car certaines entrées non spécifiées ont pu être implémentées comme impossibles.

Enfin le dernier point de la définition traite des ensembles possibles d'entrées. Selon le contexte de test, les ensembles $Poss_{min}(I\ after\ \sigma)$ ne peuvent pas toujours être déterminés par observation. Nous revenons sur ce point au début de la section 4.

L'inclusion d'ensembles ne peut pas être utilisée pour comparer les ensembles $Poss_{min}(S\ after\ \sigma)$ et $Poss_{min}(I\ after\ \sigma)$: comme les entrées non spécifiées peuvent être implémentées comme étant possibles, l'ensemble $Poss_{min}(I\ after\ \sigma)$ d'une implémentation peut, tout en étant conforme, contenir des ensembles plus restrictifs que $Poss_{min}(S\ after\ \sigma)$. Par exemple, si l'ensemble minimal des ensembles possibles d'une spécification après une trace σ est {{a,b},{c},{d,b}}, alors une implémentation dont l'ensemble minimal correspondant est {{b},{c}} peut accepter une action appartenant à n'importe lequel des ensembles de $Poss_{min}(S\ after\ \sigma)$. En effet, si elle accepte une action de l'ensemble $\{b\}$, alors elle peut obligatoirement accepter une action d'un ensemble plus grand contenant b.

Un ensemble $Poss_{min}(A)$ est non vide si et seulement aucun état de A n'est quiescent. Si ce n'est pas le cas, il n'existe pas d'ensemble possible d'entrées puisque les états non quiescents n'acceptent aucune entrée. Le troisième point de la relation $rioco$ est alors trivialement vrai.

Ce qui est requis par la relation de conformité à propos des ensembles possibles d'actions est que tout ensemble appartenant à $Poss_{min}(S\ after\ \sigma)$ est possible dans I après la trace σ : pour tout A appartenant à $Poss_{min}(S\ after\ \sigma)$, soit A appartient à $Poss_{min}(I\ after\ \sigma)$, soit il existe un ensemble strictement inclus dans A qui appartient à $Poss_{min}(I\ after\ \sigma)$.

Les ensembles $S\ after\ \sigma$ et $I\ after\ \sigma$ considérés ici ne sont jamais vides puisque les Straces σ considérées sont exécutables par I et par S. Donc d'après la remarque 1, les ensembles d'actions $Poss_{min}(S\ after\ \sigma)$ et $Poss_{min}(I\ after\ \sigma)$ ne contiennent pas l'ensemble vide. Par contre, ces ensembles eux-mêmes peuvent être vides si aucun ensemble possible n'existe, i.e. s'il existe un état non quiescent dans l'ensemble d'états considéré. Le fait que l'ensemble vide ne puisse pas appartenir aux ensembles $S\ after\ \sigma$ pour une trace σ de S, est utilisé dans la preuve donnée dans la version longue de l'article (http ://www.lri.fr/~lestienn/rioco.ps).

4. Hypothèses de test et ensemble exhaustif de test

Dans cette section, nous définissons la forme des tests, la façon dont ils sont soumis et les verdicts correspondants. Une exécution de test est la mise en parallèle d'un test (ou testeur) avec l'implémentation. Les verdicts associés à un test définissent quelles sont les exécutions qui sont des succès, des échecs ou qui sont inconclusives. Ce dernier verdict est utilisé pour gérer les implémentations non déterministes et l'occurence de sorties non attendues pour un test précis, mais malgré tout correctes.

Les deux notations suivantes servent à décrire le test d'une action a :

– $\tilde{a}$ est le test du refus de a. Tout comme dans [PHI 87], nous supposons qu'il est possible de détecter le refus (aussi bien physique que logiciel) des actions. Cette notation est étendue à un ensemble d'actions A : $\tilde{A} = \{\tilde{a} \mid a \in A\}$.

– $\overline{a}$, le miroir de a, est le test de l'acceptation de a. Si a est une sortie alors $\overline{a}$ est l'entrée correspondante et vice versa. Cette notation est étendue à une trace σ pour laquelle $\overline{\sigma}$ est le miroir de la trace σ.
Dans la suite, les actions de sortie (resp. d'entrée) feront référence aux sorties (resp. entrées) du point de vue de l'implémentation.

Nous considérons que le rôle d'un processus de test est d'observer les comportements du système sous test sans les influencer. Aussi, nous pensons qu'il doit accepter toutes les sorties que le système produit. C'est pourquoi nos tests sont réceptifs et sont modélisés par des $IOLTS$. L'ensemble des actions d'entrée de ces tests est $\mathcal{I}' = \mathcal{U} \cup \{\delta\}$ et l'ensemble des actions de sortie est $\mathcal{U}' = \mathcal{I} \cup \tilde{\mathcal{I}}$.

L'opérateur $\rceil\!|\!|$ utilisé pour la composition parallèle synchronisée d'un $IOLTS$ et d'un $RIOLTS$ est l'opérateur classique de composition parallèle synchronisée auquel on rajoute les règles suivantes.

Définition

$$t \xrightarrow{\delta} t', \forall a \in \mathcal{I} \cup \mathcal{U} : t \overset{a}{\not\longrightarrow} \ ou\ i \overset{a}{\not\longrightarrow} \vdash \ t\rceil\!|\!|i \xrightarrow{\delta} t'\rceil\!|\!|i$$

$$\forall a \in \mathcal{I} : t \xrightarrow{\tilde{a}} t'\ et\ i \overset{a}{\not\longrightarrow} \vdash \ t\rceil\!|\!|i \xrightarrow{\tilde{a}} t'\rceil\!|\!|i$$

La première règle concerne la détection de la quiescence : lorsqu'un test t vérifie la quiescence et que l'implémentation i ne produit aucune sortie, alors le test progresse tandis que l'état de l'implémentation reste inchangé. La seconde règle concerne la

détection des actions refusées : lorsqu'un test vérifie le refus d'une entrée, et que l'implémentation refuse cette entrée, alors le test progresse et l'état de l'implémentation reste inchangé.

Comme les tests sont des $IOLTS$, on ne va pas pouvoir observer certains ensembles $Poss_{min}(I\ after\ \sigma)$ pour tous les systèmes car les sorties produites par l'implémentation peuvent interférer. En effet, un processus de test envoyant à l'implémentation une entrée pour vérifier qu'elle est bien acceptée, ne peut différencier deux états pouvant tout deux produire une sortie, mais l'un refusant l'entrée proposée et l'autre pouvant l'accepter : Quand une sortie est produite, on ne sait pas si l'implémentation pouvait accepter l'entrée ou non. On ne peut donc pas toujours établir qu'un ensemble d'entrées est ou non possible après une trace. Le fait de ne pas pouvoir déterminer certains ensembles $Poss_{min}(I\ after\ \sigma)$ fait que le troisième point de la relation $rioco$ portant sur les ensembles possibles ne peut être vérifié.

Pour cette raison, nous nous limitons au test basé sur des spécifications dîtes $IO-exclusives$ i.e. dont aucun état ne donne le choix entre actions d'entrée et actions de sortie. Les états sont donc divisés en deux classes : les états quiescents d'une part et les états pour lesquels ne sont spécifiées que des sorties d'autre part.

Cette restriction sur la spécification n'est pas aussi forte qu'elle paraît. Par exemple la classe de systèmes spécifiables est plus importante que dans le cas des FSM où une paire d'entrée-sortie constitue une action atomique : dans le modèle correspondant, le seul moyen de changer d'état est par des entrées.

Cependant, cette restriction exclut les systèmes contenant des boucles composées uniquement d'actions de sortie [1] pouvant être interrompues par une entrée à tout moment (par exemple : une horloge avec un reset, ou un économiseur d'écran). En effet dans ce cas, entrées et sorties peuvent être exécutées dans un même état. De tels systèmes réactifs sont d'une certaine façon "plus que réceptifs" pour certaines entrées : nous appellerons ces entrées *entrées urgentes*. Elles ne sont actuellement pas prises en compte dans notre modèle, ni dans les $IOLTS$. Ce sera l'objet d'un travail ultérieur.

Définition

Soit un $RIOLTS\ S = (Q, q_0, \mathcal{I}, \mathcal{U}, T, T_\times)$, S est $IO-exclusif$ ssi :
$$\nexists q \in Q \mid a \in \mathcal{I}, b \in \mathcal{U}, q \xrightarrow{a}, q \xrightarrow{b}$$

Dans un $RIOLTS\ IO-exclusif$, tout état non quiescent ne peut accepter aucune entrée. Notons qu'à cause du non déterminisme, il se peut qu'une même trace σ mène aux deux différents types d'états : l'ensemble d'états $S\ after\ \sigma$ peut contenir des états quiescents et des états pour lesquels aucune entrée n'est spécifiée. Dans ce cas, l'ensemble $Poss(S\ after\ \sigma)$ est vide par définition.

Dans la suite de cette section, étant donnés la relation $rioco$ et le $RIOLTS\ S$ d'une spécification, et conformément à l'approche suivie dans [BER 91], nous définissons :

– un ensemble exhaustif de tests $exhaustif_{rioco}(S)$

1. Ce cas est également exclus dans [HUO 04]

– un verdict associé

– des hypothèses de test sur l'implémentation I

Si les hypothèses de test sont vérifiées, alors le fait que le verdict d'aucune execution de test ne soit *échec* est équivalent à $I\ rioco\ S$.

L'ensemble exhaustif de tests est souvent beaucoup trop grand pour pouvoir être utilisé dans son intégralité. C'est une base à partir de laquelle des stratégies de sélection peuvent être appliquées : celles-ci peuvent être modélisées comme une sélection d'un sous ensemble de l'ensemble exhaustif de tests et un renforcement des hypothèses de test. La sélection est habituellement faite en appliquant des hypothèses de régularité sur la longueur des tests et des hypothèses d'uniformité [GAU 99, LES 02], ou à l'aide d'objectifs de test [FER 97]. De même, à partir de cette sélection, on peut éliminer des redondances par factorisation des tests de préfixe commun.

Dans la suite de cette section, nous définissons $exhaustif_{rioco}$ en trois étapes correspondant aux trois parties de la définition de la relation $rioco$ (cf section 3). Nous présentons d'abord les tests et les verdict associés pour vérifier les sorties, les entrées possibles et les entrées impossibles. Comme dans [TRE 96], ces tests sont indépendants et partent de l'état initial de la spécification. Nous supposons donc que le système peut être réinitialisé correctement. Ceci fait partie des hypothèses de test résumées en section 4.4. Nous donnons ensuite un algorithme de génération de tests.

4.1. *Test des sorties après une trace* σ

Les tests utilisés pour le test des actions de sortie sont arborescents. Chaque test est associé à une trace σ de la spécification. Il consiste en un arbre linéaire correspondant à $\overline{\sigma}$ complété par toutes les actions de sortie sur chaque noeud, et complété, pour toute action a, par $\overline{a}$ sur les noeuds où $\overline{a}$ est présent. Le test associé à une trace σ dans le cas du test de sorties est noté T_σ et est réutilisé par la suite.

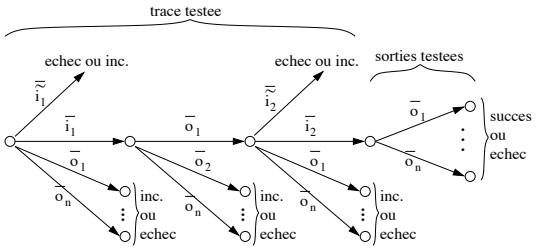

Figure 2. *Schéma d'un test d'un ensemble* $exhaustif_{rioco-o}$ *pour une trace* σ=i_1, o_1, i_2

Soit $exhaustif_{rioco-o}(S)$ l'ensemble de tous les tests utilisés pour le test de sorties pour une spécification S :

$$exhaustif_{rioco-o}(S) = \{T_\sigma \mid \sigma \in Straces(S)\}$$

La figure 2 donne le schéma d'un test de l'ensemble $exhaustif_{rioco-o}$ d'une spécification pour la trace σ=i_1, o_1, i_2. Les actions i_n (resp. o_n) correspondent aux actions d'entrée (resp. actions de sortie). Un test comporte deux parties : une première qui exécute la trace σ considérée et une seconde qui vérifie la sortie produite.

Soit σ_1, préfixe strict de la trace σ considérée. Le verdict d'une exécution d'un test est :

– $inconclusif$ si, après σ_1, l'implémentation produit une sortie appartenant à $Out(S\ after\ \sigma_1)$, mais différente de l'action suivant directement σ_1 dans σ. Ceci peut se produire en raison du non déterminisme possible de l'implémentation.

– *échec* si, après σ_1, l'implémentation produit une sortie n'appartenant pas à $Out(S\ after\ \sigma_1)$.

– $inconclusif$ si, après σ_1, l'implémentation refuse l'entrée a suivant directement σ_1 dans σ, et si l'ensemble $\{a\}$ n'appartient pas à $Poss_{min}(S\ after\ \sigma_1)$.

– *échec* si, après σ_1, l'implémentation refuse l'entrée a suivant σ_1 dans σ, et si l'ensemble $\{a\}$ appartient à $Poss_{min}(S\ after\ \sigma_1)$.
En fin de test, lorsque l'on vérifie la sortie produite, le verdict est :

– *succès* si la sortie effectuée appartient à $Out(S\ after\ \sigma)$,

– *échec* sinon.

Remarque : en factorisant ces tests et en supprimant les verdicts inconclusifs, on obtient des tests qui permettent d'adapter l'exécution du test à la réaction du système.

4.2. *Test des ensembles possibles d'entrées après une trace σ*

Ces tests ont la même base que ceux de $exhaustif_{rioco-o}$, c'est à dire T_σ. Pour chaque ensemble A de $Poss_{min}(S\ after\ \sigma)$, des transitions sont ajoutées pour vérifier qu'au moins l'une des actions de A est acceptée. Ainsi, à chaque fois qu'une entrée est refusée, une autre est proposée jusqu'à ce que toutes les actions de A ait été essayées. Notons C_A ce processus de vérification pour un ensemble d'entrées A. La figure 3 représente un test d'ensemble d'entrées.

Soit S une spécifcation, l'ensemble $exhaustif_{rioco-p}(S)$ est défini par :

$$exhaustif_{rioco-p}(S) = \{T_\sigma\ ;C_A \mid \sigma \in Straces(S),\ A \in Poss_{min}(S\ after\ \sigma)\}$$

Le symbole ' ;' utilisé entre T_σ et C_A correspond à la concaténation de la dernière action de σ et de C_A.

Le verdict concernant la partie du test dédiée au parcours de la trace σ est le même que pour l'ensemble $exhaustif_{rioco-o}$. La partie destinée à vérifier les actions de l'ensemble d'entrées A a pour verdict :

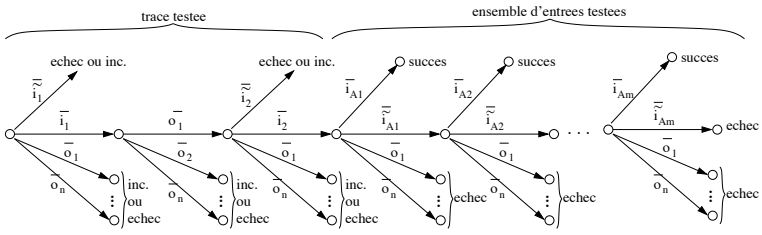

Figure 3. *Schéma d'un test d'un ensemble* $exhaustif_{rioco-p}$ *pour une trace* σ=i_1, o_1, i_2 *et un ensemble d'entrées testé* $A = \{i_{A1}, ..., i_{Am}\}$

– *échec* si I effectue une action de sortie,

– *succès* si l'une des actions d'entrée proposées est acceptée,

– *échec* si aucune des actions d'entrée proposées n'est acceptée.

4.3. *Test des entrées impossibles après une trace* σ

Les tests utilisés dans la vérification des entrées impossibles sont également basés sur T_σ. Pour chaque action de l'ensemble $Impo(S\ after\ \sigma)$, T_σ est complété par deux transitions afin de tester l'entrée considérée.

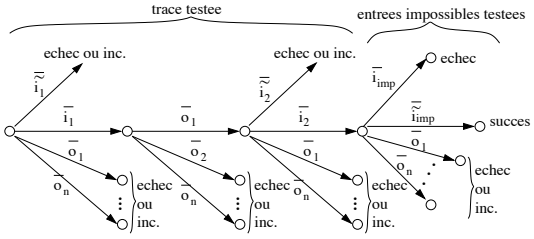

Figure 4. *Schéma d'un test d'un ensemble* $exhaustif_{rioco-i}$ *pour une trace* σ=i_1, o_1, i_2 *et une entrée impossible testée* i_{imp}

Soit S une spécification, l'ensemble $exhaustif_{rioco-i}(S)$ est défini par :

$exhaustif_{rioco-i}(S) = \{T_\sigma\ ;\ (\overline{a}\ [\ ]\ \widetilde{a}) |\ \sigma \in Straces(S), a \in Impo(S\ after\ \sigma)\}$[2]

Le verdict de la première partie est encore identique à celui des tests de l'ensemble $exhaustif_{rioco-o}$. Le verdict du test d'une entrée impossible en fin de test est :

– *échec* si I effectue une action de sortie n'appartenant pas à $Out(S\ after\ \sigma)$,

– $inconclusif$ si I effectue une action de sortie appartenant à $Out(S\ after\ \sigma)$,

2. Le symbole [ ] dénote ici l'opération de choix.

– *succès* si l'entrée est refusée,

– *échec* si l'entrée est acceptée.

Nous définissons l'ensemble $exhaustif_{rioco}(S)$ d'une spécification S comme l'union des trois ensembles définis précédemment :

$$exhaustif_{rioco}(S) = exhaustif_{rioco-o}(S) \cup exhaustif_{rioco-p}(S) \cup exhaustif_{rioco-i}(S)$$

Comme précisé dans la définition de la relation $rioco$, les traces de la spécification ne pouvant être exécutées par l'implémentation ne sont pas considérées, cette dernière pouvant être plus déterministe que la spécification. L'ensemble de tests $exhaustif_{rioco}$ est cependant basé sur toutes les traces de la spécification puisqu'il n'est pas possible de savoir quelles sont les traces pouvant être exécutées par l'implémentation avant leurs soumissions. Ce problème est traité par le verdict associé aux tests.

Notons que $exhaustif_{rioco}$ est une construction théorique qui a pour but de servir de base par sélections et factorisations à la construction de jeux de test utilisables.

4.4. *Hypothèses de test*

L'ensemble exhaustif de tests pour la relation $rioco$ est basée sur les hypothèses suivantes :

1) Le refus d'une entrée n'a pas d'effet sur l'état du système sous test.

2) Pour pallier au non déterminisme possible de l'implémentation, nous faisons l'hypothèse qu'après un nombre suffisant d'exécutions du même test, tous les chemins correspondant à ce test ont été empruntés. Cette hypothèse classique est connue sous le nom d'*hypothèse de test complet*. Elle assure que l'implémentation ne peut avoir d'autre comportement que ceux observés pendant la phase de test. Chaque test est donc exécuté plusieurs fois, et un test est passé avec succès si et seulement si le verdict de chacune de ces exécutions est *succès* ou $inconclusif$.

3) Afin de pouvoir parler des traces d'une implémentation, nous devons supposer que les actions sont atomiques ou au moins observables comme si elles étaient atomiques. De plus, elles sont supposées suivre le modèle d'entrelacement du parallélisme.

4) Les tests étant indépendants les uns des autres, il nous faut faire l'hypothèse que le système peut être réinitialisé correctement avant chaque exécution de test.

5) Les implémentations doivent être fortement convergentes, c'est à dire qu'elles ne peuvent exécuter de séquence d'actions internes infinie. Cette hypothèse nous permet de pouvoir détecter les états $quiescents$ du système.

A part la première hypothèse, ces hypothèses sur les implémentations testées sont classiques dans le domaine du test basé sur des modèles, quoique pas toujours explicitées. La preuve de l'exhaustivité de $exhaustif_{rioco}(S)$ sous ces hypothèses est donnée dans la version longue du papier (http ://www.lri.fr/~lestienn/rioco.ps).

4.5. *Algorithme de génération de tests*

Dans cette section, nous proposons dans la lignée de [TRE 96] un algorithme non déterministe générant des tests appartenant à l'ensemble $exhaustif_{rioco}$.

Soient S une spécification d'état initial s_0, Set un ensemble d'états non vide initialisé à $\{s_0\}$. Un test t_{Set} est obtenu par un nombre fini d'applications récursives des règles suivantes :

1. Terminer le test par une action de sortie :

$t_{Set} := \underset{a \in Out(Set)}{[\,]}(\overline{a}; succès) \quad \underset{a \in (\mathcal{U} \cup \{\delta\}) \setminus Out(Set)}{[\,]}(\overline{a}; échec)$

2. Terminer le test par une action d'entrée impossible :

$t_{Set} := (\overline{a}_2; succès)\ [\,]\ (\overline{a}_2; échec) \quad \underset{a \in Out(Set)}{[\,]}(\overline{a}; inconclusif) \quad \underset{a \in \mathcal{U} \setminus Out(Set)}{[\,]}(\overline{a}; échec)$

où $a_2 \in Impo(Set)$

3. Terminer le test avec un ensemble possible d'actions d'entrée :

$t_{Set} := C_A\ \underset{a \in \mathcal{U}}{[\,]}(\overline{a}; échec)$ où $A \in Poss_{min}(Set)$

et C_A est obtenu en utilisant l'algorithme $GenC_A$ avec l'ensemble d'actions d'entrée A

4. Poursuivre le test avec une action de sortie :

$t_{Set} := (\overline{o}; t_{Set'}) \quad \underset{a \in Out(Set) \setminus \{o\}}{[\,]}(\overline{a}; inconclusif) \quad \underset{a \in (\mathcal{U} \cup \{\delta\}) \setminus Out(Set)}{[\,]}(\overline{a}; échec)$

où $o \in Out(Set)$ et $t_{Set'}$ est obtenu récursivement en appliquant l'algorithme avec $Set' = Set\ after\ o$

5. Poursuivre le test avec une action d'entrée :

$t_{Set} := (\overline{a}_2; t_{Set'})\ [\,]\ (\overline{a}_2; \text{verdict}) \quad \underset{a \in Out(Set)}{[\,]}(\overline{a}; inconclusif) \quad \underset{a \in \mathcal{U} \setminus Out(Set)}{[\,]}(\overline{a}; échec)$

où $a_2 \in Spec(Set)$, $t_{Set'}$ est obtenu récursivement en appliquant l'algorithme avec $Set' = Set\ after\ a_2$, et si $\{a_2\} \notin Poss_{min}(Set)$ alors verdict=*succès* sinon verdict=$inconclusif$

La fin C_A d'un test destiné à vérifier un ensemble possible d'entrées A est obtenu en utilisant l'algorithme $GenC_A$ suivant :

- si A=$\emptyset$ alors C_A :=*échec*
- sinon $C_A := \underset{a \in \mathcal{U}}{[\,]}(\overline{a}; échec)\ [\,]\ (\overline{a}_2; succès)\ [\,]\ (\overline{a}_2; C_{A'})$ où $a_2 \in A$ et $C_{A'}$ est obtenu en appliquant $GenC_A$ à l'ensemble A'=$A \setminus \{a_2\}$

5. Travaux similaires

Une autre variante des $IOLTS$ a été proposée par Heerink et Tretmans dans [HEE 98, HEE 97]. Le modèle $MIOTS$ sert à modéliser des systèmes ayant des interfaces distribuées dont les canaux de communication peuvent être désactivés. Dans ce modèle l'ensemble des actions d'entrée et l'ensemble des actions de sortie sont tout deux partitionnés, et dans tout état d'un $MIOTS$, soit toutes les actions d'un sous-ensemble sont possibles, soit aucune ne l'est. Le partitionnement est généralement obtenu par rapport aux canaux de communication du système.

Par rapport au modèle $MIOTS$, l'ensemble des actions dans le modèle $RIOLTS$ n'est pas partitionné, et chaque action est indépendante. On peut obtenir un résultat similaire avec les $MIOTS$ en utilisant le partitionnement le plus fin, i.e. celui pour lequel il n'y a qu'une action par sous-ensemble.

Mais il y a deux différences majeures entre l'approche $MIOTS/mioco$ et la nôtre. Tout d'abord, il y a trois classes d'entrées dans le modèle $RIOLTS$(possibles, impossibles et non spécifiées) alors qu'il n'y en a que deux dans le modèle $MIOTS$ (spécifiés et non spécifiées). Dans notre approche, il n'est pas nécessaire de compléter la spécification. Les entrées non spécifiées peuvent librement être implémentées comme possibles ou impossibles. De plus, notre relation de conformité $rioco$ impose que les entrées impossibles soient effectivement implémentées comme telles. La relation $mioco$ n'exige pas que les ensembles de refus soient refusés par l'implémentation : si un ensemble d'entrées est refusé par la spécification, alors l'implémentation est autorisée par $mioco$ à l'accepter ou à le refuser.

Une autre approche pour modéliser le test d'$IOLTS$ a été proposé par Huo et Petrenko [HUO 04]. Les systèmes pouvant bloquer des actions d'entrée sont modélisés par des $IOLTS$ avec des files d'attente FIFO en entrée et en sortie servant d'interfaces. On peut ainsi spécifier des systèmes pour lesquels dans certains états la file en entrée n'est pas lue, ainsi toute entrée est bloquée. Dès qu'une entrée est spécifiée dans un état, la file d'attente peut être lue, et donc toute entrée est acceptée dans cet état. Certaines sont spécifiées, les autres non, tout comme pour les $RIOLTS$. Cela conduit à une notion d'entrée non spécifiée similaire à la nôtre. Les $RIOLTS$ sont différents dans le sens où ils ne supposent pas de communication bufferisée avec l'environnement. Certains problèmes d'observabilité mentionnés dans [HUO 04], comme l'entrelacement d'entrées et de sorties, sont ainsi écartés dans les $RIOLTS$. Une dernière différence est que dans [HUO 04] les systèmes considérés doivent être $input-progressifs$ i.e. sans boucle d'actions de sortie. Pour notre part, nous acceptons de tels systèmes, mais introduisons une autre restriction : les systèmes doivent être $IO-exclusifs$.

6. Conclusion

Dans cet article, nous avons proposé un nouveau modèle pour spécifier les systèmes réactifs non réceptifs. Nous avons défini une relation de conformité basée sur ce modèle. Cette relation a été appelée $rioco$ et prend en compte les entrées interdites, spécifiées, non spécifiées, ainsi que les sorties. Nous avons développé une méthode de test pour cette nouvelle relation, à savoir :

- des hypothèses de test
- un ensemble exhaustif de tests
- un verdict associé
- un algorithme de génération de tests

Nous avons prouvé qu'à condition que les hypothèses de test soient vérifiées et que la spécification soit $IO-exclusive$, le non échec de tous les tests de $exhaustive_{rioco}$ est équivalent à la conformité de l'implémentation testée vis à vis de la relation $rioco$.

Cette approche est radicalement différente de celle de $mioco$ puisque les entrées interdites doivent être implémentées comme telles et qu'il y a une distinction entre entrée interdite, entrée spécifiée et entrée non spécifiée. Nous avons également vu que l'approche diffère de celle qui consiste à tester les $IOLTS$ par le biais de file d'attente FIFO comme cela est fait dans [HUO 04]. Nous projetons d'étudier comment affaiblir la restriction sur les spécifications $IO - exclusives$ et comment introduire la notion d'*entrée urgente* (cf section 3). En fait, ces différentes approches considèrent toutes des classes de systèmes réactifs qui diffèrent selon la manière dont les entrées-sorties sont prises en compte. Ceci plaide pour une classification précise de ces systèmes et des méthodes de test correspondantes.

7. Bibliographie

[BER 91] BERNOT G., GAUDEL M.-C., MARRE B., « Software testing based on formal specifications : a theory and a tool », *Software Engineering Journal*, 1991, p. 387-405.

[FER 97] FERNANDEZ J.-C., JARD C., JÉRON T., VIHO C., « An experiment in automatic generation of test suites for protocols with verification technology », *Science of Computer Programming*, vol. 29 (1-2), 1997, p. 123-146.

[GAU 99] GAUDEL M.-C., JAMES P., « Testing Algebraic Data Types and Processes : a unifying theory », *Formal Aspects of Computing, 10(5-6)*, 1999, p. 436-451.

[HEE 97] HEERINK L., TRETMANS J., « Refusal Testing for Classes of Transition Systems with Inputs and Outputs. », *FORTE*, 1997, p. 23-38.

[HEE 98] HEERINK L., « Ins and Outs in Refusal Testing », PhD thesis, University of Twente, Enschede, The Netherlands, 1998.

[HUO 04] HUO J. L., PETRENKO A., « On Testing Partially Specified IOTS through Lossless Queues. », *TestCom*, 2004, p. 76-94.

[LES 02] LESTIENNES G., GAUDEL M.-C., « Testing Processes from Formal Specifications with Inputs, Outputs and Data Types. », *ISSRE*, 2002, p. 3-14.

[LYN 89] LYNCH N., TUTTLE M., « An Introduction to Input/Output automata », *CWI-Quarterly*, vol. 2, n° 3, 1989, p. 219–246.

[NIC 84] NICOLA D., HENNESSY M., « Testing equivalences for processes », *Theoretical Computer Science*, vol. 34, 1984, p. 83-133.

[PET 02] PETRENKO A., YEVTUSHENKO N., « Queued Testing of Transition Systems with Inputs and Outputs », *Proceedings of the workshop on Formal Approaches to Testing of Software (FATES'02)*, 2002, p. 79-93.

[PHI 87] PHILLIPS I., « Refusal testing », *Theoretical Computer Science*, vol. 50(2), 1987, p. 241-284.

[SEG 93] SEGALA R., « Quiescence, Fairness, Testing and the notion of Implementation (extended abstract) », *LNCS*, vol. 715, CONCUR '93, Hildesheim, Germany, 1993.

[TRE 92] TRETMANS J., « A Formal Approach to Conformance Testing », PhD thesis, University of Twente, Enschede, The Netherlands, 1992.

[TRE 96] TRETMANS J., « Test Generation with Inputs, Outputs and Repetitive Quiescence », MARGARIA T., STEFFEN B., Eds., *LNCS*, vol. 1055, TACAS'96, 1996, p. 127-146.