Vérification de Comportement de Machine d'Exécution

Charles André — Hédi Boufaïed
Laboratoire I3S – Université de Nice/CNRS
41 Bd Napoléon III - 06041 NICE cedex
andre@@unice.fr

RÉSUMÉ. *On dispose d'outils efficaces pour vérifier les propriétés de sûreté des programmes synchrones. En pratique, lorsqu'un programme synchrone doit piloter une application réelle, il utilise une machine d'exécution. Un rôle essentiel de cette machine est d'interfacer le programme avec l'environnement asynchrone. Cet interfaçage introduit des distorsions temporelles entre signaux physiques et signaux logiques. Nous étudions dans cet article ces déformations et nous analysons dans quelle mesure les propriétés du programme sont également des propriétés de la réalisation.*

ABSTRACT. *There exist efficient tools for verifying safety properties of synchronous programs. When a synchronous program drives a real-world application it has to rely on some execution machine. One of the missions of the execution machine is to interface the program with its environment. Interfacing entails distortions between the (actual) physical signals and the logical signals the program deals with. In this paper we give a simple model of these distortions and we explain how to check safety properties on the (simplified) execution machine, and not on the program only.*

MOTS-CLÉS : *programmation synchrone, validation, sûreté, asynchronisme.*

KEY WORDS : *synchronous programming, validation, safety, asynchrony.*

1. Introduction

Programmation synchrone et preuves

La programmation synchrone permet l'expression de comportements réactifs complexes. Le compilateur d'un langage comme ESTÉREL n'accepte que des programmes réactifs et déterministes. Le contrôle d'un programme correct peut donc être exprimé sous la forme d'un automate (déterministe) à état fini ou d'un système d'équations booléennes. Ces deux types de représentations se prètent bien à des analyses formelles de propriétés.

Aussi séduisante que puisse paraître cette approche, elle n'est pas totalement satisfaisante pour l'utilisateur. En fait, on sait prouver des propriétés d'un programme, alors qu'un utilisateur veut valider son implémentation.

Mise en œuvre des programmes synchrones

En pratique, un programme synchrone doit piloter une application réelle. Les évolutions de l'environnement étant généralement asynchrones par rapport à celles du code synchrone, il faut ajouter des traitements d'acquisition, d'action et de synchronisation. Ces fonctions sont réalisées par une machine d'exécution. La synchronisation d'événements asynchrones ainsi que les divers traitements d'entrée/sortie entrainent des "distorsions" entre les signaux physiques et les signaux ESTÉREL qui en sont une représentation abstraite.

Propriétés de "bout en bout"

Dans cet article, nous abordons le problème de la vérification de propriété de "bout-en-bout", c'est à dire en partant des événements physiques. Ce problème est a priori très complexe car il fait intervenir les traitements d'acquisition. Pour cette raison, nous nous limitons à une classe de machines d'exécution : celles que nous avons développées [BOU 98]. Cette classe est suffisamment générique pour couvrir de nombreuses implémentations réelles.

Pour illustrer la méthode proposée nous avons choisi un contrôleur de distributeur automatique de billets (ATM ou Automatic Teller Machine). Notre objectif n'étant pas ici la modélisation, nous ne détaillerons pas le programme du contrôleur. Nous préciserons par contre le type de propriétés à vérifier. Le comportement attendu a été spécifié en SYNCCHARTS [AND 98], traduit automatiquement en programme ESTÉREL. Le programme résultant, auquel on ajoute une description de l'interfaçage (entrées/sorties physiques et choix des gérants associés), nous permet de construire une machine d'exécution dédiée à l'application [BOU 98]. Nous voulons vérifier des propriétés de cette réalisation.

Nous suivrons le plan suivant :

— Présentation sommaire de l'exemple et des propriétés attendues. Nous montrerons les insuffisances du langage naturel pour exprimer ces propriétés.

— Principe des preuves sur programmes synchrones en utilisant la notion d'observateur. L'analyse des propriétés s'appuie sur l'outil XEVE [BOU 97].

— Modélisation des distorsions introduites par les acquisitions.

— Analyse du programme modifié pour tenir compte des biais introduits par la réalisation. Les performances et limitations sont présentées.

2. Application

2.1. *ATM*

Un ATM est une interface entre un client (local) et une banque (distante). Après avoir inséré une carte, le client doit taper son code puis indiquer la somme désirée. L'identité de la carte ainsi que la somme sont communiquées à la banque qui décide si la transaction peut avoir lieu. À tout moment en dehors de certaines phases critiques, comme l'établissement de la connexion avec la banque, le client doit pouvoir annuler la transaction. Le comportement de l'ATM s'exprime de façon concise et lisible en SYNCCHARTS (les détails sont disponibles dans la thèse de H. Boufaïed [BOU 98]).

2.2. *Propriétés attendues*

Les propriétés que nous voulons vérifier pour l'ATM sont les suivantes :

P1 : Si le client tape 3 fois consécutives un code erroné, alors la carte est confisquée par le contrôleur du distributeur.

P2 : En cas d'annulation de la transaction, aucune somme d'argent n'est délivrée.

P3 : En cas de livraison d'espèces, le contrôleur émet à destination de la banque un signal confirmant la fourniture.

P4 : Si le client annule la transaction, le contrôleur du distributeur émet à destination de la banque un signal demandant l'annulation.

D'autres propriétés pourraient être testées. Ces quatre propriétés ont été retenues car elles sont pertinentes pour l'application et leur formulation pose des problèmes différents.

2.3. *Propriétés de sûreté*

Les propriétés ont été exprimées en langage naturel. Elles contiennent des ambiguités que nous allons lever. Nous recherchons des propriétés du type sûreté (safety) c'est à dire des propriétés qui établissent qu'une situation donnée (indésirable) ne se produira jamais ou inversement qu'une condition (désirée) sera toujours vraie. De telles propriétés peuvent être testées sur l'ensemble des états accessibles sans qu'il soit nécessaire d'expliciter les chemins d'exécution.

Réponse en temps borné

Le premier point à assurer est que les propriétés sont effectivement des propriétés de sûreté. La propriété **P3**, par exemple, pourrait être interprétée comme une propriété de vivacité : " après que les espèces ont été fournies, le contrôleur émettra (finalement) un signal de confirmation ". Il est clair que ce type de propriété est sans intérêt pratique pour le distributeur de billets. Dans ce cas, et de façon plus générale, pour les systèmes temps-réel, les effets attendus d'une cause doivent se produire dans un temps borné. Ce type de propriété est connu sous le nom de " bounded responsiveness " (voir les exemples d'utilisation du logiciel TempEst de C. Puchol [JAG 95]). Il s'agit d'une propriété de sûreté et non pas de vivacité.

Une formulation plus correcte de la propriété **P3** serait

P3 : En cas de livraison d'espèces, le contrôleur doit émettre à destination de la banque un signal confirmant la fourniture, dans les x secondes qui suivent.

La durée x doit être donnée par le cahier des charges du contrôleur.

Choix des événements

Il demeure une autre ambiguité liée au fait que nous adoptons une modélisation à événements discrets : quel est l'événement qui marque la fourniture des billets? Nous avons le choix entre, au moins, deux événements :

— l'ouverture de la trappe par où sortent les billets,

— la prise des billets par le client.

Le banquier préfèrera certainement le premier. Le client pencherait plutôt pour le second. Dans notre réalisation nous avons choisi d'émettre de façon synchrone (c'est à dire au même instant) l'ordre d'ouverture de la trappe et l'envoi d'un message à destination de la banque, pour l'informer que les billets vont être délivrés. Pour que la banque reçoive effectivement ce signal il faut que la liaison soit fiable, ce que nous admettrons dans cette étude. La propriété **P3** est alors reformulée en

P3 : En cas de livraison d'espèces, le contrôleur doit émettre à destination de la banque un signal lors de l'ouverture de la trappe.

La propriété revient alors à tester que ce signal est émis quand et seulement quand la trappe s'ouvre.

Comportements répétitifs

L'énoncé de la propriété **P2** contient un autre type d'imprécision. Cette propriété est implicitement limitée à une transaction : l'annulation doit entraîner la non-délivrance d'argent pour la session en cours. Bien sûr, le fait d'introduire une nouvelle carte, offrira à nouveau la possibilité de retirer de l'argent.

P2 devrait être complétée ainsi :

P2 : En cas d'annulation de la transaction, aucune somme d'argent, liée à cette transaction, ne doit être délivrée avant l'acceptation d'une nouvelle carte.

En résumé, les propriétés de sûreté peuvent être délicates à formuler en langage naturel. Leur expression à l'aide d'un formalisme synchrone (textuel ou graphique) est bien plus précise et sans ambiguité pour un expert (voir la section suivante). Nous adoptons ce point de vue en ayant conscience qu'il ne résout pas le problème du client qui peut être tout à fait étranger aux expressions formelles de comportements.

3. Principe des preuves

3.1. *Observateur*

Les observateurs [HAL 93] ont été introduits pour faciliter l'étude des propriétés de sûreté. Ce sont des modules placés en parallèle avec le programme à vérifier : ils "observent" les signaux du programme sans les modifier. Ils signalent immédiatement toute violation de la propriété qu'ils doivent surveiller. Avec cette technique, les propriétés sont exprimées dans le même formalisme que le programme lui-même, ceci constitue un réel avantage.

À titre d'exemple nous donnons le comportement de l'observateur de la propriété **P2**, exprimé en ESTEREL :

```
loop
   weak abort
     await immediate CANCEL;
     await immediate DELIVERY;
     % should not reach this point
     emit DELIVERY_after_CANCEL;
     pause % purely technical
   when CARD % insertion of a new card
end loop
```

`DELIVERY_after_CANCEL` est le signal émis en cas de violation de la propriété **P2**.

3.2. *L'outil* XEVE

XEVE [BOU 97] est un logiciel de vérification pour programmes ESTÉREL qui utilise une représentation sous forme de machines à états finis. Il permet par une analyse des états accessibles, de tester si un signal est potentiellement émis ou potentiellement non émis. Pour établir une propriété de sûreté, on fournit à XEVE le résultat de la composition d'un programme et de son observateur. Ainsi, prouver une propriété de sûreté, revient à montrer que le signal de sortie de l'observateur (signal de violation) n'est pas potentiellement émis.

Un point à souligner pour les utilisateurs : en cas de violation, XEVE fournit une séquence conduisant à la violation. De plus, cette séquence est directement exécutable par l'outil de simulation interactive (XES). Le débogage des programmes est ainsi grandement facilité.

3.3. *Passage à un programme* ESTÉREL *pur*

L'outil XEVE ne peut travailler correctement que sur des programmes ESTÉREL purs. Il faut dans un premier temps transformer notre programme écrit en ESTÉREL standard, en programme ESTÉREL pur.

Il faut donc éliminer les parties transformationnelles du programme sans en modifier le contrôle. Les modifications portent sur

— Les entités transformationnelles qui n'ont pas d'influence directe sur le comportement réactif du programme (certaines variables, fonctions et procédures)

— Les variables, fonctions ou procédures qui interviennent dans le contrôle doivent être remplacées par des signaux purs en nombre suffisant pour assurer les mêmes discriminations.

— Les tâches sont assez analogues aux procédures mais comme il existe un signal de retour associé à leur terminaison le traitement est plus délicat.

Ces transformations ne sont pas automatisables dans le cas général. Elles demandent beaucoup d'expérience et une compréhension fine du contrôle de l'application à réaliser. C'est une phase critique dans notre approche.

4. Distorsions introduites par la machine d'exécution

4.1. *Problèmes d'asynchronisme*

Le programme synchrone travaille sur des signaux qui sont des abstractions des signaux physiques. La machine d'exécution est chargée de fournir au cœur synchrone une image des signaux réels. L'extérieur évoluant en asynchronisme par rapport à la machine, cette image peut être différente de la vision idéalisée qui est celle des programmes synchrones (vecteur de valeurs échantillonnées instantanément).

Le problème le plus critique est celui de la simultanéité de signaux d'entrée [1]. En pratique, deux signaux asynchrones très proches dans le temps peuvent parfois être perçus à deux instants consécutifs par la machine d'exécution.

Pour les langages synchrones flots de données (SIGNAL et LUSTRE), une solution existe pour éviter les décalages temporels “non significatifs”. Cette solution a été déve-

1. La notion de simultanéité entre signaux locaux à un programme n'est pas en cause car ces derniers n'ont d'existence que vis à vis du programme.

loppée pour répondre aux besoins d'exécution répartie de ces programes. Elle consiste à calculer des “horloges” qui assurent le respect des précédences réelles [BEN 97]. Ici, nous traitons un problème différent qui est lié à la discrétisation des instants. Ce problème est bien connu en circuit : si une entrée varie lorsque l'horloge commande l'échantillonnage des entrées, la bascule peut passer dans un état métastable et en sortir, après une durée aléatoire, dans un état quelconque. En pratique, il convient de maintenir les entrées stables pendant une durée dite de pré-positionnement (setup time) avant et pendant une durée dite de maintien (hold time) après la commande d'échantillonnage. Dans le cas où on néglige les temps de pré-positionnement et de maintien, on peut retenir l'hypothèse de promptitude **H4**, considérée dans le paragraphe suivant. Lorsque ces conditions ne peuvent pas être satisfaites, on utilise des circuits “synchronisateurs” (bascules D en général) et on maintient les niveaux pendant au moins deux périodes d'horloge. Il faudrait alors opter pour la variante **H4'** de l'hypothèse de promptitude.

4.2. *Modèle d'acquisition*

Représentons le temps physique par $\mathbb{R}^+$ et le temps logique par $\mathbb{N}$. Notons proj$_M$ la correspondance établie par la machine d'exécution M entre les temps physique et logique.

Soit $H_M = \{t_k \in \mathbb{R}^+ \mid k = 0, 1, \cdots\}$ une suite non décroissante de dates appelée horloge de M. Ces dates sont les instants d'activations de M. δ (resp. Δ) est le plus petit (resp. le plus grand) intervalle entre deux activations consécutives. Notons I_p un événement physique d'entrée et I le signal d'entrée ESTÉREL correspondant. $t(I_p) \in \mathbb{R}^+$ est la date d'occurrence de I_p, $\theta(I) \in \mathbb{N}$ est l'instant de présence du signal I.

Le modèle retenu doit rester simple tout en exprimant l'effet d'asynchronisme.

Nos hypothèses se résument ainsi :

H1 : proj$_M$: $\mathbb{R}^+ \longrightarrow \mathbb{N}$ est une fonction.

H2 : $\forall t_k \in H_M : \mathrm{proj}_M(t_k) = k$

H3 : Monotonie : $\forall t, t' \in \mathbb{R}^+ : t < t' \implies \mathrm{proj}_M(t) \leq \mathrm{proj}_M(t')$

H4 : Promptitude :

$$\forall I_p\ t = t(I_p);\ t_k, t_{k-1} \in H_M\ t_{k-1} < t \leq t_k \implies \theta(I) = k$$

H4' : Promptitude' :

$$\forall I_p\ t = t(I_p);\ t_k, t_{k-1} \in H_M\ t_{k-1} < t \leq t_k \implies \theta(I) \in \{k, k+1\}$$

À ces hypothèses sur la relation temps physique/temps logique, il faut rajouter des hypothèses concernant le fonctionnement de la machine en temps réel vis à vis du système à contrôler :

$\forall k \in \mathbb{N} : x_k \leq (t_k - t_{k-1}) \ll \tau$ avec $t_k, t_{k-1} \in H_M$ (x_k est la durée d'exécution de la k$^{\text{ième}}$ réaction, τ est la plus petite constante de temps du système à contrôler; la machine est donc “suffisament” rapide),

$\forall k \in \mathbb{N} : t(I_p^{(k+1)}) - t(I_p^{(k)}) > \Delta$ (la machine ne perd pas des occurrences successives du même événement).

L'hypothèse **H4**, qui est plus simple que la variante **H4'**, suffit pour mettre en évidence le phénomène de distorsion des événements. Elle nous permet d'expliquer les modifications à apporter aux observateurs. Le traitement retenant l'hypothèse **H4'** est plus complexe mais relève des mêmes techniques. Nous nous limitons donc à **H4**.

4.3. *Conséquences*

Partant des hypothèses **H1**–**H4** on établit :

$$|\theta(A) - \theta(B)| \in \left\{ k \in \mathbb{N} \,\middle|\, \left\lfloor \frac{|t(A_p) - t(B_p)|}{\Delta} \right\rfloor \leq k \leq \left\lceil \frac{|t(A_p) - t(B_p)|}{\delta} \right\rceil \right\} (1)$$

Cette formule montre que même lorsque $|t(A_p) - t(B_p)| < \delta$, $|\theta(A) - \theta(B)|$ peut prendre les valeurs 0 (A et B simultanément présents) ou 1 (A et B présents dans deux instants successifs).

5. Analyse modifiée

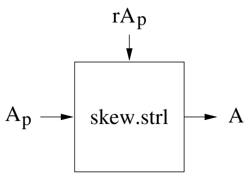
Pour établir les propriétés désirées sur la machine d'exécution de l'exemple, le biais introduit à l'acquisition doit être pris en compte. Des modules ESTÉREL qui modélisent ces distorsions sont placés en parallèle sur le programme et les propriétés sont réévaluées pour le programme résultant.

5.1. *Module de prise en compte du biais*

On associe à chaque signal d'entrée un module de biais (figure 1). A_p représente le signal physique, A le signal d'entrée (ESTÉREL) et rA_p est un signal rajouté au modèle pour prendre en compte l'indéterminisme introduit par l'acquisition. Bien sûr ce signal n'a pas de réalité physique, il n'est qu'un intermédiaire utile pour établir les propriétés. Le code ESTÉREL sur la figure 1 exprime le comportement du module de biais.

5.2. *Cas des signaux de retour*

Les signaux de retour des tâches proviennent de l'environnement d'exécution. Ils sont donc susceptibles de subir des distorsions analogues aux signaux d'entrée. Les assimiler à des signaux d'entrée dont les occurrences seraient filtrées par le module de biais (skew) conduirait à des comportements irréalistes. Un signal de retour ne peut en aucun cas se produire tant que la tâche associée n'est pas active. De plus, une tâche




```
loop
  await immediate Ap;
  present rAp then
    emit A;
    pause
  else
    pause;
    emit A
  end present
end loop
```

Figure 1. *Module de biais pour un signal d'entrée*

étant lancée, le signal de retour peut ne jamais être présent si la tâche est tuée par le programme avant qu'elle ne termine. Un module de comportement des retours biaisés est donc nettement plus complexe que le module de la figure 1. Pour les détails nous renvoyons le lecteur à la description complète de l'exemple.

5.3. *Résultats de l'expérience*

— L'expérimentation a d'abord mis en évidence la nécessité d'une modélisation réaliste des signaux physiques, en particulier pour le retour des tâches. Des modules de biais insuffisamment contraints conduisent rapidement à une explosion de l'espace des états accessibles. Dans le cas de l'ATM, une première modélisation a conduit à 8 millions d'états. Une modélisation plus judicieuse des retours a ramené cet espace à une centaine de milliers d'états.

— Les résultats des tests ont montré que certaines propriétés du programme initial n'étaient plus strictement vérifiées par la machine d'exécution. . En fait, il a fallu modifier les observateurs pour tenir compte des incertitudes introduites par la discrétisation du temps. La formule (1) doit servir de base pour quantifier ces incertitudes.

— Le non respect de propriétés peut également conduire à modifier le programme initial pour le rendre plus robuste [2].

Références

[AND 98] C. ANDRÉ, H. BOUFAÏED, and S. DISSOUBRAY. SYNCCHARTS : un modèle graphique synchrone pour systèmes réactifs complexes. In Real-Time Systems (RTS'98), pages 175–196, Paris (France), Janvier 1998. Teknea.

[BEN 97] A. BENVENISTE, P. LE GUERNIC, and P. AUBRY. Compositionality in dataflow synchronous languages: Specification and code generation. Technical Report PI–1150, IRISA, November 1997.

2. Robuste est pris dans le sens de peu sensible aux petites incertitudes temporelles sur les signaux d'entrées

[BOU 97] A. BOUALI. XEVE: an ESTEREL verification environment. Technical Report draft, CMA - ENSMP, December 1997.

[BOU 98] H.BOUFAÏED. Machines d'exécution pour langages synchrones. PhD thesis, Université de Nice-Sophia Antipolis, November 1998.

[HAL 93] N. HALBWACHS. Synchronous Programming of Reactive Systems. Kluwer Academic Publishers, Amsterdam, 1993.

[JAG 95] L J. JAGADEESA, C. PUCHOL, and J. VON OLNHAUSEN. Safety property verification of esterel programs and applications to telecommunication software. Liege, Belgium, July 1995. Conference on Computer Aided Verification (CAV'95).