



HAL
open science

Une sémantique du langage ESTEREL

Frédéric Boussinot

► **To cite this version:**

Frédéric Boussinot. Une sémantique du langage ESTEREL. [Rapport de recherche] RR-0577, INRIA. 1986. inria-00075977

HAL Id: inria-00075977

<https://inria.hal.science/inria-00075977>

Submitted on 24 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

IRIA

CENTRE
SOPHIA ANTIPOLIS

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
BP 105
78153 Le Chesnay Cedex
France

Té (1) 39 63 55 11

Rapports de Recherche

N° 577

**UNE SÉMANTIQUE
DU LANGAGE ESTEREL**

Frédéric BOUSSINOT

Octobre 1986

Frédéric BOUSSINOT

Ecole Nationale Supérieure des Mines de Paris
Centre de Mathématiques Appliquées
Sophia-Antipolis
06560 Valbonne

Résumé

On décrit une sémantique opérationnelle du langage réactif synchrone ESTEREL dans le cadre algébrique des systèmes de transitions. Cette approche utilise la notion de critère d'observation lié à celle de bisimulation pour exprimer le comportement instantané d'un programme.

Abstract

We describe operational semantics for the ESTEREL synchronous reactive language, expressed in the theory of algebraic calculi of processes. This approach uses the notions of abstraction criterion and bisimulation to reflect instantaneous behaviour of ESTEREL programs.

1. Introduction

Le langage ESTEREL [Berry-Cosserat] est un langage de programmation de *systèmes réactifs* dont le fonctionnement est "rythmé" par une suite d'activations provenant du monde extérieur: à chaque activation un programme ESTEREL reçoit des signaux d'entrée (instruction **present...then...else**) et il réagit en émettant des signaux de sortie (instruction **emit**).

L'hypothèse de *fort synchronisme* de [Berry-Cosserat] est que la réaction d'un programme est synchrone avec son activation. En d'autres termes on suppose que l'on dispose d'une machine infiniment rapide ou de manière équivalente, que le passage du contrôle ne prend aucun temps.

L'hypothèse de fort synchronisme induit la présence de programmes inconsistants c'est-à-dire auxquels on ne peut donner de signification compatible avec l'hypothèse, par exemple le programme P_1 émet un signal si celui-ci n'est pas présent:

present s then else emit s end

Un autre exemple de programme inconsistant lié à l'opérateur de parallélisme d'ESTEREL, est le programme P_2 :

```

present s1 then ... else emit s2 end
||
present s2 then emit s1 else ... end

```

le signal s2 est émis si s1 est absent, mais alors la présence de s2 provoque l'émission de s1.

Les deux programmes précédents sont rejetés par [Berry-Cosserat] comme manifestant des *cycles de causalité* (par exemple en ce qui concerne P_2 , s1 inhibe s2 qui cause s1). Le rejet des programmes cycliques conduit également à rejeter les programmes ayant plusieurs significations distinctes compatibles avec l'hypothèse de fort synchronisme. Un programme sans cycle de causalité est déterministe: il n'a qu'une réaction possible à chacune des activations.

Dans ce papier on adopte une approche dans laquelle les instants, c'est à dire l'intervalle de temps entre l'activation du programme et sa réaction, ne sont pas considérés comme de durée nulle. Dans cette vision un programme consistant est un programme qui peut être exécuté avec la contrainte suivante: tout signal émis pendant un instant *est nécessairement reçu* et avec la *même valeur* par tous les composants (i.e. les branches d'instructions parallèle) en attente de sa réception au cours de ce même instant. Cette approche et celle de [Berry-Cosserat] ne sont pas fondamentalement divergentes dans la mesure où on obtient pour les programmes sans cycle, la même sémantique si l'on décide de n'observer que le début et la fin des instants.

Par rapport à [Berry-Cosserat] l'approche de ce papier diffère par plusieurs autres aspects:

- (1) La sémantique du langage est décrite dans le formalisme des systèmes de transitions en utilisant la notion de *critère d'observation* issue de la théorie des calculs algébriques de processus [Boudol]. Dans ce formalisme, les programmes sont des termes d'une algèbre interprétés comme des systèmes de transitions: La relation fondamentale est celle qui exprime la transformation d'un programme p en un autre programme p' en effectuant l'action a , ce que l'on note

$$p \xrightarrow{a} p'$$

La signification des primitives du langage est décrite par un ensemble de règles de réécriture qui peuvent être conditionnées par des prédicats portant uniquement sur les actions (cf [Plotkin]). Par exemple, la règle suivante exprime le comportement de l'instruction **present** lorsque le signal est présent

$$\frac{E^S(s) = \text{present}}{\text{present } s \text{ then } i_1 \text{ else } i_2 \text{ end} \xrightarrow{\langle E, E \rangle} i_1}$$

Dans cette règle la prémisse est un prédicat sur l'environnement E et l'action est le couple $\langle E, E \rangle$ qui indique que l'environnement n'est pas transformé.

Les critères d'observation permettent d'observer abstraitement les systèmes de transition (et donc les programmes) en définissant les actions abstraites comme ensembles de séquences d'actions élémentaires. Un programme p effectue une action abstraite A et se transforme en un programme p' si p se transforme en p' après une séquence de transformations appartenant à l'action abstraite A :

$$p \xrightarrow{a_1} p_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} p_n \xrightarrow{a_{n+1}} p'$$

$$a_1 \dots a_{n+1} \in A$$

Le critère que l'on considère dans le cas d'ESTEREL consiste à définir les actions abstraites comme des séquences d'actions élémentaires dont la dernière indique la fin de l'exécution dans l'instant. En d'autres termes, un comportement instantané d'un programme est un comportement abstrait d'un critère d'observation définissant la notion d'instant.

- (2) Un programme inconsistant est caractérisé par le fait qu'il n'a pas de sémantique observable: il ne peut réaliser aucune séquence d'actions se terminant. Par exemple les programmes P_1 et P_2 précédents sont inconsistants. Ainsi on exprime directement le rejet des programmes inconsistants dans la sémantique, sans avoir à définir une sémantique statique. Par ailleurs, on est conduit à donner une signification à des programmes qui tout en étant consistants sont rejetés comme cycliques par [Berry-Cosserat]. Par exemple le programme P_3 suivant :

```
present s1 then ... else emit s2 end
||
present s2 then ... else emit s1 end
```

Ce programme est cyclique (l'émission de s2 dépend de l'absence de s1 et inversement) et non déterministe: il possède deux significations distinctes, la première en supposant que s1 est absent et s2 présent, la seconde en supposant l'inverse.

- (3) On considère un ensemble de primitives légèrement différent de celui utilisé dans [Berry-Cosserat] en remplaçant **upto** et **uptonext** par **present** et **pause**, cette dernière instruction indiquant le passage à l'instant suivant. On montre en utilisant l'approche algébrique et la notion de bisimulation, que la puissance d'expression du langage n'en est pas réduite. Pour cela on spécifie **upto** et **upto next** sous forme de règles de réécriture, on les traduit à l'aide de **pause** et **present** et on montre que la spécification et la traduction obtenues pour chacune d'elles sont indiscernables par toute action abstraite (elles sont équivalentes observationnellement). Enfin on considère pour simplifier la version en signaux purs du langage: seule la présence ou l'absence d'un signal dans l'instant est significative.

2. Description informelle

Dans la suite, par exécution d'une instruction on entend *exécution dans l'instant* de cette instruction. Les programmes sont construits à l'aide des primitives suivantes:

2.1. Instruction vide: **nothing**

Le contrôle passe en séquence en arrivant sur l'instruction **nothing**.

2.2. Passage d'un instant: **pause**

Le contrôle ne passe pas en séquence en arrivant sur l'instruction **pause**. Par contre, il repart en séquence à l'instant suivant.

2.3. Réception de signal: **present**

Exécuter l'instruction **present s then i_1 else i_2 end** signifie passer le contrôle à i_1 si le signal s est émis pendant l'instant, à i_2 sinon.

2.4. Emission de signal: **emit**

Exécuter l'instruction **emit s** signifie émettre le signal s et passer le contrôle en séquence.

2.5. Déclaration de signal: **signal**

Exécuter l'instruction **signal s in i end** signifie exécuter i en considérant le signal s comme local.

2.6. Séquencement

Exécuter l'instruction $i_1; i_2$ signifie exécuter l'instruction i_1 puis si celle ci passe le contrôle en séquence, exécuter l'instruction i_2 .

2.7. Boucle: **loop**

Dans l'instruction **loop i end** l'exécution de i ne doit pas passer le contrôle en séquence. Sa signification est celle de l'instruction $i; \text{loop } i \text{ end}$.

2.8. Trappe: **tag**

Exécuter l'instruction **tag l in i end** signifie exécuter l'instruction i . Le contrôle passe en séquence si l'une des deux conditions suivantes est vérifiée:

- l'instruction i passe le contrôle en séquence.
- une instruction **exit l** a été exécutée et aucune instruction **exit** correspondant à une trappe englobante n'a également été exécutée.

2.9. Exit: **exit**

Une instruction **exit l** ne peut apparaître que sous la portée d'une trappe de même nom. Le contrôle est bloqué en arrivant sur l'instruction **exit l** .

2.10. Parallélisme

Dans l'instruction $i_1 || i_2$, i_1 et i_2 ne partagent aucune variable: l'émission et la réception de signaux est le seul moyen de communication entre i_1 et i_2 .

Exécuter l'instruction $i_1 || i_2$ signifie exécuter un mélange ("interleaving") des instructions i_1 et i_2 . Le contrôle passe en séquence si i_1 et i_2 le passent en séquence toutes les deux.

3. Système de transition

3.1. Actions

Un *environnement* est un couple $E = \langle S, T \rangle$ où

- S est une fonction qui à chaque signal ESTEREL associe une des valeurs $?$, *emis*, *present* ou *absent*. Initialement tous les signaux ont la valeur $?$. Celle ci devient *emis* lorsque le signal est émis pour la première fois. La fixation d'un signal permet de tester sa présence ou son absence. Fixer un signal dont la valeur est $?$ et qui donc n'a pas été émis, signifie lui donner la valeur *absent*. Fixer un signal dont la valeur est *emis* signifie lui donner la valeur *present*.
- T est un ensemble de signaux de terminaison, formé de noms de trappes et éventuellement de *Stop* ou *Seq*.

On note dans la suite E^S la première composante de l'environnement E et E^T sa seconde composante. On note $f[s \leftarrow v]$ la fonction g semblable à f sauf pour $g(s) = v$.

L'ensemble A des actions atomiques est l'ensemble des couples $\langle E, E' \rangle$ d'environnements. On note la transition $i \xrightarrow{\langle E, E' \rangle} i'$ par $\langle i, E \rangle \rightarrow \langle i', E' \rangle$.

3.2. Règles de réécriture

3.2.1. Instruction vide

Le signal de terminaison *Seq* est produit par la réécriture de l'instruction **nothing**.

$$\frac{}{\langle \text{nothing}, E \rangle \rightarrow \langle \text{nothing}, \langle E^S, E^T \cup \{Seq\} \rangle \rangle}$$

3.2.2. Passage d'un instant

Le signal de terminaison *Stop* est produit par la réécriture de l'instruction **pause**. La nouvelle instruction obtenue est **nothing**.

$$\frac{}{\langle \text{pause}, E \rangle \rightarrow \langle \text{nothing}, \langle E^S, E^T \cup \{Stop\} \rangle \rangle}$$

3.2.3. Test de présence de signal

Un signal ne peut être testé que s'il est fixé.

$$\frac{E^S(s) = \text{present}}{\langle \text{present } s \text{ then } i_1 \text{ else } i_2 \text{ end}, E \rangle \rightarrow \langle i_1, E \rangle}$$

$$\frac{E^S(s) = \text{absent}}{\langle \text{present } s \text{ then } i_1 \text{ else } i_2 \text{ end}, E \rangle \rightarrow \langle i_2, E \rangle}$$

3.2.4. Emission de signal

Un signal ne peut pas être émis après avoir été fixé.

$$\frac{E^S(s) \neq \text{absent}, \text{present}}{\langle \text{emit } s, E \rangle \rightarrow \langle \text{nothing}, \langle E^S[s \leftarrow \text{emis}], E^T \rangle \rangle}$$

3.2.5. Déclaration de signal

La valeur d'un signal déclaré localement est mémorisée à l'entrée de la déclaration et est rétablie à la sortie.

$$\frac{}{\langle \text{signal } s \text{ in } i \text{ end}, E \rangle \rightarrow \langle \text{signal}_{E(s)} s \text{ in } i \text{ end}, \langle E^S[s \leftarrow ?], E^T \rangle \rangle}$$

$$\frac{\langle i, E \rangle \rightarrow \langle i_3, E_3 \rangle, E_3^T \neq \{Seq\}}{\langle \text{signal}_\alpha s \text{ in } i \text{ end}, E \rangle \rightarrow \langle \text{signal}_\alpha s \text{ in } i_3 \text{ end}, E_3 \rangle}$$

$$\frac{\langle i, E \rangle \rightarrow \langle i_3, E_3 \rangle, E_3^T = \{Seq\}}{\langle \text{signal}_\alpha s \text{ in } i \text{ end}, E \rangle \rightarrow \langle \text{nothing}, \langle E^S[s \leftarrow \alpha], E^T \rangle \rangle}$$

La première règle correspond à la mémorisation de la valeur du signal et la troisième au rétablissement de son ancienne valeur.

3.2.6. Séquencement

L'instruction $i_1; i_2$ se transforme en i_2 si l'exécution de i_1 passe le contrôle en séquence, c'est-à-dire si le seul signal de terminaison est Seq .

$$\frac{\langle i_1, E \rangle \rightarrow \langle i_3, E_3 \rangle, E_3^T = \{Seq\}}{\langle i_1; i_2, E \rangle \rightarrow \langle i_2, E \rangle}$$

$$\frac{\langle i_1, E \rangle \rightarrow \langle i_3, E_3 \rangle, E_3^T \neq \{Seq\}}{\langle i_1; i_2, E \rangle \rightarrow \langle i_3; i_2, E_3 \rangle}$$

3.2.7. Boucle

Une boucle peut toujours se réécrire en une séquence formée de son corps suivi d'elle même.

$$\frac{}{\langle \text{loop } i \text{ end}, E \rangle \rightarrow \langle i; \text{loop } i \text{ end}, E \rangle}$$

3.2.8. Trappe

Le contrôle passe en séquence de l'instruction $\text{tag } t \text{ in } i \text{ end}$ si i passe le contrôle en séquence ou bien si l' exit le plus externe ayant été exécuté correspond à la trappe t .

$$\frac{\langle i, E \rangle \rightarrow \langle i_1, E_1 \rangle, E_1^T = \phi}{\langle \text{tag } t \text{ in } i \text{ end}, E \rangle \rightarrow \langle \text{tag } t \text{ in } i_1 \text{ end}, E_1 \rangle}$$

$$\frac{\langle i, E \rangle \rightarrow \langle i_1, E_1 \rangle, E_1^T \neq \phi, \sim TR(t, E_1^T)}{\langle \text{tag } t \text{ in } i \text{ end}, E \rangle \rightarrow \langle \text{tag } t \text{ in } i_1 \text{ end}, \langle E_1^S, E_1^T - \{t\} \rangle \rangle}$$

$$\frac{\langle i, E \rangle \rightarrow \langle i_1, E_1 \rangle, TR(t, E_1^T)}{\langle \text{tag } t \text{ in } i \text{ end}, E \rangle \rightarrow \langle \text{nothing}, E \rangle}$$

où le prédicat TR est défini par

$$TR(t, E) \Leftrightarrow E = \{Seq\} \text{ ou } E = \{t\} \text{ ou } E = \{t, Seq\} \text{ ou } E = \{t, Stop\}$$

3.2.9. Exit

L'instruction **exit** t se réécrit en **nothing**; le nom t est ajouté à l'environnement.

$$\frac{}{\langle \text{exit } t, E \rangle \rightarrow \langle \text{nothing}, \langle E^S, E^T \cup \{t\} \rangle \rangle}$$

3.2.10. Fixation de signal

On fixe un signal en lui donnant la valeur *present* s'il a été émis et en lui donnant la valeur *absent* sinon.

$$\frac{E^S(s) = ?}{\langle i, E \rangle \rightarrow \langle i, \langle E^S[s \leftarrow \text{absent}], E^T \rangle \rangle}$$

$$\frac{E^S(s) = \text{emis}}{\langle i, E \rangle \rightarrow \langle i, \langle E^S[s \leftarrow \text{present}], E^T \rangle \rangle}$$

3.2.11. Parallélisme

Exécuter $i_1 \parallel i_2$ consiste à exécuter i_1 ou i_2 tant qu'ils ne terminent pas; le parallèle ne termine que lorsque ses deux branches terminent.

$$\frac{\langle i_1, E \rangle \rightarrow \langle i_3, E_3 \rangle, E_3^T = \phi}{\langle i_1 \parallel i_2, E \rangle \rightarrow \langle i_3 \parallel i_2, E_3 \rangle}$$

$$\frac{\langle i_2, E \rangle \rightarrow \langle i_3, E_3 \rangle, E_3^T = \phi}{\langle i_1 \parallel i_2, E \rangle \rightarrow \langle i_1 \parallel i_3, E_3 \rangle}$$

$$\frac{\langle i_1, E \rangle \rightarrow \langle i_3, E_3 \rangle, \langle i_2, E \rangle \rightarrow \langle i_4, E_4 \rangle, E_3^T \neq \phi, E_4^T \neq \phi}{\langle i_1 \parallel i_2, E \rangle \rightarrow \langle i_3 \parallel i_4, \langle E^S, E_3^T \cup E_4^T \rangle \rangle}$$

La dernière règle indique qu'une instruction parallèle passe le contrôle en séquence uniquement quand ses deux branches le passent également.

La terminaison d'une instruction et la transformation de la valeur d'un signal ne peuvent pas être effectuées lors d'une même réécriture: on a la

Propriété

Si $\langle i_1, E_1 \rangle \rightarrow \langle i_2, E_2 \rangle$ avec $E_1^T = \phi$ et $E_2^T \neq \phi$, alors $E_1^S = E_2^S$.

Cette propriété du système de règles permet de supposer que dans la dernière règle du parallèle, E^S , E_3^S et E_4^S sont égaux.

De même, dans la première règle du séquençement ou dans la dernière de l'instruction **tag**, on ne risque pas "d'oublier" une émission de signal en supposant que l'environnement n'est pas transformé.

4. Sémantique

Un *critère d'observation* \mathbb{O} (ou *critère d'abstraction*) est une partition partielle de l'ensemble des séquences d'actions (cf [Boudol]). On appelle \mathbb{O} -*bisimulation* une relation d'équivalence sur l'ensemble des états d'un système de transition, telle que deux états équivalents ne peuvent pas être distingués par un élément de \mathbb{O} . On note $\sim_{\mathbb{O}}$ la \mathbb{O} -bisimulation la plus grossière (appelée \mathbb{O} -*équipollence*) et on dit que c'est une *congruence* (ou une *sémantique*) si elle est compatible avec la structure algébrique sur laquelle les états sont construits.

4.1. Critère d'observation

L'observation naturelle en ESTEREL (en tant que langage réactif synchrone) est celle qui consiste à observer la transformation de l'environnement lors de chaque instant.

A un instant correspond donc une suite de réécritures à partir d'un environnement E non terminé jusqu'à un environnement terminé F . L'hypothèse de fort synchronisme de [Berry-Cosserat] correspond au critère d'observation \mathbb{C} dont les actions abstraites ont la forme

$$c_{E,F} = \{ \langle E, E_1 \rangle \langle E_1, E_2 \rangle \dots \langle E_{n-1}, E_n \rangle \langle E_n, F \rangle / F^T \neq \phi \ \& \ \forall i \in \{1, \dots, n\} \ E_i^T = \phi \}$$

qui consiste à observer uniquement la transformation d'environnement entre le début et la fin d'un instant.

On note $i \xrightarrow[F]{E} i'$ si il existe une action abstraite $e \in c_{E,F}$ telle que $i \xrightarrow{e} i'$.

L'exécution au cours des divers instants d'une instruction i est alors définie par:

$$i \xrightarrow[F_1]{E_1} i_1 \xrightarrow[F_2]{E_2} \dots \xrightarrow[F_n]{E_n} i_n \xrightarrow[F_{n+1}]{E_{n+1}} \dots$$

Le critère \mathbb{C} ne définit pas une congruence sur ESTEREL: il n'est pas compatible avec l'opérateur de parallélisme. En effet, considérons l'exemple suivant (du à G. Gonthier):

read e ≡ present e then nothing else nothing end

$i_1 \equiv \text{read } e ; \text{emit } f$

$i_2 \equiv \text{emit } f ; \text{read } e$

$i_3 \equiv \text{read } f ; \text{emit } e$

On a

$$i_1 \sim_{\mathbb{C}} i_2$$

mais

$$\exists i \exists F (i_2 \parallel i_3) \xrightarrow[F]{\phi} i \quad \sim \quad \exists i \exists F (i_1 \parallel i_3) \xrightarrow[F]{\phi} i$$

donc

$$i_1 \parallel i_3 \not\sim_{\mathbb{C}} i_2 \parallel i_3$$

4.2. Congruence

On va raffiner le critère \mathbb{C} pour obtenir une congruence \mathbb{D} . Dans ce nouveau critère \mathbb{D} , on observe (en plus de la transformation d'environnement entre le début et la fin d'un instant), les (premières) émissions de signaux ainsi que leur fixation. Rappelons que pour chaque réécriture, au plus un signal peut voir sa valeur transformée.

Soit O l'ensemble des éléments de la forme $!s$ (interprété comme émission du signal s) ou $?s$ (interprété comme fixation de s) où s est un signal ÉSTEREL, et $H = O^*$ l'ensemble des séquences finies d'éléments de O . On définit la fonction μ des séquences d'actions dans H par

- * Fixation de signal: $\mu(\langle E_1, E_2 \rangle) = ?s$ si E_1 est semblable à E_2 sauf pour $E_1^S(s) = ?$ et $E_2^S(s) = absent$ ou $E_1^S(s) = emis$ et $E_2^S(s) = present$
- * Emission de signal: $\mu(\langle E_1, E_2 \rangle) = !s$ si E_1 est semblable à E_2 sauf pour $E_1^S(s) = ?$ et $E_2^S(s) = emis$
- * Action non observable: $\mu(\langle E, E \rangle) = \varepsilon$ (ε désigne la séquence vide).
- * $\mu(a_1 \dots a_n) = \mu(a_1) \dots \mu(a_n)$.

Remarque

Le critère dont les actions abstraites ont la forme

$$p_{E,F,X} = \{e \in c_{E,F} \in \mathbb{C} / \mu(e) = X\}$$

est trop fin: il distingue par exemple les deux programmes

emit e; emit f emit f; emit e

Le premier peut faire des actions abstraites de la forme $p_{E,F,X}$ avec $X = !e!f$ tandis que le second ne le peut pas.

De même, les deux programmes suivants sont distingués

read e; read f read f; read e

La remarque précédente indique qu'on veut faire "commuter" les émissions de signaux entre elles et les fixations de signaux entre elles. Soit T le *monoïde partiellement commutatif* (cf [Perrin] et également la notion reliée de *trace* dans [Mazurkiewicz]) sur O engendré par la relation de commutation c suivante:

$$\langle o_1, o_2 \rangle \in c \iff (o_1 = !s_1 \text{ et } o_2 = !s_2) \text{ ou } (o_1 = ?s_1 \text{ et } o_2 = ?s_2)$$

En d'autres termes, T est le quotient de H par la relation symétrique c .

La définition de T revient à dire que l'on ne tient pas compte de l'ordre des émissions entre deux fixations de signaux, ni de l'ordre des fixations entre deux émissions de signaux. On note la classe d'un élément de H en ôtant les symboles $!$ qui se suivent: $!s_1!s_2 \dots !s_n$ devient $!s_1s_2 \dots s_n$, et en faisant de même pour les symboles $?$ qui se suivent: $?s_1?s_2 \dots ?s_n$ devient $?s_1s_2 \dots s_n$. Par exemple, la classe de

!a!b!c?d!e!f?g?h?i

est notée

!abc?d!ef?ghi

Dans cette notation, l'ordre des lettres entre deux symboles $?,!$ est indifférent. Par exemple

!abc?d!ef?ghi = !bac?d!ef?ihg

Soit e une suite d'actions. On note $\sigma(e) \in T$ la classe de $\mu(e)$.

On peut maintenant définir le critère \mathbb{D} comme étant formé des actions abstraites de la forme

$$d_{E,F,X} = \{e \in c_{E,F} \in \mathbb{C} / \sigma(e) = X\}$$

On note $i \xrightarrow[X]{E,F} i'$ si $i \xrightarrow{e} i'$ avec $e \in d_{E,F,X}$.

Proposition

L'équipollence $\sim_{\mathbb{D}}$ définit une congruence sur ESTEREL.

4.3. Remarque sur la compilation séparée

Permettre la compilation séparée en ESTEREL signifie implémenter une congruence du langage, permettant d'utiliser indifféremment dans tout contexte, un programme (ou plus précisément, un module) ESTEREL ou bien le résultat de sa compilation. Actuellement, l'observation qui est implémentée dans le système v2.1 [v2.1] correspond à $\sim_{\mathbb{C}}$ et ne permet donc pas de compiler séparément un programme ESTEREL quelconque.

Par contre, les modules ESTEREL qui vérifient la restriction suivante pourraient être compilés séparément

Restriction

Il ne doit pas y avoir de test d'un signal d'input après l'émission du premier signal d'output (on interdit toute émission d'un signal d'input). En d'autres termes, à chaque instant, on doit d'abord tester les inputs avant d'émettre les outputs.

En effet, toute observation (qui ne concerne que les signaux d'input ou d'output des modules, cf règle de la définition locale des signaux) a la forme $?i_1 \dots ?i_n !o_1 \dots o_p$ ce qui entraîne que pour deux programmes p_1 et p_2 vérifiant la restriction, on a $p_1 \sim_{\mathbb{D}} p_2$ ssi $p_1 \sim_{\mathbb{C}} p_2$.

Cette restriction revient en fait à interdire les dialogues instantanés (au cours d'un même instant) entre les branches d'un parallèle. Par exemple le programme

```

....
emit QUESTION;
await REPONSE
....
emit RESULTAT
||
....
await QUESTION;
emit REPONSE
....

```

ne vérifie pas la restriction.

4.4. Remarque sur le non déterminisme en ESTEREL

La sémantique $\sim_{\mathbb{D}}$ donne une signification à certains programmes rejetés comme cyclique par [Berry-Cosserat]. Par exemple le programme suivant est cyclique car non déterministe: il a par $\sim_{\mathbb{D}}$ deux significations dans l'environnement vide, suivant que s ou u est fixé (à *absent*) en premier.

```
present s then nothing else emit u end
```

```
||
```

```
present u then nothing else emit s end
```

Le non déterminisme est du à la possibilité de fixer un signal qui ne l'est pas déjà, à n'importe quelle étape de calcul.

Un moyen de rejeter les programmes non déterministes consiste à *retarder* la fixation des signaux: on ne fixe un signal que si par une analyse statique du programme, on est assuré qu'il ne peut plus être émis dans l'instant. Cette règle de retard fournit une implémentation d'ESTEREL consistant à exécuter toutes les réécritures autres que les fixations avant celles-ci (situation de blocage) puis à fixer les signaux ne pouvant plus être émis (déblocage) et ainsi de suite jusqu'à la terminaison de l'exécution dans l'instant. Un programme est rejeté dans cette implémentation si aucun signal ne peut être fixé dans une situation de blocage. Par exemple, le programme précédent est rejeté car toute exécution doit nécessairement commencer par la fixation (à *absent*) de s ou u et ces deux signaux peuvent être tous les deux émis.

Cet algorithme par blocage/déblocage est implémenté dans un compilateur de la version 2.1 d'ESTEREL [vfb2.1].

5. Puissance d'expression

On ne montrera pas que les programmes acceptés par [Berry-Cosserat] ont une sémantique équivalente par $\sim_{\mathbb{D}}$. Par contre, on va exprimer dans ce paragraphe les primitives **await**, **upto** et **upto next** de [Berry-Cosserat] à l'aide de **pause** et **present**.

5.1. Instruction await

On spécifie l'instruction **await** de la manière suivante

$$\frac{E^S(s)=present}{\langle \text{await } s, E \rangle \rightarrow \langle \text{nothing}, \langle E^S, E^T \cup \{Seq\} \rangle \rangle}$$

$$\frac{E^S(s)=absent}{\langle \text{await } s, E \rangle \rightarrow \langle \text{await } s, \langle E^S, E^T \cup \{Stop\} \rangle \rangle}$$

Proposition

await *s*

$\sim_{\mathbb{D}}$

*Att*_{*s*} \equiv **tag** *fini* **in**

loop

present *s* **then** **pause** **else** **exit** *fini* **end**

end

end

Preuve

On montre que **await** *s* \sim *Att*_{*s*}, ce qui entraîne le résultat.

5.2. Instruction upto

On spécifie l'instruction **upto** de la manière suivante

$$\frac{E^S(s)=present}{\langle \text{do } i \text{ upto } s, E \rangle \rightarrow \langle \text{nothing}, \langle E^S, E^T \cup \{Seq\} \rangle \rangle}$$

$$\frac{E^S(s)=absent, \langle i, E \rangle \rightarrow \langle i_1, E_1 \rangle, E_1^T = \phi}{\langle \text{do } i \text{ upto } s, E \rangle \rightarrow \langle \text{do } i_1 \text{ upto } s, E_1 \rangle}$$

$$\frac{E(s)=absent, \langle i, E \rangle \rightarrow \langle i_1, E_1 \rangle, E_1^T \neq \phi}{\langle \text{do } i \text{ upto } s, E \rangle \rightarrow \langle \text{do } i_1 \text{ upto } s, \langle E_1^S, E_1^T \cup \{Stop\} \rangle \rangle}$$

La traduction de **do** *i* **upto** *s* consiste à exécuter en parallèle un processus qui en l'absence du signal *s* émet le signal local *g* et un processus obtenu à partir de *i* en le contrôlant par la réception du signal *g*. Ce contrôle est obtenu en remplaçant dans *i* toutes les occurrences de **pause** par **pause;await** *g* (cette traduction n'est donc pas "structurelle" mais correspond plutôt à un mécanisme de "macro").

Lemme 1

Soit la spécification

$$\frac{\langle i, E \rangle \rightarrow \langle i_1, E_1 \rangle, E_1^T = \phi}{\langle i \langle g \rangle, E \rangle \rightarrow \langle i_1 \langle g \rangle, E_1 \rangle}$$

$$\frac{\langle i, E \rangle \rightarrow \langle i_1, E_1 \rangle, E^T = \{Seq\}}{\langle i \langle g \rangle, E \rangle \rightarrow \langle nothing, E \rangle}$$

$$\frac{\langle i, E \rangle \rightarrow \langle i_1, E_1 \rangle, E^T \neq \emptyset, E^T \neq \{Seq\}}{\langle i \langle g \rangle, E \rangle \rightarrow \langle await\ g; i_1 \langle g \rangle, E_1 \rangle}$$

alors si g n'apparait pas dans i , on a

$$i \langle g \rangle \sim i[\text{pause}; \text{await } g / \text{pause}, \text{exit } t; \text{await } g / \text{exit } t]$$

où la notation $i[a/b]$ désigne la substitution textuelle de a à toutes les occurrences de b .

Lemme 2

$$(\text{await } g; i_1) \parallel (\text{await } g; i_2) \sim \text{await } g; (i_1 \parallel i_2)$$

Proposition

```

do i upto s
  ~D
  signal g in
    tag fini in
      await g; i[pause; await g/pause]
    ||
    loop
      present s then exit fini else emit g; pause end
    end
  end
end
end

```

5.3. Instruction upto next

On spécifie l'instruction **upto next** à l'aide de **upto par**

$$\frac{\langle i, E \rangle \rightarrow \langle i_1, E_1 \rangle, E_1^T = \emptyset}{\langle do\ i\ upto\ next\ s, E \rangle \rightarrow \langle do\ i_1\ upto\ next\ s, E_1 \rangle}$$

$$\frac{\langle i, E \rangle \rightarrow \langle i_1, E_1 \rangle, E_1^T \neq \emptyset}{\langle do\ i\ upto\ next\ s, E \rangle \rightarrow \langle do\ i_1\ upto\ s, E \cup \{Stop\} \rangle}$$

Proposition

```

do i upto next s
  ~D
  signal g in
    tag fini in
      i[pause; await g/pause]
    ||
    pause;
    loop
      present s then exit fini else emit g; pause end
    end
  end
end
end
end

```

Remarque

Les primitives **pause** et **present** permettent d'exprimer facilement les automates produits par la compilation des programmes ESTEREL. Si on autorise ces primitives dans le langage lui-même, la compilation devient fondamentalement une opération d'élimination des parallèles (c'est-à-dire de la communication interne).

6. Conclusion

Un interpréteur/compilateur d'ESTEREL a été réalisé suivant l'approche décrite dans ce papier. Il n'utilise pas la passe statique de construction du graphe de dépendance des signaux implémentée dans le système ESTEREL v2.1. [v2.1]; le test de consistance d'un programme est effectué dynamiquement au cours de la construction de l'automate qui lui est associé.

Ce compilateur vérifie une propriété de complétude: si un programme est observable dans un environnement, alors le compilateur construit une transition correspondant à l'exécution du programme dans cet environnement.

Une version du compilateur rejetant les programmes non déterministes a également été réalisée et interfacée avec le système v2.1 (cf [vfb2.1]).

Remerciements

Ce travail est issu de discussions multiples avec les membres du projet "Parallélisme, Synchronisation et Temps réel" commun à l'INRIA et au CMA. Je tiens à remercier particulièrement G. Boudol, G. Gonthier et R. De Simone pour leur remarques et suggestions.

Bibliographie

- [Berry-Cosserat] G. Berry, L. Cosserat *The Synchronous Programming Language ESTEREL and its Mathematical Semantics*, in "Seminar on Concurrency", S. Brookes and G. Winskel editors, Springer-Verlag LNCS 197 (1985)
- [Boudol] G. Boudol *Notes on Algebraic Calculi of Processes*, Logics and Models of Concurrent Systems (K. Apt Ed.), NATO ASI Series F13, (1985), p. 261-303
- [Mazurkiewicz] A. Mazurkiewicz *Traces, Histories, Graphs: Instances of a Process Monoid*, MFCS 1984, Springer-Verlag LNCS 176 (1984)
- [Perrin] D. Perrin *Words over a Partially Commutative Alphabet*, Combinatorial Algorithms on Words (Apostolics & Galil Eds.) NATO ASI Series F12, (1985), p. 329-340
- [Plotkin] G. Plotkin *A structural approach to operational semantics*, Report Daimi FN-19, Comput. Sci. Dept., Aarhus Univ. (1981)
- [v2.1] G. Berry, P. Couronné, G. Gonthier *ESTEREL v2.1 System Manuals*, Ecole des Mines, Centre de Mathématiques Appliquées, Sophia-Antipolis (1986)
- [vfb2.1] F. Boussinot *ESTEREL vfb2.1 System Manuals*, Ecole des Mines, Centre de Mathématiques Appliquées, Sophia-Antipolis (1986)

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique

