

PH. DARONDEAU

**Une critique de la notion de test de processus fondée sur
la non séparabilité de certaines classes de langages**

Informatique théorique et applications, tome 20, n° 3 (1986),
p. 291-318

http://www.numdam.org/item?id=ITA_1986__20_3_291_0

© AFCET, 1986, tous droits réservés.

L'accès aux archives de la revue « Informatique théorique et applications » implique l'accord avec les conditions générales d'utilisation (<http://www.numdam.org/conditions>). Toute utilisation commerciale ou impression systématique est constitutive d'une infraction pénale. Toute copie ou impression de ce fichier doit contenir la présente mention de copyright.

NUMDAM

Article numérisé dans le cadre du programme
Numérisation de documents anciens mathématiques

<http://www.numdam.org/>

UNE CRITIQUE DE LA NOTION DE TEST DE PROCESSUS FONDÉE SUR LA NON SÉPARABILITÉ DE CERTAINES CLASSES DE LANGAGES (*)

par Ph. DARONDEAU ⁽¹⁾

Communiqué par A. ARNOLD

Résumé. – Une classe de langages est dite séparable s'il y existe pour toute paire de langages distincts L' et L'' un langage L tel que $L \cap L' = \emptyset \Leftrightarrow L \cap L'' \neq \emptyset$. Nous résolvons par la négative la question de la séparabilité pour un large éventail de classes de langages infinitaires, de Alg^ω à Σ_1^1 . Nous prouvons ensuite, pour le calcul des systèmes communicants, qu'il n'existe pas de notion de test qui sépare tous les agents différant par leurs suites infinies d'actions visibles.

Abstract. – A class of languages is separable if, for each pair of non identical languages L' and L'' , there exists some language L such that $L \cap L' = \emptyset \Leftrightarrow L \cap L'' \neq \emptyset$. We give a negative answer to the question of separability for a large variety of classes of infinitary languages, from Alg^ω to Σ_1^1 . We then prove that there exists for the calculus of communicating systems no notion of testing which can separate every pair of agents which differ by their infinite sequences of visible actions.

I. INTRODUCTION

Un processus peut être défini comme un ensemble de suites d'actions allié à un dispositif de production de ces suites, déterministe si l'ensemble est un singleton ou non déterministe. Dans la vision traditionnelle des processus, les actions sont présumées autonomes et le dispositif de production opère indépendamment de tout environnement. En extension, un processus se confond avec un langage, finitaire ou infinitaire, image de l'ensemble de ses suites d'actions par un morphisme qui efface les actions réputées invisibles, et laisse les autres actions inchangées (si l'on suppose une interprétation libre de ces dernières). Dans la vision des systèmes communicants introduits par R. Milner [11], les processus sont de nature plus complexe: le dispositif de

(*) Reçu en juin 1985, révisé en janvier 1986.

(¹) I.R.I.S.A., Campus de Beaulieu, 35042 Rennes Cedex.

production est asservi à un environnement, qu'il conditionne à son tour par le jeu des potentialités d'actions. En extension, un processus est un ensemble de comportements observables, chacun caractérisé par une suite d'actions visibles et par un ensemble de conditions sur l'environnement, nécessaires à la production de cette suite.

Le propos de la présente étude est de déterminer, dans chacun des deux cadres précédents, en quelle mesure des processus qui diffèrent par leur extension peuvent être distingués par des ensembles de tests de même nature que ces processus.

S'agissant de processus autonomes, nous suggérons $|p| \cap |t| \neq \emptyset$ comme critère de la satisfaction du test t par le processus p , où $| \quad |$ indique l'extension. Le problème ci-dessus est donc un problème de séparabilité. Pour chaque classe de langages définie par un type particulier de dispositif de production, le problème s'énonce ainsi : étant donnés deux langages L' et L'' distincts l'un de l'autre, est-il toujours possible de trouver un langage L tel que les assertions $(L \cap L' = \emptyset)$ et $\sim (L \cap L'' = \emptyset)$ soient équivalentes ? Une notion de séparabilité forte peut également être considérée : étant donnés un langage L et un mot u n'appartenant pas à L , peut-on toujours trouver un langage L_u contenant u et disjoint de L ? (Les classes de langages n'étant généralement pas fermées pour l'union dénombrable, ces questions de séparabilité faible ou forte ne sont pas toujours de nature topologique). Nos problèmes de séparabilité ont des solutions immédiates pour les classes de langages finitaires, et nous ne les poserons dans la suite que pour des langages purement infinitaires. Dans ce cadre restreint, la séparation entre L' et L'' resp. entre u et L est immédiate si $\text{Adh}(L') \neq \text{Adh}(L'')$ resp. si $u \notin \text{Adh}(L)$, où $\text{Adh}(L)$ est la clôture de L dans l'espace métrique complet induit par la distance ultramétrique sur les mots [1]. Curieusement cependant, la quasi totalité des classes de langages infinitaires que nous étudierons sont non séparables. Ce résultat négatif vaut pour la plupart des classes construites par fermetures de Kleene [3]. Il vaut aussi pour la classe Σ_1^1 [12], ce qui laisse présager l'insuffisance de toute notion de test pour toute espèce de processus suffisamment généraux.

Dans le cadre des processus communicants, nous limiterons notre étude à la classe connue sous l'appellation C.C.S. Nous nous proposons d'étendre légèrement les notions de test et d'équivalence de test introduites par M. Hennessy et R. de Nicola [5]. Selon ces auteurs, un test est un processus qui s'exécute en parallèle avec le processus testé, les deux processus communiquant et formant un système isolé. Dans ce système où chaque processus est l'environnement de l'autre, certaines exécutions du processus de test sont réputées gagnantes. Le critère de succès est un critère finitaire. Le critère

d'échec est infinitaire: une exécution du processus de test échoue lorsque aucun de ses préfixes n'indique le succès. Deux processus sont équivalents si pour tout test t , chacune des assertions « x satisfait t dans toute exécution » (x must t) et « x satisfait t dans certaines exécutions » (x may t) est simultanément vérifiée ou falsifiée par ces deux processus. Fondés sur cette équivalence de test \simeq ont été proposés pour C.C.S. et pour sa variante synchrone S.C.C.S. des modèles dénotationnels ayant la propriété de pleine abstraction [10], i. e. tels que les significations $\mathcal{M}(p)$ et $\mathcal{M}(q)$ calculées pour p et q soient égales s.s.i. $C[p] \simeq C[q]$ pour tout contexte C de la classe considérée (C.C.S. dans [5], S.C.C.S. dans [6]). La notion de test apparaît ainsi dans sa fonction première qui est celle de critère de validité d'un modèle sémantique. C'est ce critère que nous nous proposons de discuter en montrant qu'un modèle sémantique pleinement abstrait vis-à-vis d'une équivalence de test identifie nécessairement certains processus qui diffèrent par leurs suites infinies d'actions visibles. Nous nous appuyons sur la non séparabilité de Σ_1^1 pour établir ce résultat, valable pour des tests généralisés et sous toutes les hypothèses plausibles d'équité ou de non équité. Le résultat annoncé indique une certaine forme d'incohérence pour toute équivalence de test reposant en partie sur la détection de propriétés des calculs infinitaires. L'équivalence d'observation à la base du modèle construit dans [2] esquive précisément cette critique.

Avant de clore cette introduction, nous tenons à faire remarquer que tout résultat sur la séparabilité des langages concerne aussi la notion générale de test. Soient en effet deux processus autonomes p et p' qui engendrent respectivement des langages L et L' : toute paire $\langle w, w' \rangle$ de mots $w \in L$ et $w' \in L'$ est susceptible d'être produite par leur exécution conjointe. Si le succès du test de p par p' est défini par l'obtention d'une paire $\langle w, w' \rangle$ ayant ses deux éléments identiques, alors « p must p' » est faux si L' n'est pas un singleton, et équivaut sinon à « p may p' », soit encore à $L \cap L' \neq \emptyset$.

La suite du texte est organisée en deux parties. Le chapitre 2 étudie la séparabilité des langages. Le chapitre 3 étudie la séparabilité de la classe des processus C.C.S. Une brève conclusion (§4) souligne, à la lumière des résultats, les limitations inhérentes aux modèles du parallélisme construits sur des domaines algébriques.

II. SÉPARABILITÉ DE CLASSES DE LANGAGES

Après quelques rappels et définitions (§2.1), nous nous intéressons aux classes de langages obtenues par fermeture au sens de Kleene des classes de

la hiérarchie de Chomsky (§ 2. 2), puis à d'autres classes de langages faisant intervenir des opérateurs de mélange finitaire ou infinitaire (§ 2. 3), enfin à la classe Σ_1^1 (§ 2. 4).

II.1. Notations et définitions

Pour tout alphabet dénombrable X , X^* est le monoïde libre sur X , d'élément neutre ε , et X^ω est l'ensemble des mots infinis sur X . On note ${}^\omega$ l'application de $(X^* - \{\varepsilon\})$ dans X^ω qui au mot fini u fait correspondre le mot infini u^ω limite de la suite $(u^i)_{i>0}$. Pour tout couple de mots (u, v) dans $X^* \times (X^* \cup X^\omega)$, uv est le mot obtenu par concaténation de u à gauche de v . Les opérations précédentes se généralisent aux ensembles de mots, ou langages. Une classe de langages sur X est finitaire si tout langage de la classe est un sous-ensemble de X^* , elle est (puremment) infinitaire si tout langage de la classe est un sous-ensemble de X^ω .

Après ces quelques rappels, nous introduisons deux notions de séparabilité applicables aux classes de langages infinitaires. Dans toute la suite, \mathcal{L} resp. \mathcal{L} désigne une classe de langages finitaires resp. infinitaires sur un alphabet X . Par définition, \mathcal{L} est fortement séparable si pour tout L dans \mathcal{L} et pour tout u dans $X^\omega - L$, il existe L_u dans \mathcal{L} tel que $u \in L_u \in X^\omega - L$. \mathcal{L} est (faiblement) séparable si pour tout doubleton $\{L', L''\}$ dans \mathcal{L} , il existe L dans \mathcal{L} tel que $L \cap L' = \emptyset \Leftrightarrow L \cap L'' \neq \emptyset$. Lorsque un tel L existe, nous disons que la paire (L', L'') est séparable et qu'elle est séparée par L .

Nous résolvons ci-dessous la question de la séparabilité pour diverses classes de langages infinitaires, en partant des plus classiques.

II.2. Clôtures de Kleene des FAL de la hiérarchie

Soit \mathcal{L} une classe de langages finitaires sur un alphabet X . Nous notons \mathcal{L}^ω la fermeture de \mathcal{L} au sens de Kleene, ou ensemble des langages définis par des expressions de la forme :

$$\bigcup_{i=1}^n A_i (B_i - \{\varepsilon\})^\omega, \quad A_i \in \mathcal{L}, \quad B_i \in \mathcal{L}.$$

Nous posons dans cette section la question de la séparabilité de \mathcal{L}^ω pour \mathcal{L} variant dans la hiérarchie des familles agréables de langages (FAL). Rappelons qu'une FAL est une classe de langages finitaires fermée pour les opérations d'union ensembliste finie, d'intersection rationnelle, de produit de concaténation et d'étoile (*) ainsi que pour les homomorphismes alphabétiques et leurs inverses. Parmi les FAL les plus connues, citons les familles

Rat et Alg des langages rationnels resp. algébriques, la famille Etol des langages engendrés par tableaux de substitution parallèle, les familles Rec et Recenum des langages récursifs resp. récursivement énumérables. Selon la hiérarchie d'inclusion, ces FAL sont ordonnées comme suit :

$$\text{Rat} \subset \text{Alg} \subset \text{Etol} \subset \text{Rec} \subset \text{Recenum}$$

Les deux théorèmes suivants règlent les problèmes de séparabilité pour cette hiérarchie.

THÉORÈME : *Rat^ω est fortement séparable*

Preuve... car fermée par complémentation d'après les théorèmes de Büchi et de Mac Naughton (cf. [3]) □

THÉORÈME : *Si \mathcal{L} contient Alg, \mathcal{L}^ω n'est pas séparable.*

Preuve : Le principe est de construire un emboîtement de langages algébriques infinitaires ayant même ensemble de mots ultimement périodiques. (On rappelle qu'un mot ultimement périodique est la limite d'une suite non stationnaire de mots finis $(uv^i)_{i>0}$.) Étant données deux lettres a, b de l'alphabet, soient E et F les deux langages algébriques donnés par :

$$E = a^+ b, F = \bigcup_{n>0} E^* a^n b E^* a^n b.$$

Clairement, E^ω et F^ω ont même ensemble de mots ultimement périodiques, et F^ω est inclus dans E^ω . Mais E^ω diffère de F^ω , car $aba^2ba^3b\dots \notin F^\omega$. Soit Fin l'ensemble des langages finitaires.

Supposons l'existence dans Fin^ω d'un langage L qui sépare E^ω de F^ω , c'est-à-dire vérifie $L \cap F^\omega = \emptyset$ et $L \cap E^\omega \neq \emptyset$. Nous montrons ci-dessous que cette supposition est absurde.

Soit $L = \bigcup_{i=1}^m A_i B_i^\omega$ où $A_i, B_i \in \text{Fin}$, et soit $w \in L \cap E^\omega$. Par définition de L , il existe $i \leq m, u \in A_i$ et une suite de mots $v_j \in B_i, j > 0$, tels que $w = uv_1 v_2 \dots v_j \dots$

Puisque w appartient à E^ω , il existe certainement deux entiers k et 1 ($k < 1$) tels que $v_{k+1} \dots v_1$ contienne la lettre b et commence ou finisse par la lettre a : si un facteur $v_p \dots v_q$ de w se termine par b , la lettre initiale de v_{q+1} est a .

Posons $f = uv_1 \dots v_k$ et $g = v_{k+1} \dots v_1$.

Par construction, fg^ω est un mot ultimement périodique commun aux langages $A_i B_i^\omega$ et E^ω . Comme E^ω et F^ω ont même ensemble de mots ultimement

périodiques, fg^{ω} appartient à l'intersection des langages L et F^{ω} , d'où la contradiction. \square

Deux voies restent ouvertes pour découvrir d'autres classes séparables de langages sans modifier radicalement leur procédé de formation : appliquer la fermeture de Kleene à des familles (finitaires) incomparables avec Alg, ou introduire une variante de cet opérateur de fermeture. Ces deux voies sont explorées dans la section suivante.

II.3. Opérateurs de mélange

Nous étudions successivement la fermeture de Kleene de Conc, une classe de langages concurrents qui est construite à l'aide de deux opérateurs de mélange fini [13], puis une variante Alg^{\odot} de Alg^{ω} construite à partir de Alg à l'aide d'un opérateur de mélange infini \odot .

DÉFINITION: $\text{Conc} = (\cdot, \cup, *, \odot, \odot^*) (X \cup \{\varepsilon\})$ est la classe des langages engendrés à partir du mot vide et des lettres de l'alphabet X par les opérateurs concurrents de mélange (\odot) et de mélange itéré (\odot^*) joints aux opérateurs rationnels ($\cdot, \cup, *$). Les opérateurs concurrents sont définis de la façon suivante sur $P(X^*)$:

$$A \odot B = \{u_1 v_1 \dots u_n v_n \mid u_i, v_i \in X^*, u_1 \dots u_n \in A, v_1 \dots v_n \in B\},$$

$$A^{\odot} = \bigcup_{i \geq 0} A^{\odot i}, A^{\odot} = \{\varepsilon\}, A^{i+1} = A^{\odot i} \odot A.$$

PROPRIÉTÉS: $\text{Alg} \not\subseteq \text{Conc}$ [13], $\text{Conc} \not\subseteq \text{Alg}$ [9].

THÉORÈME: Si \mathcal{L} contient Conc, \mathcal{L}^{ω} n'est pas séparable.

Preuve: L'idée est ici encore de construire sur deux lettres a et b un emboîtement de langages concurrents (infinitaires) ayant même ensemble de mots ultimement périodiques. La définition de ces langages fait intervenir les constantes suivantes, à valeur dans Conc:

$$A = ((ab)^{\odot} \odot a^+),$$

$$B = ((ba)^{\odot} \odot b^+),$$

$$C = (a \odot b)^{\odot},$$

$$D = b(a \odot b)^{\odot} a,$$

$$E = a(a \odot b)^{\odot} b.$$

Les langages L et L' que nous allons considérer sont donnés par

$$L = (a^* b^*)^* (A^\omega \cup B^\omega \cup C^\omega),$$

$$L' = (a^* b^*)^* (A^\omega \cup B^\omega) \cup ((a \odot b)^{\odot} \odot (a^+ \cup b^+)) (D^\omega \cup E^\omega).$$

Clairement, L' est inclus dans L , car D et E sont deux sous-ensembles de C qui n'est autre que le Dyck généralisé sur deux lettres. Réciproquement, L n'est pas inclus dans L' : le mot $aba^2b^2 \dots$ est dans $C^\omega - L'$ (lemme 1).

Soit Fin l'ensemble des langages finitaires.

Supposons l'existence de F et G dans Fin vérifiant $FG^\omega \cap L \neq \emptyset$ et $FG^\omega \cap L' = \emptyset$. Soit alors $w \in (L - L')$ un mot admettant la décomposition $uv_1 v_2 \dots v_i \dots$, $u \in F$ et $v_i \in G$, $i > 0$. Nous poursuivons par analyse de cas.

Cas 1 ($\exists i$) ($|v_i|_a > |v_i|_b$)

Soit v l'un des préfixes de v_i pour lequel la valeur $|v|_a - |v|_b$ est minimale, soit $v_i = vv'$ et soit $u' = uv_1 \dots v_{i-1}$.

Alors $v'v \in A$ et $u'v(v'v)^\omega \in L'$, donc $u'v_i^\omega \in FG^\omega \cap L'$, ce qui est contradictoire.

Cas 2 ($\exists i$) ($|v_i|_b > |v_i|_a$)

Ce cas est semblable au précédent.

Cas 3 ($\forall i$) ($|v_i|_a = |v_i|_b$)

Soit v l'un quelconque des mots v_i non vides, alors $uv^\omega \in FG^\omega \cap L'$ (lemme 2), ce qui est à nouveau contradictoire. \square

LEMME 1: Soit $w = aba^2b^2 \dots$, alors $w \notin L'$.

Preuve du lemme: Tout facteur u de w dans A est de la forme $u = a^i b^i \dots a^j b^j a v$, où $v \in a^*$ et $0 \leq i \leq j$. La concaténation $u_1 \dots u_n$ de n mots de cette forme ne peut résulter en un facteur de w que si tous les u_i sauf éventuellement u_1 sont des mots sans b . Or tout facteur droit de w contient une infinité d'occurrences b . Par conséquent, $w \notin (a^* b^*)^* A^\omega$.

Tout facteur u de w dans B est de la forme $u = b^i a^j$, $0 < i$ et $0 \leq j \leq i - 1$. La concaténation $u_1 \dots u_n$ de n mots de cette forme ne peut résulter en un facteur de w que si tous les facteurs u_i sauf éventuellement u_n sont des mots sans a , or tout facteur droit de w contient une infinité d'occurrences a . Par conséquent, $w \notin (a^* b^*)^* B^\omega$.

Supposons $w=uv$ et $v \in D^\omega$, alors $v=v_1 v_2 v_3 \dots$ où les v_i sont des mots de D . Par définition de w , puisque v_1 finit par une occurrence a et que v_3 commence par une occurrence b , v_2 admet une décomposition de la forme $v_2=b^i a^{i+1} \dots b^{i+k} a^{i+k+1}$, $k \geq 0$. On a $|v_2|_a = |v_2|_b + k + 1 > |v_2|_b$ donc $v_2 \notin C$. Comme D est inclus dans C , $v_2 \notin D$ (contradiction).

Supposons $w=uv$ et $v \in E^\omega$, alors $v=v_1 v_2 v_3 \dots$ où les v_i sont des mots de E . Par définition de w , puisque v_1 finit par une occurrence b et que v_3 commence par une occurrence a , v_2 admet une décomposition de la forme $v_2=a^i b^i \dots a^{i+k} b^{i+k}$. Le mot v_1 , qui est à la fois facteur de w et élément du Dyck généralisé, présente alors une forme semblable. Par conséquent, u s'écrit $aba^2 b^2 \dots a^j b^j$, d'où

$$|u|_a = |u|_b \quad \text{et} \quad u \notin (a \odot b)^{\odot} \odot (a^+ \cup b^+) \quad \square$$

LEMME 2: Soit v un mot non vide sur les lettres a et b , tel que $|v|_a = |v|_b$, alors $uv^\omega \in L'$ pour tout $u \in \{a, b\}^*$.

Preuve du lemme: Le mot v^2 admet par hypothèse une décomposition $v^2 = fa^x b^y a^z g$ pour laquelle x, y, z sont tous non nuls.

Si $|uf|_b \neq x + |uf|_a$, posons

$$u' = ufa^x \quad \text{et} \quad v' = b^y a^z gfa^x$$

alors u' et v' appartiennent respectivement à $((a \odot b)^{\odot} \odot (a^+ \cup b^+))$ et à D . Puisque $uv^\omega = u'v'^\omega$, $uv^\omega \in L'$.

Si $x + |uf|_a \neq y + |uf|_b$, posons

$$u' = ufa^x b^y \quad \text{et} \quad v' = a^z gfa^x b^y$$

alors $uv^\omega = u'v'^\omega$ et $u'v'^\omega \in L'$ pour les mêmes raisons que ci-dessus.

Finalement, l'un des deux cas précédents est nécessairement vérifié, car la propriété $y \neq 0$ rend impossible la double égalité

$$|uf|_b = x + |uf|_a = y + |uf|_b \quad \blacksquare$$

Le principe des preuves développées ci-dessus est d'utiliser la propriété suivante des (classes de) langages L défini(e)s par clôture de Kleene: si $w \in L$ alors il existe une infinité de factorisations uv de w telles que $u(v)^\omega \in L$ pour une infinité de facteurs gauches v' de v . Afin d'obtenir des classes séparables, une issue possible est de remplacer l'opération $^\omega$ sur les langages par une variante $^{\odot}$ dans laquelle le mélange joue le rôle usuel de la concaténation.

Aussi introduisons nous ci-dessous deux nouveaux opérateurs de mélange. L'opérateur de mélange infini \odot est à \otimes ce que ω est à $*$. L'opérateur de mélange infinitaire \odot permet de mélanger mots finis et mots infinis en respectant les relations $L^\otimes = L^\omega \odot L^\otimes$.

DÉFINITION: Pour tout alphabet X , nous notons \otimes l'opérateur de $P(X^+)$ dans $P(X^\omega)$ qui à tout langage finitaire L (dépourvu du mot vide) fait correspondre le langage infinitaire L^\otimes déterminé par l'ensemble

$$\{w \in X^\omega \mid \exists f: \mathbb{N} \rightarrow (X \cup \{\varepsilon\}): f(i) = f_i, \exists g: \mathbb{N} \rightarrow \mathbb{N},$$

$$[w = \lim_j (f_{g(0)}(0) f_{g(1)}(1) \dots f_{g(j)}(j))]$$

$$\& [\forall i, \forall j, f_i(j) \neq \varepsilon \Leftrightarrow g(j) = i] \& [\forall i, \lim_j (f_i(0) f_i(1) \dots f_i(j)) \in L]\}.$$

DÉFINITION: Pour tout alphabet X , \odot est l'opérateur de $P(X^*) \times P(X^\omega)$ dans $P(X^\omega)$ qui au couple A, B fait correspondre le langage $A \odot B$ déterminé par l'ensemble:

$$\{w \in X^\omega \mid \exists f, g: \mathbb{N} \rightarrow (X \cup \{\varepsilon\}), \exists h: \mathbb{N} \rightarrow \{f, g\}: h(i) = h_i,$$

$$[w = \lim_j (h_0(0) h_1(1) \dots h_j(j))]$$

$$\& [\forall i, (h(i) = f \Leftrightarrow f(i) \neq \varepsilon) \& (h(i) = g \Leftrightarrow g(i) \neq \varepsilon)]$$

$$\& [\lim_j (f(0) \dots f(j)) \in A \& \lim_j (g(0) \dots g(j)) \in B]\}.$$

DÉFINITION: Pour toute classe \mathcal{L} de langages finitaires on note \mathcal{L}^\otimes la classe des langages infinitaires décrits par des expressions de la forme

$$\bigcup_{i=1}^n A_i \text{op}_i^1 (B_i \text{op}_i^2 (C_i - \{\varepsilon\}))^\otimes$$

où $\{\text{op}_i^1, \text{op}_i^2\} = \{., \odot\}$ et A_i, B_i, C_i varient dans \mathcal{L} pour tout $i \in \{1 \dots n\}$.

THÉORÈME: Si \mathcal{L} contient Alg, \mathcal{L}^\otimes n'est pas séparable.

Preuve: On considère les langages suivants sur $\{a, b, c\}$:

$$L = (ca^+ b)^\otimes \cup \{a, b, c\}^* ba \{a, b, c\}^\otimes$$

$$L' = A \odot (ca^+ b)^\otimes \cup \{a, b, c\}^* ba \{a, b, c\}^\otimes$$

où $A = \bigcup_{n>0} (ca^n b \odot ca^n b)$.

On montre aisément que A est algébrique (lemme 3); L' est par ailleurs inclus dans L (car $B^{\odot} = B^{\odot} \odot B^{\odot}$ pour tout n), mais l'inclusion opposée est invalide puisque le mot $cabca^2bca^3b \dots$ n'appartient pas à L' .

Supposons l'existence de trois langages finitaires F, G, H et d'un mot w dans $\{a, b, c\}^{\omega}$ tels qu'en posant

$$L'' = F \text{op}^1 (G \text{op}^2 (H - \{\varepsilon\})^{\odot}),$$

où $\{\text{op}^1, \text{op}^2\} = \{., \odot\}$, on obtienne

$$L' \cap L'' = \emptyset \text{ et } w \in (L \cap L'').$$

Comme $w \notin L'$, ba n'est pas facteur de w , et w est donc élément de $(ca^+ b)^{\odot}$. Puisque w appartient aussi à L'' et que ce langage est asymptotiquement égal à H^{\odot} , H doit contenir à la fois des mots présentant des occurrences a et des mots présentant des occurrences b (éventuellement les mêmes). Il s'ensuit que ba est facteur de certains mots de H^{\odot} et donc de certains mots de L'' . Or tout mot présentant le facteur ba est dans L' , d'où nécessairement $L' \cap L'' \neq \emptyset$ en contradiction des hypothèses \square

LEMME 3: $A = \bigcup_{n>0} (ca^n b \odot ca^n b)$ est algébrique.

Preuve du lemme: Soit B le langage algébrique: $\bigcup_{n>0} \{ca^n bca^n b\}$,

Alors $A = B \cup C$ où C est le langage reconnu par la grammaire suivante munie de l'axiome S_0 :

$$\begin{aligned} S_0 &\rightarrow c S_1 b, \\ S_1 &\rightarrow a S_1 a + c S_2 + S_3 b, \\ S_2 &\rightarrow a a S_2 + S'_2, \\ S_3 &\rightarrow S_3 a a + S'_3, \\ S'_2 &\rightarrow a S'_2 a + b, \\ S'_3 &\rightarrow a S'_3 a + c. \quad \blacksquare \end{aligned}$$

A ce stade on pourrait envisager d'autres classes de langages infinitaires engendrées à partir des précédentes par certains ensembles spécifiques d'opérateurs. Nous ne poursuivrons pas dans cette voie, ne disposant d'aucun critère général pour retenir certains opérateurs plutôt que d'autres. Aussi,

choisissons-nous de considérer directement dans la section suivante la plus complexe des classes de langages infinitaires susceptibles d'être associés à des mécanismes de production (§ 3), à savoir la classe Σ_1^1 .

II.4. Langages infinitaires de la classe Σ_1^1

Identifions l'alphabet X utilisé jusque à présent et l'ensemble \mathbb{N} des entiers naturels. Un langage (purent) infintaire n'est alors rien d'autre qu'un sous-ensemble de \mathcal{F} , l'ensemble des fonctions totales de \mathbb{N} dans \mathbb{N} , dont l'élément f est le mot $f(0) f(1) \dots$. Le propos de la présente section est d'étudier la séparabilité de la classe des sous-ensemble Σ_1^1 de \mathcal{F} . Nous procédons au préalable, en nous inspirant de l'ouvrage de H. Rogers [12], au rappel des notions indispensables.

Σ_1^1 est par définition la classe des sous-ensembles de \mathcal{F} exprimables sous la forme $\{f \mid (\exists g)(\forall x) R(f, g, x)\}$ où R est une relation récursive à deux variables fonctionnelles et une variable entière. Ces ensembles ne sont qu'un cas particulier d'ensembles analytiques (de fonctions). Est un ensemble analytique tout ensemble obtenu à partir d'une relation récursive par une suite finie d'opérations de projection ou de complémentation. Les ensembles analytiques sont exactement ceux qu'expriment les formes prédictives dans lesquelles des chaînes finies de quantificateurs du premier ou du second ordre sont appliquées à des relations récursives. Les classes de complexité analytique, dont Σ_1^1 , sont déterminées par les alternations de quantificateurs du premier ou du second ordre dans les préfixes de ces expressions. Ces classes sont préservées par les transformations suivantes sur les suites de quantificateurs (\exists^0 et \forall^0 pour le premier ordre, \exists^1 et \forall^1 pour le second ordre) :

$$\begin{aligned} & \left\{ \begin{array}{l} \dots \forall^0 \forall^0 \dots \rightarrow \dots \forall^0 \dots, \\ \dots \exists^0 \exists^0 \dots \rightarrow \dots \exists^0 \dots \end{array} \right. \\ & \left\{ \begin{array}{l} \dots \forall^1 \forall^1 \dots \rightarrow \dots \forall^1 \dots, \\ \dots \exists^1 \exists^1 \dots \rightarrow \dots \exists^1 \dots \end{array} \right. \\ & \left\{ \begin{array}{l} \dots \forall^0 \dots \rightarrow \dots \forall^1 \dots, \\ \dots \exists^0 \dots \rightarrow \dots \exists^1 \dots \end{array} \right. \\ & \left\{ \begin{array}{l} \dots \forall^0 \exists^1 \dots \rightarrow \dots \exists^1 \forall^0 \dots, \\ \dots \exists^0 \forall^1 \dots \rightarrow \dots \forall^1 \exists^0 \dots \end{array} \right. \\ & \left\{ \begin{array}{l} \dots \exists^1 \rightarrow \dots \exists^0, \\ \dots \forall^1 \rightarrow \dots \forall^0. \end{array} \right. \end{aligned}$$

Les classes Σ_1^0 et Π_1^0 sont ainsi caractérisées par les suites réduites de quantificateurs $\exists^0 \forall^0$ et $\forall^0 \exists^0$. De façon analogue, les classes Σ_1^1 et Π_1^1 sont respectivement caractérisées par les suites réduites de quantificateurs $\exists^1 \forall^0$ et $\forall^1 \exists^0$. La classe Σ_1^1 a la propriété d'être fermée par union et intersection finies. La classe Π_1^1 est formée des complémentaires des Σ_1^1 . On note Δ_1^1 la classe des ensembles hyperarithmétiques qui sont à la fois Σ_1^1 et Π_1^1 . Nous disposons à présent des éléments nécessaires à l'établissement du

THÉORÈME : Σ_1^1 n'est pas séparable.

Preuve : Il existe un ensemble Π_1^0 non vide qui n'admet aucun élément Δ_1^1 (Rogers, p. 419, corollaire XLI (b)). Soit X un tel ensemble et soit μ l'élément minimal de X pour l'ordre lexicographique sur les fonctions (ou sur les mots). Posons $Y = X - \{\mu\}$. Puisque X est Π_1^0 , $\{\mu\}$ est Π_1^1 et son complémentaire est Σ_1^1 . Puisque tout ensemble Π_1^0 est aussi Σ_1^1 , Y est Σ_1^1 en tant qu'intersection de deux Σ_1^1 . Maintenant, si X et Y étaient séparables, il existerait Z dans Σ_1^1 tel que $Z \cap X = \{\mu\}$. Ainsi, μ serait à la fois Σ_1^1 et Π_1^1 , ce qui est impossible puisque X n'admet pas d'élément Δ_1^1 . \square

COROLLAIRE : $\Sigma_1^1 \cap P(\{0, 1\}^\omega)$ n'est pas séparable.

Preuve : Soit $\Phi : \Sigma_1^1 \rightarrow P(\{0, 1\}^*)^\omega$ l'extension ensembliste de la fonction

$$\varphi : \mathbb{N}^\omega \rightarrow (0 \{1\}^*)^\omega : \varphi(f) = \psi(f(0)) \cdot \psi(f(1)) \cdot \dots \cdot \psi(f(i)) \cdot \dots,$$

où $\psi : \mathbb{N} \rightarrow (0 \{1\}^*)$ code l'entier i par $\psi(i) = 0^i$.

Clairement, pour tout F dans Σ_1^1 , $\Phi(F)$ est dans Σ_1^1 et réciproquement, pour tout F' dans $\Sigma_1^1 \cap P(\{0, 1\}^*)^\omega$, $\Phi^{-1}(F')$ est dans Σ_1^1 . Soient maintenant F_1 et F_2 deux ensembles non séparables de la classe Σ_1^1 , alors $\Phi(F_1)$ et $\Phi(F_2)$ ne peuvent pas être séparés dans $\Sigma_1^1 \cap P(\{0, 1\}^*)^\omega$. En effet, si H séparait $\Phi(F_1)$ et $\Phi(F_2)$ dans $\Sigma_1^1 \cap P(\{0, 1\}^*)^\omega$, alors $\Phi^{-1}(H \cap \{0, 1\}^*)^\omega$ séparerait F_1 et F_2 dans Σ_1^1 . \blacksquare

Une construction explicite de deux ensembles inséparables X et Y peut être donnée en utilisant la notion de Σ_1^1 -index. Il existe en effet une relation récursive $R \subseteq \mathbb{N} \times \mathcal{F} \times \mathcal{F} \times \mathbb{N}$ telle qu'à chaque ensemble E_z de la classe Σ_1^1 soit associé un index z vérifiant $E_z = \{f \mid (\exists g)(\forall k) R(z, f, g, k)\}$. Posons $G = \{g \mid (\forall i)(\exists f)(\exists j)(f \in E_{g(i)} \cap E_{f(j)})\}$. Soient alors X l'ensemble des fonctions strictement croissantes de G , et Y l'ensemble des éléments non minimaux de X pour l'ordre lexicographique sur les mots. X et Y sont deux ensembles

Σ_1^1 et leur différence $X - Y$ est un singleton μ qui n'est pas Σ_1^1 (confer. rapport I.N.R.I.A., n° 425).

Le chapitre 3 étudie les conséquences, en matière de test des processus communicants, de la non séparabilité de la classe Σ_1^1 .

III. TEST DE PROCESSUS COMMICANTS

Après une brève introduction au calcul des systèmes communicants (§ 3.1), nous montrons que la classe des langages infinitaires définis par les ensembles de suites d'actions des processus de ce calcul est presque exactement Σ_1^1 (§ 3.2), puis nous précisons nos hypothèses sur les tests (§ 3.3) et utilisons les constructions de la section 3.2 afin de prouver la non séparabilité de C.C.S. (§ 3.4).

III.1. Introduction au calcul des systèmes communicants

Nous présentons ci-dessous une version effective du calcul des systèmes communicants (C.C.S.), vu comme un langage applicatif de définition de systèmes de transitions. Syntaxiquement, C.C.S. est une algèbre de termes, auxquels s'identifient les états des systèmes de transitions. Sémantiquement, C.C.S. est un ensemble de règles de transitions, fixant une notion de calcul.

La syntaxe fait intervenir un ensemble d'opérateurs $\Sigma = \cup \{\Sigma_k \mid k \geq 0\}$ qui dépend de deux paramètres : un ensemble Λ de noms d'actions, union disjointe de trois sous-ensembles Δ , $\bar{\Delta}$ et $\{\perp, \tau\}$ respectivement indexés par $2\mathbb{N}+2$, $2\mathbb{N}+3$ et $\{0, 1\}$, et un ensemble REN de renommages sur ces actions, indexé par \mathbb{N} . τ symbolise un événement invisible, produit synchronisé de deux actions $\alpha \in \Delta$ (d'index $2n+2$) et $\bar{\alpha} \in \bar{\Delta}$ (d'index $2n+3$). \perp est le symbole de l'indéfini. Nous utiliserons μ et λ pour désigner les éléments de $\Lambda - \{\perp\}$ et de $\Delta \cup \bar{\Delta}$, avec la convention $\bar{\bar{\lambda}} = \lambda$. Un renommage est une fonction récursive (totale) S de Λ dans Λ telle que

$$\begin{aligned} ((S(\mu) = S(\mu') \Rightarrow (S(\mu) = \perp \vee \mu = \mu')) \\ \& (S(\tau) = \tau) \& (S(\perp) = \perp) \& (S(\lambda) \neq \perp \Rightarrow S(\bar{\lambda}) = \overline{S(\lambda)})). \end{aligned}$$

Pour toute action $\lambda \in \Lambda$, le renommage identique sur $\Lambda - \{\lambda\}$ et indéfini en λ représente ainsi la restriction $/\lambda$ de [11].

Intervient également dans la syntaxe un ensemble X — indexé par \mathbb{N} — de variables utilisées pour les définitions récursives.

Un terme t de C.C.S. est soit une variable $x (\in X)$, soit une expression $\text{op}(t_1, \dots, t_k)$ dans laquelle $\text{op} \in \Sigma_k$ et les t_i sont des termes, soit une définition

récursive $\text{rec}_i \mathbf{x}. \mathbf{t}$ dans laquelle $\mathbf{x}=(x_1 \dots x_n)$ est un vecteur de variables, $\mathbf{t}=(t_1 \dots t_n)$ est un vecteur de termes, et i est inférieur ou égal à n . Un programme C.C.S. est un terme sans variable libre, i. e. un terme qui a toutes ses occurrences de variables liées par des occurrences correspondantes des combinateurs rec_i . La signature Σ de l'algèbre de termes est la suivante :

$$\begin{aligned} \Sigma_0 &= \{\text{NIL}\}, & \Sigma_1 &= \{\mu \mid \mu \in \Lambda\} \cup \{\{S\} \mid S \in \text{REN}\}, \\ \Sigma_2 &= \{+, | \}, & \Sigma_k &= \emptyset \text{ pour } k \geq 3, \end{aligned}$$

où $+$ et $|$ sont les symboles respectifs du choix non déterministe et de la composition parallèle.

Une exécution, ou calcul clos, est une suite finie ou infinie de transitions $p_i \xrightarrow{\tau} p_{i+1}$, pouvant chacune résulter de la synchronisation des mouvements $q'_i \xrightarrow{\bar{\lambda}} q'_{i+1}$ et $q''_i \xrightarrow{\lambda} q''_{i+1}$ de deux composantes parallèles q'_i et q''_i de p_i . Un calcul clos est maximal s'il est infini ou s'il ne peut être prolongé par aucune transition. Les relations $\xrightarrow{\mu}$ sont les plus petites relations binaires (sur les programmes) définies par les axiomes et règles d'inférence 1 à 8 :

1. $\vdash \mu p \xrightarrow{\mu} p$,
2. $p_1 \xrightarrow{\mu} q \vdash p_1 + p_2 \xrightarrow{\mu} q$,
3. $p_2 \xrightarrow{\mu} q \vdash p_1 + p_2 \xrightarrow{\mu} q$,
4. $p_1 \xrightarrow{\mu} q \vdash p_1 | p_2 \xrightarrow{\mu} q | p_2$,
5. $p_2 \xrightarrow{\mu} q \vdash p_1 | p_2 \xrightarrow{\mu} p_1 | q$,
6. $p_1 \xrightarrow{\lambda} q_1, p_2 \xrightarrow{\bar{\lambda}} q_2 \vdash p_1 | p_2 \xrightarrow{\tau} q_1 | q_2$,
7. $p \xrightarrow{\mu} q \vdash p[S] \xrightarrow{S(\mu)} q[S] \text{ — si } S(\mu) \neq \perp \text{ —}$,
8. $t_j [\text{rec}_i \mathbf{x}. \mathbf{t}/x_i]_{i=1}^n \xrightarrow{\mu} q \vdash \text{rec}_j \mathbf{x}. \mathbf{t} \xrightarrow{\mu} q$

(où $t_j [\text{rec}_i \mathbf{x}. \mathbf{t}/x_i]_{i=1}^n$ est le terme obtenu par substitution de $\text{rec}_i \mathbf{x}. \mathbf{t}$ à chacune des occurrences libres de x_i dans t_j , $i=1 \dots n$).

On peut encore imposer aux calculs clos des programmes C.C.S. diverses contraintes d'équité [2,8]. Étant donnée une contrainte d'équité EQ, un EQ-calcul clos est un calcul clos qui satisfait cette contrainte.

III.2. Relations entre C.C.S. et la classe Σ_1^1

Notre premier objectif est de montrer que Σ_1^1 borne la complexité des ensembles de calculs et de traces des programmes C.C.S.

DÉFINITION : Soit \mathcal{C} un (EQ-) calcul maximal clos du programme $p_0 \mid q_0$. Une décomposition parallèle de \mathcal{C} est une paire

$$((p_0 \xrightarrow{\mu'_1} p_1 \xrightarrow{\mu'_2} \dots p_i \xrightarrow{\mu'_{i+1}} \dots), (q_0 \xrightarrow{\mu''_1} q_1 \xrightarrow{\mu''_2} q_2 \dots q_i \xrightarrow{\mu''_{i+1}} \dots))$$

telle que \mathcal{C} soit déductible par les règles 4 à 6 de ces deux suites de mouvements, appelées (EQ-) calculs ouverts (finis ou infinis). La trace $\eta(\mu'_1) \eta(\mu'_2) \dots \eta(\mu'_i) \dots$ du premier de ces deux calculs est définie par $\eta(\lambda) = \lambda$ et $\eta(\tau) = \varepsilon$. Par convention, un calcul ouvert d'extrémité p_n est suffixé par la suite $(p_n \xrightarrow{\perp})^\omega$; de même, une trace finie est suffixée par \perp^ω . \square

Nous supposons désormais fixées pour les actions et programmes deux fonctions de codage quelconques sur \mathbb{N} (à ceci près que 0 code le programme indéfini), et confondrons dans la suite actions et programmes avec leurs codes. Ainsi, l'entier $\langle p, \langle \mu, q \rangle \rangle$ représente le mouvement $p \xrightarrow{\mu} q$ (s'il existe); de même, la fonction f telle que $\pi_1 \circ f(i) = p_i$ et $\pi_2 \circ f(i) = \mu_i$ pour tout i représente le calcul ouvert $p_0 \xrightarrow{\mu_0} p_1 \dots \xrightarrow{\mu_i} p_{i+1} \dots$ (π_1 et π_2 sont les fonctions récursives de décodage des paires $\langle x, y \rangle$, i. e. $\pi_1 \langle x, y \rangle = x$ et $\pi_2 \langle x, y \rangle = y$).

Afin de répondre à certains besoins techniques, nous introduisons maintenant la notion d'arbres de preuve associés aux mouvements des programmes.

DÉFINITION : L'ensemble \mathcal{A} des arbres de preuve est le plus petit ensemble d'entiers qui vérifie les conditions suivantes :

- pour tous μ et p , $\langle \langle 1, 0 \rangle, \langle \mu p, \langle \mu, p \rangle \rangle \rangle \in \mathcal{A}$;
- si $\theta \in \mathcal{A}$ et si $\pi_2(\theta) \vdash \sigma$ est une instance de l'une des règles 2 à 5 ou 7 ou 8 alors $\langle \langle i, \theta \rangle, \sigma \rangle \in \mathcal{A}$, où i est le numéro de cette règle;
- si $\theta_1, \theta_2 \in \mathcal{A}$ et si $(\pi_2(\theta_1), \pi_2(\theta_2)) \vdash \sigma$ est une instance de la règle 6 alors $\langle \langle 6, \langle \theta_1, \theta_2 \rangle \rangle, \sigma \rangle \in \mathcal{A}$. \square

Clairement, \mathcal{A} est un ensemble récursif, car l'ensemble des instances de chacune des règles 1 à 8 est un ensemble récursif. Cette remarque nous permet d'énoncer la...

PROPOSITION : *L'ensemble des calculs ouverts d'un programme C.C.S. est dans Σ_1^1 ainsi que l'ensemble des traces de ces calculs.*

Preuve : Soient $R_\tau(f, g, i) \subset \mathcal{F}^2 \times \mathbb{N}$ et $\bar{R}_\tau(f, i, j, k) \subset \mathcal{F} \times \mathbb{N}^3$ les relations définies par

$$R_\tau(f, g, i) \Leftrightarrow [g(i) \in \mathcal{A} \ \& \ \pi_2 \circ g(i) = \langle \pi_1 \circ f(i), \langle \pi_2 \circ f(i), \pi_1 \circ f(i+1) \rangle \rangle],$$

$$\bar{R}_\tau(f, i, j, k) \Leftrightarrow [\pi_1 \circ f(i) = \pi_1 \circ f(i+1) \ \& \ \langle j, \langle \pi_1 \circ f(i), \langle \tau, k \rangle \rangle \rangle \notin \mathcal{A}].$$

Ces deux relations sont récurrentes.

Soit $R(f, g) \subset \mathcal{F}^2$ (lire « f est un calcul maximal clos prouvé par g ») la relation définie par $(\forall i)(\forall j)(\forall k)$

$$[(\pi_2 \circ f(i) = \tau \ \& \ R_\tau(f, g, i)) \vee (\pi_2 \circ f(i) = \perp \ \& \ \bar{R}_\tau(f, i, j, k))].$$

Si $\langle f, g \rangle$ est la fonction telle que $\langle f, g \rangle(i) = \langle f(i), g(i) \rangle$ pour tout i , alors l'ensemble des paires $\langle f, g \rangle$ correspondant à des éléments (f, g) de R est dans Σ_1^1 . Cette propriété peut être généralisée à l'ensemble des quadruplets $\langle f', f'', f, g \rangle$ tels que (f', f'') soit une décomposition parallèle du calcul maximal clos f prouvé par g , et reste valide lorsque l'on restreint $R(f, g)$ par la contrainte $\pi_1 \circ f(0) = p$, où p est un programme particulier. La proposition suit alors sans difficulté. \square

La prise en compte d'une contrainte d'équité n'entraîne pas de modification des propriétés précédentes. Une telle contrainte peut être décomposée sur les divers agents qui apparaissent dans un calcul clos et qui y sont identifiés par un codage des redex associés à leurs mouvements [7]. (Il existe une fonction récursive qui fait correspondre à chaque transition donnée par son arbre de preuve le ou les redex des agents engagés dans cette transition). En présence d'une contrainte d'équité EQ, la relation $R(f, g)$ de la preuve donnée ci-dessus devient $(\forall i)(\forall j)(\forall k)(\forall l)$

$$[- \ \& \ [EQ_1(f, g, i, l) \Rightarrow EQ_2(f, g, i, l)]]$$

où EQ_1 et EQ_2 spécifient des propriétés relatives à l'agent l dans le i -ième suffixe du calcul f prouvé par g . Or ces propriétés sont à notre connaissance toujours exprimables par quantification au premier ordre de relations récursives.

Afin d'étayer notre discussion ultérieure sur les tests, nous établissons dans la fin de cette section une sorte de réciproque de la proposition précédente. De nouvelles définitions sont indispensables à cet effet.

DÉFINITION : Soit \mathcal{C} un calcul ouvert représenté par une fonction f . \mathcal{C} est maximal si pour tous $i \in \mathbb{N}$ et $\mu \in \wedge - \{\perp\}$, $(\pi_2 \circ f(i) = \perp) \Rightarrow \sim (\pi_1 \circ f(i) \xrightarrow{\mu})$, où $p \xrightarrow{\mu}$ est l'abréviation de $(\exists q)(p \xrightarrow{\mu} q)$. \mathcal{C} est un calcul simple s'il est infini ($\forall i, \pi_2 \circ f(i) \neq \perp$) et si les implications suivantes sont vérifiées pour tous $i \in \mathbb{N}$ et $\lambda \in \Delta \cup \bar{\Delta}$:

- $(\pi_1 \circ f(i) \xrightarrow{\lambda}) \Rightarrow (\pi_2 \circ f(i) = \lambda)$;
- $(\pi_2 \circ f(i) = \lambda) \Rightarrow (\exists j)[j < i] \& (\forall k)[j \leq k \& k < i \Rightarrow (\pi_2 \circ f(k) = \tau)]$.

Un programme p est un programme simple si tout calcul ouvert maximal de ce programme est un calcul simple. \square

Les idées intuitives qui sous tendent ces définitions sont les suivantes. Nous désirons établir la non séparabilité de C.C.S. par les tests comme une conséquence directe de la non séparabilité de Σ_1^1 . Pour ceci, il nous faut déduire une paire de processus inséparables d'une paire de langages inséparables. Mais les processus ne peuvent pas en général être correctement représentés par leurs ensembles de traces ! Aussi rechercherons nous une paire inséparable dans un sous-ensemble restreint de processus, définis par des programmes qui ont la particularité de ne jamais offrir aucune forme de choix à leur environnement : l'équivalence de traces implique pour ces programmes simples l'équivalence de comportement. (La seconde condition de la définition est nécessaire afin que l'implication reste valide sous la contrainte d'équité faible.) Les lemmes ci-dessous formulent plus précisément ces propriétés.

DÉFINITION : Soit p un programme simple et soit \mathcal{C} un calcul ouvert de ce programme. Si \mathcal{C} est infini, \mathcal{C} est sans résidu. Si \mathcal{C} est un calcul fini d'extrémité q et si $q \xrightarrow{\lambda}$, \mathcal{C} a le résidu λ . Sinon \mathcal{C} a le résidu τ .

LEMME 6 : Soient p et q deux programmes simples. Si l'ensemble des traces des calculs simples de p est inclus dans l'ensemble des traces des calculs simples de q , alors à tout calcul ouvert de p correspond un calcul ouvert de q ayant même trace et même résidu.

Preuve : découle directement des définitions \square

DÉFINITION : Étant donné un contexte C et un calcul maximal clos \mathcal{C} de $C[\tau p]$ où p est un programme, on note $\mathcal{C} \setminus C$ la fonction qui à i fait correspondre la suite \mathcal{C}_i des mouvements du $i+1$ -ième agent p apparu en

position active dans \mathcal{C} (sans résulter du déploiement d'un autre agent p); on pose par convention $(\mathcal{C} \setminus C)(i) = 0 \xrightarrow{\perp} 0 \xrightarrow{\perp} \dots$ pour i supérieur ou égal au nombre total de ces agents.

LEMME 7 : *Étant donné un contexte C et un programme simple p , soit \mathcal{C} un calcul maximal clos de $C[\tau p]$. Pour tout programme simple q et pour toute famille de calculs ouverts \mathcal{C}_i de q tels que \mathcal{C}_i et $(\mathcal{C} \setminus C)(i)$ aient même trace et même résidu, on peut former un calcul maximal clos de $C[\tau q]$ en remplaçant les suites de mouvements $(\mathcal{C} \setminus C)(i)$ de p par les suites de mouvements \mathcal{C}_i de q sans autrement altérer le calcul \mathcal{C} .*

Preuve : Laissée au lecteur. \square

Notons que le lemme 7 reste valide sous toutes les contraintes usuelles d'équité. Les lemmes 6 et 7 seront utilisés pour prouver la non-séparabilité de C.C.S. en s'appuyant sur la...

PROPOSITION : *Pour tout langage L de la classe Σ_1^1 inclus dans $\{0, 1\}^\omega$, il existe un programme simple P_L tel que l'ensemble des traces de ses calculs simples soit égal à $L \cup \{0, 1\}^* \perp^\omega$.*

Nous donnons une preuve informelle de cette proposition en ébauchant la construction du programme P_L . Les principales étapes de la construction sont décrites par une série de lemmes démontrés en annexe.

LEMME 8 : *On sait simuler en C.C.S. une pile de booléens avec test des passages à vide.*

LEMME 9 : *Il existe un programme simple FIN tel que l'ensemble des traces de ses calculs simples soit égal à $\{0, 1\}^* \perp^\omega$.*

LEMME 10 : *On sait en C.C.S. simuler une machine de Turing en rendant compte de l'arrêt de la machine par l'arrêt du simulateur.*

LEMME 11 : *Pour tout langage L de la classe Σ_1^1 inclus dans $\{0, 1\}^\omega$, il existe un programme simple q_L tel que l'ensemble Q des traces de ses calculs simples satisfasse $L \subset Q$ et $Q \subset L \cup \{0, 1\}^* \perp^\omega$.*

Preuve de la proposition : $P_L = \tau \text{FIN} + \tau q_L$. \blacksquare

III.3. Hypothèses sur les tests

Étant donné deux programmes p et q , nous appelons q -expérience sur p tout calcul ouvert \mathcal{C}_q intervenant dans la décomposition parallèle $(\mathcal{C}_p, \mathcal{C}_q)$ de

l'un des calculs clos maximaux de $p \mid q$. Puisque les calculs ouverts sont représentés par des fonctions $f \in \mathcal{F}$, tout prédicat total $\mathcal{T}(f)$ à une variable fonctionnelle définit une bipartition {Succès, Échecs} de l'ensemble des calculs ouverts. Nous appelons critère de test un prédicat $\mathcal{T}(f)$ qui s'exprime par quantification des variables du premier ordre d'une relation récursive $R(f, i_1, \dots, i_n)$. Supposons avoir fixé un tel critère. Par définition, $p \text{ must } q$ si l'ensemble des q -expériences \mathcal{C}_q sur p est inclus dans Succès, $p \text{ may } q$ si l'ensemble des q -expériences \mathcal{C}_q sur p n'est pas inclus dans Échecs. Deux programmes p et p' sont indiscernables par test si les assertions $C[p] \text{ sat } q$ et $C[p'] \text{ sat } q$ sont équivalentes pour tout programme q , pour tout contexte $C[]$, pour tout degré de satisfaction $\text{sat} \in \{\text{may}, \text{must}\}$ et pour tout critère de test \mathcal{T} .

Afin de fixer les idées, nous donnons quelques exemples de prédicats

$\mathcal{T}(q_0 \xrightarrow{\mu_1} q_1 \xrightarrow{\mu_2} \dots)$ significatifs : $\exists i, q_i \xrightarrow{\omega} -$ critère utilisé dans [5] —, $\forall i, \mu_i \neq \perp \ \& \ (\exists j > i, \mu_j \neq \tau) -$ critère de coopération infinie —, $\forall i, q_i \xrightarrow{\omega'} \Rightarrow (\exists j > i, q_j \xrightarrow{\omega'}) -$ critère de test conditionnel —. Il est à noter que la notion précise de test développée dans [5] ne rentre pas tout à fait dans le cadre des définitions ci-dessus : sa partie « must » suppose l'inspection directe de certaines propriétés de divergence du programme soumis au test (non détectables par le programme testeur). Néanmoins, si p et q sont indiscernables par test selon nos définitions, une propriété semblable vaut pour les programmes $p \mid \text{rec } x. \tau x$ et $q \mid \text{rec } x. \tau x$ avec les tests de Hennessy et de Nicola.

III.4. Non séparabilité de C.C.S.

Comme annoncé, nous allons appliquer ici les résultats du chapitre 2 à la théorie des tests.

THÉORÈME : *Certains programmes qui diffèrent par l'ensemble des traces associées à leurs calculs ouverts (i.e. qui diffèrent par leurs suites finies ou infinies d'actions visibles) sont néanmoins indiscernables par test.*

La démonstration de ce résultat, indépendant des critères de test, repose sur une série de lemmes qui utilisent les notations proposées ci-dessous.

DÉFINITION : Soit $f : \mathbb{N} \rightarrow \mathcal{F} : i \rightarrow f_i$ une suite de fonctions. On note f^Δ la fonction de \mathbb{N} dans \mathbb{N} que représente le mot $f_0(0)f_0(1)f_1(0)f_0(2)f_1(1)f_2(0)\dots$. Inversement, si $f^\Delta = g$, on pose $g^{\vee i} = f_i$ pour tout i . On étend ces notations aux sous-ensembles F de $\mathcal{F} : F^\Delta$ est l'ensemble

des éléments f^Δ pour lesquels f énumère une partie de F , et $F^{\nabla i}$ est l'ensemble des éléments $f^{\nabla i}$ pour lesquels f appartient à F .

Nous supposons dorénavant avoir choisi un critère de test, fixant ainsi arbitrairement la partition $\{\text{Succès}, \text{Échecs}\}$ de \mathcal{F} .

LEMME 12 : *Étant donnés deux programmes p et q , soit \mathcal{P} l'ensemble des calculs ouverts \mathcal{C}_p de p tels que \mathcal{C}_q appartienne à Succès (resp. Échecs) pour $(\mathcal{C}_p, \mathcal{C}_q)$ l'une des décompositions parallèles de calculs clos maximaux de $p \mid q$, alors \mathcal{P} est dans Σ_1^1 .*

Preuve : Soit $\text{DEC}(f, f', f'', g)$ la relation récursive qui est vérifiée si et seulement si le calcul maximal clos f prouvé par g admet la décomposition parallèle (f', f'') – voir la preuve de la première proposition donnée dans la section 3.2 –. On a alors

$$\mathcal{P} = \{f' \mid (\exists f)(\exists f'')(\exists g)[\mathcal{T}(f'') \& \text{DEC}(f, f', f'', g)]\},$$

où $\mathcal{T}(f'')$ exprime le succès ou l'échec de f'' par un prédicat de la forme

$$(\forall \exists_1 i_1) \dots (\forall \exists_n i_n) R(f'', i_1, \dots, i_n).$$

Or chacun des préfixes $(\exists f)(\exists f'')(\exists g)(\forall \exists_1 i_1) \dots (\forall \exists_n i_n)$, où $\forall \exists_j \in \{\forall, \exists\}$, admet la forme réduite $\exists^1 \forall^0$ \square

LEMME 13 : *Pour tout contexte C et pour tout programme p , si Q est un ensemble Σ_1^1 de calculs ouverts de $C[\tau p]$, alors $\{(\mathcal{C} \setminus C)^\Delta \mid \mathcal{C} \in Q\}$ est dans Σ_1^1 .*

Preuve : On décore les règles opérationnelles de C.C.S. par un jeu d'exposants et d'indices, respectivement utilisés pour totaliser le nombre des agents p apparus à une certaine étape d'un calcul et pour distinguer ces agents les uns des autres. Les indices sont créés avec les agents p et sont hérités textuellement par leurs descendants. Les agents qui ne dérivent pas de p ne portent pas d'indice. On introduit les variantes suivantes des règles d'action :

$$(\tau p)^{\langle i \rangle} \xrightarrow{\tau} (p)^{\langle i+1 \rangle} \text{ – création d'indice –}$$

$$(\mu q)^{\langle i \rangle}_j \xrightarrow{\mu} (q)^{\langle i \rangle}_j \quad \text{si } q \neq p.$$

Les règles d'inférence sont adaptées en conséquence.

De

$$t^{\langle i \rangle} \xrightarrow{\alpha} t'^{\langle i+1 \rangle}, \quad u^{\langle i \rangle} \xrightarrow{\alpha} u'^{\langle i \rangle} \quad \text{et} \quad v^{\langle i \rangle} \xrightarrow{\bar{\alpha}} v'^{\langle i \rangle}$$

se déduisent ainsi

$$(u \mid v)^{\langle i \rangle} \xrightarrow{\tau} (u' \mid v')^{\langle i \rangle} \quad \text{et} \quad (t \mid v)^{\langle i \rangle} \xrightarrow{\tau} (t' \mid v')^{\langle i+1 \rangle}.$$

Étant donné ce nouveau système de règles d'action, soit Q' l'ensemble des suites de mouvements qui se réduisent par oubli des indices et exposants à des calculs de Q . Si Q est dans Σ_1^1 , alors Q' l'est également, et une vérification de routine permet d'établir le lemme à partir de cette propriété. \square

LEMME 14 : *Il existe dans la classe Σ_1^1 deux ensembles non séparables X et Y tels que X^Δ et Y^Δ soient non séparables.*

Preuve : Nous savons qu'il existe dans la classe Σ_1^1 deux ensembles non séparables X et Y tels que X contienne Y et que $X - Y$ soit un singleton $\{\mu\}$. Nous prouvons que les conditions du lemme sont vérifiées pour ces ensembles.

Nous montrons d'abord que pour tout A dans Σ_1^1 , A^Δ est aussi dans Σ_1^1 . Pour cela, on note $\Delta(i, j)$ le rang occupé par la paire (i, j) dans l'énumération usuelle $(0,0), (0,1), (1,0), (0,2), (1,1), (2,0)...$ de $\mathbb{N} \times \mathbb{N}$.

Clairement, Δ est une fonction récursive. Soit alors

$$A = \{ f \mid (\exists g)(\forall i) R_A(f, g, i) \}$$

où R_A est récursive. On a

$$A^\Delta = \{ f \mid (\forall j)(\exists f)(\exists g)(\forall i)[R_A(f, g, i) \& f(\Delta(j, i)) = f(i)] \}.$$

Le préfixe $\forall^0 \exists^1 \exists^1 \forall^0$ de cette expression se réduit comme prévu à $\exists^1 \forall^0$.

Nous adoptons pour la suite de la démonstration le procédé du raisonnement par l'absurde, en supposant l'existence dans Σ_1^1 d'un ensemble E tel que $E \cap X^\Delta \neq \emptyset$ et $E \cap Y^\Delta = \emptyset$. Puisque $E \cap X^\Delta$ est dans Σ_1^1 nous pouvons supposer $E \subset X^\Delta$ sans perte de généralité. Maintenant, pour toute fonction $e \in E$, si l'on pose $e_i = e^{\forall i}$ pour tout i , l'ensemble $\{e_0 \dots e_i \dots\} \cap (X - Y)$ est nécessairement le singleton $\{\mu\}$, c.a.d. l'ensemble $X - Y$! L'idée consiste alors à utiliser l'identité

$$X - Y = \{ h \mid (\exists e \in E)(\forall i)[(\sim(e_i \in Y)) \Rightarrow (e_i = h)] \}$$

pour montrer que ce singleton est dans Σ_1^1 , d'où contradiction puisque X et Y sont non séparables.

Posons

$$E = \{ e \mid (\exists g)(\forall j) R_E(e, g, j) \}$$

et

$$Y = \{ h \mid (\exists f)(\forall j) R_Y(h, f, j) \}$$

où R_E et R_Y sont deux relations récursives. On a

$$X - Y = \{ h \mid (\exists e)(\exists g)(\forall j)(\forall i)(\exists e_i)(\forall k)(\exists f)(\forall m)(\forall n) [R_E(e, g, j) \& e_i(k) = e(\Delta(i, k)) \& [R_Y(e_i, f, m) \vee (e_i(n) = h(n))]] \}.$$

Le préfixe $\exists^1 \exists^1 \forall^0 \forall^0 \exists^1 \forall^0 \exists^1 \forall^0 \forall^0$ de cette forme prédicative se réduit en $\exists^1 \forall^0$, d'où la démonstration. \square

COROLLAIRE : Soit $\Phi : \Sigma_1^1 \rightarrow P((0\{1\}^*)^\omega)$ l'extension ensembliste de la fonction $\varphi : \mathbb{N}^\omega \rightarrow (0\{1\}^*)^\omega$:

$$\varphi(f) = \psi(f(0)). \psi(f(1)). \dots \psi(f(i)). \dots,$$

où $\psi : \mathbb{N} \rightarrow (0\{1\}^*)$ code l'entier i par $\psi(i) = 0^i 1$.

Alors $(\Phi(X))^\Delta$ et $(\Phi(Y))^\Delta$ sont deux ensembles non séparables de la classe Σ_1^1 .

(Même démonstration que ci-dessus, en utilisant le fait que $\Phi(X)$ et $\Phi(Y)$ sont deux ensembles non séparables de la classe Σ_1^1 .)

Forts de cette série de lemmes, nous abordons maintenant la partie centrale de la...

Démonstration du théorème : Soient $P = \Phi(X)$ et $Q = \Phi(Y)$, où X et Y sont les ensembles de fonctions du lemme 14. Puisque P et Q sont dans Σ_1^1 , on peut faire correspondre à ces ensembles de chaînes binaires deux programmes simples p et q admettant respectivement $P \cup \{0,1\}^* \perp^\omega$ et $Q \cup \{0,1\}^* \perp^\omega$ comme ensembles des traces de leurs calculs simples. Notre intention est de montrer que τp et τq ne sont alors pas distinguables par test, ce qui suffit à prouver le théorème puisque $P - Q$ est non vide.

Supposons τp et τq séparables. Soient alors C, t et sat les contexte, programme de test et degré de satisfaction qui les distinguent, i.e. vérifient $C[\tau p] sat t \not\Leftarrow C[\tau q] sat t$. Posons $x m\phi y$ si $\sim (x may y)$ alors nous pouvons supposer $sat \in \{ m\phi y, must \}$.

En vertu des lemmes 6 et 7, puisque Q est inclus dans P , on a alors

$$C[\tau p] sat t \Rightarrow C[\tau q] sat t.$$

Donc $C[\tau q] sat t \not\Leftarrow C[\tau p] sat t$.

Soit \mathcal{P} l'ensemble des calculs ouverts \mathcal{C}_p de $C[\tau p]$ pour lesquels l'un des calculs clos maximaux de $(C[\tau p]|t)$ se décompose en $(\mathcal{C}_p, \mathcal{C}_t)$ avec \mathcal{C}_t dans Echecs si $sat = must$, ou dans Succes si $sat = m\#y$. Par les lemmes 12 et 13, l'ensemble \mathcal{P}' égal à $\{(\mathcal{C} \setminus C)^\Delta \mid \mathcal{C} \in \mathcal{P}\}$ est dans Σ_1^1 . Pour toute fonction f , soit $traces(f)$ la fonction telle que $(traces(f))^{\forall i}$ soit égale pour tout i à la trace du calcul ouvert représenté par $f^{\forall i}$ (s'il existe). Alors clairement, l'ensemble $\mathcal{P}'' = \{traces(f) \mid f \in \mathcal{P}'\}$ est dans Σ_1^1 .

Maintenant, puisque p et q sont des programmes simples qui ont les mêmes traces finies, l'assertion $(C[\tau q] \text{ sat } t \neq C[\tau p] \text{ sat } t)$ entraîne par le lemme 7 la conclusion $\mathcal{P}'' \cap (Q \cup \{0, 1\}^* \perp^\omega)^\Delta = \emptyset$, tandis que $\mathcal{P}'' \cap (P \cup \{0, 1\}^* \perp^\omega)^\Delta \neq \emptyset$ par construction. Soit R l'ensemble de traces infinies donné par

$$\{g \mid (\exists f \in \mathcal{P}'') (\exists i) (\forall j) (g = f^{\forall i} \& g(j) \neq \perp)\}.$$

R et R^Δ sont deux ensembles de la classe Σ_1^1 , et R^Δ sépare nécessairement P^Δ et Q^Δ , ce qui est impossible. ■

Remarque : La non séparabilité a été démontrée pour la version standard (non équitable) de C.C.S. Un examen détaillé montre que la démonstration qui a été donnée reste valide en remplaçant la notion de calcul clos par celle d'EQ-calcul clos pour l'une quelconque des contraintes usuelles d'équité.

IV. QUELQUES CONCLUSIONS

Parvenus au terme d'une étude dans laquelle nous avons amalgamé langages et processus, il nous reste à justifier ce rapprochement et à tirer les enseignements de nos résultats quant aux modèles du parallélisme.

L'exemple de la séparabilité montre que certaines propriétés essentielles des classes de langages infinitaires sont à ce jour insuffisamment explorées : celles dont la transposition finitaire est triviale. La considération de ces propriétés peut être motivée par des questions liées aux théories des processus communicants et de leur observation, sources possibles d'un enrichissement de la problématique des langages. (Le peu de méthodes disponibles pour former des classes de langages infinitaires à partir de classes finitaires montre en outre l'intérêt des calculs de processus communicants pour la définition directe de ces classes).

Le rapport entre (classes de) processus et (classes de) langages infinitaires est plus étroit que ne l'indiquent les remarques précédentes. Nous donnons ci-dessous quelques arguments à l'appui de la thèse selon laquelle l'extension

d'un processus est ou s'apparente à un langage infinitaire dans toute sémantique « observationnelle » des calculs de processus communicants.

Un temps infini est certainement nécessaire au constat du refus continu d'une action par un processus. Si les refus de ce type sont observables, alors les suites infinies d'actions d'un processus le sont aussi, car leur observation ne demande pas davantage de temps. Les domaines ω -algébriques [4,14] ne sont donc pas appropriés à la construction de sémantiques « observationnelles » des calculs de processus. Tout objet d'un domaine ω -algébrique est par définition la borne supérieure de ses approximants finis, représentant autant de propriétés finies. Or on connaît des processus qui présentent exactement les mêmes propriétés finies (mêmes suites et ensembles d'actions possibles, garanties ou refusées...) et qui diffèrent uniquement par leurs suites infinies d'actions visibles (e. g. les processus p et q considérés dans la preuve du dernier théorème).

Comme la notion d'algébricité, la notion d'ordre d'amélioration [4,14] qui sous-tend la plupart des travaux classiques en sémantique dénotationnelle nous semble mal appropriée à la modélisation des calculs de processus. Un préordre fondé sur l'observation est défini par un critère qui permet de juger, étant donnés deux processus, si l'un d'eux coopère plus ou mieux que l'autre avec un processus tiers (l'expérimentateur). Ce critère peut être simplement l'inclusion des ensembles de comportements respectivement permis au processus tiers par les processus soumis à la comparaison : le préordre considéré est alors un préordre d'implémentation. Dans tout autre cas de figure, le critère utilisé est un critère de test car il est nécessaire d'attribuer à chaque exécution du processus expérimentateur un poids dans une échelle de succès ou d'échec. Les processus qui ne sont pas distingués par test ont alors même représentation dans le modèle du préordre, or nous avons montré que certains de ces processus diffèrent par leurs suites infinies d'actions visibles. Dans le cas d'un préordre d'implémentation, le domaine sémantique le plus naturel est un domaine ensembliste, ordonné par inclusion, dans lequel les éléments des ensembles représentent les propriétés observables des processus. Puisque les suites infinies d'actions visibles des processus sont des propriétés observables, ces ensembles sont ou s'apparentent à des langages infinitaires. Le modèle équitable de C.C.S. présenté dans [2] a été construit selon ce principe.

ANNEXE

Nous donnons ici les preuves des lemmes 8 à 11. La notation $(x_1 = t_1, \dots, x_n = t_n)$ doit être interprétée comme

$$\text{rec}_1(x_1 \dots x_n) \circ (t_1 \dots t_n).$$

S_0, S_1, S_2 et S_3 désignent les renommages les moins définis tels que $S_0(\alpha) = S_1(\alpha) = S_2(\alpha) = \alpha$ et $S_3(\alpha) = \alpha'$ pour $\alpha \in \{0, 1\}$, $S_0(2) = 2$ et $S_2(2) = S_3(2) = 2'$.

Preuve du lemme 8 : Si 2 est le signal de détection du passage à vide, le comportement recherché est celui du programme PILE donné par la définition récursive

$$\begin{aligned} (z = ((1 \cdot (x_1 | x[S_3]) + 0 \cdot (x_0 | x[S_3]))[S_2] | 2' \cdot \bar{2} \cdot z)[S_0], \\ x_0 = \bar{0} \cdot (2' \cdot x + 1' \cdot x_1 + 0' \cdot x_0) + 0 \cdot (\bar{0}' \cdot x_0) + 1 \cdot (\bar{0}' \cdot x_1), \\ x_1 = \bar{1} \cdot (2' \cdot x + 1' \cdot x_1 + 0' \cdot x_0) + 0 \cdot (\bar{1}' \cdot x_0) + 1 \cdot (\bar{1}' \cdot x_1), \\ x = \bar{2} \cdot \text{NIL} + 0 \cdot (x_0 | x[S_3]) + 1 \cdot (x_1 | x[S_3])). \end{aligned}$$

Preuve du lemme 9 : Le comportement recherché est celui du programme donné par la définition récursive

$$\begin{aligned} (z = (\text{PILE}[S_3] | x_0)[S_1], \\ x_0 = \tau \cdot \bar{0}' \cdot x_1 + \tau \cdot \bar{1}' \cdot x_1 + \tau \cdot y_1, \\ x_1 = \tau \cdot \bar{0}' \cdot x_1 + \tau \cdot \bar{1}' \cdot x_1 + \tau \cdot y_0, \\ y_0 = 0' \cdot 0 \cdot y_0 + 1' \cdot 1 \cdot y_0 + 2' \cdot y_1, \\ y_1 = \tau \cdot y_1). \end{aligned}$$

Preuve du lemme 10 : Il suffit de construire un programme C.C.S. dans lequel coopèrent un automate d'états finis, qui représente le programme de la machine et mémorise son état programme, et deux piles qui représentent la bande de la machine, scindée en ses portions à gauche et à droite de la tête de lecture (PILE[T_g] et PILE[T_d]). Or la correspondance entre automates d'états finis et systèmes de définitions récursives est évidente. L'arrêt de la machine avec le résultat n peut en particulier être simulé par une transition C.C.S. dont le terme d'arrivée est observationnellement équivalent [11] à un terme de la forme : (PILE[T_g] | ($z = \text{start} \dots$)) | (PILE[T_d]), dans lequel ($z = \text{start} \dots$) est l'automate d'états finis en attente du lancement d'une nouvelle exécution par la commande start, et PILE $\xrightarrow{\bar{0}} (\bar{1})^{n+1}$ PILE[T_d].

Preuve du lemme 11 : A chaque langage A de la classe Σ_1^1 , on sait faire correspondre une relation récursive R_A telle que

$$A = \{ f \mid (\exists h)(\forall i) R_A(f[0 \dots i], h[0 \dots i]) \},$$

où $g[0 \dots i] = \langle g(0), \langle g(1), \dots, g(i) \rangle \dots \rangle$.

Notons $\hat{h}(j)$ la chaîne $1^{1+h(j)}$ et posons :

$$h \text{ jusque } i = \hat{h}(0).0.\hat{h}(1).\dots.\hat{h}(i).0.$$

Si f varie dans $\mathbb{N} \rightarrow \{0, 1\}$, l'ensemble de chaînes

$$\{ f(0).f(1).\dots.f(i).h \text{ jusque } i \mid R_L(f[0 \dots i], h[0 \dots i]) \},$$

où R_L correspond au langage L , est récursif. Soit Φ_L sa fonction caractéristique, et soit PHI_L l'automate ($z = \text{start} \dots$) de la machine de Turing qui calcule Φ_L .

Alors le programme q_L a la structure

$$(\text{PILES}_g \mid (\text{PHI}_L \mid \text{CONTROLE}) \mid \text{PILES}_d) [S_1],$$

où

$$\text{PILES}_g = \text{PILE}[T_g] \mid \text{PILE}[F_g] \mid \text{PILE}[H_g]$$

et

$$\text{PILES}_d = \text{PILE}[T_d] \mid \text{PILE}[F_d] \mid \text{PILE}[H_d].$$

Le couple $(\text{PILE}[T_g], \text{PILE}[T_d])$ simule la bande de la machine de Turing. Les autres jeux de piles appariées simulent deux bandes auxiliaires utilisées pour représenter respectivement $f(0) \dots f(i)$ et h jusque i . Afin de fixer les notations, posons $S_y(\alpha) = \alpha_y^s$ pour $S \in \{T, F, H\}$ et $y \in \{g, d\}$. Le paragraphe suivant donne quelques indications sur l'algorithme du programme **CONTROLE** donné à la fin du texte.

A l'initialisation, la métavariable i vaut 0 et toutes les piles sont vides. Au début de la $n+2$ -ième étape, la métavariable i vaut $n+1$, les données $f(0) \dots f(n)$ et h jusque n sont rangées dans les piles de gauche réservées à f et h , et les autres piles sont vides. Pour chacune des valeurs successives de i , les opérations suivantes sont effectuées :

1. choisir aléatoirement $f(i)$ et l'empiler à gauche;
2. choisir aléatoirement $h(i)$ et empiler à gauche $\hat{h}(i).0$;

3. recopier h jusque i sur la bande de la machine de Turing (PILE [T_d]) et dupliquer cette donnée dans la pile de droite réservée à h ;

4. recopier $f(0) \dots f(i)$ sur la bande de la machine de Turing et dupliquer cette donnée dans la pile de droite réservée à f ;

5. lancer l'exécution de PHI_L (commande start) et attendre le résultat; si ce résultat est 0 (codé par 1.0 dans la pile), adopter le comportement purement divergent (τ^0), sinon passer à la suite;

6. transférer le contenu des piles de droite dans les piles de gauche;

7. imiter le comportement $f(i)$.

CONTROLE est donné par la définition récursive

$$\begin{aligned} x_1 &= \bar{0}_g^f \cdot x_2 + \bar{1}_g^f \cdot x_2, \\ x_2 &= \bar{1}_g^h \cdot y_2, y_2 = \bar{1}_g^h \cdot y_2 + \bar{0}_g^h \cdot x_3, \\ x_3 &= 0_g^h \cdot \bar{0}_d^h \cdot \bar{0}_d^t \cdot x_3 + 1_g^h \cdot \bar{1}_d^h \cdot \bar{1}_d^t \cdot x_3 + 2_g^h \cdot x_4, \\ x_4 &= 0_g^f \cdot \bar{0}_d^f \cdot \bar{0}_d^t \cdot x_4 + 1_g^f \cdot \bar{1}_d^f \cdot \bar{1}_d^t \cdot x_4 + 2_g^f \cdot x_5, \\ x_5 &= \text{start} \cdot 1_d^t \cdot (0_d^t \cdot y_5 + 1_d^t \cdot 0_d^t \cdot x_6), y_5 = \tau \cdot y_5, \\ x_6 &= 0_d^h \cdot \bar{0}_g^h \cdot x_6 + 1_d^h \cdot \bar{1}_g^h \cdot x_6 + 2_d^h \cdot y_6, \\ y_6 &= 0_d^f \cdot \bar{0}_g^f \cdot y_6 + 1_d^f \cdot \bar{1}_g^f \cdot y_6 + 2_d^f \cdot x_7, \\ x_7 &= 0_g^f \cdot \bar{0}_g^f \cdot 0 \cdot x_1 + 1_g^f \cdot \bar{1}_g^f \cdot 1 \cdot x_1. \end{aligned}$$

REMERCIEMENTS

Je tiens à remercier ici L. Kott et S. Yoccoz, avec lesquels a été conduite une étude préliminaire de la notion de séparabilité forte pour quelques classes de la hiérarchie.

BIBLIOGRAPHIE

1. L. BOASSON et M. NIVAT, *Adherences of Languages*, Comp. Syst. Sc., vol. 20, 1980, p. 285-309.
2. Ph. DARONDEAU, *About Fair Asynchrony*, T.C.S., vol. 37, 1985, p. 305-336.
3. S. EILENBERG, *Automata, Languages and Machines*, vol. A, Academic Press, 1974.
4. I. GUESSARIAN, *Algebraic Semantics*, Springer-Verlag, LNCS, 99, 1981.
5. M. HENNESSY et R. DE NICOLA, *Testing Equivalences for Processes*, Theor. Comp. Sc., vol. 34, 1984, p. 83-134.
6. M. HENNESSY, *Synchronous and Asynchronous Experiments on Processes*, Inform. Contr., vol. 59, 1983, p. 36-83.

7. M. HENNESSY, *Modelling Finite Delay Operators*, University of Edinburgh, Dept of Computer Science, Internal Report CSR 153-183, 1983.
8. M. HENNESSY, *An Algebraic Theory of Fair Asynchronous Communicating Processes*, ICALP 85, Springer-Verlag LNCS 194, ed. W. Brauer, 1985.
9. T. KIMURA, *An Algebraic System for Process Structuring and Interprocess Communication*, proc. 8th Ann. ACM Symposium on Theory of Computing, 1976, p. 92-100.
10. R. MILNER, *Fully Abstract Models of Typed Lambda-Calculi*, Theor. Comp. Sc., vol. 4, 1977, p. 1-23.
11. R. MILNER, *A Calculus of Communicating Systems*, Springer-Verlag LNCS, 92, 1980.
12. H. ROGERS, *Theory of Recursive Functions and Effective Computability*, McGraw-Hill, 1967.
13. A. C. SHAW, *Software Descriptions with Flow Expressions*, IEEE Trans. Soft. Engineer., vol. 3, 1978, p. 242-254.
14. J. E. STOY, *Denotational Semantics*, M.I.T. Press, 1977.