Algebre de processus et synchronisation

Didier Austry, Gérard Boudol

INRIA

CENTRE
SOPHIA ANTIPOLIS

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P 105
78153 Le Chesnay Cedex
France
Tél. 954 90 20

Rapports de Recherche

N° 187

ALGÈBRE DE PROCESSUS ET SYNCHRONISATION

Didier AUSTRY
Gérard BOUDOL

Février 1983

INRIA

CENTRE
SOPHIA ANTIPOLIS

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
BP 105
78153 Le Chesnay Cedex
France
Tél. 954 90 20

Rapports de Recherche

N° 187

ALGÈBRE DE PROCESSUS ET SYNCHRONISATION

Didier AUSTRY
Gérard BOUDOL

Février 1983

ALGEBRE DE PROCESSUS ET SYNCHRONISATION

Didier AUSTRY* - Gérard BOUDOL**

---

* CMA - ENSMP - SOPHIA ANTIPOLIS
06565 VALBONNE

** INRIA - SOPHIA ANTIPOLIS
06565 VALBONNE

RESUME

Nous introduisons ici un calcul de processus fondé sur une composition parallèle asynchrone, et quelques primitives de synchronisation. Nous montrons qu'on peut y formuler de nombreux mécanismes de synchronisation ; en particulier il s'avère équivalent au calcul synchrone de R. Milner. Nous évaluons sa puissance d'expression en termes de langages de comportements.

ABSTRACT

We introduce here a calculus of processes, involving an asynchronous parallel composition and some synchronization primitives. It is shown that in this calculus, many synchronization mechanisms can be formulated ; in fact, this calculus is equivalent to the R. Milner's synchronous calculus. We evaluate its expressive power by means of languages of behaviours.

Mots Clés : sémantique, parallélisme, synchronisation, calcul de processus

Keywords : semantics, parallelism, synchronization, calculus of processes.

ALGEBRE DE PROCESSUS ET SYNCHRONISATION

D.Austry * - G. Boudol **

* CMA - ENSMP - Sophia Antipolis
06565 VALBONNE

** INRIA - Sophia Antipolis
06565 VALBONNE

1. INTRODUCTION

Notre propos est de présenter dans cet article un modèle formel de la notion de processus, l'objectif étant d'élaborer un cadre adapté à la description des mécanismes de synchronisation. Il y a de nombreuses approches possibles de cette problématique de la synchronisation des processus : nous adoptons ici complètement le point de vue algébrique des calculs de R. Milner [6,8]. Un processus p y est conçu abstraitement comme un objet qui accomplit certaines actions a (dont nous ignorons ici la sémantique) et ce faisant se reconfigure en un autre processus p', relation dénotée par

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

Nous considérons ici que les actions sont instantanées, c'est-à-dire atomiques du point de vue temporel. Traiter de la synchronisation suppose :

(1) rendre compte de la simultanéité : un système de processus mis en parallèle doit pouvoir accomplir une action résultant de l'activité simultanée de ses composants ;

(2) rendre compte de contraintes temporelles : ceci suppose un échange d'information, une certaine forme de communication (qui n'altère pas autrement le comportement d'un processus qu'en contraignant le développement de ses actions dans le temps).

Ces deux points sont formalisés par l'idée fondamentale de R. Milner [7,8] :

- l'ensemble des actions forme un semi-groupe abélien : l'action qui résulte de la co-occurrence temporelle de deux actions a et b est leur produit a.b . Cette opération de composition est naturellement commutative et associative ;

- certaines actions sont des actions d'échange, qui ont un inverse: la communication, élément neutre pour la composition, résulte de la co-occurence de deux actions d'échanges inverses l'une de l'autre.

Etant intéressés ici par la synchronisation pure, nous sommes conduits à adopter comme fondement de notre calcul la structure de l'ensemble des actions décrite comme suit :

- pour écrire des programmes parallèles, nous disposons d'un ensemble $\mathbb{A}$ d'actions (ou plutôt : schémas d'actions) de calcul, tels "calculer", "mettre", "prendre", etc. ;

- nous nous donnons par ailleurs un ensemble S de signaux, chaque signal $\underline{\alpha}$ déterminant deux actions de synchronisation α et $\overline{\alpha}$, qui forment l'ensemble $\mathbb{S}$;

- l'ensemble $\mathbb{M}$ des actions est alors le quotient du monoide commutatif libre (les paquets) engendré par $\mathbb{A} \cup \mathbb{S}$ par la congruence donnée par

$$\alpha.\overline{\alpha} = \overline{\alpha}.\alpha = 1 \qquad \text{pour } \underline{\alpha} \in S$$

(1 étant l'élément neutre).

On peut donc également décrire $\mathbb{M}$ comme le produit d'un monoide commutatif libre (engendré par $\mathbb{A}$) et d'un groupe abélien libre (engendré par S). On voit que le rôle des actions α et $\bar{\alpha}$ est parfaitement symétrique ($\bar{\bar{\alpha}} = \alpha$), même si leur connotation resp. dans ce papier est celle de $\alpha?$ (réception du signal $\underline{\alpha}$) et $\alpha!$ (émission du signal $\underline{\alpha}$).

En optant pour cette présentation du monoide d'actions, nous avons un point de vue un peu plus restrictif que celui de R. Milner. Cette structure de produit fait apparaître le rôle ambigu de l'élément neutre: d'un côté action de ne rien faire, ou d'attendre, pendant un instant ; de l'autre action de communication. De fait nous n'utiliserons jamais explicitement, sauf à retrouver les constructions du calcul synchrone de R. Milner [8], l'action 1 comme action d'attente. Ceci nous amène à une seconde différence avec SCCS : dans ce calcul (voir aussi [7]) sont introduits des opérateurs de désynchronisation explicite.

A l'inverse nous voulons essayer de synchroniser des processus a priori temporellement indépendants. Cette dernière hypothèse se traduit dans la définition opérationnelle de la composition parallèle décrite par :

$$\frac{p \xrightarrow{a} p'}{(P//q) \xrightarrow{a} (p'//q)} \qquad \frac{p \xrightarrow{a} p' \ , \ q \xrightarrow{b} q'}{(p//q) \xrightarrow{a.b} (p'//q')} \qquad \frac{q \xrightarrow{b} q'}{(p//q) \xrightarrow{b} (p//q')}$$

Ceci signifie que

$$(p//q) \xrightarrow{c} r \quad \underline{\text{ssi}} \quad \text{(i)} \ \exists\, p' : p \xrightarrow{c} p' \text{ et } r=(p'//q) \quad \text{ou bien}$$

$$\text{(ii)} \ \exists\, a,b \ \exists\, p',q' : p \xrightarrow{a} p' \ , \ q \xrightarrow{b} q'$$

$$\text{et } c=a.b, \quad r=(p'//q') \quad \text{ou bien}$$

$$\text{(iii)} \ \exists\, q' : q \xrightarrow{c} q' \text{ et } r=(p//q')$$

c'est-à-dire que les premières actions de (p//q) sont celles de p ou q, ou une action résultant de l'activité simultanée de p et q. Ainsi on

voit que l'un des deux peut "attendre" par rapport à l'autre, sans accomplir pour autant l'action 1.

Précisons maintenant les traits caractéristiques de l'approche algébrique adoptée :

(1) un processus, ou plus exactement un agent, est un terme d'une algèbre libre. Nous avons déjà vu l'un des constructeurs de ce langage : la composition parallèle ; nous décrivons les autres plus loin ;

(2) les transitions initiales possibles d'un agent sont décrites, ainsi que nous l'avons vu pour (p//q), par induction structurelle. Nous suivons là le style de G. Plotkin [9] qui montre que la sémantique opérationnelle d'un langage s'exprime très naturellement par ce genre de système de réécriture de termes conditionnel ;

(3) comme ce sont plus les comportements que les termes qui nous intéressent, les processus sont les éléments d'une algèbre quotient par une congruence compatible avec les transitions.

Cette congruence induit de nombreuses propriétés, ce qui justifie le terme de "calcul" pour l'algèbre des processus. Celui que nous introduisons dans la seconde section est baptisé MEIJE. On peut, de façon quelque peu arbitraire, y distinguer deux types de primitives :

(1) les constructeurs proprement dits, qui sont :

- le préfixage par une action a, opération qui s'écrit a:p et dont la sémantique est donnée par $a:p \xrightarrow{a} p$

- la composition parallèle, déjà évoquée

- la déclaration ou définition récursive. Elle prends la forme :

$$(x_i \text{ where } x_1 <= p_1 \ldots, x_n <= p_n)$$

où les x_i sont des identificateurs et p_i des termes. Sans donner le détail de la sémantique de cette construction, disons que ce

terme se comporte comme le corps p_i de la définition de x_i, dans lequel les identificateurs sont récursivement liés à leur définition.

(2) Les primitives de synchronisation, qui visent à modifier le comportement d'un processus :

- le renommage par un endomorphisme φ du monoide d'action : $\langle\varphi\rangle p$

- la restriction sur un signal $\underline{\alpha}$, qui s'écrit $p\backslash\underline{\alpha}$. Cette opération a pour effet d'interdire toute action qui comporte (de façon irréductible) l'émission ou la réception de ce signal.

Cette famille d'opération est plus restrictive qu'en SCCS puisque nous n'autorisons ici des restrictions qu'à certains sous-monoides de $\mathbb{M}$. Par contre nous introduisons deux primitives de synchronisation qui n'apparaissent pas explicitement dans les calculs de R. Milner (mais y sont définissables) :

- Le déclenchement d'un agent p sur une action de synchronisation $s \in \mathbb{S}$, noté (s => p), qui vise à lier les premières actions possibles de p à l'émission ou réception simultanée d'un signal :

$$\frac{p \xrightarrow{a} p'}{(s \Rightarrow p) \xrightarrow{s.a} p'}$$

- Le pilotage de p sur $s \in \mathbb{S}$, que nous notons $(s^{@}p)$, qui lie toutes les actions possibles à partir de p à l'action de synchronisation s:

$$\frac{p \xrightarrow{a} p'}{(s^{@}p) \xrightarrow{s.a} (s^{@}p')}$$

Cette opération consiste en quelque sorte à régler le comportement d'un agent sur une horloge.

Un intérêt de ce type de formalisme est qu'il n'y a là aucune différence entre processus et système (synchronisé) de processus. En effet un mécanisme de synchronisation apparait comme un opérateur (ou plus généralement un système d'opérateurs) qui organise les comportements d'un certain nombre de processus. Ici l'on peut spécifier un tel mécanisme comme sont décrites les primitives : par la description structurelle de sa sémantique opérationnelle. Par exemple on peut ainsi décrire la somme de deux processus :

$$\frac{p \xrightarrow{a} p'}{(p+q) \xrightarrow{a} p'} \qquad \frac{q \xrightarrow{b} q'}{(p+q) \xrightarrow{b} q'}$$

ou la composition synchrone de SCCS :

$$\frac{p \xrightarrow{a} p' \;,\; q \xrightarrow{b} q'}{(p \times q) \xrightarrow{a.b} (p' \times q')}$$

Mais il est aussi possible de définir (ou réaliser) ces mécanismes comme des opérations dérivées dans le langage : il s'agit là de trouver un contexte (expression du langage avec des variables libres) tel que, lorsqu'on y plonge les composants du système, le terme obtenu se comporte exactement (à la congruence près) comme l'indique la spécification axiomatique du mécanisme de synchronisation. En général un tel contexte apparait sous la forme d'une mise en parallèle des composants du système, soumis à des contraintes de réception de signaux, avec un agent synchronisateur (souvent une horloge) qui distribue les signaux. Ainsi par exemple nous verrons que l'on peut définir la somme :

$$(p+q) = ((\alpha \Rightarrow p) // (\alpha \Rightarrow q) // \overline{\alpha}:0) \setminus \underline{\alpha}$$

(où 0 est un agent qui n'a pas d'action) et la composition synchrone :

$$(p \times q) = (\alpha * p // \alpha * q // (x \text{ where } x <= \bar{\alpha}.\bar{\alpha}:x)) \setminus \alpha$$

où $h = (x \text{ where } x <= \bar{\alpha}.\bar{\alpha}:x)$ est une horloge qui émet deux tops simultanément.

Nous consacrons la 3ème section de ce papier à illustrer cette démarche par de nombreux exemples, en montrant la validité des constructions proposées et en donnant quelques propriétés algébriques des opérateurs dérivés (avec lesquels la congruence est compatible puisqu'ils sont définis par une expression du langage). Nous retrouverons en particulier les constructions finitaires de SCCS, c'est-à-dire la version de ce calcul donnée en [7]. Nous montrons qu'en fait ce calcul et MEIJE sont équivalents : chacun est sous-calcul de l'autre, au sens de [8]. Ceci signifie que les deux approches : désynchroniser des processus synchrones (SCCS) ou synchroniser des processus asynchrones (MEIJE) ont la même puissance d'expression.

Nous évaluons dans la 4ème partie cette puissance d'expression : modélisant le comportement des processus par des langages de suite d'actions (et l'on retrouve là une approche largement utilisée, voir par exemple [2,4,5,10]), nous montrons que tout langage récursivement énumérable apparait comme le comportement, en ce sens, d'un processus de notre calcul. En particulier pour tout mécanisme de synchronisation décrit (comme par exemple celui qui régit les rapports de production/consommation à travers une file d'attente) par une grammaire algébrique (ie "context-free") on peut construire un processus qui le réalise.

2. SYNTAXE ET SEMANTIQUE

Ainsi que nous l'avons dit, notre calcul repose sur la présentation du monoide M des actions (dans lequel la composition est notée . et le neutre 1) comme quotient du monoide commutatif libre engendré par A ∪ $ par la congruence donnée par

$$\alpha.\bar{\alpha} = 1 \quad \text{pour} \quad \underline{\alpha} \in S$$

où - $\mathbb{A}$ est un ensemble d'actions de calcul

- S est un ensemble (disjoint de $\mathbb{A}$) de signaux, désignés par $\underline{\alpha}, \underline{\beta}, \underline{\sigma} \ldots$ et $\$ = \{\alpha, \bar{\alpha} \ / \ \underline{\alpha} \in S\}$ est l'ensemble (supposé disjoint de $\mathbb{A} \cup S$) des actions (atomiques) de synchronisation.

D'une façon générale, un élément de $\mathbb{M}$ sera désigné, à une permutation près, par un mot réduit de $(\mathbb{A} \cup \$)^*$, et lorsque ce mot m n'est composé que d'actions de synchronisation, $\bar{m}$ désigne son inverse, de sorte que $\bar{\bar{m}} = m$. Par abus de langage encore, on dira qu'une action a est facteur d'une action b (ou divise b) si, à une permutation près, le mot réduit a est facteur du mot réduit b. Formellement :

pour $B \subseteq \mathbb{A} \cup \$$ on note par

- $B^{@}$ le quotient par la congruence engendré par $\{\alpha.\bar{\alpha} = 1 \ / \ \underline{\alpha} \in S\}$ du sous-semigroupe engendré par B du monoide commutatif libre engendré par $\mathbb{A} \cup \$$. C'est-à-dire que $B^{@}$ est l'ensemble des paquets non vides d'éléments de B, que l'on réduit (on peut donc avoir $1 \in B^{@}$).

Exemple : pour $a \in \mathbb{A}$, $a^{@} = \{a, a.a, a.a.a, \ldots\}$ et si $\underline{\alpha} \in S$:

$$\{a.\alpha \ , \ \bar{\alpha}\}^{@} = \{a.\alpha, \ \bar{\alpha}, \ a, \ a.\alpha.a.\alpha, \ \bar{\alpha}.\bar{\alpha}, \ a.a, \ \ldots\}$$

- $B^{@} = B^{@} \cup \{1\}$ (donc $\mathbb{M} = (\mathbb{A} \cup \$)^{@}$ par définition)

- $\mathbb{M}\backslash B = ((\mathbb{A} \cup \$) - B)^{@}$

Par abus de notation, pour $B \subseteq \mathbb{A} \cup S$, on désignera aussi par $B^{@}$, B^{*} , $\mathbb{M}\backslash B$ resp. les structures $C^{@}$, C^{*} , $\mathbb{M}\backslash C$ où $C = (B \cap \mathbb{A}) \cup \{\alpha, \bar{\alpha} \ / \ \underline{\alpha} \in B\}$.

La structure adoptée pour le monoide des actions nous permet de décrire les endomorphismes : ceux-ci sont entièrement déterminés par leur image sur les générateurs, c'est-à-dire sur $\mathbb{A} \cup \{\alpha / \underline{\alpha} \in S\}$ puisque

$$\varphi(\overline{\alpha}) = \overline{\varphi(\alpha)}$$

(une action de synchronisation ne peut être renommée que par un paquet de telles actions, c'est-à-dire que $\varphi(\$) \subseteq \$^{\circledS}$).

Pour écrire les processus nous nous donnerons un ensemble X (disjoint de $\mathbb{M}$) d'identificateurs de processus, ou variables, généralement désignés par x,y,z... (éventuellement indexés). Les expressions du langage (que nous dénoterons par p,q,r, ...) sont les éléments de l'algèbre libre $\mathcal{L}$ dont la syntaxe est décrite par :

(i) 0 est une expression de $\mathcal{L}$ (où $0 \notin X \cup \mathbb{M}$) ainsi que tout identificateur de processus.

(ii) Si $a \in \mathbb{M}$ et $p \in \mathcal{L}$, alors a:p est une expression du langage, le préfixage de p par a.

(iii) Si p et q sont des expressions, alors (p//q) est une expression, la composition parallèle de p et q.

(iv) Si $\{x_1,\ldots,x_n\} \subseteq X$ et $\{p_1,\ldots,p_n\} \subseteq \mathcal{L}$ alors pour tout $i(1 \leq i \leq n)$

$$(x_i \ \underline{\text{where}} \ x_1 <= p_1, \ldots, x_n <= p_n)$$

est une expression (qu'on écrira souvent (x_i where R)).

(v) Si p est une expression et φ un endomorphisme de $\mathbb{M}$ alors $\langle\varphi\rangle p$ est une expression.

(vi) Pour $\underline{\alpha} \in S$, si p est une expression alors $(p \setminus \underline{\alpha})$ est une expression (restriction de p sur $\underline{\alpha}$).

(vii)Pour $s \in \$$, si p est une expression alors (s => p) et (s * p) sont des expressions, resp. déclenchement et pilotage de p par s.

Remarques :

(1) On verra qu'on peut définir l'expression 0, mais il est commode d'en disposer. On pourrait également limiter les opérations de préfixage au cas où $a \in \mathbb{A} \cup \$$, ce qui serait peut être plus naturel.

(2) Nous n'utiliserons en fait l'opération $\langle\varphi\rangle$ que pour des endomorphismes tels que $\{a/a \in \mathbb{A} \cup \$ \ \& \ \varphi(a) \neq a\}$ est un ensemble fini. Si cet ensemble est $\{a_1,\ldots,a_n\}$ et si $\varphi(a_i) = b_i$, on notera $\langle\varphi\rangle p$ par

$$\langle b_1/a_1,\ldots,b_n/a_n\rangle\ p$$

(cette opération n'est pas une substitution).

(3) L'un de nos objectifs est de développer un système de manipulation de processus formels. Aussi nous n'avons pas de primitive "infinitaire" comme dans [8]. Par contre nous admettons des primitives qui pourraient être réduites (point (1)), pour obtenir plus de souplesse dans l'écriture.

Dans ce papier, nous prendrons quelques libertés avec la syntaxe formelle qui vient d'être donnée en ce qui concerne le parenthésage (sans pour autant expliciter les priorités entre opérateurs). Comme il est usuel, on peut définir inductivement les notions d'occurrence libre et liée d'une variable dans une expression, et de variable libre ou liée. Le nom d'une variable muette est évidemment sans importance, et nous considèrerons que deux expressions sont syntaxiquement identiques, ce que nous notons $\equiv$, si elles ne diffèrent que par le nom des variables liées (c'est la α-conversion du λ-calcul). Cela étant, on peut définir inductivement l'opération de substitution d'expressions $p_1,\ldots,p_n$ à des variables $y_1,\ldots,y_n$ en toutes leurs occurrences libres dans l'expression p (avec un changement de variables muettes dans p s'il y a lieu d'éviter des conflits). Le résultat de cette opération (défini modulo $\equiv$) est noté

$$[p_1/y_1,\ldots,p_n/y_n]p$$

ou, lorsqu'il n'y a pas d'ambiguité, $[p_i/y_i]p$.

Sans définir formellement ces notions, notons seulement que

- $[q_1/y_1,\ldots,q_n/y_n]\ (x_i \underline{\text{where}}\ x_1 <= p_1,\ldots,\ x_k <= p_k) \equiv$

$(x'_i \underline{\text{where}}\ x'_1 <= [q_j/y_j][x'_h/x_h]p_1,\ldots,\ x'_k <= [q_j/y_j][x'_h/x_h]p_k)$

où chaque x'_h est une variable qui n'est ni un y_j, ni une variable libre d'un q_j.

- On considère aussi que la restriction est une opération liante :

$$[p_i/x_i](p\backslash\underline{\alpha}) \equiv ([p_i/x_i][\underline{\beta}/\underline{\alpha}]p)\backslash\underline{\beta}$$

où $\underline{\beta}$ n'apparait dans aucun p_i, c'est-à-dire n'est pas dans leur sorte, comme nous le verrons.

Nous appellerons <u>agents</u> les expressions closes (sans variable libre), qui forment l'ensemble $\mathcal{A}$.

Les transitions possibles d'une expression sont données par les relations $\xrightarrow{a}$ pour $a \in \mathbb{M}$, qui sont les plus petites relations satisfaisant la définition inductive (ou structurelle, cf [9], donnée ci-dessous. Indiquons d'abord que lorsque nous écrivons

$$\frac{p_1 \xrightarrow{a_1} p'_1,\ldots,p_n \xrightarrow{a_n} p'_n}{q_1 \xrightarrow{b_1} q'_1,\ldots,q_k \xrightarrow{b_k} q'_k}$$

c'est là un élément de la définition inductive de la (plus petite) relation $T \subseteq \mathcal{L} \times \mathbb{M} \times \mathcal{L}$ qui satisfait (entre autres) :

$$(p_1,a_1,p'_1) \in T \wedge \ldots \wedge (p_n,a_n,p'_n) \in T \Rightarrow$$

$$(q_1,b_1,q'_1) \in T \vee \ldots \vee (q_k,b_k,q'_k) \in T$$

La sémantique opérationnelle des constructeurs de $\mathcal{L}$ est donc spécifiée par :

(i) $\quad a:p \xrightarrow{a} p$

(ii) $\quad \dfrac{p \xrightarrow{a} p'}{(p//q) \xrightarrow{a} (p'//q)} \qquad \dfrac{p \xrightarrow{a} p' \ , \ q \xrightarrow{b} q'}{(p//q) \xrightarrow{a.b} (p'//q')} \qquad \dfrac{q \xrightarrow{b} q'}{(p//q) \xrightarrow{b} (p//q')}$

(iii) $\quad \dfrac{[(x_j \text{ where } R)/x_j]\ p_i \xrightarrow{a} q}{(x_i \text{ where } R) \xrightarrow{a} q}$ où $R = x_1 <= p_1,\ldots,x_n <= p_n$

(iv) $\quad \dfrac{p \xrightarrow{a} p'}{\langle\varphi\rangle p \xrightarrow{\varphi(a)} \langle\varphi\rangle p'}$

(v) pour $\underline{\alpha} \in S$:

$$\frac{p \xrightarrow{a} p' \ , \ a \in \mathbb{M} \setminus \alpha}{(p \setminus \underline{\alpha}) \xrightarrow{a} (p' \setminus \underline{\alpha})}$$

(vi) pour $s \in \$$:

$$\frac{p \xrightarrow{a} p'}{(s \Rightarrow p) \xrightarrow{s.a} p'} \qquad \frac{p \xrightarrow{a} p'}{(s^{\#}p) \xrightarrow{s.a} (s^{\#}p')}$$

Remarque : puisque les relations de transition $\xrightarrow{a}$ sont les plus petites satisfaisant ces propriétés, on peut voir que les

expressions 0 et x pour $x \in X$ n'ont pas de transition.

Exemple

Soit $p = (a:(\bar{\alpha}^*(b:c:0))//d:(\alpha => c:0)) \setminus \underline{\alpha}$ où a, b, c, d, sont dans A et $\underline{\alpha} \in S$.

On a par exemple, en posant $p = (q \setminus \underline{\alpha})$

$$\frac{a:(\bar{\alpha}^*(b:c:0)) \xrightarrow{a} \bar{\alpha}^*(b:c:0) \quad , \quad d:(\alpha => c:0) \xrightarrow{d} (\alpha => c:0)}{q \xrightarrow{a.d} (\bar{\alpha}^*(b:c:0)//\alpha => c:0)}$$

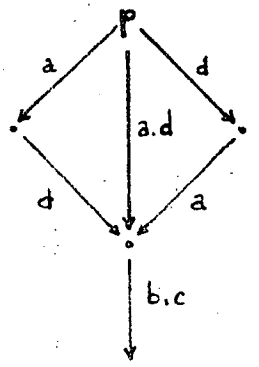
$$p \xrightarrow{a.d} (\bar{\alpha}^*(b:c:0)//\alpha => c:0) \setminus \underline{\alpha}$$

puis :

$$\frac{b:c:0 \xrightarrow{b} c:0}{\bar{\alpha}^*(b:c:0) \xrightarrow{\bar{\alpha}.b} \bar{\alpha}^*(c:0)} \qquad \frac{c:0 \xrightarrow{c} 0}{\alpha => (c:0) \xrightarrow{\alpha.c} 0}$$

$$\frac{(\bar{\alpha}^*(b:c:0)//\alpha => c:0) \xrightarrow{b.c} (\bar{\alpha}^*(c:0)//0)}{(\bar{\alpha}^*(b:c:0)//\alpha => c:0) \setminus \underline{\alpha} \xrightarrow{b.c} (\bar{\alpha}^*(c:0)//0) \setminus \underline{\alpha}}$$

et l'on peut tracer le graphe des transitions possibles à partir de p (en omettant les expressions intermédiaires)



$(\bar{\alpha}^*c:0//0) \setminus \underline{\alpha}$

et l'expression $(\bar{\alpha}@c:\mathbb{0}//\mathbb{0}) \setminus \underline{\alpha}$ est bloquée.

Exemple

soit p = (x where x <= (a:α => b:x//c:$\bar{\alpha}$ => d:$\mathbb{0}$) \ $\underline{\alpha}$)

Posons p_1 = a: (α => b:x)

p_2 = c:($\bar{\alpha}$ => d:$\mathbb{0}$)

et $q = (p_1//p_2) \setminus \alpha$

de sorte que p = (x where x <= q).

On a

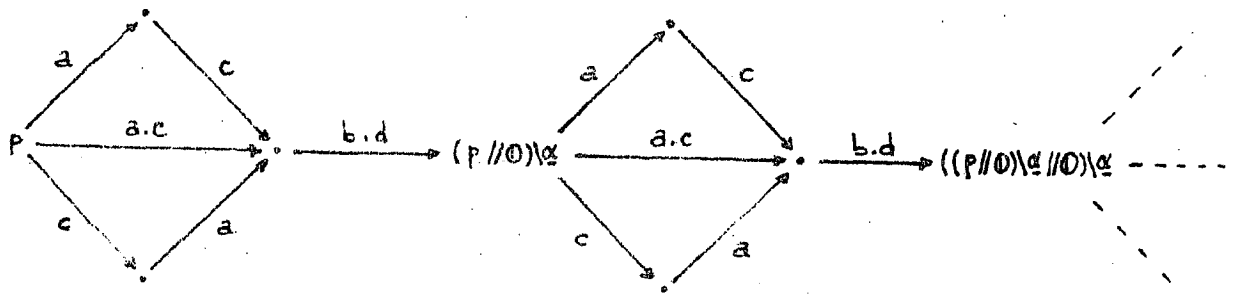
$$[(x \text{ where } x <= q)/x]p_1 \xrightarrow{a} \alpha => b:(x \text{ where } x <= q)$$

$$[(x \text{ where } x <= q)/x]p_2 \xrightarrow{c} \bar{\alpha} => d:\mathbb{0}$$

d'où $[(x \text{ where } x <= q)/x]q \xrightarrow{a.c} q'$ par exemple (le lecteur pourra écrire l'expression q') et donc

$$(x \text{ where } x <= q) \xrightarrow{a.c} q'$$

et le graphe des transitions possibles à partir de p est donné par :



Exemple

Soit p = (x where x <= (a:$\mathbb{0}$//x))

On peut voir que

$$p \xrightarrow{a} (\mathbb{0}//p) \quad \text{donc}$$
$$p \xrightarrow{a.a} (\mathbb{0}//(\mathbb{0}//p)) \quad \text{etc.}$$

C'est-à-dire que $\forall n \in \mathbb{N}\ \exists\ p_n : p \xrightarrow{a^n} p_n$.

Cet exemple montre que l'ensemble des actions possibles d'un agent et de ses actions initiales peut être infini (branchement infini). Cependant ces actions ne sont composées que d'un nombre fini d'actions atomiques. On peut le voir en définissant la sorte d'une expression, partie de $\mathbb{A} \cup S$:

(i) sorte $(\mathbb{0}) = \phi =$ sorte (x) pour tout $x \in X$

(ii) sorte $(a:p) = B \cup$ sorte (p) où B est le plus petit sous-ensemble de $\mathbb{A} \cup S$ tel que $a \in B^{\circledast}$ (donc par exemple $B = \emptyset$ si $a = 1$)

(iii) sorte $(p//q) =$ sorte $(p) \cup$ sorte (q)

(iv) sorte $(x_i \text{ where } R) = \bigcup_{1 \le j \le n}$ sorte $([(x_k \text{ where } R)/x_k]p_j)$

où $R = x_1 \Leftarrow p_1, \ldots, x_n \Leftarrow p_n$

(ici on pourrait avoir une définition plus restrictive, en ne faisant intervenir que les indices j des variables x_j dont la définition contribue effectivement aux actions de l'expression considérée)

(v) sorte $(\langle\varphi\rangle p) = B$ où B est le plus petit sous-ensemble de $\mathbb{A} \cup S$ tel que φ(sorte $(p)) \subseteq B^{\circledast}$

(vi) sorte $(p \setminus \underline{\alpha}) =$ sorte $(p) - \{\underline{\alpha}\}$

(vii) sorte $(s \Rightarrow p) =$ sorte $(p) \cup \{\underline{\alpha}\} =$ sorte $(s^{\#}p)$ si $s \in \{\alpha, \overline{\alpha}\}$.

proposition :

Pour tout $p \in \mathcal{L}$, sorte (p) est une partie finie de $\mathbb{A} \cup S$ telle que

$$p \xrightarrow{a} p' \Rightarrow a \in \text{sorte}(p)^{\circledast} \text{ et sorte}(p') \subseteq \text{sorte}(p)$$

preuve: par induction sur la structure de p et la définition des relations de transition, routine■

Remarques :

- sorte (p) est, modulo les renommages et restrictions, l'ensemble des actions qui apparaissent de façon irréductible dans l'expression p.

- sorte (p) n'est pas en général le plus petit sous-ensemble de $\mathbb{A} \cup S$ satisfaisant la précédente proposition. Par exemple la seule action possible de l'expression $(\alpha \Rightarrow \bar{\alpha}:0)$ est τ, mais sa sorte est $\{\underline{\alpha}\}$.

- A l'inverse il se peut que sorte (p) = $\emptyset$ mais que p ne soit pas pour autant bloqué. Il ne peut alors effectuer que des actions τ. C'est le cas par exemple pour $(\alpha:0 // \underline{\alpha}:0) \setminus \underline{\alpha}$.

La question de savoir si, pour une expression p, l'ensemble

$$\{(a,q) \;/\; (a,q) \in \mathbb{M} \times \mathcal{L} \;\&\; p \xrightarrow{a} q\}$$

est vide ou non n'est donc pas complètement triviale (essentiellement à cause des restrictions, car pour les termes écrits sans cette opération, cette question est décidable).

Il ne sera peut-être pas inutile de préciser ici la définition de la substitution $[\underline{\beta} / \underline{\alpha}]$ qui intervient dans l'opération de substitution de termes à des variables libres :

pour $\underline{\alpha}$, $\underline{\beta}$ dans S

(i) $[\underline{\beta} / \underline{\alpha}]0 = 0$ et $[\underline{\beta} / \underline{\alpha}]x = x$ pour $x \in X$

(ii) $[\underline{\beta} / \underline{\alpha}](a:p) = \varphi(a):[\underline{\beta} / \underline{\alpha}]p$ où φ est l'endomorphisme déterminé par $\varphi(\alpha) = \beta$

(iii) $[\underline{\beta} / \underline{\alpha}](p // q) = ([\underline{\beta} / \underline{\alpha}]p // [\underline{\beta} / \underline{\alpha}]q)$

(iv) $[\underline{\beta} / \underline{\alpha}](x_i \text{ where } x_1 <= p_1 , \ldots , x_n <= p_n) =$
$(x_i \text{ where } x_1 <= [\underline{\beta} / \underline{\alpha}]p_1 , \ldots , x_n <= [\underline{\beta} / \underline{\alpha}]p_n)$

(v) $[\underline{\beta} / \underline{\alpha}](\langle\varphi\rangle p) = \langle\psi \circ \varphi\rangle p$ où ψ est l'endomorphisme déterminé par $\psi(\alpha) = \beta$

(vi)

$$[\underline{\beta} / \underline{\alpha}](p\backslash\underline{\Upsilon}) = \begin{cases} (p\backslash\underline{\Upsilon}) & \text{si } \underline{\Upsilon} = \underline{\alpha} \\ ([\underline{\beta} / \underline{\alpha}][\underline{\Upsilon}' / \underline{\Upsilon}]p) \setminus \underline{\Upsilon}' & \text{où } \underline{\Upsilon}' \in \{\underline{\beta}, \underline{\alpha}\} \text{ sinon} \end{cases}$$

(vii) $[\underline{\beta} / \underline{\alpha}](s => p) = \varphi(s) => [\underline{\beta} / \underline{\alpha}]p$ et
$[\underline{\beta} / \underline{\alpha}](s * p) = \varphi(s) * [\underline{\beta} / \underline{\alpha}]p$

où φ est l'endomorphisme défini par $\varphi(\alpha) = \beta$.

Remarque: cette opération n'est pas un renommage, (c'est une sorte de renommage avec portée).

On a par exemple

$$[\alpha a:0/x , \bar{\alpha} b:0/y](((\langle\alpha a/a\rangle x // \langle\bar{\alpha} b/b\rangle y) \setminus \underline{\alpha}) = ((\langle\beta a/a\rangle \alpha a:0 // \langle\bar{\beta} b/b\rangle \bar{\alpha} b:0) \setminus \underline{\beta}$$

On a vu dans les exemples précédents que, par exemple, le terme $(\bar{\alpha} * c:0 // 0) \setminus \underline{\alpha}$, tout comme 0, n'a pas d'actions, que (dans le second exemple) $(p // 0) \setminus \underline{\alpha}$ a le même graphe de transition que p (au nom des noeuds près).

Il importe, pour rendre compte de la signification intuitive des

opérations choisies, de ne pas distinguer ces termes. C'est pourquoi nous définissons, en nous inspirant de la notion de bisimulation de R. Milner [8] et d'observation [3], des équivalences entre agents (rappelons ici que l'ensemble $\mathcal{A}$ des agents est formé des termes clos, c'est-à-dire sans variable libre). Chacune de ces équivalences est relative à un critère d'observation, qui est une famille d'"observables"; chaque observable à son tour est un ensemble de suites (finies) d'actions (séquences dans le temps) observables mais indistinguables entre elles. Formellement :

> un <u>critère d'observation</u> est une famille E de parties de $\mathbb{M}^*$; chaque élément de e est un <u>observable</u> dans le critère considéré.

Ici $\mathbb{M}^*$ désigne le monoide libre (ensemble des suites finies) engendré par $\mathbb{M}$. Nous désignerons par ε son élément neutre, et la concaténation d'un mot u et d'un mot v sera dénotée u:v (alors que la composition instantanée d'actions a et b est parfois notée ab plutôt que a.b).

On peut étendre naturellement les relations de transition aux séquences finies d'actions, en définissant $\xrightarrow{w}$ pour $w \in \mathbb{M}^*$:

$$p \xrightarrow{w} p' \ \underline{\text{ssi}} \quad \text{(i)} \ \ w = \varepsilon \ \text{et} \ p' = p \ \text{ou bien}$$
$$\text{(ii)} \ \ w = a:u \ \&\ \exists\, q:\ p \xrightarrow{a} q \ \&\ q \xrightarrow{u} p'$$

Du point de vue d'un critère E, un agent p satisfait une observation $e \in E$ <u>ssi</u> , par définition :

$$\exists\, u \in e \ \exists\, p':p \xrightarrow{u} p'$$

relation que l'on peut noter, suivant [3]: $p \overset{e}{\Rightarrow} p'$ (transition modulo E).

On peut alors dire qu'une relation d'équivalence R sur l'ensemble $\mathcal{A}$ des agents satisfait le critère d'observation E si deux termes équivalents (modulo R) sont indistinguables via les observables de E. Autrement dit: si les relations de transition modulo E passent au quotient $\mathcal{A}/R$ (suivant R. Milner [8], on pourrait appeler une telle équivalence une

E-bisimulation). Formellement :

une équivalence R sur $\mathcal{A}$ <u>satisfait</u> le critère d'observation E <u>ssi</u>

$$(p,q) \in R \Rightarrow \forall e \in E \; \forall p' \; \underline{si} \; p \overset{e}{\Rightarrow} p' \; \underline{alors} \; \exists q' : q \overset{e}{\Rightarrow} q' \; \& \; (p',q') \in R$$

(propriété de commutation : $R \circ \overset{e}{\Rightarrow} \subseteq \overset{e}{\Rightarrow} \circ R$ pour $e \in E$).

Appelons $\mathcal{C}$ le treillis complet des équivalences sur $\mathcal{A}$ pour l'ordre de finesse $\leq$ (c'est-à-dire l'inclusion inverse) et $\mathcal{C}_E$ l'ensemble des équivalences sur $\mathcal{A}$ qui satisfont E. Rappelons que dans $\mathcal{C}$ la borne inférieure d'une famille non vide $\{R_i / i \in I\}$ est donnée par

$$\bigwedge_{i \in I} R_i = \left(\bigcup_{i \in I} R_i \right)^+ \quad \text{(fermeture transitive de l'union)}$$

Il est facile de voir que, si $\{R_i / i \in I\} \subseteq \mathcal{C}_E$, alors c'est là encore la borne inférieure de cette famille dans $\mathcal{C}_E$. Comme la diagonale de $\mathcal{A} \times \mathcal{A}$ (la relation $\{(p,p)/p \in \mathcal{A}\}$) est une équivalence satisfaisant E plus fine que toute autre, à tout élément R de $\mathcal{C}$ on peut associer l'équivalence $f_E(R)$ satisfaisant E la plus grossière plus fine que R :

$$f_E(R) = \bigwedge \{R' / R' \in \mathcal{C}_E \; \& \; R \leq R'\}$$

Cette application de $\mathcal{C}$ vers $\mathcal{C}_E$ est une fermeture de $\mathcal{C}$, ie :

(i) $R \leq R' \Rightarrow f_E(R) \leq f_E(R')$

(ii) $R \leq f_E(R)$

(iii) $f_E \circ f_E = f_E$

En particulier, elle est monotone, et donc admet (théorème de Knaster-Tarski) un plus petit point fixe $\sim_E$ qui est donné (puisqu'évidemment $f_E(R) = R \Leftrightarrow R \in \mathcal{C}_E$) par :

$$\sim_E = (\bigcup \{R / R \in \mathcal{C}_E\})^+$$

Comme nous l'avons dit, toutes les équivalences satisfaisant E sont points fixes de f_E, cette équivalence (qui satisfait E) est la plus grossière de toutes :

$$R \in \mathcal{T}_E \Rightarrow P \subseteq \sim_E$$

Nous l'appelerons <u>équivalence de E-observation</u> (et nous pourrons parler en général d'équivalence d'observation). La propriété qui vient d'être énoncée fonde un principe de preuve, bien connu sous le nom d'induction de Park (voir [8]). Nous utiliserons ici :

<u>principe de preuve</u> (relatif à un critère d'observation E) :

pour que $p \sim_E q$, il (faut et il) suffit qu'il existe une relation $P \subseteq \mathcal{A} \times \mathcal{A}$ symétrique, telle que $(p,q) \in P$, qui satisfait (le principe de preuve)

$$(p,q) \in P \Rightarrow \forall e \in E \; \forall p' : \underline{\text{si}} \; p \overset{e}{\Rightarrow} p' \; \underline{\text{alors}} \; \exists q' : q \overset{e}{\Rightarrow} q' \text{ et } (p',q') \in (P \cup \sim_E)^+$$

<u>validité</u>: il suffit de voir qu'alors $(P \cup \sim_E)^+$ est dans $\mathcal{T}_E$, d'où $P \subseteq \sim_E$.

Sans entreprendre ici une étude systématique de la notion d'équivalence d'observation, on peut donner quelques exemples (voir [8]):

(1) La plus grossière équivalence d'observation est $\mathcal{A} \times \mathcal{A}$, déterminée par le critère trivial $\perp = \{M^*\}$.

(2) A l'opposé, si l'on se donne comme critère d'observation que toutes les séquences d'actions sont observables, et toutes distinguées, c'est-à-dire

$$T = \{\{w\} \; / \; w \in M^*\}$$

alors on obtient la plus fine de toutes les équivalences d'observation, que nous noterons simplement $\sim$, et qui sera (provisoirement) désignée sous le nom d'équivalence d'observation

forte.

Notons que le critère d'observation T est équivalent à (détermine la même équivalence que)

$$\{\{a\} \ / \ a \in \mathbb{M}\}$$

et c'est cette présentation de l'équivalence forte qui sera adoptée lorsqu'il s'agira de faire des preuves.

(3) Ce que R. Milner appelle équivalence d'observation (cf [3,8]) est celle, notée $\approx$, qui est associée au critère

$$O = \{\pi^{-1}(w) \ / \ w \in (\mathbb{M} - \{1\})^{\#}\}$$

où π est la projection (morphisme) du monoide $\mathbb{M}^{\#}$ sur lui-même déterminée par $\pi(1) = \varepsilon$ (et $\pi(a) = a$ pour $a \in \mathbb{M} - \{1\}$).

La signification intuitive de ce critère d'observation est que l'action 1 est interprétée comme une action d'attente ou de délai, qui, entre les autres actions, n'est pas observable : toute autre action est observable en elle même, mais non le moment exact auquel elle intervient, puisque les observables sont ici

$$\{1\}^{\#} \text{ et } \{1\}^{\#} a_1 \{1\}^{\#} \ldots \{1\}^{\#} a_n \{1\}^{\#} \quad \text{pour } a_i \in \mathbb{M} - \{1\}$$

R. Milner montre ([8]) que ce critère est équivalent à

$$\{\{1\}^{\#}, \{1\}^{\#} a \quad \text{pour } a \in \mathbb{M} - \{1\}$$

Par contre on n'obtient pas la même chose si l'on décide que l'attente n'est pas observable, c'est-à-dire avec

$$O' = \{\{1\}^{\#} a_1 \{1\}^{\#} \ldots \{1\}^{\#} a_n \{1\}^{\#} \ / \ n>0 \ \& \ a_i \in \mathbb{M} - \{1\}\}$$

ou $\{\{1\}^{\#} a \ / \ a \in \mathbb{M} - \{1\}\}$.

Par exemple, on a (en utilisant la somme qui sera définie plus loin)

$$(1: 0 + a: 0) \sim_0 a: 0$$

alors que ces deux termes n'entretiennent pas la relation $\approx$. On a aussi :

$$1: 0 \approx 0 \quad \text{mais} \quad 1: 0 \not\sim 0$$

(4) On peut imaginer d'autres critères d'observation. Par exemple on peut admettre que l'action 1 (qui exprime aussi, nous l'avons vu, une communication interne) est invisible, mais prends du temps ; c'est-à-dire que le temps d'attente d'une action observable est lui-même observable, ce qui est traduit dans

$$E = \{\{1^n a\} \ / \ n \in \mathbb{N} \ \& \ a \in \mathbb{M} - \{1\}\}$$

Ici encore on a

$$1: 0 \sim_E 0 \quad \text{et} \quad (1: 0 + a: 0) \sim_E a: 0$$

Par contre $1: a: 0 \approx a: 0$ alors que $1: a: 0 \not\sim_E a: 0$.

Si l'on voulait exprimer que deux actions a et b n'entrent pas en conflit et que l'ordre de leur exécution relatif n'est pas pertinent, on peut avoir un observable comme

$$\{a:b \ , \ ab \ , \ b:a\}$$

Ou encore, si une action a "fait la même chose" qu'une séquence $b_1 :...: b_k$, on admettra comme observable

$$\{a, b_1 :...: b_k\}$$

La notion d'observable permet donc d'introduire un peu de sémantique des

actions.

Les équivalences d'observation concernent les agents. On peut les étendre aux expressions du langage, en considérant celles-ci comme des contextes, ou plus exactement comme des schémas de fonctions sur l'ensemble des agents : si $\{x_{i_1}, \ldots, x_{i_k}\}$ est l'ensemble des variables libres de l'expression $q \in \mathcal{L}$, cette expression définit une fonction à k arguments sur $\mathcal{A}$, avec

$$\hat{q}(p_1, \ldots, p_k) = [p_1/x_{i_1}, \ldots, p_k/x_{i_k}]\, q \quad \text{(substitution)}$$

D'où la définition de l'extension d'une équivalence d'observation $\sim_E$ à $\mathcal{L}$:

pour p,q dans $\mathcal{L}$:
$p =_E q$ <u>ssi</u> pour toute substitution close (de X dans $\mathcal{A}$)
$\rho : [\rho]p \sim_E [\rho]q$

Sur $\mathcal{L}$, cette relation est évidemment une équivalence stable par substitution (quelconque) compatible avec les relations de transition, dans laquelle chaque identificateur constitue une classe.

D'une façon générale, un calcul de processus (voir [8]) apparaît comme la donnée :

- d'une syntaxe : les opérateurs primitifs
- d'une sémantique opérationnelle structurelle
- d'une équivalence d'observation $=_E$ qui est une congruence.

Les deux premiers points qui définissent le calul que nous appelons MEIJE ont déjà été introduits ; il reste à voir que :

<u>proposition</u>:

$=$ est une congruence (sur $\mathcal{L}$) : la <u>congruence forte</u>.

La preuve en est exactement la même que celle que donne R. Milner pour la proposition analogue concernant SCCS [8]. Nous ne la répétons pas. Indiquons seulement dans deux cas (simples) comment y est utilisé le

principe de preuve :

- on veut montrer par exemple que

$$p \simeq p' \ \& \ q \simeq q' \ \Rightarrow \ (p//q) \simeq (p'//q')$$

Pour cela on considère la relation (symétrique) $P \subseteq \mathcal{A} \times \mathcal{A}$ constituée des couples $((p//q),(p'//q'))$ pour p,p',q,q' dans $\mathcal{A}$ tels que $p \sim p'$ et $q \sim q'$. On montre que P satisfait le principe de preuve relatif au critère d'observation qui définit l'équivalence forte:

$$(r,r') \in P \Rightarrow \forall a \in \mathbb{M} \ \forall t \ \underline{\text{si}} \ r \xrightarrow{a} t \ \underline{\text{alors}} \ \exists t' :$$

$$r' \xrightarrow{a} t' \ \& \ (t,t') \in (P \cup \sim)^+$$

Par définition des relations de transitions, on a ici trois cas :

$$(p//q) \xrightarrow{a} r \Leftrightarrow \text{(i)} \ p \xrightarrow{a} r_1 \ \& \ r = (r_1//q) \text{ ou bien}$$

$$\text{(ii)} \ \exists \ b,c \in \mathbb{M} : a=b.c \quad \text{et}$$

$$p \xrightarrow{b} r_1 \ , \ q \xrightarrow{c} r_2 \ \& \ r = (r_1//r_2) \text{ ou bien}$$

$$\text{(iii)} \ q \xrightarrow{a} r_2 \ \& \ r = (p//r_2)$$

Dans le second cas par exemple, comme on a par hypothèse $p \sim p'$ et $q \sim q'$, et puisque, par définition, $\sim$ satisfait le critère d'observation $\{\{a\}/a \in \mathbb{M}\}$:

$$p \xrightarrow{b} r_1 \ \& \ p \sim p' \Rightarrow \exists r'_1 : p' \xrightarrow{b} r'_1 \ \& \ r'_1 \sim r_1$$

$$q \xrightarrow{c} r_2 \ \& \ q \sim q' \Rightarrow \exists r'_2 : q' \xrightarrow{c} r'_2 \ \& \ r'_2 \sim r_2$$

D'où $(p'//q') \xrightarrow{a} r' = (r'_1//r'_2)$ et $(r,r') \in P$, ce qu'il fallait montrer. Les deux autres cas se traitent de la même façon. D'où $P \subseteq \sim$ et dans $\mathcal{L}$

$$p = p' \ \& \ q = q' \Rightarrow (p//q) = (p'//q')$$

par définition de $=$.

- pour montrer, pour $s \in \$$, que

$$p = q \Rightarrow (s \Rightarrow p) = (s \Rightarrow q)$$

on considère

$$P = \{((s \Rightarrow p),(s \Rightarrow q)) \ / \ p,q \in \mathcal{A} \ \& \ p \sim q\}$$

On a :

$$(s \Rightarrow p) \xrightarrow{c} r \Leftrightarrow \exists\, b: p \xrightarrow{b} r \ \& \ c = s.b$$

Donc si $q \sim p$: $\exists\, r': q \xrightarrow{b} r' \ \& \ r' \sim r$

Par conséquent :

$$(s \Rightarrow q) \xrightarrow{c} r' \ \& \ r' \sim r$$

et là encore P satisfait le principe de preuve ∎

On appelera <u>processus</u> (du calcul MEIJE) un élément de l'algèbre quotient $\mathcal{P} = \mathcal{A}/\sim$, l'algèbre $\mathcal{S} = \mathcal{L}/=$ étant celle des schémas de processus. Les opérations quotient et relations de transition quotient y sont dénotées comme dans $\mathcal{L}$. Le reste de cette section est consacré à montrer quelques propriétés de cette algèbre.

On a vu que la congruence $=$ est stable par substitution. En fait les opérations de substitution (la classe d'un identificateur étant

réduite à cet identificateur) passent au quotient puisque :

propriété 1 :

si ρ et ρ' sont des substitions telles que $\rho = \rho'$
(ie $\forall x \in X : \rho(x) = \rho'(x)$.)
alors $p = q \Rightarrow [\rho]p = [\rho']q$

preuve: puisque $p = q \Rightarrow [\rho']p = [\rho']q$, il suffit de montrer que

$$\rho = \rho' \Rightarrow \forall p \in \mathcal{L} : [\rho]p = [\rho']p$$

En fait on montre que

$$p \equiv p' \ \& \ \rho = \rho' \Rightarrow [\rho]p = [\rho']p'$$

par induction structurelle sur p, en utilisant le fait que $=$ est une congruence ▪

La congruence forte contient d'ailleurs l'identité syntaxique :

propriété 2 :

$$p \equiv q \Rightarrow p = q$$

(par induction structurelle).

Notons que ces deux propriétés, ne tenant qu'à la définition de l'équivalence sur $\mathcal{L}$, sont vraies de toute congruence observationnelle.

On peut voir également que modulo la congruence forte, le n-uple de termes

$$(x_i \ \underline{\text{where}} \ x_1 <= p_1 , \ldots , x_n <= p_n)_{1 \leq i \leq n}$$

est solution du système d'équations $\begin{cases} x_1 = p_1 \\ \vdots \\ x_n = p_n \end{cases}$

C'est-à-dire que :

propriété 3 :

pour $R = x_1 <= p_1 , \ldots . x_n <= p_n$ et $1 \leq i \leq n$:
$(x_i \text{ where } R) = [(x_j \text{ where } R) / x_j]p_i$

preuve: on vérifie aisément que la fermeture symétrique de la relation

$$P = \{([\rho](x_i \text{ where } R) , [\rho][(x_j \text{ where } R) / x_j]p_i) /$$

$$1 \leq i \leq n \text{ et } \rho \text{ substitution close}\}$$

satisfait le principe de preuve.

D'une façon générale la preuve des propriétés données ici s'obtient en appliquant le principe de preuve à la fermeture symétrique des relations données par les couples de termes dont on veut montrer l'équivalence. On obtient par exemple :

propriétés 4 :

$(\mathcal{S}, //, 0)$ est un monoide abélien.

Ceci signifie que, pour p,q,r expressions du langage :

$(p//(q//r)) = ((p//q)//r)$
$(0//p) = (p//0) = p$
$(p//q) = (q//p)$

De ce fait on pourra utiliser la notation $(p_1 //\ldots// p_n)$. Il y a là une différence essentielle avec SCCS, dans lequel l'élément neutre de la composition parallèle (synchrone) est donné par :

$$1 =_{def} (x \text{ where } x <= 1:x)$$

qui n'est donc pas un terme "fini". Il n'y a pas non plus ici de relation simple entre préfixage et composition parallèle (comparer avec la proposition 5-3 de [8]).

Concernant les restrictions, on a :

propriétés 5 :

(1) $(p \setminus \underline{\alpha}) \setminus \underline{\beta} = (p \setminus \underline{\beta}) \setminus \underline{\alpha}$

$(p \setminus \underline{\alpha}) \setminus \underline{\alpha} = (p \setminus \underline{\alpha})$

(2) $\underline{\alpha} \notin \text{sorte}(p) \Rightarrow (p \setminus \underline{\alpha}) = p$

(3) $(a:p) \setminus \underline{\alpha} = \begin{cases} a: (p \setminus \underline{\alpha}) & \text{si } a \in M \setminus \underline{\alpha} \\ 0 & \text{sinon} \end{cases}$

(4) $(\langle \beta / \alpha \rangle p) \setminus \underline{\beta} = (p \setminus \underline{\alpha})$

(5) $(s \Rightarrow p) \setminus \underline{\alpha} = \begin{cases} s \Rightarrow (p \setminus \underline{\alpha}) & \text{si } s \notin \{\alpha, \overline{\alpha}\} \\ 0 \text{ si } s \in \{\alpha, \overline{\alpha}\} & \text{et } \underline{\alpha} \notin \text{sorte}(p) \end{cases}$

$(s * p) \setminus \underline{\alpha} = \begin{cases} s * (p \setminus \underline{\alpha}) & \text{si } s \notin \{\alpha, \overline{\alpha}\} \\ 0 \text{ si } s \in \{\alpha, \overline{\alpha}\} & \text{et } \underline{\alpha} \notin \text{sorte}(p) \end{cases}$

(dans les autres cas, ie si $s \in \{\alpha, \overline{\alpha}\}$ et $\underline{\alpha} \in \text{sorte}(p)$, on ne peut rien dire de général).

Il n'y a pas de relation simple entre les restrictions et la composition parallèle ; en général :

$(p // q) \setminus \underline{\alpha} \neq (p \setminus \underline{\alpha}) // (q \setminus \underline{\alpha})$ (par exemple $p = \alpha:0$ et $q = \overline{\alpha}:0$)

Cependant cette propriété de distributivité est vraie si $\underline{\alpha} \notin \text{sorte}(p) \cap \text{sorte}(q)$.

La propriété 5-1 autorise l'écriture d'expressions de la forme

$(p \setminus \underline{\alpha}_1, \ldots, \underline{\alpha}_n)$ pour $\{\underline{\alpha}_1, \ldots, \underline{\alpha}_n\} \subseteq S$

ou même (p\B) si B est une partie finie de S.

En utilisant les propriétés ci-dessus on peut montrer le fait, déjà mentionné, que certaines primitives du calcul pourraient être réduites :

propriétés 6 :

(1) $0 = (x \text{ where } x <= x)$

(2) $a.b:x = (\alpha => a:0 \;//\; \overline{\alpha} => b:x) \setminus \underline{\alpha}$ où $a,b \in \mathbb{M} \setminus \underline{\alpha}$

$1:x = (\overline{\alpha}:0 \;//\; \alpha:x) \setminus \underline{\alpha}$

Pour φ endomorphisme de $\mathbb{M}$:

propriétés 7 :

(1) si $a \in$ sorte(p) => $\varphi(a)=a$ alors $\langle\varphi\rangle p = p$ (pour $p \in \mathcal{A}$)
En particulier : $\langle\varphi\rangle\, 0 = 0$

(2) $\langle\varphi\rangle(a:p) = \varphi(a):\langle\varphi\rangle p$

(3) $\langle\varphi\rangle(p//q) = (\langle\varphi\rangle p \;//\; \langle\varphi\rangle q)$

(4) $\langle\varphi\rangle(\langle\psi\rangle p = \langle\varphi \circ \psi\rangle p$

(5)

$$\langle\varphi\rangle(s => p) = \begin{cases} (s_1 => \ldots (s_n => \langle\varphi\rangle p)\ldots) & \text{si } \varphi(s) = s_1 \ldots s_n \\ \langle\varphi\rangle p & \text{si } \varphi(s) = 1 \end{cases}$$

$$\langle\varphi\rangle(s * p) = \begin{cases} (s_1 *\ldots(s_n * \langle\varphi\rangle p)\ldots) & \text{si } \varphi(s) = s_1 \ldots s_n \\ \langle\varphi\rangle p & \text{si } \varphi(s) = 1 \end{cases}$$

propriétés 8 :

(1) $(s*0) = (s=>0) = 0$

(2) $(s => (a:p)) = s.a:p$
$(s * (a:p)) = s.a: (s*p)$

(3) $(s => (s'*p)) = (s' * (s=>p))$

Par contre en général

$$s => (p//q) \neq (s=>p) // (s=>q)$$
$$s * (p//q) \neq (s*p) // (s*q)$$

(faire par exemple $p = a:0$, $q = b:0$ avec $a,b \in M \setminus \{s,\bar{s}\}$). On peut noter qu'il y a des contextes permettant d'annuler l'effet du déclenchement et du pilotage sur action de synchronisation :

si $s \in \{\alpha,\bar{\alpha}\}$

$x = ((s=>x) // \bar{s}:0) \setminus \underline{\alpha}$

$x = ((s*x) // (y \text{ where } y <= \bar{s}:y)) \setminus \underline{\alpha}$.

3. SYNCHRONISATIONS

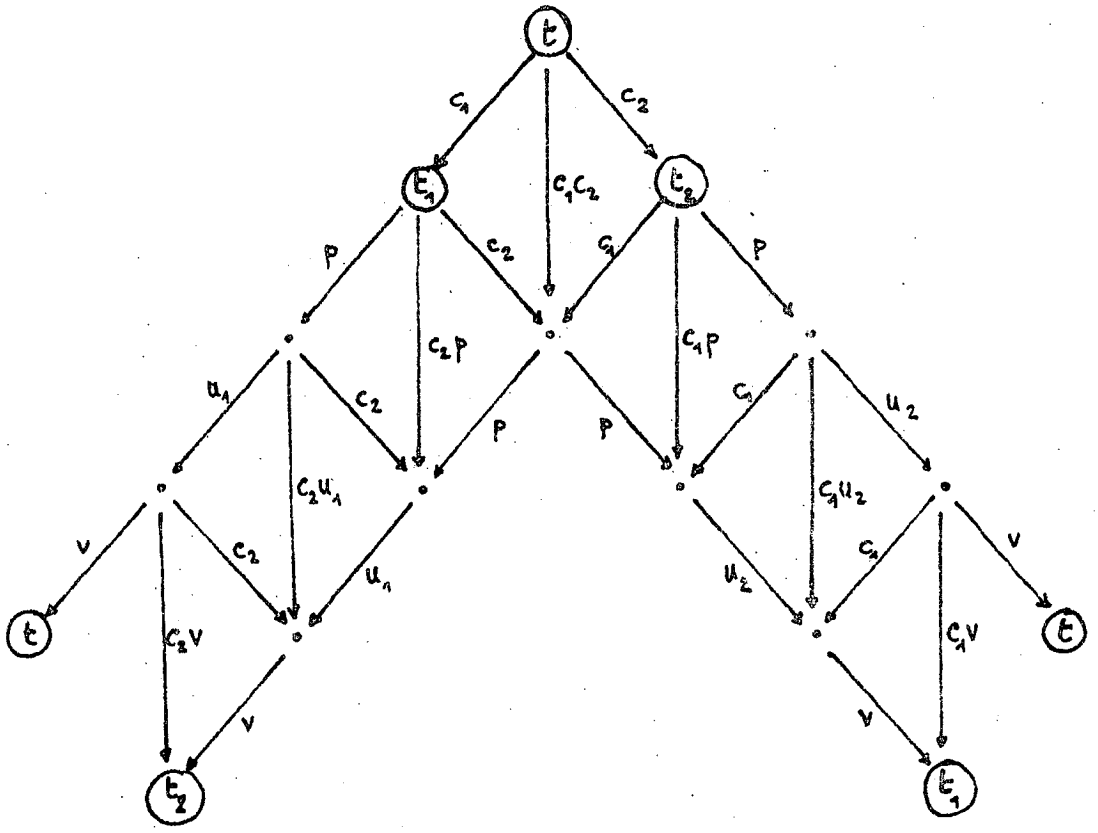
Un exemple typique de synchronisation entre processus est celui qui régit le partage d'une ressource à accès mutuellement exclusif. Supposons par exemple que ce partage doive se faire entre deux processus qui exécutent de manière cyclique la séquence d'actions :

"calculer" : "prendre" : "utiliser" : "relacher".

On peut les formaliser par les expressions (pour i = 1,2)

$$q_i = (x \text{ where } x <= c_i : p : u_i : v : x)$$

Ce que l'on veut obtenir, c'est un système synchronisé t, mettant q_1 et q_2 en parallèle, qui réalise le graphe de transitions :



Formuler cette synchronisation dans notre langage consiste à trouver une expression qui, lorsqu'on y plonge q_1 et q_2, réalise ce graphe de transitions. Pour ce faire, on va soumettre, par renommage, la possibilité de prendre et relacher la ressource à la reception de signaux _ et . Ces signaux seront émis cycliquement dans l'ordre convenable par un processus synchronisateur (sémaphore), l'échange des signaux étant obligé par restriction. Le terme recherché ici est donc :

$$(\langle\varphi\rangle q_1 \;//\; \langle\varphi\rangle q_2 \;//\; (x \;\underline{\text{where}}\; x <= \bar{\alpha};\bar{\beta};x)) \setminus \underline{\alpha}, \underline{\beta}$$

où $\varphi(p) = \alpha.p$ et $\varphi(v) = \beta.v$ (ici on suppose que c_1, c_2, p, v, u_1, u_2 sont dans A).

Un inconvénient de cette façon de faire est qu'en général (pour d'autres processus que q_1 et q_2), l'un des agents peut sortir de sa section critique sans y être entré, lorsque l'autre y est entré. Avant de

remédier à cela, on peut cependant voir un avantage de la sémantique adoptée ici pour la composition parallèle : lorsque l'un des processus est entré dans sa section critique, le processus synchronisateur attends (sans rien faire) la demande de sortie, sans bloquer l'agent actif. De même si l'autre processus demande à entrer dans sa section critique, il attends sans bloquer le système. Ici l'on n'a pas à introduire, comme dans SCCS [7], des attentes explicites désynchronisant les processus, ce qui en général conduit à des attentes infinies.

Pour résoudre la petite difficulté soulevée ci-dessus, il faut pouvoir distinguer, dans le synchronisateur, à quel processus est donnée l'autorisation d'entrer ou de sortir de sa zone critique. Pour ce faire nous avons besoin d'une opération de <u>somme</u>, notée +, qui exprime que (p+q) se comporte comme l'un ou l'autre des processus p et q. Formellement, la sémantique opérationnelle de cette construction est spécifiée par (cf [7]) :

$$\frac{p \xrightarrow{a} p'}{(p+q) \xrightarrow{a} p'} \qquad \frac{q \xrightarrow{b} q'}{(p+q) \xrightarrow{b} q'}$$

Là encore, nous voulons définir cette opération comme une construction dérivée. C'est-à-dire trouver un contexte qui, à la congruence près, la réalise. L'idée est simple : il suffit de déclencher les agents sur la réception d'un signal, et de ne donner qu'une seule "autorisation" (ie disposer d'un synchronisateur qui émet un signal et puis s'arrête). Formellement :

$$(x+y) =_{def} (\alpha\Rightarrow x \;//\; \alpha\Rightarrow y \;//\; \bar{\alpha}:0) \setminus \underline{\alpha}$$

(où $x,y \in X$).

Il faut montrer que cette définition est correcte, c'est-à-dire satisfait bien la spécification annoncée. Supposons que, pour p et q dans

$$[p/x \;,\; q/y](\alpha\Rightarrow x \;//\; \alpha\Rightarrow y \;//\; \bar{\alpha}:0) \setminus \underline{\alpha} \xrightarrow{a} r$$

En examinant les cas possibles (où, compte tenu de la définition de la substitution, on peut supposer $\underline{\alpha} \notin sorte(p) \cup sorte(q)$), on voit que ceci équivaut à

(i) $p \xrightarrow{a} p' \;\&\; r = (p' \;//\; \alpha => q \;//\; 0) \setminus \underline{\alpha}$ ou bien

(ii) $q \xrightarrow{a} q' \;\&\; r = (\alpha => p \;//\; q' \;//\; 0) \setminus \underline{\alpha}$

Dans le premier cas :

$$\begin{aligned}(p' \;//\; \alpha => q \;//\; 0) \setminus \underline{\alpha} &= (p' \;//\; \alpha => q) \setminus \underline{\alpha} \text{ (propriété 4)} \\ &= (p' \;//\; (\alpha => q) \setminus \underline{\alpha}) \text{ (car } \underline{\alpha} \notin sorte(p'), \text{ et on applique prop. 5-2)} \\ &= (p' \;//\; 0) \text{ (prop. 5-5, car } \underline{\alpha} \notin sorte(q)) \\ &= p' \text{ (prop. 4)}\end{aligned}$$

Et de même dans le cas (ii) on a $r = q'$. Par conséquent l'expression choisie répond bien, modulo la congruence forte, à la spécification de la somme. On peut donc manipuler cette opération, avec laquelle l'équivalence forte reste compatible, par sa spécification. Avant d'énoncer quelques unes de ses propriétés, montrons comment l'utiliser pour le problème du partage de ressource évoqué au début de cette section. En notant pour la circonstance par χ cette synchronisation :

$$(x \chi y) =_{def} (<\varphi_1>x \;//\; <\varphi_2>y \;//\; sem) \setminus \underline{\alpha}_1 , \underline{\beta}_1 , \underline{\alpha}_2 , \underline{\beta}_2$$

où $\varphi_i(p) = \alpha_i p$, $\varphi_i(v) = \beta_i v$ pour $i = 1,2$ et

$$sem = (z \;\underline{where}\; z <= (\bar{\alpha}_1 : \bar{\beta}_1 : z + \bar{\alpha}_2 : \bar{\beta}_2 : z)) \text{ (sémaphore)}$$

(on pourrait vérifier que c'est là une opération commutative et associative, qui admet 0 pour élément neutre). Donnons maintenant des propriétés de la somme (en notant (p+q) pour $[p/x , q/y](x+y)$, c'est-à-dire pour $[p/x , q/y]((\alpha=>x \;//\; \alpha=>y \;//\; \bar{\alpha}:0) \setminus \underline{\alpha})$)

<u>propriétes 9</u> :

(1) $\text{sorte}(p+q) = \text{sorte}(p) \cup \text{sorte}(q)$

(2) $((p+q)+r) \simeq (p+(q+r))$

$(p+\emptyset) \simeq (\emptyset+p) \simeq p$

$(p+q) \simeq (q+p)$

$(p+p) \simeq p$

(3) $(p+q) \setminus \underline{\alpha} \simeq (p\setminus\underline{\alpha}) + (q\setminus\underline{\alpha})$

$\langle\varphi\rangle(p+q) \simeq \langle\varphi\rangle p + \langle\varphi\rangle q$

$s \Rightarrow (p+q) \simeq (s\Rightarrow p) + (s\Rightarrow q)$

$s \circledast (p+q) \simeq (s \circledast p) + (s \circledast q)$

Ces propriétés se montrent soit par raisonnement équationnel (c'est le cas par exemple pour l'idempotence), en utilisant les propriétés déjà connues de la congruence forte, soit en utilisant le principe de preuve.

Dans la suite de cette section nous traitons ainsi de nombreux exemples de synchronisation :

- un mécanisme de synchronisation apparait comme une fonction sur les processus (ou plus généralement sur $\mathcal{S}$). Nous en spécifions la sémantique opérationnelle dans le style structurel de [9] adopté ici.

- nous donnons une expression du langage qui, modulo la congruence forte, réalise cette fonction. (Pour être tout-à-fait rigoureux, il faudrait utiliser ici la relation de transition modulo $\simeq$: $p \overset{a}{\leadsto} q$ <u>ssi</u> $\exists\, q' : p \xrightarrow{a} q'$ & $q' \simeq q$).

Nous omettrons en général les preuves de l'adéquation de l'expression proposée et des propriétés de l'opérateur dérivées (et là deux techniques sont possibles : en utilisant le principe de preuve, compte tenu de la nouvelle spécification, ou par raisonnement équationnel sur les expressions).

On peut par exemple étendre les opérations de déclenchement et de pilotage, sur des actions $a \in \mathbb{M}$ quelconques. La spécification en est

évidente ; pour les réaliser, il suffit de déclencher (resp. piloter) simultanément le processus considéré et un synchronisateur qui exécute (répétitivement) l'action a :

$$(a{=>}x) =_{def} (\alpha{=>}x \;//\; \bar{\alpha}.a{:}0) \setminus \underline{\alpha}$$

$$(a{*}x) =_{def} (\alpha{*}x \;//\; (y \;\underline{\text{where}}\; y <= \bar{\alpha}a{:}y)) \setminus \underline{\alpha}$$

où $\underline{\alpha}$ est tel que $a \in \mathbb{M} \setminus \underline{\alpha}$.

Ces opérations ont des propriétés analogues à celles de s=> et s*, à quoi on peut ajouter :

propriétés 8' :

$$a => (b => p) = (ab => p)$$
$$a * (b * p) = (ab * p)$$

Une autre opération très souvent considérée dans la littérature ([3,5,10]) est l'entrelacement ("interleaving", ou "shuffle", ou encore "merge") (p I q) de p et q. C'est une composition parallèle qui consiste à entremêler les séquences d'actions de p et q, en interdisant leur simultanéité :

$$\frac{p \xrightarrow{a} p'}{(p \;I\; q) \xrightarrow{a} (p' \;I\; q)} \qquad \frac{q \xrightarrow{b} q'}{(p \;I\; q) \xrightarrow{b} (p \;I\; q')}$$

Pour définir cette opération dans le langage, il suffit de piloter p et q sur la réception d'un même signal, avec un synchronisateur qui ne délivre qu'un seul signal à la fois :

$$(x \;I\; y) =_{def} (\alpha{*}x \;//\; \alpha{*}y \;//\; (z \;\underline{\text{where}}\; z <= \bar{\alpha}{:}z)) \setminus \underline{\alpha}$$

En notant encore par (p I q) le résultat de la substitution de p à x et de q à y dans l'expression en membre droit, on a les propriétés :

propriétés 10 :

(1) $((p \mid q) \mid r) \simeq (p \mid (q \mid r))$

$(p \mid 0) \simeq (0 \mid p) \simeq p$

$(p \mid q) \simeq (q \mid p)$

(2) $(p \mid q) \setminus \underline{\alpha} \simeq (p\setminus\underline{\alpha}) \mid (q\setminus\underline{\alpha})$

$\langle\varphi\rangle(p \mid q) \simeq ((\langle\varphi\rangle p \mid \langle\varphi\rangle q)$

$(s^{@}(p \mid q)) \simeq ((s^{@}p) \mid (s^{@}q))$

Par contre (nous laissons au lecteur le soin de trouver les contre exemples) en général :

$(p//(q \mid r)) \not\simeq (p//q) \mid (p//r)$

$(p+q) \mid r \not\simeq (p \mid r) + (q \mid r)$

$s \Rightarrow (p \mid r) \not\simeq (s\Rightarrow p) \mid (s\Rightarrow r)$

Une autre forme de composition parallèle a été introduite par R. Milner ([7,8]), c'est la composition synchrone (p x q) qui au contraire ne peut qu'exécuter des actions qui résultent de l'activité simultanée de p et de q :

$$\frac{p \xrightarrow{a} p' \ , \ q \xrightarrow{b} q'}{(p \times q) \xrightarrow{ab} (p' \times q')}$$

Le principe de réalisation de cette synchronisation est le même que pour l'entrelacement, mais ici le synchronisateur doit forcer l'activité simultanée, c'est-à-dire émettre en même temps les deux signaux qui pilotent p et q :

$$(x \times y) =_{def} (\alpha^{@}x \;//\; \alpha^{@}y \;//\; (z \text{ where } z \Leftarrow \bar{\alpha}\bar{\alpha};z)) \setminus \underline{\alpha}$$

On peut aussi avoir ici une solution "distribuée" :

$$(x \times y) \simeq (\alpha^{@}x) \;//\; \bar{\alpha}^{@}y) \setminus \underline{\alpha}$$

On retrouve pour cette opération les propriétés données par R. Milner :

propriétés 11 :

(1) $((p \times q) \times r) = (p \times (q \times r))$
$(p \times q) = (q \times p)$
$(\mathbb{1} \times p) = p$ (où, rappelons-le : $\mathbb{1} = (x \text{ where } x <= 1:x)$.)
$(\mathbb{0} \times p) = \mathbb{0}$

(2) $(a:p) \times (b:q) = ab : (p \times q)$
$\langle\varphi\rangle(p \times q) = (\langle\varphi\rangle p \times \langle\varphi\rangle q)$
$a => (p \times q) = (a => p) \times q$
$a * (p \times q) = (a * p) \times q$
$p \times (q + r) = p \times q + p \times r$

(3) $(a => p) = ((p \times (x \text{ where } x <= a:1))$
$(a * p) = (p \times (x \text{ where } x <= a:x))$

Dans les exemples qui viennent d'être traités (dont certains sont des cas particuliers d'ordonnancement pur, qui prend la forme $(\alpha^* p \;//\; \beta^* q \;//\; syn) \setminus \underline{\alpha}, \underline{\beta}$ où les actions possibles de syn sont dans $\{\overline{\alpha}, \overline{\beta}\}^{@}$), on voit apparaitre une forme commune de processus synchronisateur. Ceux-ci font une action donnée $\tau \in \mathbb{M}$ et se reconfigurent en eux-mêmes, c'est-à-dire que leur spécification est :

$$h_\tau \xrightarrow{\tau} h_\tau$$

C'est pourquoi nous les appelerons des horloges (déterministes, l'action τ étant le "top"); on a déjà vu comment les définir :

$$h_\tau =_{def} (x \text{ where } x <= \tau:x)$$

Par exemple $\mathbb{1}$ est une horloge, et l'on a évidemment :

$$h_\tau = \tau * \mathbb{1}$$

Le sémaphore du premier exemple (seconde version) de cette section est un cas particulier d'horloge non-déterministe. La forme générale de la spécification est ici, pour $B = \{b_1, \ldots, b_n\} \subseteq \mathbb{M}$:

$$\forall\, b \in B : h_B \xrightarrow{b} h_B$$

(on verra plus loin comment, dans certains cas particuliers, réaliser des horloges non-déterministes à branchement infini). On peut les définir par

$$h_B =_{def} (x \text{ where } x <= (b_1:x + \ldots + b_n:x))$$

mais la forme la plus satisfaisante de définition est peut-être :

$$h_B =_{def} (h_{b_1} \; I \ldots I \; h_{b_n})$$

Revenant à SCCS, on peut retrouver les opérateurs de désynchronisation, par exemple le <u>délai initial</u> $\delta(p)$ [7,8] qui autorise à attendre avant d'activer p :

$$\delta(p) \xrightarrow{1} \delta(p) \quad \& \quad \frac{p \xrightarrow{a} p'}{\delta(p) \xrightarrow{a} p'}$$

Comme le note R. Milner [8], cette opération peut-être définie par :

$$\delta(x) =_{def} (y \text{ where } y <= 1:y + x)$$

Ou encore, pour poursuivre dans le style de ce papier (avec un synchronisateur qui peut attendre et à tout moment décider de passer le contrôle au processus mis en délai):

$$\delta(x) =_{def} (\alpha => x \;//\; (y \text{ where } y <= 1:y + \bar{\alpha}:0)) \setminus \underline{\alpha}$$

et l'on retrouve les propriétés :

alors, par définition de δ :

$$\exists\, r' : \delta\delta(p) \xrightarrow{a} r' \;\&\; r' = r \qquad \blacksquare$$

L'autre opération de désynchronisation utilisée dans la première version du calcul synchrone de R. Milner [7] est Δ, qui introduit des attentes entre les actions d'une séquence. De façon imagée, si l'on a :

$$p \xrightarrow{a_1} \cdot \xrightarrow{a_2} \cdots$$

alors on aura :

$$\Delta p \xrightarrow{a_1} \underset{1}{\circ} \xrightarrow{a_2} \underset{1}{\circ} \cdots$$

La spécification de cet opérateur est (cf [7])

$$\frac{p \xrightarrow{a} p'}{\Delta p \xrightarrow{a} \delta \Delta p'}$$

Pour formuler la définition de cet opérateur dans notre langage, l'idée est de réaliser, après les premières action de p, l'entrelacement des reconfigurations de p avec un processus qui ne fait qu'attendre, et qui n'est autre que $\mathbb{1}$. Pour imposer de commencer par p, on lie les premières actions de p au premier top de l'horloge qui constitue le synchronisateur de l'entrelacement, par un déclenchement mutuel :

$$\Delta(x) =_{def} (\alpha \Rightarrow (\tau \cdot x) \;//\; \tau \cdot \mathbb{1} \;//\; \overline{\alpha} \Rightarrow h_{\overline{\tau}}) \setminus \underline{\alpha}, \underline{\tau}$$

A titre d'exemple montrons l'adéquation de cette définition ; on a :

$$\Delta(p) \xrightarrow{a} r \Leftrightarrow a \in \mathbb{M} \setminus \underline{\alpha}, \underline{\tau} \quad r = (r' \setminus \underline{\alpha}, \underline{\tau}) \quad \text{avec}$$

propriétés 12 :

$$\delta(0) = \mathbb{1} \ , \ \delta(p) = p \leftarrow 1 : \delta(p) = p \leftarrow \delta(p)$$
$$\delta(p \setminus \underline{\alpha}) = \delta(p) \setminus \underline{\alpha}$$
$$\delta(\langle\varphi\rangle p) = \langle\varphi\rangle\, \delta(p)$$
$$\delta\delta(p) = \delta(p)$$

A titre d'exemple, donnons la preuve de cette dernière propriété : on applique le principe de preuve à la fermeture symétrique de la relation

$$P = \{(\delta\delta(p) \ , \ \delta(p)) \ / \ p \in \mathcal{A}\}$$

en utilisant la spécification de l'opérateur de délai initial (à la congruence près). On a donc

$$\delta\delta(p) \xrightarrow{a} r \iff (i)\ a=1 \ \&\ r = \delta\delta(p) \text{ ou bien}$$

$$(ii)\ \delta(p) \xrightarrow{a} r' \ \&\ r = r'$$

On a aussi

$$\exists\, r'' : \delta(p) \xrightarrow{1} r'' \ \&\ r'' = \delta(p)$$

donc dans le premier cas :

$$\delta\delta(p) \xrightarrow{1} r \Rightarrow \exists\, r'' : \delta(p) \xrightarrow{1} r'' \ \&\ (r'', r) \in = \circ p^{-1} \circ =$$

Dans le second cas, on a directement (par définition de la spécification) :

$$\delta\delta(p) \xrightarrow{a} r \Rightarrow \exists\, r' : \delta(p) \xrightarrow{a} r' \ \&\ r' = r$$

Maintenant si

$$\delta(p) \xrightarrow{a} r$$

$$(\alpha \Rightarrow (\tau^{*}p) \;//\; \tau^{*}\mathbb{1} \;//\; \bar{\alpha} \Rightarrow h_{\bar{\tau}}) \overset{a}{\dashrightarrow} r'$$

Compte tenu des spécifications de $\mathbb{1}$ et de $h_{\bar{\tau}}$, on voit en examinant les sept cas possibles qu'en fait on a nécéssairement

$$p \overset{a}{\dashrightarrow} p' \quad \text{et} \quad r' = (\tau^{*}p' \;//\; \tau^{*}\mathbb{1} \;//\; h_{\bar{\tau}})$$

c'est-à-dire que, comme attendu : $r = (p' \; I \; \mathbb{1})$ (la restriction sur $\underline{\alpha}$ ne servant plus à rien).

Utilisant la spécification de δ, I et $\mathbb{1}$ et le fait que, par définition de Δ, comme on vient de le voir :

$$\Delta(p) \overset{a}{\dashrightarrow} r \Leftrightarrow p \overset{a}{\dashrightarrow} p' \;\&\; r = (p' \; I \; \mathbb{1})$$

il faut donc montrer qúe

$$\forall\, q : \delta\Delta(q) = (q \; I \; \mathbb{1})$$

On vérifie alors que la fermeture symétrique de la relation

$$P = \{(\delta\Delta(q) \;,\; (q \; I \; \mathbb{1})) \;/\; q \in \mathcal{A}\}$$

satisfait le principe de preuve :

- on a $\delta\Delta(q) \overset{a}{\dashrightarrow} r$ <u>ssi</u>

(i) $a = 1$ et $r = \delta\Delta(q)$, au quel cas, puisque $\mathbb{1} \overset{1}{\dashrightarrow} \mathbb{1}$:

$(q \; I \; \mathbb{1}) \overset{1}{\dashrightarrow} r' \;\&\; r' = (q \; I \; \mathbb{1})$ donc

$\exists\, r' : (q \; I \; \mathbb{1}) \overset{1}{\dashrightarrow} r' \;\&\; (r',r) \in = \circ P^{-1} \circ =$

ou bien :

(ii) $\Delta(q) \xrightarrow{a} r'$ et $r' = r$, donc

$\exists\, q' : q \xrightarrow{a} q'$ & $r' = (q' \mid \mathbb{1})$

Alors par définition de I :

$\exists\, r'' : (q \mid \mathbb{1}) \xrightarrow{a} r''$ & $r'' = r' = r$

- on procède de même pour $(q \mid \mathbb{1}) \xrightarrow{a} r$. Les détails sont laissés au soin du lecteur ∎

Nous ne répèterons pas ici les propriétés de cet opérateur, qui sont données dans [7].

Remarque : il semblerait que la désynchronisation "totale" soit plutôt l'opérateur $\nabla = \delta\Delta$, soit encore $\nabla(x) = (x \mid \mathbb{1})$. Comme le note R. Milner [7], la notion d'asynchronisme qui pourrait être liée à ∇ n'est pas respectée par la somme. Ceci s'explique ici par le fait que cette notion n'est évidemment pas compatible avec les opérations de synchronisation, comme le déclenchement. Il est intéressant de noter cependant que c'est par cet opérateur de désynchronisation que l'on peut définir en SCCS notre composition parallèle (compte tenu qu'on peut y définir nos primitives de synchronisation, propriété 11.(3)). En effet on a :

$$(x//y) = (\nabla(\alpha^{\#}x) \times \nabla(\beta^{\#}y) \times (z \text{ where } z \Leftarrow \overline{\alpha}:z + \overline{\beta}:z + \overline{\alpha\beta}:z)) \setminus \underline{\alpha},\underline{\beta}$$

(cette définition est due à R. de Simone).

Cette propriété explicite la signification de l'équivalence entre SCCS de [7] et MEIJE, c'est-à-dire la dualité (interdéfinissabilité) de deux calculs fondés :

- l'un sur la composition parallèle synchrone x, avec des opérateurs de désynchronisation (ou d'attente) δ, Δ

- l'autre sur la composition parallèle asynchrone //, avec des opérateurs de synchronisation s=> et $s^{\#}$.

L'un des mécanismes de synchronisation les plus utilisés (voir

[2],[4], [10]) est le rendez-vous sur une action, dénommée pour la circonstance événement, ou opérateur de diffusion de cet événement (pour R. Milner [8] : conjonction sur cet événement). Il s'agit là, étant donné $e \in \mathbb{A} \cup \$$, de faire fonctionner deux processus en parallèle de façon que lorsque l'un exécute une action contenant e, l'autre doit faire de même simultanément : et le système synchronisé effectue alors une action qui contient un seul événement e : il reste donc disponible pour une synchronisation du même type avec l'environnement. Formellement, l'opérateur de diffusion $\&_e$ est spécifié par :

si $a,b \in B^{\circledast}$ avec :

$$B = \begin{cases} (\mathbb{A} \cup \$) - \{e\} & \text{si } e \in \mathbb{A} \\ (\mathbb{A} \cup \$) - \{e,\bar{e}\} & \text{si } e \in \$ \end{cases}$$

alors :

(1)
$$\frac{p \xrightarrow{a} p'}{(p \,\&_e\, q) \xrightarrow{a} (p' \,\&_e\, q)}$$

$$\frac{p \xrightarrow{a} p' \;,\; q \xrightarrow{b} q'}{(p \,\&_e\, q) \xrightarrow{ab} (p' \,\&_e\, q')}$$

$$\frac{q \xrightarrow{b} q'}{(p \,\&_e\, q) \xrightarrow{b} (p \,\&_e\, q')}$$

(2)
$$\frac{p \xrightarrow{ae} p' \;,\; q \xrightarrow{be} q' \;,\; a,b \in B^{\circledast}}{(p \,\&_e\, q) \xrightarrow{abe} (p' \,\&_e\, q')}$$

L'expression de cet opérateur se fait comme en SCCS :

$$(x \,\&_e\, y) =_{def} (\langle \alpha/e \rangle x \;//\; \langle \beta/e \rangle y \;//\; h_{\alpha\beta e}) \setminus \underline{\alpha},\underline{\beta}$$

(où $e \notin \{\alpha, \bar{\alpha}, \beta, \bar{\beta}\}$).

On peut généraliser cet opérateur à la diffusion d'un événement e non atomique ($e \in \mathbb{M} - \{1\}$) ; dans cette famille de synchronisation, la plus interessante est sans doute celle qui rends compte des rendez-vous sur paquets d'événements : c'est, étant donné un ensemble fini $E = \{e_1, \ldots, e_n\}$ d'actions (que l'on supposera atomiques - mais ceci n'est pas vraiment une restriction -, ie $E \subseteq \mathbb{A} \cup \$$, et telles que E ne contienne pas deux actions inverses l'une de l'autre, ie $1 \notin E^@$), l'opérateur $\&_E^*$ qui a la même spécification, mais où l'on considère que e est un élément quelconque de $E^@$ (un paquet - non vide - d'événements de E). Il est aisément défini dans le langage :

$$(x \ \&_E^* \ y) =_{def} (\langle\varphi\rangle x \ // \ \langle\psi\rangle y) \setminus \underline{\alpha}_i$$

où $\varphi(e_i) = \alpha_i$, $\psi(e_i) = \bar{\alpha}_i \ e_i$ $\quad (E = \{e_1, \ldots, e_n\})$

avec $E \cap \{\alpha_i, \bar{\alpha}_i \ / \ 1 \leq i \leq n\} = \emptyset$.

Cet opérateur a des propriétés remarquables ; en particulier, il admet un élément neutre : c'est l'agent qui accepte tous les rendez-vous sur $e \in E^@$. Sa spécification est donc du type horloge (non déterministe, avec branchement infini) :

$$\forall \ e \in E^@ : H_E^* \xrightarrow{e} H_E^*$$

Pour définir cet agent dans le langage, il faut utiliser les définitions non gardées (qui sont une manière de réaliser des sommes infinies, voir [8]). D'une façon générale :

pour $B = \{a_1, \ldots, a_k\} \subseteq \mathbb{M}$ soit
$H_B^* =_{def} (x \ \underline{\text{where}} \ x <= (a_1{:}0 \ // \ldots // \ a_k{:}0 \ // \ x))$

Montrons que cet agent satisfait la spécification :

$$\forall \ b \in B^@ : H_B^* \xrightarrow{b} H_B^*$$

Soit T l'ensemble d'agents défini inductivement comme le plus petit tel que

(1) $H_B^* \in T$

(2) si $p \in T$ alors, pour $\{i_1,\ldots,i_n\} \subseteq \{1,\ldots,k\}$:

$$(a_{i_1} : 0 \;//\ldots//\; a_{i_n} : 0 \;//\; p) \in T$$

(3) si $p \in T$ et $q \sim p$ alors $q \in T$

Le premier point à montrer est :

$$(i)\quad p \in T \;\&\; p \xrightarrow{a} r \;\Rightarrow\; a \in B^{@} \;\&\; r \in T$$

par induction sur la définition de T :

- dans le cas où $p = H_B^*$, on procède par induction sur l'inférence de $H_B^* \xrightarrow{a} r$. Par définition, on a :

$$H_B^* \xrightarrow{a} r \Leftrightarrow (a_1:0 \;//\ldots//\; a_k:0 \;//\; H_B^*) \xrightarrow{a} r$$

alors :

-- soit $\exists\, \{i_1,\ldots,i_n\} \subseteq \{1,\ldots,k\} (n \neq 0)$ tel que $a = a_{i_1} \ldots a_{i_n}$ et, si $\{j_1,\ldots,j_l\} = \{1,\ldots,k\} - \{i_1,\ldots,i_n\}$:

$$r \sim (a_{j_1}:0 \;//\ldots//\; a_{j_l}:0 \;//\; H_B^*)$$

donc $r \in T$ et la propriété est vraie dans ce cas.

-- soit, avec les mêmes notations :

$$\exists\, a' \exists\, r' : H_B^* \xrightarrow{a'} r' \quad \text{et} \quad a = a_{i_1} \ldots a_{i_k}\, a'$$

et $r \sim (a_{j_1} : 0 \,//\ldots//\, a_{j_l} : 0 \,//\, r')$

et on applique là l'hypothèse de récurrence.

- si $\exists\, q \in T : p = (a_{i_1} : 0 \,//\ldots//\, a_{i_n} : 0 \,//\, q)$

ou si $\exists\, q \in T : p \sim q$

on applique de façon évidente l'hypothèse d'induction sur q.

Ensuite on vérifie que

(ii) $\forall\, p \in T \;\; \forall\, i \;\; \exists\, r \in T : p \overset{a_i}{\dashrightarrow} r$ et

$\forall\, p \in T \;\; \forall\, i \;\; \forall\, a \in B^{@}$ <u>si</u> $\exists\, r \in T : p \overset{a}{\dashrightarrow} r$ <u>alors</u> $\exists\, q \in T : p \overset{aa_i}{\dashrightarrow} q$

(par induction sur la définition de T)

d'où l'on tire la propriété :

(iii) $\forall\, p \in T \;\; \forall\, b \in B^{@} \;\; \exists\, r \in T : p \overset{b}{\dashrightarrow} r$

Grace à (i) et (iii) on montre enfin :

(iv) $(p,q) \in T \Rightarrow p \sim q$

en appliquant le principe de preuve à $P = T \times T$

d'où finalement :

$$H_B^* \overset{b}{\dashrightarrow} o \sim r \iff b \in B^{@} \;\&\; r \sim H_B^*$$

ce qu'on voulait montrer ∎

propriétés 13 :

(1) $(\mathcal{L}/=, \&_E^*, H_E^*)$ est un monoide abélien :

$((p \&_E^* q) \&_E^* r) = (p \&_E^* (q \&_E^* r))$

$(H_E^* \&_E^* p) = (p \&_E^* H_E^*) = p$

$(p \&_E^* q) = (q \&_E^* p)$

(2) de plus, pour $p,q \in \mathcal{A}$, avec

$E' = (E \cap \mathbb{A}) \cup \{\underline{\alpha} / \{\alpha,\bar{\alpha}\} \cap E \neq \emptyset\}$

- si $sorte(p) \subseteq E'$ alors $(p \&_E^* p) \sim p$

- si $(sorte(p) \cup sorte(q)) \cap E' = \emptyset$:
  $(p \&_E^* q) \sim (p//q)$

Un autre mécanisme d'ordonnancement (inverse en quelque sorte du rendez-vous) se rencontre fréquement (accès à une ressource partagée par exemple) : c'est celui qui rend compte de l'exclusion mutuelle d'actions. Supposons que $B = \{a_1,\ldots, a_n\} \subseteq \mathbb{A}$ soit un ensemble d'actions de calcul atomiques deux à deux incompatibles, c'est-à-dire qui ne doivent pas avoir lieu simultanément (chaque action est incompatible avec elle-même - mais c'est une hypothèse dont on pourrait se passer : on considère généralement qu'une relation de compatibilité est réflexive et symétrique) ; alors on peut spécifier l'opérateur $\#_B$ de mise en parallèle avec exclusion mutuelle sur B par :

C dénotant l'ensemble des actions qui contiennent au plus une occurence d'une action de B, ie $C = (\{1\} \cup B)\,\mathbb{M}\backslash B$:

$$\frac{p \xrightarrow{a} p' \ , \ a \in C}{(p \#_B q) \xrightarrow{a} (p' \#_B q)} \qquad \frac{q \xrightarrow{b} q' \ , \ b \in C}{(p \#_B q) \xrightarrow{b} (p \#_B q')}$$

$$\frac{p \xrightarrow{a} p' \ , \ q \xrightarrow{b} q' \ , \ a,b \in C \text{ et } a \in \mathbb{M}\backslash B \text{ ou } b \in \mathbb{M}\backslash B}{(p \#_B q) \xrightarrow{ab} (p' \#_B q')}$$

Cette spécification est satisfaite par :

$$(x \#_B y) =_{def} (\langle\varphi\rangle x \;//\; \langle\varphi\rangle y \;//\; h_{\underline{\alpha}}) \setminus \underline{\alpha}$$

où $\varphi(a_i) = \alpha a_i \quad$ pour $1 \leq i \leq n$

(si on voulait seulement interdire les actions de (p//q) qui ont un facteur $a_i a_j$ pour $i \neq j$, on prendrait

$$(\langle\psi\rangle x \;//\; \langle\psi\rangle y \;//\; \text{syn}) \setminus \underline{\alpha}_1, \ldots, \underline{\alpha}_n$$

où $\psi(a_i) = \alpha_i a_i$ et $\text{syn} = (H^*_{\bar{\alpha}_1} \; I \ldots I \; H^*_{\bar{\alpha}_n})$

propriétés 14 :

$$(p \#_B (q \#_B r)) = ((p \#_B q) \#_B r)$$

$$(p \#_B q) = (q \#_B p)$$

$$(p + q) \#_B r = (p \#_B r) + (q \#_B r)$$

Nous nous servirons dans la suite d'une opération de composition séquentielle, qui fait appel à une notion de terminaison ; aussi nous réserverons pour toute la suite un signal distingué $\underline{\sigma}$ qui sera le signal de terminaison. On peut déjà définir l'arrêt sur l'émission de ce signal d'un processus p, opération notée $(p \setminus \underline{\sigma})$: ce terme se comporte comme p tant qu'il ne signale pas qu'il a terminé, et bloque p lorsqu'il émet $\underline{\sigma}$ (on considère comme anormale l'émission simultanée de plusieurs signaux de terminaison). La spécification est donc :

$$\frac{p \xrightarrow{a} p' \;,\; a \in \mathbb{M} \setminus \sigma}{(p \setminus \underline{\sigma}) \xrightarrow{a} (p' \setminus \underline{\sigma})} \qquad \frac{p \xrightarrow{a\bar{\sigma}} p' \;,\; a \in \mathbb{M} \setminus \sigma}{(p \setminus \underline{\sigma}) \xrightarrow{a\bar{\sigma}} 0}$$

Il s'agit donc de bloquer l'agent p dès qu'il a émis $\underline{\sigma}$: pour cela on va piloter p sur un signal et le contrôler par un synchronisateur qui donne

l'autorisation correspondante tant qu'il n'enregistre pas la terminaison de p. Cependant il faut que $(p \setminus \underline{\sigma})$ signale lui aussi sa terminaison à l'environnement : on est donc conduit à renommer cette action dans p ; d'où la définition :

$$(x \setminus \underline{\sigma}) =_{def} (\alpha * (\langle\sigma'/\sigma\rangle x) \;//\; (y \text{ where } y <= \bar{\alpha}:y + \bar{\alpha}\sigma'\bar{\sigma}:\emptyset)) \setminus \underline{\alpha,\sigma'}$$

(rappelons que le renommage $\langle\sigma'/\sigma\rangle$ renomme $\bar{\sigma}$ en $\bar{\sigma'}$).

propriétés 15 :

$$((p \setminus \underline{\sigma}) \setminus \underline{\sigma}) = (p \setminus \underline{\sigma})$$

$$(a:p) \setminus \underline{\sigma} = \begin{cases} a : (p \setminus \underline{\sigma}) & \text{si } a \in \mathbb{M} \setminus \underline{\sigma} \\ a : \emptyset & \text{si } \exists\, b \in \mathbb{M} \setminus \underline{\sigma} : a = b\bar{\sigma} \\ \emptyset & \text{sinon} \end{cases}$$

$$\begin{aligned} (p+q) \setminus \underline{\sigma} &= (p \setminus \underline{\sigma}) + (q \setminus \underline{\sigma}) \\ \delta(p \setminus \underline{\sigma}) &= \delta(p) \setminus \underline{\sigma} \\ (p \;\&_{\bar{\sigma}}\; q) \setminus \underline{\sigma} &= (p \setminus \underline{\sigma}) \;\&_{\bar{\sigma}}\; (q \setminus \underline{\sigma}) \end{aligned}$$

Remarquer qu'en général on n'a pas distributivité de cette opération sur les compositions parallèles //, I et x.

Une classe d'expressions qui paraît particulièrement intéressante est celle des $p \in \mathcal{L}$ qui satisfont $p = (p \setminus \underline{\sigma})$. On peut par exemple formuler là des propriétés de blocage.

On a dit que l'opération $(p \setminus \underline{\sigma})$ ("p jusqu'à $\bar{\sigma}$") nous servait à définir la composition séquentielle p;q. La signification de cette synchronisation est évidente ; précisons seulement que la transmission du contrôle est instantannée, c'est-à-dire que la terminaison de p est simultanée avec l'activation de q. De plus l'expression p;q doit signaler sa terminaison par $\bar{\sigma}$. La spécification est :

$$\frac{p \xrightarrow{a} p' \quad , \quad a \in \mathbb{M}\setminus\sigma}{(p;q) \xrightarrow{a} (p';q)} \qquad \frac{p \xrightarrow{a\bar{\sigma}} p' \quad , \quad q \xrightarrow{b} q' \quad , \quad a \in \mathbb{M}\setminus\sigma}{(p;q) \xrightarrow{ab} q'}$$

Pour réaliser cela, il suffit de déclencher q sur la terminaison de p (que nous devons renommer):

$$(x;y) =_{def} (\langle\sigma'/\sigma\rangle(x \setminus \underline{\sigma}) \;//\; \sigma' \Rightarrow y) \setminus \underline{\sigma'}$$

(où $\underline{\sigma'} \neq \underline{\sigma}$).

propriétés 16 :

$(p;q);r = p;(q;r)$ (associativité)

$(p;q) = (p \setminus \underline{\sigma});q$

$p;(q \setminus \underline{\sigma}) = (p;q) \setminus \underline{\sigma}$ (la terminaison de p;q est celle de q)

$(\bar{\sigma}:\mathbf{0});p = p$

$p;(\bar{\sigma}:\mathbf{0}) = (p \setminus \underline{\sigma})$

$a:p = (a:\bar{\sigma}:\mathbf{0});p$ si $a \in \mathbb{M} \setminus \underline{\sigma}$

$s \Rightarrow (p;q) = (s \Rightarrow p);q$ si $s \notin \{\sigma,\bar{\sigma}\}$

On voit donc que la composition séquentielle respecte la propriété $p = (p \setminus \underline{\sigma})$, et que dans la classe de tels termes, $\bar{\sigma} : \mathbf{0}$ est élément neutre de la composition séquentielle.

Pour terminer cette section, nous donnerons un dernier exemple de synchronisation, dont la spécification est un peu plus élaborée que celles déjà traitées. Il s'agit d'un mécanisme fort utilisé dans le contrôle des processus, celui du chien de garde (ou "time-out". Nous nous inspirons là de la construction donnée par R. Milner [8] dans un cas particulier). Intuitivement, étant donné un délai qui est un entier k, c'est un opérateur $\theta_k(p,q,r)$ qui alloue à p un "temps de travail" k. Ici le temps est celui d'une horloge $h_\tau(\tau \in \mathbb{A} \cup \$)$, et θ_k décompte les occurences du top τ dans la suite des actions de p (on considère comme anormal un agent qui produit simultanément plusieurs tops) de telle façon que :

- si p termine (ie émet $\underline{\sigma}$) dans les détails impartis, on passe à la suite normale, c'est-à-dire que le contrôle est transmis instantanément à q

- sinon l'alarme r est déclenchée au $k^{\text{ème}}$ top enregistré et p est instantanément bloqué.

Formellement, $k \in \mathbb{N}$ étant donné, on peut spécifier mutuellement les θ_j pour $0 \leq j \leq k$ par :

$$\frac{r \xrightarrow{b} r'}{\theta_0(p,q,r) \xrightarrow{b} r'}$$

et, pour $j>0$ (si $k>0$), en notant

$$B = \{\underline{\sigma}\} \cup (\{\tau\} \cap \mathbb{A}) \cup \{\underline{\alpha} \;/\; \{\tau\} \cap \{\alpha, \overline{\alpha}\} \neq \emptyset\} :$$

$$\frac{p \xrightarrow{a} p' \;,\; a \in \mathbb{M} \setminus B}{\theta_j(p,q,r) \xrightarrow{a} \theta_j(p',q,r)} \qquad \frac{p \xrightarrow{a\tau} p' \;,\; a \in \mathbb{M} \setminus B}{\theta_j(p,q,r) \xrightarrow{a\tau} \theta_{j-1}(p',q,r)}$$

$$\frac{p \xrightarrow{a\overline{\sigma}} p' \;,\; q \xrightarrow{b} q' \;,\; a \in \mathbb{M} \setminus \sigma}{\theta_j(p,q,r) \xrightarrow{ab} q'}$$

Pour réaliser cette spécification, on va piloter p sur la réception d'un signal $\underline{\Upsilon}$; le contrôle est assuré par un synchronisateur qui émet $\underline{\Upsilon}$ tout en décomptant éventuellement les τ (par rendez-vous $\&_\tau$) et qui, lorsque le délai imparti est écoulé, déclenche, sur un signal spécial $\underline{\alpha}$, l'alarme r :

soient $t_0 = \overline{\sigma}\,\overline{\alpha} : 0$

et pour $0 < j \leq k$:

$$t_j = (v \text{ where } v <= (\bar{\gamma}:v + \bar{\gamma}\tau:t_{j-1}))$$

alors :

$$\Theta_k(x,y,z) =_{def} ((\gamma^{\omega}x \;\&_{\tau}\, t_k) \setminus \underline{\gamma} \;;\; (q + \alpha => r)) \setminus \underline{\alpha}$$

(où $\tau, \bar{\sigma} \notin \{\alpha, \bar{\alpha}, \gamma, \bar{\gamma}\}$)

(le lecteur pourra développer dans cette formule les opérateurs $\&_{\tau}$, ; et + en fonction des primitives).

La preuve de correction de cette définition (par récurrence sur j, $0 \leq j \leq k$, et par cas, en raisonnant sur la spécification des opérateurs utilisés) est facile mais fastidieuse ; nous l'omettrons ici.

Remarque : on pourrait considérer un autre contrôle $\Theta'_k(p,r)$ dont la terminaison est celle de p si celle-ci se produit dans le temps imparti, ou sinon est celle de l'alarme r. Mais en fait on a :

$$\Theta'_k(p,r) = \Theta_k(p, \bar{\sigma}:0, r)$$

4. LANGAGES DE COMPORTEMENTS

Dans cette section nous établissons un lien entre notre calcul et une autre approche de la notion de processus couramment adoptée, où ceux-ci sont représentés par l'ensemble des suites finies ou infinies d'actions qu'ils peuvent exécuter. Cette approche est naturellement très liée avec celle des réseaux d'automates, systèmes de transitions, et expressions algébriques de description des langages, telles les expressions de chemin, de flot, d'événements (voir par exemple [2,4,5,10]). Il est très naturel ici d'associer à une expression $p \in \mathcal{L}$ un langage $L^{\infty}(p)$ de séquences finies ou infinies d'actions (l'ensemble de ces suites étant $\mathbb{M}^{\infty} = \mathbb{M}^{*} \cup \mathbb{M}^{\omega}$), par :

$$L^{\infty}(p) = L(p) \cup L^{\omega}(p) \qquad \text{où}$$

$$L(p) = \{w/w \in \mathbb{M}^* \ \& \ \exists\, p' \in \mathcal{L} \ : \ p \xrightarrow{w} p'\} \quad \text{et}$$

$$(a_i)_{i\in\mathbb{N}} \in L^{\omega}(p) \iff \exists\, (p_i)_{i\in\mathbb{N}} \in \mathcal{L}^{\omega} \text{ telle que}$$

$$p_0 = p \ \& \ \forall i \in \mathbb{N} : p_i \xrightarrow{a_i} p_{i+1}$$

Si de plus on considère les comportements (finis) "terminés" de p comme formant l'ensemble

$$\Lambda(p) = \{w/w{:}\bar{\sigma} \in L(p) \ \& \ w \in (\mathbb{M}\backslash\underline{\sigma})^*\}$$

(ce qui n'a en fait d'intérêt que pour les expressions qui satisfont $L(p) \subseteq ((\mathbb{M}\backslash\underline{\sigma}) \cup \{\bar{\sigma}\})^*$, et telles que $p = p \setminus \underline{\sigma}$) alors le triplet $\langle L(p) , \Lambda(p) , L^{\omega}(p)\rangle$ forme un processus au sens d'Arnold & Nivat [2].

De même on peut associer à un agent p un automate $\mathcal{A}(p)$ dont l'ensemble (en général infini) des états est (en notant $\ulcorner q \urcorner$ la classe dans $\mathcal{P}$ de l'agent q) :

$$Q(p) = \{\ulcorner q \urcorner \ / \ \exists\, w \in \mathbb{M}^* \ : \ p \xrightarrow{w} q\}$$

dont la fonction de transition λ (de $Q(p) \times \mathbb{M}$ dans l'ensemble des parties de Q(p)) est donnée par

$$\ulcorner r \urcorner \in \lambda(\ulcorner q \urcorner, a) \iff \exists\, r' \ : \ q \xrightarrow{a} r' \ \& \ r' \sim r$$

dont l'état initial est $\ulcorner p \urcorner$, de telle sorte que si $p = p \setminus \underline{\sigma}$ et $L(p) \subseteq ((\mathbb{M}\backslash\underline{\sigma}) \cup \{\bar{\sigma}\})^*$ alors $\Lambda(p)$ est reconnu par cet automate avec pour ensemble d'états finaux :

$$Q'(p) = \{\ulcorner q \urcorner \ / \ \ulcorner q \urcorner \in Q(p) \ \& \ \exists\, q' \ : \ q \xrightarrow{\bar{\sigma}} q'\}$$

Remarque: il est facile de voir que $p \sim q \Leftrightarrow \mathcal{Q}(p) = \mathcal{Q}(q)$.

L'objet de ce paragraphe est d'évaluer la puissance de notre calcul à travers l'application Λ. Pour ce faire nous introduisons un "sous-calcul" (au sens de [8]) de MEIJE, qui est celui des (schémas de) processus séquentiels non déterministes:

on considère que $\mathbb{A}$ est un alphabet fini d'actions. $\mathcal{N}$ est le plus petit sous-ensemble de $\mathcal{S}$ qui contient (la classe de) $\bar{\sigma}:0$ et les identificateurs, satisfaisant :

(i) si $p \in \mathcal{N}$ et $a \in \mathbb{A}$ alors $a:p \in \mathcal{N}$

(ii) si $p \in \mathcal{N}$ et $q \in \mathcal{N}$ alors $(p+q) \in \mathcal{N}$ et $(p;q) \in \mathcal{N}$

(iii) si $\{x_1, \ldots, x_n\} \subseteq X$ et $\{p_1, \ldots, p_n\} \subseteq \mathcal{N}$ alors
pour $1 \leq i \leq n$: $(x_i \text{ where } x_1 <= p_1, \ldots, x_n <= p_n) \in \mathcal{N}$

lemme 1:

si $p \in \mathcal{N}$ et $p \xrightarrow{a} p'$ alors $p' \in \mathcal{N}$ et $a \in \mathbb{A} \cup \{\bar{\sigma}\}$,

et si de plus $a = \bar{\sigma}$ alors $p' = 0$.

preuve: par induction sur la définition de $\mathcal{N}$ et, dans le cas d'une définition récursive, par induction sur la longueur de l'inférence de $p \xrightarrow{a} p'$ ∎

On a donc :

$$p \in \mathcal{N} \Rightarrow p = p \setminus \sigma \text{ et } L(p) \subseteq (\mathbb{A} \cup \{\bar{\sigma}\})^*, \text{ donc aussi } \Lambda(p) \subseteq \mathbb{A}^*$$

Les opérations constitutives de $\mathcal{N}$ sont naturellement reliées aux opérations sur les langages sur $\mathbb{A}$:

lemme 2: pour p,q dans $\mathcal{N}$:

$\Lambda(\bar{\sigma}:0) = \{\varepsilon\}$

$\Lambda(a:p) = \{a\} \Lambda (p)$ pour $a \in \mathbb{A}$

$\Lambda(p+q) = \Lambda(p) \cup \Lambda(q)$

$\Lambda(p;q) = \Lambda(p) \Lambda (q)$ (produit de concaténation des langages)

$\Lambda(x_i \text{ where } R) = \Lambda([(x_j \text{ where } R) / x_j]p_i)$ où

$R = x_1 <= p_1 ;\ldots; x_n <= p_n$ (et $\forall j\ p_j \in \mathcal{N}$)

preuve: triviale (modulo la spécification opérationnelle des constructeurs de $\mathcal{N}$). Pour le dernier point, on utilise la propriété 3 puisque

$$p = q \Rightarrow \Lambda(p) = \Lambda(q)$$

lemme 3:

pour toute grammaire algébrique G sur $\mathbb{A}$ engendrant un langage L, on peut construire une expression p de $\mathcal{N}$ telle que $\Lambda(p) = L$.

preuve: soit $G = (\Xi, \xi, P)$ la grammaire ("context-free") considérée, que l'on peut supposer (si $L \neq \emptyset$, mais on a évidemment $\Lambda(x) = \emptyset$ pour $x \in X$) propre et réduite, où:

- $\Xi = \{\xi_1, \ldots, \xi_n\}$ est l'alphabet non-terminal
- $\xi \in \Xi$ est l'axiome ($L = L_G(\xi)$)
- P, partie finie de $\Xi \times (\Xi \cup \mathbb{A})^*$, est l'ensemble des productions.

On définit l'application $\eta : (\Xi \cup \mathbb{A})^* \to \mathcal{N}$ par :

$\eta(\varepsilon) = \bar{\sigma} : 0$

$\eta(a:w) = a:\eta(w)$ pour $a \in \mathbb{A}$

$$\eta(\xi_j : w) = \begin{cases} x_j \text{ (element de X)} & \text{si } w = \varepsilon \\ x_j : \eta(w) & \text{sinon} \end{cases}$$

Définissons alors, pour $1 \leq i \leq n$:

$p_i = (x_i \text{ where } x_1 <= q_1, \ldots, x_n <= q_n)$ où

$$q_j = \eta(w_1^j) + \ldots + \eta(w_{k_j}^j) \quad \text{si}$$

$$\{w_1^j, \ldots, w_{k_j}^j\} = \{w/(\xi_j, w) \in P\}$$

On verifie alors grâce au lemme 2 que le n-uple de langages $\langle \Lambda(p_1), \ldots, \Lambda(p_n) \rangle$ est une solution de P qui, la grammaire étant propre, admet une seule solution (théorème de Schutzenberger). D'où le lemme si l'on pose $p = p_i$ pour i tel que $\xi = \xi_i$ ∎

Ce résultat (ou plutôt cette construction) est intéressant en lui-même dans la mesure où il permet de rendre compte dans notre calcul de synchronisations qui s'expriment naturellement par un langage algébrique (ordonnancement). Supposons par exemple que p et q soient deux processus qui communiquent par une file d'attente non bornée, et que les actions de "mettre" et de "prendre" sur cette file soient resp. a et b (dans $\mathbb{A}$). Les processus p et q fonctionnant en parallèle, il faut synchroniser leur accès à la file, de façon à ce qu'une action b ne puisse avoir lieu que si la file est non vide. Pour ce faire on va renommer a et b de façon à lier à l'émission d'un signal $\underline{\rho}$ (production) et b à la réception d'un signal $\underline{\gamma}$ (consommation). Le synchronisateur chargé de distribuer les signaux correspondants doit être tel que

$$L(syn) = \{w/w \in \{\rho, \bar{\gamma}\}^* \text{ et pour tout facteur gauche } u \text{ de } w : |u|_\rho \geq |u|_{\bar{\gamma}}\}$$

Ce langage (algébrique) n'est rien d'autre que l'ensemble des facteurs gauches des mots du langage de Dyck restreint sur une lettre. Comme tout facteur gauche d'un mot de L(syn) est encore un mot de L(syn), il suffit en fait de trouver un agent r tel que $\Lambda(r)$ soit le langage de Dyck évoqué, engendré par la grammaire :

$$\xi = \varepsilon + \rho : \xi : \bar{\gamma} : \xi$$

D'où le terme

$$r = (x \text{ where } x <= \bar{\sigma}:0 + (\rho:x) ; (\bar{\gamma}:x))$$

Naturellement on veut que le synchronisateur ne s'arrête pas ; on posera donc (syn = r^{ω}, voir plus loin)

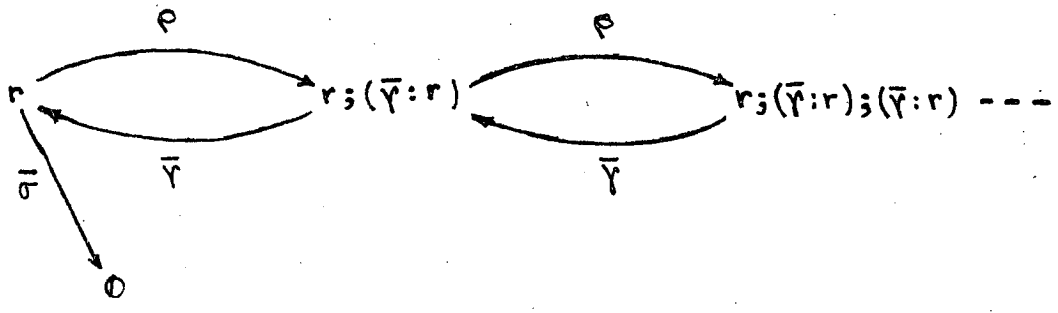
$$syn = (y \text{ where } y <= r;y)$$

Finalement le système synchronisé est décrit par :

$$(\langle \bar{\rho}a/a, \gamma b/b\rangle x \;//\; \langle \bar{\rho}a/a, \gamma b/b\rangle y \;//\; syn) \setminus \underline{\rho}, \underline{\gamma}$$

(En fait cette synchronisation vaut pour un nombre quelconque de processus en parallèle partageant une même file d'attente ; on pourrait aussi l'écrire en terme de $\&^{*}_{a,b}$, le synchronisateur étant alors une expression de chemin).

On peut figurer l'automate (à pile) de l'agent r :



En fait on peut obtenir par Λ beaucoup plus que les langages algébriques (si l'on considère des processus qui ne sont pas dans $\mathcal{N}$). Par exemple on peut construire $p \in \mathcal{A}$ tel que

$$\Lambda(p) = \{a^n b^n c^n \;/\; n \in \mathbb{N}\} \quad (\{a,b,c\} \subseteq \mathbb{A})$$

L'idée est de dupliquer l'agent qui permet d'obtenir $\{a^n b^n \;/\; n \in \mathbb{N}\}$, et qui est donné (grâce au lemme 3) par:

$$q = (x \text{ where } x <= \overline{\sigma}:\emptyset + (a:x) \ ; \ (b:\overline{\sigma}:\emptyset))$$

en liant b à l'émission d'un signal $\underline{\alpha}$ qui va forcer une copie de q (dans laquelle on renomme a par $\overline{\alpha}$ et b par c) à produire (pour terminer) autant de c que q produit de a et de b :

$$p = (\langle\overline{\alpha}/b \ / \ b\rangle q \ \&_{\overline{\sigma}} \ \langle\alpha/a \ , \ c/b\rangle q) \setminus \underline{\alpha}$$

(ici on peut avoir un blocage de p, par exemple sur a:a:b:c). D'ailleurs on peut voir facilement que :

<u>lemme 4</u>

si sorte(p) ∪ sorte(q) $\subseteq \mathbb{A} \cup \{\underline{\sigma}\}$ alors

$$\Lambda(p \ \&^{*}_{\mathbb{A}\cup\{\overline{\sigma}\}} q) = \Lambda(p) \cap \Lambda(q)$$

Et on peut réaliser dans le calcul de nombreuses opérations sur les langages (qui font éventuellement sortir de la famille des langages algébriques), qui apparaissent comme des opérations de synchronisation "globales" (par opposition à la méthode de spécification sémantique adoptée dans la 3ème section).

Par exemple :

(pour des agents p,q tels que $p = p \setminus \underline{\sigma}$ et $L(p) \subseteq (\mathbb{A} \cup \{\overline{\sigma}\})^{*}$)

$$\Lambda(p)^{*} = \Lambda(x \text{ where } x <= \overline{\sigma}:\emptyset + p;x)$$
$$\Lambda(p)^{+} = \Lambda(x \text{ where } x <= p + p;x)$$
$$\Lambda(p)^{\omega} = L^{\omega}(x \text{ where } x <= p;x)$$

et si, comme il est usuel (cf [5]), ⧢ dénote l'opération de mélange ("shuffle") :

$$\Lambda(p) \sqcup\!\sqcup \Lambda(q) = \Lambda(p \ I_{\overline{\sigma}} \ q)$$

où $I_{\overline{\sigma}}$ est un opérateur d'entrelacement avec rendez-vous sur le signal de terminaison :

$$(x\ I_{\overline{\sigma}}\ y) =_{def} (\langle\alpha/\sigma\rangle(\tau^{*}x)\ //\ \langle\beta/\sigma\rangle(\tau^{*}y)\ //\ h_{\{\overline{\tau},\alpha\beta\overline{\tau}\overline{\tau}\overline{\sigma}\}})\setminus \underline{\alpha},\underline{\beta},\underline{\tau}$$

et le mélange itéré † est donné par

$$\Lambda(p)^{\dagger} = \Lambda(x\ \underline{\text{where}}\ x <= (\overline{\sigma}:0 + (p\ I_{\overline{\sigma}}\ x)))$$

Par contre il est moins évident de rendre compte des opérations qui comportent la notion d'effacement (par exemple "inverse shuffle, cancellation", cf [5]). Quoique nous n'en ayons pas de preuve formelle, ceci semble inhérent au caractère synchrone de notre calcul (muni de la congruence forte) : les possibilités de discrimination temporelle sont trop fortes pour que l'on puisse effacer autre chose que l'image commutative d'un langage (donc quelque chose qui ne peut guère décrire de synchronisation). On trouve par exemple dans les expressions de flot ou d'événements (cf [10,5]) des mécanismes de ce type : mais là encore, la synchronisation prends du temps, que l'on décide d'ignorer. C'est pourquoi on retrouve ici un résultat analogue à ceux de [1,5] : modulo l'équivalence observationnelle $\approx$, tout langage récursivement énumérable est réalisable par une expression MEIJE :

<u>proposition</u>:

> Soit A un alphabet fini d'actions. Pour tout langage récursivement énumérable $L \subseteq A^{*}$ il existe $p \in \mathcal{A}$ tel que $L = \pi(\Lambda(p))$ où π est la projection de $(A \cup \{1\})^{*}$ sur A^{*} donnée par $\pi(1) = \varepsilon$.

<u>preuve</u>: il est bien connu que tout langage récursivement énumérable L est image par un homomorphisme φ de l'intersection de deux langages algébriques. On a déjà vu comment obtenir les langages algébriques et comment réaliser l'intersection. Il faut donc montrer, en supposant l'alphabet A assez grand pour que φ puisse être considéré comme un endomorphisme de A^{*}, que pour $q \in \mathcal{A}$ tel que

$L(q) \subseteq (\mathbb{A} \cup \{\bar{\sigma}\})^*$ il existe $p \in \mathcal{A}$ tel que

$L(p) \subseteq (\mathbb{A} \cup \{\bar{\sigma}, 1\})^*$ et $\pi(\Lambda(p)) = \varphi(\Lambda(q))$

Pour cela on pose :

$$p = (\langle\psi\rangle q \;//\; syn) \setminus \{\underline{\alpha}_a / a \in \mathbb{A}\}$$

où $\{\underline{\alpha}_a / a \in \mathbb{A}\} \subseteq S$

$\psi(a) = \alpha_a$ pour tout $a \in \mathbb{A}$

$syn = (x \text{ where } x <= \sum_{a \in \mathbb{A}} w_a : x)$ avec

$$w_a = \begin{cases} \bar{\alpha}_a a_1 : \ldots : a_n & \text{si } \varphi(a) = a_1 : \ldots : a_n \\ \bar{\alpha}_a & \text{si } \varphi(a) = \varepsilon \end{cases}$$

5. CONCLUSION

Le calcul MEIJE que nous avons présenté est équivalent au calcul SCCS de R. Milner, dans sa version finitaire [7] : dans le sens formel de [8], les constructeurs de l'un des deux calculs sont définissables par une expression de l'autre, la congruence étant la même (chacun est un sous-calcul de l'autre). Ce résultat est intéressant dans la mesure où il montre l'équivalence de deux points de vue : l'un consiste à penser les processus comme des unités "autonomes", donc asynchrones (dans un sens qui n'est pas le sens formel de [7,8]), et à se donner les moyens d'organiser la cohérence d'un système de processus. De l'autre on voit tous les processus réglés sur une horloge universelle, et l'on se donne des outils pour rompre cette organisation. Le choix entre les deux modèles n'est qu'une question de convenance.

Ceci nous amène à une seconde remarque : notre objectif était d'élaborer un calcul qui permette de traiter commodément des problèmes de synchronisation. Nous pensons que la troisième section de ce papier montre que nos primitives sont là assez bien adaptées. Cependant pour

[7] R. Milner : "On Relating Synchrony and Asynchrony", CSR-75-80, Computer Science Dept., Edinburgh Univ. (1981).

[8] R. Milner : "Calculi for Synchrony and Asynchrony", CSR-104-82, Computer Science Dept., Edinburgh Univ. (1982). A paraître dans TCS.

[9] G. Plotkin : " A Structural Approach to Operational Semantics", Daimi FN-19, Computer Science Dept., Aarhus Univ. (1981).

[10] A.C. Shaw : "Software Specification Languages Based on Regular Expressions", Rapp. ETH Zurich et Software Development Tools Workshop Conf., Pingree Park, Colorado (1979).

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique

être entièrement convaincants sur ce point, il nous faudrait définir formellement la notion de synchronisation (effective) et de spécification, et prouver que toute synchronisation est réalisable dans notre calcul. L'étude menée dans la quatrième section est un premier pas dans ce sens. Elle n'est cependant pas entièrement satisfaisante, même s'il était vrai que tout langage de comportements récursivement énumérable est "réalisable". Car un résultat de ce type ne concerne que l'ordonnancement des tâches, au sens strict. Il ne dit rien des simultanéités possibles (et souhaitables, si l'on veut vraiment traiter du parallélisme). Quoiqu'il en soit notre calcul paraît être un cadre bien adapté pour aborder cette problématique.

Remerciements :

Nous remercions R. de Simone pour ces critiques et suggestions qui nous ont permis d'améliorer une première version de ce papier.

REFERENCES :

[1] T. Araki & N. Tokura : "Flow Languages Equal Recursively Enumerable Languages", Acta Informatica 15 (1981), 209-217.

[2] A. Arnold & M. Nivat : "Comportements de Processus", Coll. AFCET : "Les Mathématiques de l'Informatique", Paris (1981).

[3] M.Hennessy & R. Milner : "On Observing Nondeterminism and Concurrency", ICALP 80, LNCS 5 (1980), 299-309.

[4] C.A.R. Hoare : "A Model for Communicating Sequential Processes", in "On the Construction of Programs" (R.M. Mc Keag, Ed.), C.U.P. (1980).

[5] M. Jantzen : "The Power of Synchronizing Operations on Strings", TCS 14 (1981) 127-154.

[6] R. Milner : "A Calculus of Communicating Systems", LNCS 92 (1980).