



# Optimal Tester Synthesis to Reduce Test Lengths for Real-Time Systems

Rachid Bouaziz  
IRIT - CNRS - Université Paul Sabatier III  
118 Route de Narbonne  
F-31062 TOULOUSE CEDEX 9, France  
bouaziz@irit.fr

Ismail Berrada<sup>\*</sup>  
Attention : L3I Université de La Rochelle  
Attention : Pôle Science et Technologie  
Attention : 17042 La Rochelle cedex 1 France  
iberrada@univ-lr.fr

## ABSTRACT

This paper shows that an optimal selection of the initial state and the input valuation of clocks can significantly reduce test lengths. An efficient method to perform this optimization is presented. Two examples are presented to illustrate our method.

## Categories and Subject Descriptors

D.2.4 [Software Engineering]: Software/Program Verification—*Formal methods, Validation*; D.2.5 [Software Engineering]: Testing and Debugging—*Error handling and recovery, Symbolic execution*

## General Terms

Verification, Reliability

## Keywords

conformance testing, timed systems, optimization, symbolic analysis and control

## 1. INTRODUCTION

L'évolution technologique a conduit au développement de systèmes informatiques complexes, dont l'impact socio-économique est devenu très fort, dans la mesure où ils occupent des places de plus en plus stratégiques au sein des organisations. De tels systèmes intègrent de nombreux composants logiciels et matériels et interagissent avec des environnements complexes. Ces systèmes sont devenus critiques tant par les conséquences de leur utilisation que par la complexité de leur développement et de leur évolution. Une classe importante des systèmes critiques est celle des systèmes réactifs, systèmes interagissant de façon continue avec un environnement.

<sup>\*</sup>This research has been supported by the european Marie Curie RTN TAROT project (MCRFN 505121).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

NOTERE 2008 June 23-27, 2008, Lyon, France

Copyright 2008 ACM 978-1-59593-937-1/08/0003 ...\$5.00.

Les méthodes formelles fournissent un cadre mathématique permettant de décrire de manière précise et stricte les systèmes et les programmes conçus. Elles ont prouvé leur efficacité dans la validation des systèmes. Parmi les méthodes formelles, nous pouvons citer : la vérification, la preuve, et la génération de cas de test. La preuve et la vérification ne visent pas à certifier n'importe quel système ou programme. Elles s'appliquent à des modèles et non à des systèmes réels. Les techniques de génération des séquences de test, quant à elles, s'appliquent au système réel. Elles consistent à extraire, à partir d'une spécification formelle du système sous test (IUT) et un critère de sélection, un ensemble de "scénarios" à appliquer sur le système réel en vue de sa validation [21, 5].

## Problématique.

Dans cet article, nous nous intéressons au test de conformité des systèmes temps réel. La particularité principale de ces systèmes vient du fait que la correction du système ne dépend pas exclusivement des occurrences des événements (actions) mais aussi des instants d'occurrences de ces derniers. La nature dense du temps physique implique que les systèmes temps réel soient souvent représentés par des modèles dont la sémantique est infinie. En conséquence, et en vue du test, des abstractions et des critères de sélection sont exigés afin de générer un nombre raisonnable et applicable de cas de test.

Le contexte actuel du test de conformité des systèmes temps réel, offre une diversité tant au niveau du modèle de base pour la description des systèmes temps réel, qu'au niveau des techniques de dérivation. De notre point de vue, le test de conformité des systèmes temps réel a atteint une maturité au niveau des techniques de dérivation, et que l'effort aujourd'hui doit porter sur l'optimisation de la génération des cas de test couplée à la prise en compte des données dans le modèle du système (ce dernier point n'est pas abordé dans cet article).

## Contributions.

Dans l'optique d'optimiser la génération de cas de test, nous proposons un cadre d'optimisation basé sur l'identification et la couverture des comportements identifiés comme critiques dans l'IUT. Notre cadre considère les éléments suivants :

- Identification des comportements critiques de l'IUT. Cette identification est réalisée à travers des observateurs modélisant les informations quantitatives et

qualitatives des comportements critiques.

- Guidage du système vers l'exécution des comportements critiques. Dans un premier temps, nous identifions les configurations qui peuvent amener le système à exécuter des comportements non critiques. Ensuite, par la modification des contraintes temporelles inspirée de la théorie de la commande par supervision des systèmes à événements discrets temporisés, nous forçons l'exécution du système vers les configurations critiques.
- Concrétisation de cas de test. Elle est réalisée en propageant les configurations temporelles suspectes le long d'une trajectoire dans l'automate d'accessibilité. Cette concrétisation peut être faite d'une façon dynamique lors de l'exécution du cas de test ou d'une façon statique dans le cas où les actions de sortie sont urgentes.

## Travaux similaires.

Le principe de sélection de tests par des propriétés, exprimées sous forme d'observateurs permettant de filtrer les tests les plus pertinents, ou par la définition de critères de couverture a été appliqué avec succès aux systèmes non temporisés [11].

Ces techniques ont été appliquées aussi au test des systèmes temps réel. En-Nouaary et al [9] étendent la méthode WP pour le test des TIOA en utilisant l'automate de grilles construit à partir du graphe des régions [1]. Cardell-Oliver [6] utilise les automates temporisés d'Uppaall et la sélection par vue (observateur) pour réduire le nombre de tests générés. Khoumsi [13] transforme l'automate temporisé en un automate non temporisé avec deux événements sur les horloges : set et expire. La sélection des tests est basée sur l'utilisation d'un objectif de test qui modélise les comportements à tester. Springintveld et al [20] discrétise le graphe des régions pour obtenir un automate de grilles à partir duquel les tests sont générés. Dans leur article, les auteurs admettent que leur approche n'est pas utilisable pour des spécifications de taille importante. Dans Higashino et al [12] transforme l'automate temporisé en un FSM et utilise la méthode UIOV pour dériver des tests. Krichen et al [14] étendent la relation de conformité ioco [21] aux systèmes temporisés. Pour la génération des tests, les auteurs distinguent entre le test off-line et le test on-line. Ils proposent pour chaque cas une méthode pour les dériver. Larsen et al [15] propose une approche de test on-line similaire à [14]. Lors de la génération de tests, les hypothèses sur l'environnement sont modélisées d'une manière explicite afin d'améliorer la qualité des tests.

La sélection de cas de test temporisé par l'utilisation des observateurs n'est malheureusement pas suffisante pour générer un nombre raisonnable de cas de test pour les systèmes temps réel. En effet, à cause de la nature dense du temps physique, l'exécution de ces tests ne permet pas toujours de décider de la conformité de l'IUT par rapport à sa spécification (verdict inconclusif lors de l'exécution). Le choix naïf des instants d'émission des actions par le testeur risque de ne pas pouvoir couvrir l'objectif pour lequel le scénario de test est construit (problème d'accessibilité des états d'acceptation de l'observateur). Afin de remédier à ce problème, nous proposons dans ce papier une approche qui exploite les techniques de contrôle temporisé pour forcer l'exécution de l'IUT

vers les comportements qui favorisent la couverture du critère de sélection.

## Organisation de l'article.

Le reste de l'article est organisé comme suit. La section 2 est consacrée au modèle des automates temporisés. La section 3 traite des notions relatives à l'observateur temps réel. Dans la section 4, nous présentons des stratégies d'optimisation et de génération de cas de test. L'étude du protocole RTEP est présentée dans la section 5. La conclusion est présentée dans la section 6.

## 2. MODÉLISATION ET NOTATIONS

Par la suite,  $\mathbb{N}$  (resp.  $\mathbb{R}^+$ ) est l'ensemble des naturels (resp. des réels positifs). Une horloge est une variable dans  $\mathbb{R}^+$  qui mémorise le passage du temps. Soit un ensemble  $X$  d'horloges.  $\mathbf{G}(X)$  est l'ensemble des contraintes d'horloges défini par la grammaire  $g := x \bowtie n \mid x - y \bowtie m \mid g \wedge g$ , avec  $n, m \in \mathbb{N}$  et  $\bowtie \in \{\leq, <, >, \geq\}$ . Par la suite, *true* est la conjonction des contraintes  $x \geq 0$  pour tout  $x \in X$ . Une valuation d'horloge est une fonction  $\nu : X \mapsto \mathbb{R}^+$  associant une valeur positive à chaque horloge  $x$  de  $X$ . Pour  $d \in \mathbb{R}^+$ ,  $r \subseteq X$ , et  $\nu$  une valuation,  $\nu + d$  et  $\nu[r := 0]$  sont les valuations définies par :

- Pour tout  $x \in X$ ,  $(\nu + d)(x) = \nu(x) + d$ .
- Pour tout  $x \in X \setminus r$ ,  $\nu[r := 0](x) = \nu(x)$ , et pour tout  $x \in r$ ,  $\nu[r := 0](x) = 0$ .

Pour  $g \in \mathbf{G}(X)$  et  $\nu \in \mathbb{R}^+$ , on note par  $\nu \models g$  ssi  $\nu$  satisfait  $g$  et  $\langle g \rangle$  représente l'ensemble  $\{\nu \in \mathbb{R}^+ \mid \nu \models g\}$ .

Une séquence temporisée  $w = (a_1, d_1)(a_2, d_2) \dots (a_n, d_n)$  est un élément de  $(\Sigma \times \mathbb{R}^+)^*$  avec  $d_1 \leq d_2 \leq \dots \leq d_n$  et  $\Sigma$  un alphabet d'actions.

Nous utilisons les automates temporisés à entrée/sortie inspiré de [1] pour modéliser un système temps réel.

**DÉFINITION 1.** *Un automate temporisé à entrée/sortie (TIOA) est un 6-uplet  $A = (L, l_0, \Sigma, X, Inv, \rightarrow)$ , avec :*

- $L$  un ensemble de localités,
- $l_0$  la localité initiale,
- $\Sigma = \Sigma_o \cup \Sigma_i$  un alphabet fini d'actions d'entrée (ou de réception) et de sortie (ou d'émission),
- $X$  un ensemble d'horloges,
- $Inv : L \mapsto \mathbf{G}$  une fonction qui associe un invariant à chaque localité
- $\rightarrow \subseteq L \times \mathbf{G}(X) \times \Sigma \times 2^X \times L$  un ensemble des transitions. Une transition  $t$  est définie par un quintuplet de la forme  $(l, g, a, r, l')$  notée  $l \xrightarrow{g, a, r} l'$  : les localités  $l$  et  $l'$  sont les localités source et destination respectivement de la transition,  $g$  est la garde que doivent satisfaire les valuations des horloges pour pouvoir franchir la transition,  $a \in \Sigma_o \cup \Sigma_i$  est l'action qui peut être reçue ou générée lors du franchissement,  $r \subseteq X$  est l'ensemble des horloges à réinitialiser lors du franchissement de la transition.

Soit  $A$  un TIOA. Un chemin de  $A$  est une séquence finie de transitions consécutives de l'état initial de  $A$ . La sémantique de  $A$  est définie par un système de transitions étiquetées  $S(A)$ . Un état (ou encore une configuration)  $s \in S$  est un couple  $(l, \nu)$ , avec  $l$  une localité et  $\nu$  une valuation. On distingue deux types de transitions :

- Transitions temporisées : Pour un état  $(l, \nu)$  et  $d \in \mathbb{R}^+$ ,  $(l, \nu) \xrightarrow{d} (l, \nu + d)$  si pour tout  $0 \leq d' \leq d$ ,  $\nu + d' \models I(l)$ .
- Transitions discrètes : Pour un état  $(l, \nu)$  et une transition  $l \xrightarrow{g, a, r} l'$ ,  $(l, \nu) \xrightarrow{a} (l', \nu[r := 0])$  si  $\nu \models g$  et  $\nu[r := 0] \models I(l')$ .

Soit  $A$  un TIOA et  $S(A)$  sa sémantique.  $A$  est dit :

- Déterministe si  $S(A)$  est déterministe.
- Complet en entrée, s'il accepte toute entrée à tout instant.
- Urgent s'il ne laisse pas le temps s'écouler dans un état lorsqu'une action est possible<sup>1</sup>

Soit  $l_f$  un état de  $A$ . Une exécution  $\rho$  de  $A$  de destination  $l_f$  est une séquence de transitions de  $S(A)$  de la forme  $\rho = (l_0, \nu_0) \xrightarrow{a_0} (l_1, \nu_1) \cdots \xrightarrow{a_n} (l_f, \nu_n)$  tel que  $a_i \in \Sigma \cup \mathbb{R}^+$  pour tout  $i \in [1, n]$ . On définit  $time(\rho) = \sum_{j \in J} a_j$  tel que  $J = \{j \mid a_j \in \mathbb{R}^+, j \in [1, n]\}$ .

Une exécution  $\rho$  d'un chemin  $p = l_0 \xrightarrow{g_1, a_1, r_1} l_1 \cdots \xrightarrow{g_n, a_n, r_n} l_n$  de  $A$ , est dite :

- au plus tôt si  $\rho$  est une exécution de  $A_p$  de destination  $l_n$  tel que pour toute exécution  $\rho'$  de  $p$  :

$$time(\rho) \leq time(\rho')$$

- au plus tard si  $\rho$  est une exécution de  $A_p$  de destination  $l_n$  tel que pour toute exécution  $\rho'$  de  $p$  :

$$time(\rho) \geq time(\rho')$$

Avec  $A_p$ , l'automate défini par le chemin  $p$  (le sous automate de  $A$  restreint aux transitions de  $p$ ).

**EXEMPLE 1.** La Figure 1 illustre un exemple d'un TIOA modélisant le comportement d'un processus de contrôle/commande de température d'un four. Le contrôleur reçoit la valeur de la température  $t$  au plus tard après 4 unités de temps (horloge  $x$ ), suivie de la valeur de la pression  $p$  au plus tard 5 unités de temps après la réinitialisation. Il envoie le signal de commande dans les 5 premières unités de temps après la réinitialisation. Les politiques de sûreté de fonctionnement suivantes ont été définies : un signal d'avertissement  $e_1$  peut être envoyé à l'environnement si le contrôleur ne reçoit pas le signal de la pression après 3 unités de temps. Un deuxième signal d'avertissement  $e_2$  peut être envoyé si le calcul de la valeur de la commande est effectué dans un temps inférieur à 2 unités de temps après réinitialisation (comportement indéterministe).

<sup>1</sup>Ceci implique que dans  $S(A)$ , les transitions sortant d'un état donnée sont soit toutes temporisées soit toutes discrètes

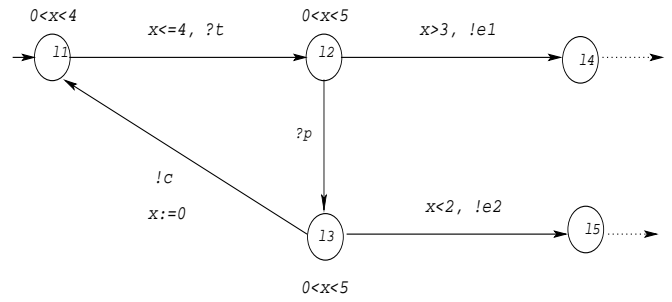


Figure 1: Exemple d'un TIOA.

### 3. OBSERVATEUR TEMPORISÉ

Un observateur modélise les comportements critiques à tester de l'IUT. Il est exprimé sous forme d'un automate temporisé à entrée/sortie avec deux localités puits : *Accept* et *Reject*. Les chemins partant de l'état initial vers l'état *Accept* représentent des comportements à tester. Si cette localité est atteinte lors de l'exécution du test, nous concluons que la fonctionnalité modélisée par l'observateur est satisfaite. *Reject* est la localité finale des chemins que le testeur ne souhaite pas tester. Ces chemins devraient être évités lors de l'exécution de tests.

Dans notre exemple de système de contrôle/commande de température (Figure 1), une fonctionnalité que nous voulons vérifier sur l'IUT peut être la suivante : dans les conditions normales de fonctionnement, un cycle de calcul de la commande dure entre 2 et 5 unités de temps. En conséquence, seuls les instants d'émission et de réception des trois signaux  $t$ ,  $p$ , et  $c$  sont mis en jeu et les signaux d'avertissement  $e_1$  et  $e_2$  doivent être écartés lors de la génération de tests.

Afin d'automatiser la construction de l'observateur, on peut l'exprimer par une formule MITL et puis la traduire en un automate temporisé [17]. Notre propriété peut être exprimée comme suit :

$$(\diamond_{1 \leq x \leq 5} ?c) \wedge \square \neg (?e_1 \vee ?e_2)$$

Dans notre approche, nous supposons que l'observateur rend explicite tous les comportements possibles de l'IUT (complétude des comportements) lors de son test. A partir d'un simple automate modélisant un comportement partiel (appelé objectif de test) de l'IUT, la procédure suivante peut être appliquée pour obtenir l'observateur complet : Pour chaque localité  $l$  de l'automate et chaque action  $a \in \Sigma$  :

1. Les transitions *Accept*  $\xrightarrow{true, a, -}$  *Accept*, *Reject*  $\xrightarrow{true, a, -}$  *Reject* sont ajoutées.
2. Si  $a$  est spécifiée dans l'une des transitions sortantes de  $l$ , la transition  $l \xrightarrow{\neg g, a, -}$  *Reject* est ajoutée, tel que  $g = \bigvee_i g_i$ , avec  $l \xrightarrow{g_i, a, r_i} l'$  est une transition de l'objectif de test <sup>2</sup>.
3. Si  $a$  n'est pas spécifiée dans les transitions sortantes de  $l$ , la boucle  $l \xrightarrow{true, a, -} l$  est ajoutée.

<sup>2</sup> $\neg g$  est la négation de la contrainte  $g$ . Cette négation peut être une disjonction d'une conjonction de contraintes élémentaires et dans ce cas, un ensemble de transitions est ajouté, une transition par conjonction de contraintes élémentaires.

Afin de marquer les comportements (désirables et indésirables) de l'observateur dans la spécification, le produit synchrone entre les deux automates doit être effectué :

**DÉFINITION 2.** Soient  $A_S(L_S, l_{0S}, X_S, \Sigma, I_S, \rightarrow_S)$  une spécification et  $A_O(L_O, l_{0O}, X_O, \Sigma, I_O, \rightarrow_O)$  un observateur. Le produit synchrone entre  $A_S$  et  $A_O$  est l'automate temporisé  $A_{SP}(L_{SP}, l_{0SP}, X_{SP}, \Sigma, I_{SP}, \rightarrow_{SP})$  défini par :

- $L_{SP} = L_S \times L_O$  est l'ensemble des localités. Les états *Accept* et *Reject* sont définis par :
  - $Accept_{SP} = L_S \times \{Accept\}$ ,
  - $Reject_{SP} = L_S \times \{Reject\}$ .
- $l_{0SP} = l_{0S} \times l_{0O}$  est la localité initiale
- $X_{SP} = X_S \cup X_O$  est l'ensemble des horloges ;
- $\rightarrow_{SP}$  est l'ensemble des transitions défini par : si  $l_S \xrightarrow{g_S, a, r_S} l'_S$  et  $l_O \xrightarrow{g_O, a, r_O} l'_O$  alors  $(l_S, l_O) \xrightarrow{g_S \wedge g_O, a, r_S \cup r_O} (l'_S, l'_O)$  et  $I((l_S, l_O)) = I(l_S) \wedge I(l_O)$ .

#### 4. OPTIMISATION DE LA GÉNÉRATION DE CAS DE TEST

La sélection de cas de test par l'utilisation des observateurs n'est malheureusement pas suffisante pour générer un nombre convenable de cas de test. Cette technique doit être complétée par le choix des instants d'occurrence des actions. Le testeur doit choisir, parmi les configurations atteignables, celles qui favorisent l'apparition d'erreurs (i.e. les configurations suspectes). Une possibilité (qui est raisonnable à notre avis) est de prendre toutes les traces d'exécution au plus tôt et au plus tard. Cependant, ces dernières peuvent amener l'IUT vers des situations non souhaitables (états *Reject* de l'observateur), ce qui conduit à la génération de tests qui n'ont aucune chance à révéler les erreurs. Dans cette section, nous montrons qu'une sélection optimal de l'état initial et des instants d'émission des actions d'entrée peut réduire considérablement le nombre de cas de test générées pour couvrir un objectif de test donnée.

**DÉFINITION 3.** Soit  $SP$  le produit synchrone construit à partir de l'automate temporisé  $A$  et de l'observateur  $O$ . Un graphe de test complet  $GT$  est un automate temporisé construit à partir de  $SP$  en transformant toute localité  $Accept_{sp}$  dans  $SP$  en localité *PASS*, et toute localité  $Rejet_{sp}$  en localité *Inconclusif*. Il est important de noter que, dans ce graphe, l'ensemble des actions sont inversés : les actions d'émission de l'IUT doivent être considérées comme des actions de réception du graphe de test, et les actions de réception de l'IUT doivent être interprétées comme des actions d'émission du graphe de test. Une localité *FAIL* et un ensemble de transitions  $\{l \xrightarrow{a, \neg g, \neg} FAIL\}$  sont ajoutés, avec  $l$ , une localité de  $GT$ ,  $a$ , une action de réception de  $GT$ ,  $g = \bigvee_{i \in \Psi} g_i$  et  $\{l \xrightarrow{g_i, a, r_i} l' \mid i \in \Psi\}$  représentant l'ensemble des transitions sortantes de  $l$  par l'action  $a$  du produit synchrone  $SP$ . Si  $a$  n'est pas spécifiée dans les transitions sortantes de  $l$ , la transition  $l \xrightarrow{a, true, \neg} FAIL$  est ajoutée au graphe de test.

L'introduction des transitions qui amènent vers la localité *FAIL* rend le graphe de test complet en entrée.

Une transition est dite *contrôlable* ssi elle est étiquetée par une action d'émission dans le graphe de test. Les contraintes temporelles associées à ce type de transitions définissent l'intervalle de temps dans lequel cette transition doit être tirée. Une transition est dite *incontrôlable* si elle est étiquetée par une action de réception dans le graphe de test. Les contraintes temporelles associées à une transition incontrôlable modélisent l'incertitude sur l'instant d'arriver de son action.

**EXEMPLE 2.** La Figure 2 illustre un testeur du comportement de notre système de contrôle/commande dans les conditions normales de fonctionnement<sup>3</sup>. Dans ces conditions, il n'est pas nécessaire de tester les politiques de sûreté de fonctionnement et on doit alors éviter l'occurrence des signaux d'avertissements  $e_1$  et  $e_2$  lors de la génération et de l'exécution de cas de test.

Pour couvrir cet objectif de test, considérons les exécutions au plus tôt et au plus tard du chemin qui amènent vers *PASS*. Due à la non contrôlabilité de la transition  $l_1 \xrightarrow{e_1, x > 3, \neg}$ , *Inconclusif*, au lieu d'avoir l'exécution au plus tard, on risque d'avoir l'exécution suivante :

$$T_{c_{max}} = (l_0, x = 0) \xrightarrow{4} (l_0, x = 4) \xrightarrow{!t} (l_1, x = 4) \xrightarrow{0.9} (l_1, x = 4.9) \xrightarrow{?e_1}$$

De la même façon, au lieu d'avoir l'exécution au plus tôt, on risque d'avoir l'exécution suivante :

$$T_{c_{min}} = (l_0, x = 0) \xrightarrow{!t} (l_1, x = 0) \xrightarrow{!p} (l_2, x = 0) \xrightarrow{?e_2}$$

On remarque qu'aucun des deux tests précédents ne peut couvrir le comportement de l'observateur. Ainsi, dans les localités  $l_1$  et  $l_2$ , selon ce qu'on reçoit de l'IUT l'état peut être *Inconclusif* ou *PASS* (problème de contrôlabilité des transitions).

L'exemple précédent montre que le problème d'identification des traces suspectes est un défi réel. Par la suite, nous proposons une approche d'identification de ces traces. Cette approche est basée principalement sur la théorie de la commande par supervision des systèmes à événements discrets temporisés [4, 19] et qui permet de :

- Forcer l'IUT (lorsque c'est possible) vers les exécutions souhaitées.
- Ecarter la génération de cas de test non pertinents.

#### Le forçage temporel d'exécution

Le forçage temporel consiste à calculer de nouveaux intervalles temporels dans lesquels le testeur doit envoyer les actions d'entrées de façon à ce que les exécutions non désirables soient écartées. Ces changements sont effectués sur le graphe de test. Pour effectuer ce calcul, nous identifions dans un premier temps l'ensemble  $M$  des localités du graphe de test qui amènent vers la localité *Inconclusif* :  $M = \{l \mid l \xrightarrow{g, a, r} \text{Inconclusif} \text{ est une transition du graphe de test}\}$ . Nous effectuons ensuite, selon le type de transitions sortantes de chaque localité  $l \in M$ , l'une des deux opérations suivantes :

<sup>3</sup>Pour ne pas surcharger la figure, les transitions qui amènent vers *FAIL* n'ont pas été représentées.

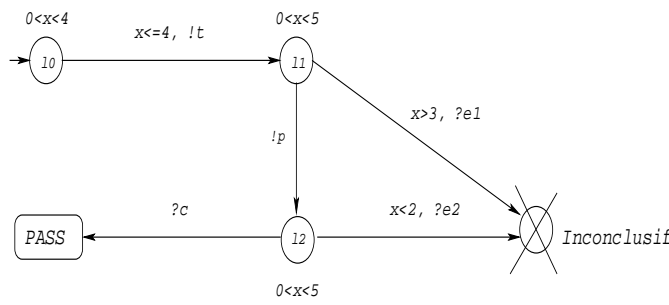


Figure 2: Graphe de test

### Opération 1.

Cette opération est utilisée ssi il existe au moins une transition contrôlable sortante de la localité  $l$  de  $M$ . Nous calculons dans un premier temps les nouvelles contraintes temporelles des transitions contrôlables, qui favorisent leur franchissement avant qu'une transition qui amène l'IUT vers la localité inconclusif soit franchissable. Nous vérifions ensuite l'accessibilité à ces nouvelles contraintes à partir de  $l$ .

Nous utilisons les techniques symboliques de résolution de contraintes pour effectuer ces calculs. Les ensembles des valuations d'horloges sont représentés d'une manière compacte par ce qu'on appelle les *zones*.

Une zone sur un ensemble d'horloge  $X$  est une conjonction de contraintes sur les horloges de la forme  $x_i - x_j < x_{i,j}$ ,  $x_i < c_{iu}$ , et  $c_{il} < x_i$ , où  $< \in \{<, \leq\}$ ,  $x_i, x_j \in X$ , et  $c_{i,j}, c_{iu}, c_{ui}$  des entiers. Un *état symbolique* est une paire  $(l, Z)$  avec  $l$ , une localité de l'automate temporisé et  $Z$ , une zone.  $Z$  représente un ensemble de valuation d'horloges, i.e., un état symbolique représente un ensemble d'états concrets :  $(l, Z) = \{(l, v) \mid v \in Z\}$ .

Pour un état symbolique  $S = \{(l, v) \mid v \models Inv(l)\}$  nous définissons les opérateurs suivants :

- $dsucc(S, a) = \{(l', v') \mid \exists (l, v) \in S \cdot (l, v) \xrightarrow{a} (l', v') \wedge v' \in \langle Inv(l') \rangle\}$ .
- $dpred(S, a) = \{(l', v') \mid \exists (l, v) \in S \cdot (l', v') \xrightarrow{a} (l, v)\}$
- $tsucc(S) = \{(l, v') \mid \exists \delta \in \mathbb{R}^+ \cdot v' = v + \delta \wedge v' \in \langle Inv(l) \rangle\}$ .
- $tpred(S) = \{(l, v') \mid \exists \delta \in \mathbb{R}^+ \cdot v = v' + \delta \wedge v' \in \langle Inv(l) \rangle\}$

où  $a \in \Sigma$ .  $dsucc(S, a)$  contient tous les états accessibles par un ensemble d'états de  $S$  en exécutant l'action  $a$ .  $dpred(S, a)$  contient les états à partir desquelles on peut atteindre  $S$  en franchissant la transition étiquetée par l'action  $a$ .  $tsucc(S)$  contient tous les états accessibles par un ensemble d'états de  $S$  en laissant passer  $\delta$  unités de temps. Et enfin,  $tpred(S)$  contient tous les états concrets à partir desquelles  $S$  peut être atteint après  $\delta$  unités de temps.

Soit  $l \in M$  et  $S_l = \{(l, v) \mid v \models Inv(l)\}$  l'ensemble des états concrets accessibles associé à  $l$ . Soit  $t = l \xrightarrow{G, !b, r} n$  une transition étiquetée par une action d'émission et  $t' = l \xrightarrow{G', ?a, r'} q$  une transition qui peut amener le système vers une localité indésirable (état Inconclusif du graphe de test). La nouvelle contrainte d'émission  $G_n$  qui garantit le franchissement de  $t$

avant que  $t'$  ne soit franchissable est donnée par l'expression suivante

$$G_n = G \wedge \neg G'$$

Ce changement de contraintes ne peut être validé que si  $\langle G_n \rangle$  est atteignable à partir de  $l$  :

$$tpred(l, \langle G_n \rangle) \cap S_l \neq \emptyset$$

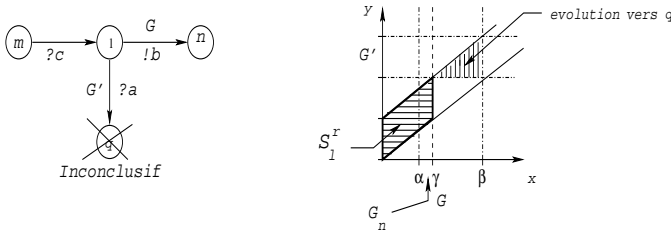
L'espace d'état initial à l'entrée de  $l$ , qui garantit le franchissement de la transition  $t = l \xrightarrow{G_n, !b, r} n$  avant que la transition  $t' = l \xrightarrow{G', ?a, r'} q$  ne soit franchissable est:

$$Int_l^r = dsucc(S_m, ?c) \cap tpred(tpred(l, \langle G_n \rangle) \cap S_l)$$

où  $S_m$  est l'espace d'états atteignable dans la localité  $m$  et  $?c$ , l'action de la transition qui mène vers  $l$ . On distingue trois cas :

- $Int_l^r = \emptyset$ , dans ce cas les nouvelles contraintes pour franchir la transition  $t = l \xrightarrow{G, !b, r} n$  ne sont pas atteignables et on ne peut donc pas forcer l'exécution du système sous test vers les états qui favorisent la détection des erreurs (les états qui couvrent l'objectif de test). La génération de test inconclusif est inévitable dans ce cas.
- $Int_l^r = dpred(S_l, c)$ , dans ce cas les nouvelles contraintes calculées garantissent que les évolutions indésirables ne sont plus atteignables depuis  $l$ . Pour éviter les tests inconclusifs, le testeur doit donc *forcer* l'envoi des actions d'émission dans ces nouvelles contraintes.
- $Int_l^r \neq \emptyset \wedge Int_l^r \neq dpred(S_l, c)$ , dans ce cas les nouvelles contraintes calculées ne peuvent pas garantir la non génération des tests inconclusifs. Afin d'éviter ce risque, il faut remonter les transitions de l'automate et calculer des nouvelles contraintes des actions d'émission de sorte que  $Int_l$  soit restreint à  $Int_l^r$ . Ce calcul est effectué par l'analyse en arrière de l'atteignabilité en utilisant les deux opérateurs  $tpred(\cdot)$  et  $dpred(\cdot, \cdot)$ . Le forçage d'exécution est possible si et seulement si  $Int_{l_0}^r \subseteq tsucc(S_{l_0})$  où  $l_0$  est la localité initiale.

La Figure 3 illustre ce calcul. Initialement l'action  $!b$  peut être envoyée dans l'intervalle temporel  $G = [\alpha, \beta]$ . À partir du point temporel ( $\gamma \in [\alpha, \beta]$ ) le système peut évoluer vers l'état  $q$  (évolution indésirable) par la réception de  $?a$ . Le



**Figure 3: Nouvelles contraintes de franchissement d'une action d'émission**

changement des conditions d'envoi de  $!b$  en  $G_n = [\alpha, \gamma]$  restreint l'espace d'état atteignable dans  $l$  en un sous espace désiré  $S_l^r$  dans lequel, l'évolution vers  $q$  est impossible.

**Remarque.** Dans le cas général (le cas où il existe plusieurs transitions de sortie à partir de  $l$  et qui mènent vers des états souhaitables) l'espace initial désiré de  $l$  est l'union de tout les espaces initiaux désirés pour les franchissements des transitions de sortie.

**Opération 2.** Cette opération est utilisée dans le cas où toutes les transitions sortantes de  $l$  sont incontrôlables (i.e. étiquetées par des actions de réceptions dans le graphe de test). Dans ce cas le testeur ne peut pas forcer l'exécution de ces actions. Une restriction de l'espace atteignable dans la localité  $l$  doit être effectuée. L'objectif est d'éviter l'évolution du système vers la localité inconclusif pendant le passage par la localité  $l$ . La restriction de l'espace atteignable de  $l$  ne peut s'effectuer que par le changement de l'espace d'entrée dans cette localité.

Soit  $l \in M$  et  $S_l = \{(l, v) \mid v \models \text{Inv}(l)\}$  l'ensemble des états concrets accessibles associé à  $l$

- $t_1 = m \xrightarrow{g_1, !a, r_1} l$ , une transition qui mène vers  $l$ .
- $t_2 = l \xrightarrow{g_2, ?b, r_2} n$  une transition étiquetée par une action de réception (une action désirable mais pas contrôlable).
- $t_2 = l \xrightarrow{g_3, ?c, r_3} q$  une transition qui peut amener le système vers la localité inconclusif (action non désirable et non contrôlable).

L'espace d'états de  $l$  qui favorise l'exécution de  $?b$  sans que la transition  $l \xrightarrow{g_3, ?c, r_3} q$  soit tirable est :

L'espace d'états de  $l$  qui permet de franchir la transition étiquetée par  $?b$  sans que la transition  $l \xrightarrow{g_3, ?c, r_3} q$  soit tirable est <sup>4</sup> :

$$S_l^f = (S_l \cap \langle g_2 \rangle) \cap \neg(\text{tpred}(S_l \cap \langle g_3 \rangle))$$

L'évitement de la localité inconclusif est possible ssi cet espace ( $S_l^f$ ) est atteignable à partir de l'espace d'états initial de  $l$ . Dans la suite on vérifie cette atteignabilité. On note par  $\text{Int}_l^f$  l'espace d'état initial qui favorise l'exécution de  $?b$  et non  $?c$ , alors :

<sup>4</sup>L'espace d'états de  $l$  qui favorise l'exécution de  $?b$  et non  $?c$  est :  $S_l^f = \text{tpred}(S_l \cap \langle g_2 \rangle) \cap \neg(\text{tpred}(S_l \cap \langle g_3 \rangle))$

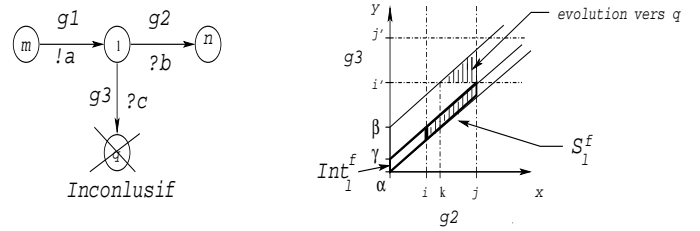
$$\text{Int}_l^f = \text{dsucc}(S_m, !a) \cap \text{tpred}(S_l^f)$$

Une exécution qui évite la localité inconclusif est possible ssi  $\text{Int}_l^f \neq \emptyset$ . Dans ce cas le testeur doit forcer l'exécution des actions contrôlables amonts de  $l$  dans les conditions temporelles qui restreignent l'espace d'entrée de  $l$  à l'espace  $\text{Int}_l^f$ . Le forçage est effectué en changeant les conditions d'envoi des actions contrôlables.  $\langle g_1 \rangle$  se restreint alors :

$$\langle g_1 \rangle = \langle g_1 \rangle \cap \text{dpred}(\text{Int}_l^f, !a)$$

Ces changements doivent être propagés jusqu'à la localité initiale.

La Figure 4 illustre un exemple de ce calcul. L'espace d'états initial à l'entrée de  $l$  est  $\text{Int}_l = [\alpha, \beta]$ . À partir du point temporel  $k \in [a, b]$ , les successeurs temporels de cet espace peuvent conduire le système sous test vers la localité  $q$ . L'espace d'état de  $l$  qui permet de franchir la transition étiquetée par  $?b$  et non  $?c$  est  $S_l^f = \{(l, v) \mid v \models (j - x \leq i' - y)\}$ . Le changement de  $\text{Int}_l$  en un sous espace  $\text{Int}_l^f = [\alpha, \gamma]$  restreint l'espace d'état de  $l$  en un espace désiré  $S_l^f = \text{tpred}(S_l^f)$  dans lequel aucun état concret ne peut conduire l'exécution de l'IUT à l'état inconclusif.



**Figure 4: Nouvelles contraintes d'entrées dans une localité suspecte**

**EXEMPLE 3.** Prenons le système de contrôle/commande de la Figure 2. Pour cet exemple, l'ensemble des états où le système peut diverger est  $M = \{l_1, l_2\}$ . Pour éviter l'occurrence de l'action  $e_1$ , nous calculons les nouveaux intervalles d'émission de l'action  $t$  (première opération de la Figure 3): L'action  $!t$  doit être envoyée dans les 3 premières unités de temps après réinitialisation au lieu de 4 unités de temps.

Pour éviter l'occurrence de l'action  $e_2$ , nous changeons les conditions temporelles d'entrée dans la localité  $l_2$  (deuxième opération faite dans la Figure 4): Le testeur ne doit pas envoyer l'action  $!p$  dans les deux premières unités de temps après réinitialisation.

La Figure 5 montre le testeur optimal de notre système de contrôle/commande. Les cas de test concrets extraits à partir de ce testeur sont efficaces dans le sens où l'exécution de chacun d'eux ne peut mener le système à réaliser les actions  $e_1$  et  $e_2$ .

Nous avons effectué et implémenté le calcul des zones forçable <sup>5</sup> d'une manière symbolique en utilisant les matrices de bornes DBM (Difference Bound Matrix) [8]. On note que l'opérateur  $(\neg)$  sur les DBM est un opérateur qui peut donner un ensemble non convexe de valuations d'horloges non

<sup>5</sup>les nouvelles contraintes temporelles dans lesquelles le testeur doit forcer l'exécution des actions contrôlables

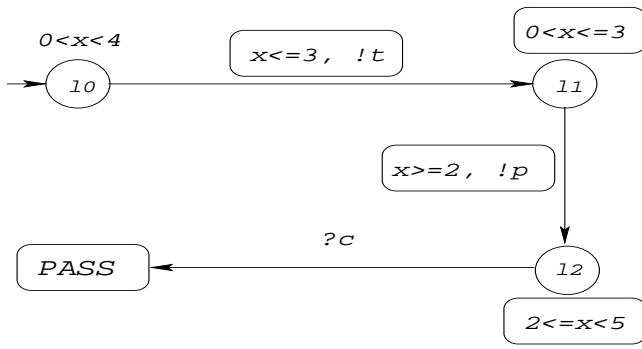


Figure 5: Le testeur optimal

représentable comme une conjonction de contraintes simples. La représentation de ce type d'ensemble exige un ensemble de DBM, ce qui rend le calcul des zones forcables coûteux.

#### 4.1 Génération de cas de test concrets

Pour générer des cas de test sous forme de séquences, le testeur doit connaître les instants précis des actions de sortie de l'IUT pour pouvoir calculer les instants des prochaines émissions. Ce calcul n'est possible que dans deux cas : Le cas où toutes les actions de sortie sont urgentes, et le cas où le test est généré dynamiquement lors de l'exécution. Dans les autres cas, un cas de test concret peut être représenté par un arbre.

**Cas de test sous forme de séquence.** Les actions urgentes sont réalisées dès que les conditions temporelles sont satisfaites. La concrétisation de cas de test est effectuée en choisissant des points temporels suspects dans les nouveaux intervalles temporels d'émissions, et en calculant leurs propagations dans le graphe optimal de test (graphe d'accessibilité).

EXEMPLE 4. À partir du testeur de la figure 5, si on suppose que l'action ?c est urgente, les deux cas de test qui correspondent aux exécutions au plus tôt et au plus tard sont

- $t_{c_{min}} = l_0 \xrightarrow{0} l'_0 \xrightarrow{!t} l_1 \xrightarrow{2} l'_1 \xrightarrow{!p} l_2 \xrightarrow{0} l'_2 \xrightarrow{?c} PASS.$
- $t_{c_{max}} = l_0 \xrightarrow{3} l'_0 \xrightarrow{!t} l_1 \xrightarrow{0} l'_1 \xrightarrow{!p} l_2 \xrightarrow{2} l'_2 \xrightarrow{?c} PASS.$

**Cas de test sous forme d'arbres.** Dans ce cas, un cas de test est construit par un calcul d'atteignabilité bornée pour chaque localité. Dans ce type d'analyse, le calcul des successeurs temporels est limité par un délai  $\Delta$  dépendant de la rapidité de réaction de l'IUT. L'automate résultant est un graphe non temporisé dans lequel le délai  $\Delta$  est considéré comme une action de sortie et chaque localité est un ensemble d'états de la spécification. Le noeud initial est  $l_0$ . Pour chaque noeud  $l$  et pour chaque action  $\alpha \in \Sigma \cup \Delta$ , un noeud  $l' \mid l \xrightarrow{\alpha} l'$  est généré et une transition  $l \xrightarrow{\alpha} l'$  est ajoutée. Les techniques d'extrapolation [2] sont appliquées pour assurer la terminaison du calcul.

L'algorithme de génération de l'arbre de test est le suivant : à partir du noeud final PASS (Accept), nous cherchons une trajectoire jusqu'au noeud initial. Cette trajectoire doit être étendue en un arbre comme suit : pour chaque noeud dans la trajectoire, et pour chaque action de sortie (y compris  $\Delta$ ), des transitions sortantes sont ajoutées. Par exemple, s'il existe une transition  $l \xrightarrow{\alpha} l'$  dans la trajectoire où

$\alpha \in \Sigma_O \cap \Delta$ , alors la transition  $l \xrightarrow{\beta} l''$  est ajoutée pour chaque action  $\beta \in \Sigma_O \cup \Delta$ . Les feuilles de l'arbre sont étiquetées par PASS sauf si la feuille est vide, dans ce cas elle est étiquetée par FAIL.

EXEMPLE 5. La Figure 6 montre un cas de test généré à partir du testeur optimal de la Figure 5. Pour ne pas surcharger la figure, nous n'avons pas représenté les transitions qui amènent vers le noeud FAIL

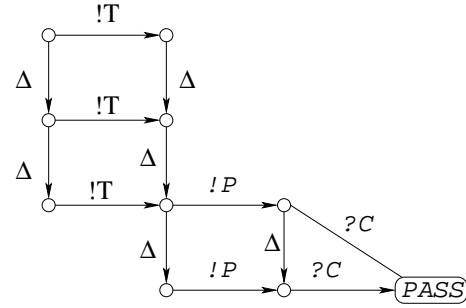


Figure 6: un exemple de cas de test sous forme d'arbre

## 5. ETUDE DE CAS DU PROTOCOLE ETHERNET TEMPS RÉEL

RTEP (Real Time Ethernet Protocol) [18] est une méthode d'accès au média conçue pour éviter les collisions dans les réseaux Ethernet par l'utilisation des jetons (*token*). Pour gérer les collisions, chaque station du réseau possède une file d'attente d'émission, dans laquelle les paquets à transmettre sont stockés par ordre de priorité, et un ensemble de files d'attente de réception. Chaque application possède sa propre file d'attente de réception. L'application doit assigner un numéro et un canal ID à chaque tâche qui demande une communication par le protocole.

Le réseau est organisé en anneau logique. Il y a deux phases : La phase d'attribution des priorités et la phase de transmission du message d'une application. Pour la transmission d'un message, une station arbitraire est désignée comme le *master\_token*. Pendant la phase d'attribution de priorité, le jeton traverse tout l'anneau en visitant toutes les stations. Chaque station vérifie les informations contenues dans le jeton afin de déterminer si un de ses propres paquets a une priorité supérieure à celle portée par le jeton. Si c'est le cas, le jeton est changé avec l'adresse de la station la plus prioritaire et la valeur de sa priorité, sinon il continue à circuler de proche en proche.

Dans la phase de transmission de message, la station *master\_token* envoie un message à la station qui possède le message le plus prioritaire afin de lui donner la permission d'envoyer son message. La station réceptrice devient dans ce cas le *master\_token*.

### 5.1 Politiques de tolérance aux fautes

Trois types de fautes sont considérés :

- La défaillance d'une station : une reconfiguration de l'anneau est exigée.
- La perte d'un paquet : une retransmission est exigée.

- Station occupée (temps de réponse long) : les paquets dupliqués sont détruits.

Le comportement temps réel est assuré dans le cas de perte des paquets. Les autres fautes sont la conséquence d'une mauvaise configuration ou d'une erreur matérielle dans le système. La méthode de recouvrement d'erreurs est basée sur l'écoute simultanée du réseau par toutes les stations. Chaque station après la transmission d'un paquet, doit recevoir un *acquiescement* (*ack*) qui est le prochain signal de sortie de la station réceptrice. Si aucun acquiescement n'est reçu après un délai *timeout*, la station émettrice suppose que le message est perdu et le retransmet. La station répète cette procédure jusqu'à la réception d'un acquiescement ou l'atteinte du nombre maximal de retransmission. Dans ce cas la station réceptrice est considérée comme défaillante et doit être exclue de l'anneau. La Figure 7 montre les différentes actions d'entrées et de sorties dans le réseau.

## 5.2 Modélisation formelle

Nous modélisons formellement les aspects temporels du protocole RTEP en utilisant les automates temporisés à entrées sorties. En se basant sur les résultats expérimentaux de [18], le temps minimal  $\alpha_{min}$  et maximal  $\alpha_{max}$  pour envoyer un jeton initial (modélisé par l'action *!Initoken*) est de 30.85  $\mu s$  et 41.16  $\mu s$  respectivement. Une station doit rester dans la localité de transmission au minimum  $\beta_{min} = 24.30 \mu s$  avant de transmettre le jeton régulier *!RegToken*. Ce temps d'attente ne doit pas dépasser les  $\beta_{max} = 41.39 \mu s$ . Le temps  $\lambda$  pour envoyer une permission de la station *master\_token* vers la station qui possède l'action la plus prioritaire (modélisé par l'action *!transToken*) est entre 24.40  $\mu s$  et 25.93  $\mu s$ . Le temps d'exécution d'une action d'envoi d'un message complet noté par  $\gamma$  est entre 37.44  $\mu s$  et 41.63  $\mu s$ , et le temps d'exécution d'une action de recouvrement d'erreur  $\delta$  est entre 10.8  $ms$  et 11.16  $\mu s$ . Enfin le temps de vérification d'un jeton  $\varepsilon$  est entre 9.32 et 18.51  $\mu s$ . La Figure 8 montre le module de transmission du protocole RTEP.

## 5.3 Test de comportements critiques

Dans le protocole RTEP, une station est considérée comme défaillante, si elle ne répond pas après 4 retransmissions d'un paquet. Si le paquet retransmis est une information, le *temps nécessaire* pour faire 4 retransmissions est  $t = 4 \cdot \gamma_{min}$ . Ce temps correspond à l'occurrence de l'action *!dk* qui signifie que la station de destination est défaillante. On note que cette hypothèse n'est valable que si  $x_{min} \leq (x_{max} - x_{min})$  où  $x$  est le temps d'exécution d'une action.

Si on s'intéresse aux politiques de recouvrement d'erreurs, on peut considérer les exemples suivants :

- Une station ne doit être exclue du réseau que si elle ne répond pas après 4 retransmissions.
- Quand une station reçoit un paquet de type information, elle doit envoyer un acquiescement dans les premiers 18.51  $ms$  après la réception.

Les Figure 9 et Figure 10 montrent les comportements basicoindents de ces propriétés et leurs observateurs.

Si on suppose que toutes les actions de sortie du RTEP sont urgentes, nous pouvons générer à partir des modules de transmission et de réception du protocole et des deux observateurs, les séquences de test suivantes :

$$Tc_1 = (S1, L1) \xrightarrow[y:=0]{!Init} (S3, L1) \xrightarrow[y:=0]{y=18.51,!Tr-Token} (S6, L1)$$

$$\xrightarrow[x:=0, w:=0, m:=0]{y=37.44,!Infos} (S9, L2) \dots$$

$$\dots (S9, L2) \xrightarrow[w:=11374, m:=11374,!dk]{} PASS.$$

$$Tc_2 = (N1, F1) \xrightarrow[m:=0]{!Init} (N2, F1) \xrightarrow[m:=0]{y=21.19,!Infos}$$

$$(N2, F2) \xrightarrow[k<18.51, m<18.51,!ack]{} Accept.$$

Le premier cas de test est généré pour tester les fonctionnalités de l'observateur de la Figure 9. Nous avons choisi les valuations maximales des horloges dans les zones atteignables des localités de transmission. Ce test signifie qu'après l'initialisation, le testeur envoie l'action *tr-Token* à l'IUT à l'instant 9.30  $\mu s$ , et devrait observer le paquet d'information à *exactement* 37.44  $\mu s$  après la transmission de l'action *tr-token*. Il attend 113.74  $\mu s$  et doit observer l'action *dk*. Pour recevoir cette dernière action, le testeur ne doit pas envoyer l'action *ack*. Le deuxième cas de test est généré à partir du module de réception du protocole (qui n'a pas été présenté dans cet article) et de l'observateur de la Figure 10.

## 6. DISCUSSION

Nous avons présenté une méthode de génération et d'optimisation de cas de test pour les protocoles temps réel. Les stratégies d'optimisation, en termes de temps et de nombre de tests générés, sont inspirées des techniques de contrôle des systèmes temps réel modélisés par les automates temporisés. Nous avons défini les cas où les tests peuvent être représentés par des séquences ou par des arbres. Deux algorithmes de concrétisation de cas de test sont présentés. Le premier dans le cas où le système sous test est urgent et le deuxième, plus pratique, dans le cas général.

Les résultats présentés dans cet article n'ont pas la prétention d'être applicables à tous les systèmes temporisés. En effet, selon l'indéterminisme présent dans le système, il n'est pas toujours possible de construire un testeur qui forcera l'exécution du système vers les comportements critiques.

## 7. REMERCIEMENT

Nous remercions Jean-Paul Bodeveix, Mamoun Filali et Pierre Michel pour leur aides et pour leurs commentaires constructifs.

## 8. REFERENCES

- [1] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183+, July 1994.
- [2] J. Bengtsson. Reducing memory usage in symbolic state-space exploration for timed systems. Technical Report 009, Dept Information Technology, Uppsala University, 2001.
- [3] I. Berrada, R. Castanet, P. Felix, and A. Salah. Test case minimization for real-time systems using timed bound traces. In *TESTCOM*, number in LNCS. Springer, July 2006.

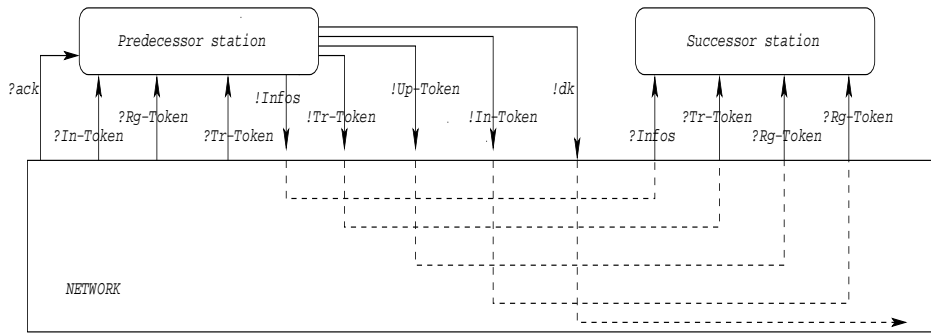


Figure 7: Fonctionnement de RTEP

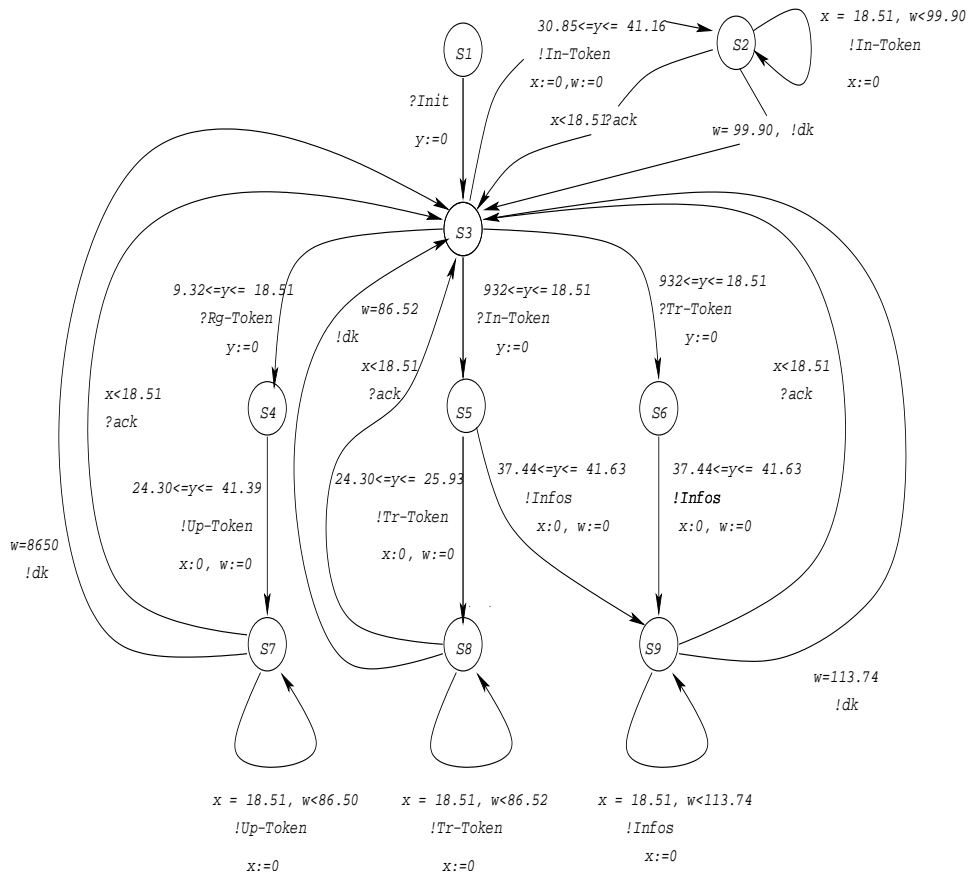


Figure 8: Le module de transmission du protocole RTEP

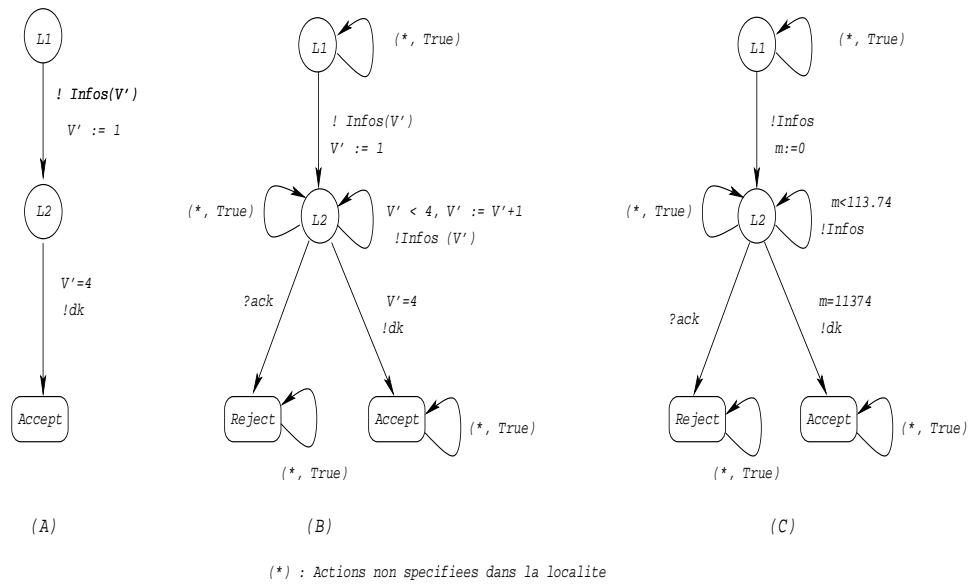


Figure 9: La première propriété: (A) son comportement basique, (B) l'observateur avec la variable ( $v'$ ) qui correspond aux nombre de retransmission, et (C) l'interprétation temporelle du variable ( $v'$ ) par l'horloge ( $m$ )

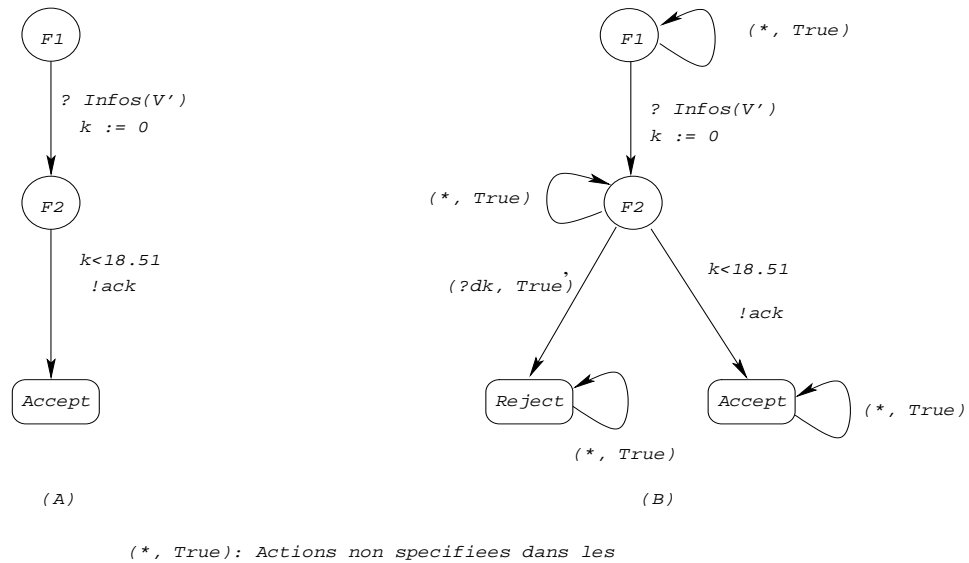


Figure 10: La deuxième propriété: (A) comportement basique, et (B) l'observateur correspondant à (A)

- [4] B. Brandin and W. W.M. Supervisory control of timed discrete event systems. *IEEE Trans. Automatic Control*, 39:329–341, 1994.
- [5] E. Brinskma. A theory for the derivation of tests. In *S. Aggarwal and Sabnani, eds. Protocol Specifications, Test, and Verifications VIII*, pages 63–74, 1988.
- [6] R. Cardell-Oliver. Conformance testing of real-time systems with timed automata specifications. *Formal Aspects of Computing*, 12(5):350–371, 2000.
- [7] D. Clarke and I. Lee. Automatic test generation for the analysis of a real-time system: Case study. In *3rd IEEE Real-Time Technology and Applications Symposium*, 1997.
- [8] D. Dill. Timing assumptions and verification of finite-state concurrent systems. In *In J. Sifakis, editor, Proceeding of first CAV*, number 407 in LNCS. Springer, 1989.
- [9] A. En-Nouaary and R. Dssouli. A guided method for testing timed input output automata. In *TestCom 2003*, number 407 in LNCS, pages 211–225. Springer, 2003.
- [10] A. En-Nouaary, R. Dssouli, F. Khenedek, and A. Elqortobi. Timed test cases generation based on state characterization technique. In *In 19th IEEE Real Time Systems Symposium (RTSS'98)*, 1998.
- [11] J. Fernandez, C. Jard, T. Jéron, and C. Viho. An experiment in automatic generation of test suites for protocols with verification technology. *Sci. Comput. Program*, 29(1-2):123–146, 1997.
- [12] T. Higashino, A. Nakata, K. Taniguchi, and A. Cavalli. Generating test cases for a timed i/o automaton model. In *TESTCOM99*. Springer, 1999.
- [13] A. Khoumsi, T. Jron, and H. Marchand. Test cases generation for nondeterministic real-time systems. In *FATES 2003*, pages 131–146. Springer, 2003.
- [14] M. Krichen and S. Tripakis. Black-box conformance testing for real-time systems. In *SPIN 2004*, pages 109–126. Springer, 2004.
- [15] K. Larsen, M. Mikucionis, and B. Nielsenn. Real-time system testing on-the-fly. In *In the 15th Nordic Workshop on Programming Theory (NWPT)*, 2003.
- [16] K. G. Larsen, P. Pettersson, and W. Yi. Diagnostic model-checking for real-time systems. In *In Proc. of WVCHS III*, number 1066 in LNCS. Springer, 1995.
- [17] O. Maler, D. Nickovic, and A. Pnueli. From mitl to timed automata. In *FORMATS 2006*, number 4202 in LNCS. Springer, 2006.
- [18] M. Martinez, M. G. Harbour, and J. J. Gutierrez. Real-time ethernet for analyzable distributed application an a minimum real-time poxis-kernel. In *2nd International Workshop on Real-Time LANs in the Internet Age. RTLIA*, 2003.
- [19] A. SAVA. Sur la synthèse de la commande des systèmes á evenement discrets temporisés. Technical report, Thèse de Doctorat LAG, Institut National Polytechnique de Grenoble, 2001.
- [20] J. Springintveld, F. Vaandrager, and P. R. D'Argenio. Testing timed automata. *Theoretical Computer Science*, 252(1-2):225–257, March 2001.
- [21] J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Software - Concepts and Tools*, 17(3):103–120, 1996.
- [22] S. Tripakis. The formal analysis of timed systems in practice. Technical report, PhD thesis, Universit Joseph Fourier, Grenoble, 1998.