CONCEPTION ET MISE EN ŒUVRE D'UN SYSTEME DE PREUVE POUR UN SOUS ENSEMBLE DE CCS SANS TRANSMISSION DE VALEURS

A. SORIANO * - J. VOIRON **

RESUME

Nous présentons les principes et la réalisation d'un système de preuve pour un sous ensemble du calcul CCS (Calculus of Communicating Systems), permettant de comparer deux descriptions d'un système communiquant. Nous considérons successivement les cas des comportements finis et des comportements infinis réguliers, sans transmission de valeur. Le système est réalisé en PROLOG-C sur VAX.

ABSTRACT

We present the principles and the realization of a proof system for a subset of the Calculus of Communicating Systems CCS. We consider finite terms and infinite terms without value passing. This system permets the comparison of two system descriptions. The system is implemented in PROLOG-C on VAX.

* Licenciado en Computación (U.C.V. 1977); sémantique, spécification, preuve, construction de programmes; systèmes communicants; intelligence artificielle. Universidad Central de Venezuela, actuellement dans le Groupe Spécification et Analyse des Systèmes Informatiques, Laboratoire Génie Informatique, IMAG .

** Maître Assistant. Université de Grenoble. Thèse 1976; compilation, sémantique, spécification, preuve, construction de programmes; systèmes communicants. Groupe Spécification et Analyse de Systèmes Informatiques. Laboratoire de Génie Informatique, IMAG.

LGI - IMAG BP 68 38410 St. Martin d'Hères Cedex France

INTRODUCTION.

Les comportements asynchrones de systèmes communicants peuvent être décrits d'une manière élégante comme termes d'un calcul. On peut alors définir dans ce calcul différentes notions d'équivalence entre comportements. Nous proposons la définition d'un système de preuve basé sur ce principe.

Nous considérons ici le calcul CCS (Calculus of Communicating Systems) proposé par R. Milner [Mi 80]. Nous nous intéressons à deux classes particulières de comportements:

- les comportements finis,
- les comportements infinis de systèmes de transitions décrits par des définitions récursives rationnelles.

Le système développé permet de prouver l'équivalence de deux comportements représentés par des expressions CCS sans transmission de valeurs; dans une démarche de conception descendante, on peut comparer la description d'un système communiquant à sa spécification. Nous étudions les cas suivants:

- Comportements finis (termes sans variables de comportement), pour lesquels nous pouvons décider si deux termes donnés sont ou non *observationnellement congrus* au sens de Milner [Mi 80].
- Comportements infinis rationnels particuliers: correspondant à des systèmes de transitions étiquetées dont l'ensemble d'états est fini. Cette restriction est réaliste dans beaucoup d'applications pratiques, par exemple: protocoles; pour ceci nous imposons certaines restrictions syntaxiques aux expressions de comportement.
  Dans ce cas, nous pouvons décider par bisimulation, si deux termes donnés sont ou non *fortement congrus* au sens Milner [Mi 80] et on peut par suite, approcher la *congruence observationnelle*, en utilisant par exemple, des techniques analogues à celles proposées par Bergstra et Klop [BK 84], (à paraitre dans un prochain rapport).

Le système est en cours de réalisation, la programmation est faite en PROLOG-C (version 1.5) [CM 81][Pe 84] sur un système UNIX 4.2 supporté par un VAX.

Nous présentons dans la suite un rappel les principaux résultats et notations de CCS, le cas des termes finis, le cas des termes infinis rationnels, et une conclusion.

I.- CCS: Calcul asynchrone des systèmes communicants.

Nous rappelons les éléments de CCS et définissons les notations utilisées dans la suite.

CCS est construit à partir:

- d'une algèbre permettant de décrire le comportement de processus parallèles,

- d'un ensemble de règles d'inférence à partir duquel est défini la semantique des opérateurs,
- et des relations d'équivalence et de congruence qui nous permettront de comparer les expressions de comportement.

La présentation de CCS faite dans la suite est purement algébrique et les relations d'équivalence et de congruence sont induites par un ensemble d'axiomes.

Algèbre pour CCS.

Soient:

- D, l'ensemble de noms *d'actions ou ports de communication* dont les éléments sont désignés par des lettres minuscules ne commençant pas par la lettre "n",
- ND, l'ensemble de noms de ports, appelés *complémentaires*, dont les éléments sont désignés par des lettres minuscules commençant par la lettre "n",
- la bijection:
  a (appartenant à D) <---> na (appartenant à ND)
  on dit que "na" est le *port complémentaire* de "a",
  ceci nous permet de définir l'interaction entre deux processus,
- t, une étiquette représentant une *communication non observable*, et "t" n'appartient ni à D ni à ND,
- P, l'ensemble d'étiquettes ou ports de communication, P = D U ND,
- F, l'ensemble de fonctions de redéfinition f telles que:
  f: L⊆P ----> M⊆P
  f est une bijection,
  f respecte les compléments définis par la bijection,
- B, l'ensemble des expressions CCS.

Soit la signature: $\Sigma = (P \cup \{t\}, nil, ., +, \|, [], \backslash)$

où:

P U {t} est l'ensemble de noms d'actions, opérateurs d'arité zéro,
nil est la constante pour les comportements,
{.,+,||,[],\} est l'ensemble des opérateurs binaires, avec:

. : P U {t} ----> B
+ : B x B ----> B
|| : B x B ----> B
[] : B x F ----> B
\ : B x P ----> B

La sémantique de ces opérateurs est définie opérationnellement en suivant les idées de Plotkin [Pl 81]:

(.): $a.E \xrightarrow{a} E$ (ACTION)

(+):

$$(1)\quad \frac{E_1 \xrightarrow{a} E'_1}{E_1 + E_2 \xrightarrow{a} E'_1}$$

(CHOIX NON DETERMINISTE)

$$(2)\quad \frac{E_2 \xrightarrow{a} E'_2}{E_1 + E_2 \xrightarrow{a} E'_2}$$

(||)

$$(1)\quad \frac{E_1 \xrightarrow{a} E'_1}{E_1 \| E_2 \xrightarrow{a} E'_1 \| E_2}$$

$$(2)\quad \frac{E_2 \xrightarrow{a} E'_2}{E_1 \| E_2 \xrightarrow{a} E_1 \| E'_2}$$

(COMPOSITION PARALLELE)

$$(3)\quad \frac{E_1 \xrightarrow{a} E'_1 \,,\; E_2 \xrightarrow{\overline{a}} E'_2}{E_1 \| E_2 \xrightarrow{i} E'_1 \| E'_2}$$

(\)

$$\frac{E \xrightarrow{a} E'}{E \backslash b \xrightarrow{a} E'}$$

(EFFACEMENT)

([])

$$\frac{E \xrightarrow{a} E'}{E[f] \xrightarrow{f(a)} E'[f]}$$

(REDEFINITION)

Relations d'équivalence et de congruence.

Soit l'ensemble d'axiomes suivant:

(+):

- (A_1): $X + (Y + Z) = (X + Y) + Z$
- (A_2): $X + Y = Y + X$
- (A_3): $X + X = X$
- (A_4): $X + nil = X$

(||): (A_5): Si U et V sont de la forme:

$U = \Sigma\ q_i\ .\ X_i$ et $V = \Sigma\ p_j\ .\ Y_j$

alors:

$$U \| V = \Sigma\ q_i\ .\ (X_i \| V) + \Sigma\ p_j\ .\ (U \| Y_j) + \Sigma\ t\ .(X_i \| Y_j)$$
$$q_i = np_j$$
$$\vee\ nq_i = p_j$$

(A_6): $X \| Y = Y \| X$

(A_7): $X \| (Y \| Z) = (X \| Y) \| Z$

(A_8): $X \| nil = X$ $(A_{8'})$: $nil \| X = X$

([]): (A_9): $(p.X)[f] = f(p).(X[f])$

(A_{10}): $(X + Y)[f] = X[f] + Y[f]$

(A_{11}): $nil[f] = nil$

(A_{12}): $X[I] = X$ (I: L —> L fonction identité)

(A_{13}): $X[f] = X[f']$ (X:L et f|`L = f'|`L)

(A_{14}): $X[f][f'] = X[f' \circ f]$

(A_{15}): $(X\|Y)[f] = X[f]\|Y[f]$

(\): (A_{16}): $X[f]\backslash f(q) = X\backslash q[f]$

(A_{17}): $(p.X)\backslash q = \begin{cases} nil & \text{si } p = q \text{ ou } p = nq \\ p.(X\backslash q) & \text{sinon} \end{cases}$

(A_{18}): $(X+Y) = X\backslash q + Y\backslash q$

(A_{19}): $nil\backslash q = nil$

(A_{20}): $X\backslash q = X$ (X:L, $q, nq \notin L$)

(A_{21}): $X\backslash q_1\backslash q_2 = X\backslash q_1\backslash q_2$

(A_{22}): $(X\|Y)\backslash q = X\backslash q \| Y\backslash q$ ($X:L_1$, $Y:L_2$, $q, nq \notin L1 \cap L2$)

(t): (A_{23}): $X+t.X = t.X$

(A_{24}): $p.(X+t.Y) = p.(X+t.Y)+p.Y$

(A_{25}): $p.t.X = p.X$

(A_{26}): $X+t.(X+Y)=t.(X+Y)$

(A_{27}): $t.B = B$

Les différentes relations d'équivalence et de congruence utilisées dans la suite sont:

- l'*équivalence observationnelle* induite par les axiomes (A_1)-(A_{27}) est notée $\approx$,
- la *congruence forte* induite par les axiomes (A_1)-(A_{22}) est notée $\sim$,
- la *congruence observationnelle* induite par les axiomes (A_1)-(A_{26}) est notée $\approx^c$,
- la congruence induite les axiomes (A_1)-(A_5), $(A_{8'})$, (A_8)-(A_{11}), (A_{17})-(A_{19}), (A_{23})-(A_{25}) notée $\underline{\approx}^c$, est appelée *congruence* ,
- la congruence induite par les axiomes (A_1)-(A_2) notée $\approx^s$, et appelée *congruence* (A_1)-(A_2).

Définitions équationnelles de comportements.

Il est possible de définir des comportements au moyen d'équations de la forme:

$$var \lessdot\!=\!= B_{var}$$

où: var est le nom d'une variable (de comportement),
B est une expression de comportement,
<=== est une relation de congruence,
de cette manière, il est possible de définir des comportements infinis.

Nous utilisons dans la suite les définitions suivantes:

On dit qu'une *occurrence d'une variable V est non gardée* dans une expression de comportement ssi elle n'est pas précédée par un nom d'action.

On dit qu'une *variable est non gardée* dans une expression de comportement ssi il existe au moins une occurrence non gardée de la variable dans l'expression.

Par exemple, X est non gardée dans les expressions suivantes:

X , a.Y + X , a.Y + X + b.Z , a.X + X

On dit qu'une *expression de comportement est bien gardée* ssi elle ne contient pas de variables non gardées, ou s'il est toujours possible de substituer chaque occurrence d'une variable par une expression dont les termes ne sont pas de variables non gardées.

Par exemple, X n'est pas une expression de comportement bien gardée dans le système d'equations:

X <=== Y + a.X
Y <=== X + b.nil

par contre X est bien gardée dans:

X <=== Y +a.X
Y <=== b.X +b.nil

et aussi dans: X <=== a.(X + b.nil).

Dans le cas des termes ne comportant pas de variables, nous décidons si deux expressions données sont ou non observationnellement congrues [Mi 80] en transformant chaque terme en une forme normale; intuitivement, la forme normale, d'une expression de comportement CCS sans transmission de valeurs, définie pour la relation de congruence observationnelle, est une expression CCS ne contenant que:

i) des opérateurs · et +,
ii) et le "minimum" de termes nécessaires à l'expression de ce comportement;

chacune de ces formes normales est le représentant d'une classe de la congruence observationnelle. Nous vérifions si ces formes canoniques correspondent ou non à la même classe de congruence, c'est-à-dire si elles sont équivalentes via la relation de congruence(A1)-(A2), notée $\approx^S$

I.- FORME NORMALE.

Pour définir la forme normale d'une expression CCS, nous définissons premièrement la notation d'*expression (écrite) sous la forme d'une somme* et nous prouvons que toute expression décrivant un comportement fini est congrue à une autre expression CCS écrite sous la forme d'une somme; puis nous donnons la définition de *comportements dérivés* d'une expression afin de pouvoir simplifier des termes d'une expression sous la forme d'une somme et finalement pouvoir caractériser la forme normale.

Définition: *expression sous la forme d'une somme.*

(i) nil est sous la forme d'une somme,
(ii) pour tout g appartenant à P U {t}, g.B est sous la forme d'une somme ssi B est sous la forme d'une somme,
(iii) B1+B2 est sous la forme d'une somme ssi B1 et B2 sont sous la forme d'une somme,
(iv) toute expression sous la forme d'une somme est obtenue à partir de (i), (ii) et (iii).

Exemples:

Les expressions suivantes sont sous la forme d'une somme:
a.b.nil+b.d.nil , a.t.(a.nil+t.(b.nil+nd.nil)) , nil

Les expressions suivantes ne sont pas sous la forme d'une somme:
d.b.nil || na.nb.nil , a.b.nil[a/b] , a.b.nil\b

Proposition 1.

Toute expression finie CCS est congrue à une expression écrite sous la forme d'une somme.

Intuitivement, si p est une action appartenant à P U {t}, les p-dérivés d'une expression correspondent aux comportements possibles lorsqu'il y a eu une p-communication.

Définition:

Pour tout p appartenant à P U {t}, le *p-dérivé* d'une expression écrite sous la forme d'une somme $E = \Sigma\, p_i.e_i$, est défini comme suit:

(i) e_i est p_i-dérivé de $\Sigma\, p_i.e_i$,

(ii) tout t-dérivé de e_i est un p-dérivé de $\Sigma\, p_i.e_i$,

(iii) si $p_i = t$ alors tout p-dérivé de e_i est un p-dérivé de $\Sigma\, p_i.e_i$.

Par exemple, soit l'expression: t.(a.nil+t.(b.nil+c.(e.nil+t.(d.nil+t.nil))))

les t-dérivés sont:

a.nil+t.(b.nil+c.(e.nil+t.(d.nil+t.nil)))

b.nil+c.(e.nil+t.(d.nil+t.nil))

les c-dérivés sont:

e.nil+t.(d.nil+t.nil)

d.nil+t.nil

nil

Proposition 3.

Si e' est congru à un p-dérivé de e, alors $p.e' + e \approx^{C} e$.

En reprenant les idées de Hennesy et Milner [HM 80] on remarque qu'une forme normale ne peut être réduite par simplification; pour garantir que l'axiome a.t.X = a.X n'est pas applicable, nous définissons d'abord la notion de forme normale propre, puis celle de forme normale impropre et en fin celle de forme normale.

Definitions:

On dit qu'une expression sous la forme d'une somme $e = \Sigma\, p_i.e_i$ est une *forme normale propre* ssi:

i) elle n'est pas de la forme t.e,

ii) chaque e_i est sous une forme normale propre,

iii) pour i différent de j, il n'existe pas de p_i-dérivé h de $p_j.e_j$ tel que $e_i \approx^{S} h$.

On dit qu'une expression sous la forme d'une somme e = t.e' , est une *forme normale impropre* ssi e' est une forme normale propre.

On dit qu'une expression sous la forme d'une somme est une *forme normale* ssi elle est une forme normale propre ou impropre.

Exemples:

Expression	Forme normale
a.nil +a.(b.nil+t.nil)	a.(b.nil+t.nil)
a.nil +t.(b.nil+t.(d.nil+a.nil))+d.nil	t.(b.nil+t(d.nil+a.nil))
a.nil+t.(nil+t.(nil+t.a.nil))	t.a.nil
(a.nil||na.nil)+t.a.na.nil	t.a.na.nil+na.a.nil+t.nil

Nous pouvons énoncer trois thèorémes dont la preuve complete est fourni dans le rapport [SV 85].

Théorème d'existence: Tout terme fini CCS est congru à une forme normale fn.

Théorème d'unicité: Pour tout f, g termes finis CCS sous forme normale, $f \underline{\approx}^{C} g$ ssi $f \approx^{S} g$.

Théorème: Pour tout f,g terme fini CCS, $f \approx^{C} g$ ssi $f \underline{\approx}^{C} g$.

Donc, la congruence observationnelle définie par R. Milner [Mi 80] est exactement la congruence utilisée dans la définition de la forme normale.

2.- ALGORITHME POUR CALCULER LA FORME NORMALE D'UNE EXPRESSION CCS.

L'algorithme qui calcule la forme normale associée à une expression CCS procède en deux étapes:

1.- Calcul de l'expression sous la forme d'une somme, associée à une expression CCS,
2.- Calcul de la forme normale associée à l'expression sous la forme d'une somme calculée.

Algorithme de transformation d'une expression sous la forme d'une somme.

Cet algorithme procède en trois étapes:

1.- Construction de l'arbre d'opérateurs associé à l'expression donnée, cet arbre tient compte de la priorité usuelle pour les opérateurs
   - les feuilles correspondent
     soit à des opérateurs CCS d'arité 0,
     soit à un ensemble de ports de communication,
     soit à une fonction de redéfinition.
   - les autres nœuds correspondent aux opérateurs binaires CCS.

2.- Transformation de l'arbre pour éliminer des opérateurs [],\ et les feuilles nil non necessaires, en utilisant les axiomes: (A_2), (A_4), (A_6), (A_8),

3.- Construction de l'arbre associé à l'expression écrite sous la forme d'une somme, en appliquant récursivement sur l'arbre réduit en partant des feuilles, la définition d'une expression sous forme d'une somme, et la propriété suivante pour éliminer l'opérateur de composition parallèle.

Propriété: Si A et B sont deux expressions CCS sous la forme d'une somme, alors, en appliquant les axiomes (A_5),(A_6) et (A_8) un nombre fini de fois, nous obtenons une expression sous la forme d'une somme observationnellement congrue à A||B .

Algorithme pour construire la forme normale associée à une expression sous la forme d'une somme.

L'algorithme qui calcule la forme normale associée à une expression sous la forme d'une somme procède en deux étapes:

- Construction de l'arbre d'opérateurs sans des 't' précédés par des noms d'actions, par application de l'axiome A_{25},
- Construction de l'arbre associé à la forme normale, en utilisant la définition d'une expression sous forme normale et la proposition 3.

Nous appliquons la définition d'une expression sous forme normale d'une manière récursive sur chaque sous-arbre, en partant des feuilles.

3.- MISE EN OEUVRE.

La réalisation a été faite en C-PROLOG (version 1.5) [Pe 84], [CM 81] sur un système UNIX 4.2 suporté par un VAX; le programme est composé d'une soixantaine de clauses PROLOG; pour avoir une idée de ses limites sur le système considéré, nous donnons le tableau suivant:

EXPRESSIONS CCS	PLACE OCUPEE PAR LES TERMES CONSTRUITS PENDANT L'EXECUTION [mots]	TEMPS MIS POUR DECIDER LA CONGRUENCE OBSERV. ENTRE LES EXPRESSIONS [sec]
t.a.(t.nil+nil)+a.nil	moins de 1188	0.15
t.(nil+t.(nil+t.a.nil))		
t.(a.nil || na.nil)	3628	0.38
t.(na.nil || a.nil)		

Les performances du programme sont évaluées en terme de nombre d'inférences logiques.

III.- TERMES INFINIS.

Nous nous restreignons dans ce cas à des expressions pouvant contenir des variables de comportement $X_1, X_2, \ldots, X_n$ définies comme suit:

$$X_1 <=== S_1(X_1, X_2, \ldots, X_n)$$
$$X_2 <=== S_2(X_1, X_2, \ldots, X_n)$$
$$\ldots$$
$$X_n <=== S_n(X_1, X_2, \ldots, X_n)$$

où $S_1(X_1, \ldots, X_n)$, $S_2(X_1, \ldots, X_n)$, $\ldots$, $S_n(X_1, \ldots, X_n)$ sont des expressions de comportement écrites sous la forme d'une somme.

L'axiomatisation (A_1)-(A_{27}) fournie pour CCS n'est complète que dans le cas de comportements finis. La méthode de preuve décrite dans la partie II ne peut être appliquée pour les comportements infinis.

Dans ce cas, une méthode consiste à construire pour chaque expression un système de transitions $S_E(E)$ (sous forme d'un graphe fini) dont le comportement est sémantiquement identique à celui décrit par l'expression E. Nous pouvons comparer par bisimulation les systèmes $S_{E1}(E1)$, $S_{E2}(E2)$, i=1, 2 ainsi construits, ceci nous permet de décider la congruence forte des termes E_1, E_2 initiaux. Par transformation sur ces systèmes de transitions, nous pouvons décider la congruence observationnelle des termes initiaux par une technique analogue, ce problème n'est pas étudié ici.

1. - CONSTRUCTION D'UN SYSTEME DE TRANSITIONS ETIQUETEES A PARTIR D'UNE EXPRESSION CCS.

Définition [Ke 76]:

Un *système de transitions étiquetées* est un triplet:

$$S_A = (Q, A, \{\xrightarrow{a}\}_{a \in A} \cup \{\xrightarrow{t}\})$$

où: Q est un ensemble d'états,
A est un vocabulaire de noms d'actions,
$\{\xrightarrow{a}\}_{a \in A} \cup \{\xrightarrow{t}\}$ est un ensemble de relations binaires sur Q.

On définit pour chaque système de transitions étiquetées un état initial $Eo \in Q$.

Soit une expression CCS bien gardée:

$$E(X1, \ldots, Xn)$$

où: $X_1 <=== S_1(X_1, X_2, \ldots, X_n)$
$X_2 <=== S_2(X_1, X_2, \ldots, X_n)$
$\ldots$
$X_n <=== S_n(X_1, X_2, \ldots, X_n)$

les variables de comportement sont définies par les expressions $S1(X1, \ldots Xn)$, $\ldots$, $Sn(X1, \ldots, Xn)$ écrites sous forme d'une somme.

Nous associons un système de transitions étiquetées S(E) à l'expression E, comme suit:

$S(E) = (Q_E(E), A_E(E), ST_E(E))$

où: $Q_E(E) = \{ B_1 : \exists B, e \ (B \xrightarrow{e} B_1 \in ST_E(E)) \} \cup \{E\}$,

$A_E(E) = \{ e : \exists B, B_1 (B \xrightarrow{e} B_1 \in ST_E(E))\}$,

$ST_E(E)$ est un ensemble de relations binaires sur Q correspondant aux dérivations de l'expression E.

Pour construire cet ensemble, nous procèdons par induction sur la structure de E :

$ST_{E0}(nil) = \emptyset$,

$ST_{E0}(a.E_i) = \{E_0 \xrightarrow{a} E_i\} \cup ST_{Ei}(E_i)$,

$ST_{E0}(E1+E2) = ST_{E0}(E1) + ST_{E0}(E2)$,

$ST_{E0}(E_i[f]) = \{E[f] \xrightarrow{f(e)} E_i[f] : E$ non identique à $E0 \ \wedge$

$E \xrightarrow{e} E_i \in ST_{E0}(E_i)\} \cup$

$\{Eo \xrightarrow{f(e)} E_i[f] : Eo \xrightarrow{e} E_i \in ST_{E0}(E_i)\}$,

$ST_{E0}(x) = STS_{E0}(B, \emptyset)$, où: x est une variable de comportement définie par:

x<=== B

où $STS_{E0}(B,S)$ est défini par:

si B est identique à $Eo \ \wedge \ \exists E_1, e : Eo \xrightarrow{e} E1 \in S$

alors $STS_{E0}(B,S) = S$

sinon $STS_{E0}(nil,S) = S$,

$STS_{E0}(p.B,S) = STS_B(B, S \cup \{Eo \xrightarrow{p} B\})$,

$STS_{E0}(B1+B2,S) = STS_{E0}(B1, STS_{E0}(B2,S))$,

$STS_{E0}(x,S) = STS_{E0}(B,S)$, si x est une variable et x<===B,

$ST_{E0}(E1 \parallel E2) = STP_{E0}(ST_{E1}(E1), ST_{E2}(E2), E1, E2, \emptyset, \emptyset)$

où STP_{E0} (S1,S2,B1,B2,F,S) est l'ensemble de transitions correspondant au système de transitions associé à l'expression E1 || E2. Cet ensemble est obtenu par l'application de l'axiome (A_5) sur les systèmes de transitions associés aux expressions E1 et E2 (définition détaillée dans [SV 85]).

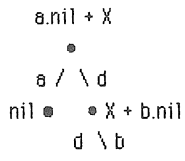
$ST_{E0}(E1 \backslash P) = STE_{E0}(E1 \backslash P)$

En définissant $STE_{E0}(E)$ comme le système de transitions étiquetées associé à l'expression E et en prenant Eo comme le nom de l'état initial (définition detaillée dans [SV 85]).

Exemple:

Soit l'expression CCS: a.nil +X
où: X<---d.(X + b.nil)

le système de transitions associé peut se représenter graphiquement comme:



2.- BISIMULATION.

Intuitivement, une bisimulation est une relation R entre deux systèmes de transitions S1 et S2, R définie entre les états de S1 et S2 comme suit:

A chaque état de S1 on associe au moins un état de S2 à partir duquel on a les mêmes transitions possibles et vice-versa.

Définition [Pa 81a]:

Soient: - deux systèmes de transitions Si = (Qi,Ri), i=1,2 ,
- une relation R de Q1 sur Q2, totale pour Q1 et Q2,

une *bisimulation* entre deux systèmes S1 et S2 (notée $R1 \overset{R}{<===>} R2$) est une relation R telle que:

$q_1 \; R \; q_2$ implique $(\forall \; q'_1 \; (q_1 \; R_1 \; q'_1$ implique $\exists q'_2 (q_2 \; R_2 \; q'_2$ et $q'_1 \; R \; q'_2))$
$\wedge \; \forall \; q'_2 \; (q_2 \; R_2 \; q'_2$ implique $\exists q'_1 (q_1 \; R_1 \; q'_1$ et $q'_1 \; R \; q'_2)))$.

Pour pouvoir décider s'il existe une relation de bisimulation entre deux systèmes de transitions, nous utilisons le resultat suivant dû à Park [Pa 81b]:
Soient B l'ensemble de comportements,
F, G : B x B ---> B définies comme:

$G(R) = \{(p,q) :$ si $p \xrightarrow{a} p'$ alors $\exists q' \; (q \xrightarrow{a} q'$ et $p' \; R \; q')\}$
$F(R) = G(R) \cap \overline{G(\overline{R})}$, où $\overline{R} = \{(p,q) : p \; R \; q\}$

F est monotone et possede un plus grand point fixe:

$F = \cup \{ R$ tels que $R \subseteq F(R)\}$.

Une extension de ce résultat aux systèmes de transitions étiquetées est:
Soient : S1 = (Q1,A1,R1) et S2 = (Q2,A2,R2) deux systèmes de transitions

$R1 \overset{R}{<===>} R2$ ssi les trois conditions suivantes sont vérifiées:

(Condition 1): il existe R tel que R est le plus grand point fixe de $R = R \cap F(R)$,

où: $F(R) = \cap \{ (q1,q2) : (\forall q'_1 (q_1 R_{1a} q'_1 \text{ implique } \exists q'_2 (q_2 R_{2a} q'_2 \wedge q'_1 R q'_2)))$
$a \in A_1 \cup A_2$
$\wedge (\forall q'_2 (q_2 R_{2a} q'_2 \text{ implique } \exists q'_1 (q_1 R_{1a} q'_1 \wedge q'_1 R q'_2)))\},$

(Condition 2): $(EO_1, EO_2) \in R$, où: EO_1 et EO_2 sont les états initiaux de S1 et S2,

(Condition 3): R est totale pour Q1 et Q2.

3.- ALGORITHME POUR PROUVER LA CONGRUENCE FORTE ENTRE DEUX EXPRESSIONS CCS.

L'algorithme de preuve de la congruence forte entre deux expressions CCS procède en trois étapes:

1.- construction des systèmes de transitions étiquetées associés aux expressions CCS données,
2.- calcul itératif d'une relation R entre les systèmes de transition construits, vérifiant la condition 1,
3.- vérification des conditions 2 et 3 pour la relation R.

Algorithme de construction des systèmes de transitions.

Pour la construction de chaque système de transitions:

1) nous construisons d'abord des arbres d'opérateurs correspondant:
   - à l'expression de comportement,
   - à chaque expression de comportement définissant une variable de comportement dans le cas d'un système équationnel.

   Soit Ao l'arbre d'opérateurs associé à une expression CCS avec variables :
   - les feuilles de cet arbre correspondent
     soit à des opérateurs CCS d'arité zéro,
     soit à un ensemble de ports,
     soit à une fonction de redéfinition,
     soit à une variable de comportement.
   - les autres nœuds de l'arbre correspondent aux opérateurs binaires CCS.

   Cet arbre tient compte de la priorité définie en CCS pour les opérateurs.

2) Pour le système de transitions, nous construisons de manière inductive l'ensemble de relations de transition définies précédemment, en partant des feuilles de l'arbre d'opérateurs.

Algorithme de calcul d'une relation R entre deux systèmes de transitions.

L'algorithme de construction d'une relation de bisimulation R entre deux systèmes de transitions $S_1 = (Q_1,A_1,R_1)$ et $S_2 = (Q_2,A_2,R_2)$ calcule itérativement le plus grand point fixe de:

$R = R \cap F(R)$,

où:

$F(R) = \cap \{(q1,q2): (\forall q'_1 \ (q_1 \ R_{1a} q'_1 \text{ implique } \exists q'_2 (q_2 \ R_{2a} \ q'_2 \wedge q'_1 \ R \ q'_2)))$
$a \in A_1 \cup A_2$

$\wedge \ (\forall q'_2 \ (q_2 \ R_{2a} q'_2 \text{ implique } \exists q'_1 (q_1 \ R_{1a} \ q'_1 \wedge q'_1 \ R \ q'_2)))\},$

en partant de la relation $R = Q_1 \times Q_2$.

Existence d'une relation de bisimulation.

Si la relation calculée R vérifie les conditions 2 et 3 alors c'est une relation de bisimulation.

L'existence d'une bisimulation entre les systèmes de transitions permet de décider si les expressions CCS données sont fortement congrues.

4.- MISE EN ŒUVRE.

La réalisation a éte faite en C-PROLOG (version 1.5) [Pe 84], [CM 81] sur un système UNIX supporté par un VAX; le programme est composé d'environ cent cinquante clauses PROLOG; pour avoir une idée de sa performance, nous donnons l'exemple suivant:

Soit le contrôleur défini dans [Mi 80]:

```
- (s||p||q)\{e,f}
        s<---ne.nil
        p<---e.na.nb.nf.p
        q<---f.nc.nd.ne.q
```

et sa spécification:

```
- x
  x<---t.a.nb.t.c.nd.x
```

la description du contrôleur est correcte (fortement congrue à l'expression de sa spécification); la place occupée par les termes construits pendant l'execution de la preuve est de 151 924 mots, le temps d'execution est de 32.98 secondes.

IV.- CONCLUSIONS.

Un système complet de preuve peut être construit en étendant les rèsultats précédents aux expressions CCS avec transmission de valeur.

D'autres calculs de systèmes communicants peuvent être pris en compte.

[BK 84] Bergstra, J.A., Klop, J.W.
A COMPLETE INFERENCE SYSTEM FOR REGULAR PROCESSES WITH SILENT MOVES.
Centre for Mathematics and Computer Science. Departement of Computer Science. Report CS R8420. November 1984.

[CM 81] Clocksin, W.F, Mellish, C.S.
PROGRAMMING IN PROLOG.
Springer-Verlag. 1981.

[HM 80] Hennessy, M., Milner, R.
Unpublished note.

[Ke 72] Keller, R. M.
VECTOR REPLACEMENT SYSTEMS: A FORMALISM FOR MODELING ASYNCHRONOUS SYSTEMS,
Princeton University, Technical Report No. 117.1972.

[Mi 80] Milner, R.
A CALCULUS FOR COMMUNICATING SYSTEMS.
Lecture Notes in Computer Science, 92. Springer-Verlag. 1980.

[Pa 81a] Park, D.
CONCURRENCY AND AUTOMATE ON INFINITE SEQUENCES.
Proc. 5$^{th.}$ G. I. Conference. Lecture Notes in Computer Science, 104. Springer-Verlag. 1981.

[Pa 81b] Park, D.
A NEW EQUIVALENCE NOTION FOR COMMUNICATING SYSTEMS.
Bulletin of European Association for Theoretical Computer Science, 14, pp. 78-80. 1981.

[Pe 84] Pereira, F.
C-PROLOG USER'MANUAL.
SRI International, Menlo Park, California. 1984.

[Pl 81] Plotkin, G. D.
A STRUCTURAL APPROACH TO OPERATIONAL SEMANTICS.
Aarthus University, Computer Science Departement, Technical report DAIMI FN-19. 1981.

[Si 84] Sifakis, J.
PROPERTY PRESERVING HOMOMORPHISMES OF TRANSITIONS SYSTEMS.
Workshop Carnegie Mellon, University Pittsburgh, P.A. 1983. Lecture Notes in Computer Science, 164, pp 458-473. Springer-Verlag. 1984.

[SV 85] Soriano, A., Voiron, J.
CONCEPTION ET MISE EN ŒUVRE D'UN SYSTEME DE PREUVE POUR UN SOUS ENSEMBLE DE CCS SANS TRANSMISSION DE VALEURS.
IMAG. Rapport de Recherche N° 530. 1985