A non-interleaving semantics for CCS based on proved transitions

Gérard Boudol, Ilaria Castellani

INRIA

UNITÉ DE RECHERCHE
INRIA-SOPHIA ANTIPOLIS

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél. (1) 39 63 55 11

Rapports de Recherche

N° 919

A NON-INTERLEAVING SEMANTICS FOR CCS BASED ON PROVED TRANSITIONS

Programme 1

Gérard BOUDOL
Ilaria CASTELLANI

Octobre 1988

RR-0919

A non-interleaving semantics for CCS based on proved transitions

Une sémantique du parallélisme dans CCS basée sur les transitions prouvées

Gérard Boudol & Ilaria Castellani
INRIA Sophia-Antipolis
06560-VALBONNE FRANCE

Résumé.

La sémantique des langages comme CCS ou TCSP est généralement donnée par un système de transitions, où les transitions sont celles qu'on peut déduire en utilisant un système de règles structurelles. Nous proposons de raffiner cette sémantique opérationnelle, en étiquetant les transitions par leur preuve – dans le système de règles considéré –, plutôt que par une simple action. Nous montrons que l'on peut utiliser cette notion de preuve pour définir une relation d'indépendance et de résidu entre transitions étiquetées par leur preuve. Ceci nous permet de d'adapter la notion d'équivalence par permutations de Berry et Lévy. Nous montrons que chaque classe de séquences de transitions peut être représentée par une transition étiquetée par un ordre partiel d'occurrences d'actions.

Abstract.

When using labelled transition systems to model languages like CCS or TCSP, one specifies transitions by a set of structural rules. We consider labelling transitions with their proofs – in the given system of rules – instead of simple actions. Then the label of a transition identifies uniquely that transition, and one may use this information to define a concurrency relation on (proved) transitions, and a notion of residual of a (proved) transition by a concurrent one. We apply Berry and Lévy's notion of equivalence by permutations to sequences of proved transitions for CCS to obtain a partial order semantics for this language.

A non-interleaving semantics for CCS based on proved transitions

Gérard Boudol & Ilaria Castellani
INRIA Sophia-Antipolis
06560-VALBONNE FRANCE

Abstract.
When using labelled transition systems to model languages like CCS or TCSP, one specifies transitions by a set of structural rules. We consider labelling transitions with their proofs – in the given system of rules – instead of simple actions. Then the label of a transition identifies uniquely that transition, and one may use this information to define a concurrency relation on (proved) transitions, and a notion of residual of a (proved) transition by a concurrent one. We apply Berry and Lévy's notion of equivalence by permutations to sequences of proved transitions for CCS to obtain a partial order semantics for this language.

1. Introduction.

A computational system evolves by elementary computations from one state to the other, in notation $s \longrightarrow s'$. Examples of state changes are transitions of a machine, β-reductions of λ-terms and rewritings in a term rewriting system. To be able to reason about such computations – e.g. to show a Church-Rosser property –, we often need to have some indication of *what* has been performed and *where* it has happened. In other words, we have to deal with labelled transitions $s \xrightarrow{w} s'$, where w denotes a specific *occurrence* of some action. Now consider two computations from a same state, $s \xrightarrow{u} s_0$ and $s \xrightarrow{v} s_1$: we may have the intuition that these two moves are *compatible*, or independent. In this case we should be able to define what remains of one move after the other, in notation v/u and u/v, in such a way that v/u can still happen in state s_0, that is $s_0 \xrightarrow{v/u} s'$, and similarly $s_1 \xrightarrow{u/v} s''$. Moreover, if u and v are really independent, it should be possible to perform them in any order without affecting the result, that is we should have $s' = s''$. This is known as the *diamond property*, or the parallel moves property. Then two sequences of transitions may be regarded as equivalent if they are equal up to commutation of compatible moves, typically:

$$s \xrightarrow{u} s_0 \xrightarrow{v/u} s' \simeq s \xrightarrow{v} s_1 \xrightarrow{u/v} s'$$

This is the essence of Berry and Lévy's *equivalence by permutations* for sequences of (elementary) computations.

This equivalence was first elaborated by Lévy in his thesis (*cf.* [15]) upon Church notion of residual for the λ-calculus, and then used for recursive program schemes in [2]. It was further extended to deterministic term rewriting systems by Huet and Lévy in [14], and to non-deterministic

ones by Boudol in [4]. In any case, this equivalence allows one to associate with each "state" a complete partial order of computations. These computations are equivalence classes of sequences of elementary moves, ordered by the prefix ordering, up to permutations. The idea of associating a poset of computations with a program is the basis of Winskel's theory of *event structures*, where an event structure determines an ordered domain of configurations [23].

In this paper we introduce an equivalence by permutations for Milner's calculus CCS. As we saw earlier, we need to this purpose a notion of *occurrence* of action. The usual operational semantics of CCS describes processes as performing transitions labelled by actions. These transitions are inferred using a system of structural rules. What we shall take here as the occurrence of action associated with a transition is the *proof* of that transition in the given system of rules. Each proof identifies uniquely one transition, and we use this information to define a relation of independency or *concurrency* on (proved) transitions, and a notion of residual of a (proved) transition by a concurrent one. Since we have a diamond property, we are then able to define the equivalence by permutations on sequences of proved transitions. Now each equivalence class of sequences may be represented as a one step transition, labelled by a *pomset* (partially ordered multiset) [20] of actions. Roughly speaking, two events (occurrences of actions) are ordered in this poset – that is *causally related* – if one precedes the other in all the sequences of the class; conversely, they are unordered if they may be permuted. We thus obtain a *partial order* semantics for CCS, which is directly derived from its usual operational semantics.

As regards the semantics of concurrency, the idea of a computation as an equivalence class of sequences was first formalised by Mazurkiewicz in his theory of *traces* [16]. Let us recall that traces are equivalence classes of sequences of actions up to commutation of independent actions. In this setting the notion of residual is simple: the residual of an action by an independent one is the action itself (as we shall see later, this is not the case for our proved transitions).

The idea of abstracting from the ordering of concurrent transitions has been applied to Petri nets by Nielsen, Plotkin and Winskel in [18]. More recently, Best and Devillers have established the correspondence between the equivalence by permutations for firing sequences and processes of Petri nets [3].

The semantical framework of computation sequences modulo permutations has been axiomatized by Stark in the notion of "concurrent transition system" [21]. A related model is that of Bednarczyk's "labelled asynchronous systems" [1]. In all these approaches the primitive notion is that of concurrency, while causality arises from the fact that one deals with sequences. A similar but somewhat dual approach consists in endowing execution sequences with an explicit causality relation on their actions: this method has been used in [12] by Degano, De Nicola and Montanari, who obtain a pomset transition from a sequence of enriched transitions that they call atomic *concurrent histories*. Similarly, van Glabbeek and Vaandrager define in [13] a partial order semantics for (one-safe) Petri nets, using the structure of the net to associate a causality ordering with each firing sequence.

The semantics we give here for CCS may be seen as an extension of our semantics for "true concurrency" in [5,6], where communication (and restriction) were not considered. In that case it was possible to define partial order transitions *directly*, by composing partial orders on top of transitions according to the structure of terms. Unfortunately that approach could not be generalised easily to the whole CCS, essentially because communication introduces conflicts: then the composition of two partial order computations may yield a nondeterministic process – instead of a simple partial order. Here, as in most of the works mentioned above, we take an indirect approach and define the partial order semantics in two steps: we start with computational sequences, and then extract partial orders from them.

We shall only be concerned in defining the partial order semantics, and not in abstracting from it by defining a bisimulation equivalence (or other equivalence relations).

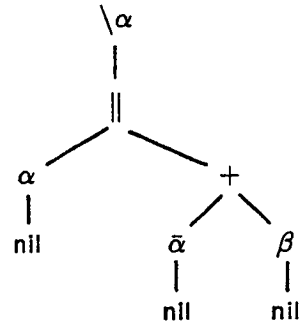
2. Pure CCS: terms and transitions.

As in [17], we assume a fixed set Δ of *names*. We use α, β,... to stand for names. We assume a set $\bar{\Delta}$ of *co-names* (complementary names), disjoint from Δ and in bijection with it: the co-name of α is $\bar{\alpha}$, while its name is $\mathsf{nm}(\bar{\alpha}) = \mathsf{nm}(\alpha) = \alpha$. Then $\Lambda = \Delta \cup \bar{\Delta}$ is the set of *labels*. We shall use λ to range over Λ, and extend the bijection so that $\bar{\bar{\lambda}} = \lambda$. As usual the set A of CCS *actions* is $A = \Lambda \cup \{\tau\}$, where τ is a new symbol, not in Λ; by convention the name of τ is τ. We use a, b, c,... to range over A. We presuppose a collection X (disjoint from A) of *identifiers*, and use x, y, z,... to range over identifiers.

We use the notation $a:p$ for the *action* construct of CCS and $(p \parallel q)$ for parallel composition. We shall not consider the relabelling operator, although it would not introduce any difficulty. The set T of (pure) CCS terms is given by the following grammar:

$$p \ ::= \ \mathsf{nil} \mid x \mid a:p \mid (p \parallel p') \mid (p+p') \mid (p\backslash\alpha) \mid \mu x.p$$

We shall use p, q, r,... to range over terms. Finite terms – built without fixpoint $\mu x.p$ – may be viewed as *finite trees*, with parallel composition and sum as binary node constructors, action and restriction as unary ones (with a parameter in A and Δ respectively), and nil as a constant. For instance the term $r = ((\alpha:\mathsf{nil} \parallel (\bar{\alpha}:\mathsf{nil} + \beta:\mathsf{nil}))\backslash\alpha)$ will be identified with the tree:



As is standard, the fixpoint construction binds the defined identifier, and substituting q for x in p may require renaming the bound variables of p in order to avoid captures; the result of such a substitution is denoted $p[q/x]$. Terms involving fixpoint define *infinite trees*, obtained by unfolding $\mu x.p$ into $p[\mu x.p/x]$ ad infinitum. In fact the construct $\mu x.p$ is just a device for defining infinite trees, and will never appear in the syntactic tree of a term. As it is usual, we assume that there is an empty tree Ω, in order to interpret diverging terms such as $\mu x.x$ for instance.

We recall now the standard transition system semantics of CCS. This is given by means of *inference rules*, allowing one to prove *transitions* of the form $p \xrightarrow{a} p'$. The transitions of a term p are exactly those which can be proved in the following system of rules:

action	$\vdash a:p \xrightarrow{a} p$
parallel composition 1	$p \xrightarrow{a} p' \ \vdash \ (p \parallel q) \xrightarrow{a} (p' \parallel q)$
parallel composition 2	$q \xrightarrow{b} q' \ \vdash \ (p \parallel q) \xrightarrow{b} (p \parallel q')$
communication	$p \xrightarrow{\lambda} p' \ , \ q \xrightarrow{\bar{\lambda}} q' \ \vdash \ (p \parallel q) \xrightarrow{\tau} (p' \parallel q')$
sum 1	$p \xrightarrow{a} p' \ \vdash \ (p+q) \xrightarrow{a} p'$
sum 2	$q \xrightarrow{b} q' \ \vdash \ (p+q) \xrightarrow{b} q'$
restriction	$p \xrightarrow{a} p' \ , \ \mathsf{nm}(a) \neq \alpha \ \vdash \ (p\backslash\alpha) \xrightarrow{a} (p'\backslash\alpha)$
fixpoint	$p[\mu x.p/x] \xrightarrow{a} p' \ \vdash \ \mu x.p \xrightarrow{a} p'$

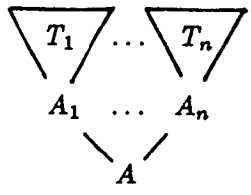
This set of rules is usually regarded as defining an *interleaving* semantics for CCS. Indeed, if we except the possibility of communication, the above rules for parallel composition describe $\|$ as an interleaving operator, allowing the components to move in any order, but not together.

Note that the semantics – the set of transitions which can be inferred for processes – contains no trace of the inference mechanism itself. We want to show that, if we keep track of the *proofs* of transitions, we can extract much more information from the same set of rules. In particular, we will be able to derive a non-interleaving semantics for the language, without having to depart from its basic operational semantics.

Let us first formalise the idea of a *proof* in the inference system of CCS. In general, in an inference system one has rules of the form:

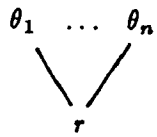
$$r: \frac{A_1, \ldots, A_n}{A}$$

to deduce an assertion from a finite set of assertions. Such rules generate proof trees of the form:



Here $T_1, \ldots, T_n$ are the proof trees for $A_1, \ldots, A_n$, and the whole tree is a proof for A.

Alternatively, one may represent the structure of a proof as a tree θ whose nodes are labelled by the *rules* which have been used in the derivation. It is not difficult to see that such a tree has the same shape as the proof tree itself. For example, if $\theta_1, \ldots, \theta_n$ represent the (rule labelled) trees for $A_1, \ldots, A_n$ and the last step of derivation consists in applying rule r, the whole proof may be represented by the tree:



Since each rule has an arity, that is a fixed finite number of premisses, a tree of this kind may be denoted by a term $r(\theta_1, \ldots, \theta_n)$, what we shall call a *proof term* – or simply a *proof* – in the following. This is the kind of notation we will use for proofs of transitions in CCS.

Our next step will be to associate proofs with assertions, and consider proved assertions like $\theta : A$, where θ is a proof of A. To manipulate proved assertions, we will use enriched rules of the kind:

$$\frac{x_1 : A_1, \ldots, x_n : A_n}{r(x_1, \ldots, x_n) : A}$$

which build up proofs of assertions as the inference process goes on. Whenever a rule is applied, the name r of the rule is recorded in the resulting proof term.

We start by giving the symbols for rules, which will serve as constructors for our proof terms θ:

a	for action (note that there is a separate rule for each action a)
$\|_0$	to prove a transition at the left of a parallel composition
$\|_1$	to prove a transition at the right of a parallel composition
κ	for communication
$+_0$	to prove a transition at the left of a sum
$+_1$	to prove a transition at the right of a sum
ρ_α	for restriction on the name α

Note that these proof constructors are just short names for the inference rules of CCS. Each constructor takes as many parameters as are the hypotheses of the corresponding rule. For example the constructor a takes no parameters, while κ takes two. We need not have a name for the fixpoint rule, since this is essentially a metarule, saying that a recursive term is to be interpreted as an infinite tree. For instance the terms $a:\mathsf{nil}$ and $\mu x.(a:\mathsf{nil})$ give rise to the same syntactic tree, and it would be odd to distinguish them by giving different proofs to their transitions $\xrightarrow{a}$. In fact, as we shall see later in more detail, proof terms are closely related to *paths* in a syntactic tree.

The syntax for *proofs* of CCS transitions is thus given by the grammar, where $a \in Act$ and $\alpha \in \Delta$:

$$\theta ::= a \mid \|_0(\theta) \mid \|_1(\theta) \mid \kappa(\theta,\theta') \mid +_0(\theta) \mid +_1(\theta) \mid \rho_\alpha$$

Note that although we call them proof terms, the θ's will not always represent *proper* proofs; for instance $\rho_\alpha(\alpha)$ and $\kappa(\alpha,\beta)$ do not correspond to any CCS transition. We give now the rules for building proper proof terms. Since any proper proof will contain the label of the corresponding transition, we define simultaneously the set of proper proof terms θ and their label $\ell(\theta)$.

The set Π of *proper proofs* and the labelling $\ell : \Pi \to Act$ are given by:

$$\begin{array}{lll}
a \in Act & \Rightarrow & a \in \Pi \quad \text{and} \quad \ell(a) = a \\
\theta \in \Pi & \Rightarrow & \|_i(\theta) \in \Pi \quad \text{and} \quad \ell(\|_i(\theta)) = \ell(\theta) \\
\theta \in \Pi & \Rightarrow & +_i(\theta) \in \Pi \quad \text{and} \quad \ell(+_i(\theta)) = \ell(\theta) \\
\theta,\theta' \in \Pi,\ \ell(\theta) = \overline{\ell(\theta)} & \Rightarrow & \kappa(\theta,\theta') \in \Pi \quad \text{and} \quad \ell(\kappa(\theta,\theta')) = \tau \\
\theta \in \Pi,\ \mathsf{nm}(\ell(\theta)) \neq \alpha & \Rightarrow & \rho_\alpha(\theta) \in \Pi \quad \text{and} \quad \ell(\rho_\alpha(\theta)) = \ell(\theta)
\end{array}$$

We may now bring together proofs and transitions. Usually one denotes by $\theta : A$ the fact that θ is a proof of the assertion A. In the inference system of CCS, assertions are transitions of the form $p \xrightarrow{a} p'$. Since we shall deal with sequences of transitions, we will prefer the notation $p \xrightarrow{a,\theta} p'$ to $\theta : p \xrightarrow{a} p'$. Note moreover that for such transitions we will always have $a = \ell(\theta)$ and thus we may omit the action a. We will then use the simpler notation $p \xrightarrow{\theta} p'$, to be interpreted as: θ is a proof of the fact that p performs the action $\ell(\theta)$ and becomes p' in doing so. We will call each $p \xrightarrow{\theta} p'$ a *proved transition*.

The rules for proved transitions are the following:

action	$\vdash a : p \xrightarrow{a} p$
parallel composition 1	$p \xrightarrow{\theta} p' \ \vdash \ (p \parallel q) \xrightarrow{\|_0(\theta)} (p' \parallel q)$
parallel composition 2	$q \xrightarrow{\theta} q' \ \vdash \ (p \parallel q) \xrightarrow{\|_1(\theta)} (p \parallel q')$
communication	$p \xrightarrow{\theta} p' \ , \ q \xrightarrow{\theta'} q', \ \ell(\theta) = \overline{\ell(\theta')} \ \vdash \ (p \parallel q) \xrightarrow{\kappa(\theta, \theta')} (p' \parallel q')$
sum 1	$p \xrightarrow{\theta} p' \ \vdash \ (p + q) \xrightarrow{+_0(\theta)} p'$
sum 2	$q \xrightarrow{\theta'} q' \ \vdash \ (p + q) \xrightarrow{+_1(\theta')} q'$
restriction	$p \xrightarrow{\theta} p' \ , \ \mathsf{nm}(\ell(\theta)) \neq \alpha \ \vdash \ (p\backslash\alpha) \xrightarrow{\rho_\alpha(\theta)} (p'\backslash\alpha)$
fixpoint	$p[\mu x.p/x] \xrightarrow{\theta} p' \ \vdash \ \mu x.p \xrightarrow{\theta} p'$

It should be clear that if we drop the proof terms – and retain their labels – we obtain exactly the rules of CCS. Note also that the proofs actually hold for the (infinite) trees that we get by unfolding the $\mu x.p$'s, since the rule for fixpoint does not introduce any special proof constructor.

Let us see an example. Take again the term $r = ((\alpha : \mathsf{nil} \parallel (\bar{\alpha} : \mathsf{nil} + \beta : \mathsf{nil}))\backslash\alpha)$. We give below the proved transition corresponding to the communication on α, $\bar{\alpha}$ (where to illustrate our technique we picture the proof tree as well):

$$\dfrac{\dfrac{}{\alpha : \mathsf{nil} \xrightarrow{\alpha} \mathsf{nil}} \qquad \dfrac{\dfrac{}{\bar{\alpha} : \mathsf{nil} \xrightarrow{\bar{\alpha}} \mathsf{nil}}}{(\bar{\alpha} : \mathsf{nil} + \beta : \mathsf{nil}) \xrightarrow{+_0(\bar{\alpha})} \mathsf{nil}}}{(\alpha : \mathsf{nil} \parallel (\bar{\alpha} : \mathsf{nil} + \beta : \mathsf{nil})) \xrightarrow{\kappa(\alpha, +_0(\bar{\alpha}))} (\mathsf{nil} \parallel \mathsf{nil})}$$

$$(\alpha : \mathsf{nil} \parallel (\bar{\alpha} : \mathsf{nil} + \beta : \mathsf{nil}))\backslash\alpha \xrightarrow{\rho_\alpha(\kappa(\alpha, +_0(\bar{\alpha})))} (\mathsf{nil} \parallel \mathsf{nil})\backslash\alpha$$

Decorating the transitions with their proofs provides us with a "maximal" concrete information. As a matter of fact, our proof terms are closely related to syntactic trees. If we look back at the syntactic tree for the term r, we may notice that the proof of a transition specifies a *path* in the tree. In the simplest case, this path leads to a subterm $a : q$, the one which performs the action a. However, if the action is a communication as in the example above, the proof is a path to a pair of complementary subterms $\lambda : q$ and $\bar{\lambda} : q'$.

To sum up, the proof of a transition $\xrightarrow{a}$ for a process p is an indication of how we get the action a from p. As it were, the proof of a transition identifies uniquely that transition: the new transition system of proved transitions is *deterministic*, that is to say, the transition relation is a (partial) function $f : (\mathbf{T} \times \Pi) \to \mathbf{T}$.

Now this concrete information can be weakened in various ways to obtain more abstract semantics. For instance we can extract from the proof θ of a transition the *local residual* associated with this proof, as defined by Castellani and Hennessy [8,9]. This will be given by a second function $g : (\mathbf{T} \times \Pi) \to \mathbf{T}$. Then one may consider decorated transitions of the form $p \xrightarrow{a \, , \, p''} p'$ where p'' is the local residual, and devise a notion of *distributed bisimulation*.

Here we shall use the information contained in proofs to define a relation of *concurrency* on transitions. This will enable us to define an equivalence by permutations on sequences of transitions, and thereby retrieve a partial order semantics for CCS.

3. Permutation of transitions.

We now introduce a notion of *concurrency* on proved transitions. Roughly speaking, two transitions are concurrent if they occur on different sides of a parallel composition, whereas they are in *conflict* (not concurrent) if they occur on different sides of a sum. However some complications arise from communication, which may introduce new conflicts. Typically, two communications will be in conflict if they share one component. Conversely, they will be concurrent if they are pairwise concurrent – i.e. they have concurrent (corresponding) components.

The relation of concurrency on proved transitions is induced from a relation of concurrency between proof terms, in notation $\theta \frown \theta'$, which we define now.

The relation $\frown$ on proof terms is the least symmetric relation *compatible with the proof constructors* which satisfies the following clauses (for any $\theta, \theta', \theta'' \in \Pi$):

(A1) $\|_0(\theta) \frown \|_1(\theta')$

(A2) $\theta \frown \theta' \Rightarrow \begin{cases} \|_0(\theta) \frown \kappa(\theta', \theta'') \\ \|_1(\theta) \frown \kappa(\theta'', \theta') \end{cases}$

As regards communication, compatibility with the constructor κ amounts to requiring:

$$\theta_0 \frown \theta_0' \text{ and } \theta_1 \frown \theta_1' \Rightarrow \kappa(\theta_0, \theta_1) \frown \kappa(\theta_0', \theta_1')$$

For instance, the two communications in the term $(\alpha : \mathsf{nil} \parallel \beta : \mathsf{nil}) \parallel (\bar{\alpha} : \mathsf{nil} \parallel \bar{\beta} : \mathsf{nil})$ are concurrent, since: $\kappa(\|_0(\alpha), \|_0(\bar{\alpha})) \frown \kappa(\|_1(\beta), \|_1(\bar{\beta}))$. Note that (A2) also expresses a kind of compatibility of $\frown$ w.r.t. the constructors $\|_i$, since a proof $\kappa(\theta', \theta'')$ stands for the co-occurrence of $\|_0(\theta')$ and $\|_1(\theta'')$. We could have used the notation $(\|_0(\theta'), \|_1(\theta''))$ in place of $\kappa(\theta', \theta'')$.

We give next the definition of concurrency on proved transitions.

DEFINITION (CONCURRENT TRANSITIONS). *Let $t_0 = p \xrightarrow{\theta_0} p_0$ and $t_1 = p \xrightarrow{\theta_1} p_1$ be two proved transitions for the same CCS term p. The transitions are concurrent, in notation $t_0 \frown t_1$, if and only if $\theta_0 \frown \theta_1$.*

By definition the concurrency relation between transitions is symmetric and irreflexive: it is easy to check that $\theta \frown \theta' \Rightarrow \theta \neq \theta'$.

To illustrate the application of clauses (A1) and (A2), we examine the relations between some of the transitions of the term $p = ((\alpha : \mathsf{nil} \parallel \alpha : \mathsf{nil}) \parallel \bar{\alpha} : \mathsf{nil})$. The two α-transitions are concurrent because $\|_0(\|_0(\alpha)) \frown \|_0(\|_1(\alpha))$, by (A1) and compatibility of $\frown$ with the constructor $\|_0$. The first α-transition and the communication on $\alpha, \bar{\alpha}$ of the remaining two components are also concurrent, because $\|_0(\|_0(\alpha)) \frown \kappa(\|_1(\alpha), \bar{\alpha})$ is an instance of (A2), with $\theta = \|_0(\alpha)$ and $\theta' = \|_1(\alpha)$. On the other hand $\|_0(\|_0(\alpha)) \not\frown \kappa(\|_0(\alpha), \bar{\alpha})$, since $\|_0(\alpha) \not\frown \|_0(\alpha)$ and thus (A2) does not apply. We also have a conflict between the two communications, since the two transitions

$$p \xrightarrow{\kappa(\|_0(\alpha), \bar{\alpha})} ((\mathsf{nil} \parallel \alpha : \mathsf{nil}) \parallel \mathsf{nil}) \quad , \quad p \xrightarrow{\kappa(\|_1(\alpha), \bar{\alpha})} ((\alpha : \mathsf{nil} \parallel \mathsf{nil}) \parallel \mathsf{nil})$$

share the same "sub-transition" $\|_1(\bar{\alpha})$.

Another case of conflict occurs in the term $r = (\alpha : \mathsf{nil} \parallel (\bar{\alpha} : \mathsf{nil} + \beta : \mathsf{nil}))\backslash\alpha$ considered earlier. Here the two transitions:

$$r \xrightarrow{\rho_\alpha(\|_1(+_1(\beta)))} (\alpha : \mathsf{nil} \parallel \mathsf{nil})\backslash\alpha \quad , \quad r \xrightarrow{\rho_\alpha(\kappa(\alpha, +_0(\bar{\alpha})))} (\mathsf{nil} \parallel \mathsf{nil})\backslash\alpha$$

are not concurrent since they made two different choices at the subterm $(\bar{\alpha} : \mathsf{nil} + \beta : \mathsf{nil})$.

We define now the *residual* θ/θ' of a proof term by a concurrent one, namely what is left of the proof θ after θ'. This residual may differ from the proof term itself because of nondeterministic choices. When a parallel composition is put in a sum-context, the choice may be made by any of the parallel components, and does not have to be solved again by the other components. This point will be made clearer by an example below.

For any concurrent proofs θ, θ', the residual θ/θ' is defined by:

$$i \neq j \;\Rightarrow\; \|_i(\theta) \,/\, \|_j(\theta') = \|_i(\theta)$$

$$\theta \smile \theta' \;\Rightarrow\; \begin{cases} \|_0(\theta)/\kappa(\theta', \theta'') = \|_0(\theta/\theta') \text{ and } \kappa(\theta', \theta'')/\|_0(\theta) = \kappa(\theta'/\theta, \theta'') \\ \|_1(\theta)/\kappa(\theta'', \theta') = \|_1(\theta/\theta') \text{ and } \kappa(\theta'', \theta')/\|_1(\theta) = \kappa(\theta''/\theta, \theta') \end{cases}$$

$$\theta \smile \theta' \;\Rightarrow\; \begin{cases} \|_i(\theta)/\|_i(\theta') = \|_i(\theta/\theta') \\ +_i(\theta)/+_i(\theta') = \theta/\theta' \\ \rho_\alpha(\theta)/\rho_\alpha(\theta') = \rho_\alpha(\theta/\theta') \end{cases}$$

$$\theta_0 \smile \theta_0' \text{ and } \theta_1 \smile \theta_1' \;\Rightarrow\; \kappa(\theta_0, \theta_1)/\kappa(\theta_0', \theta_1') = \kappa(\theta_0/\theta_0', \theta_1/\theta_1')$$

Let us look at an example, which shows in which way residuals are affected by choices. The term $p = ((a : \mathsf{nil} \parallel b : \mathsf{nil}) + c : \mathsf{nil})$ may do the proved transitions:

$$p \xrightarrow{+_0(\|_0(a))} (\mathsf{nil} \parallel b : \mathsf{nil}), \qquad p \xrightarrow{+_0(\|_1(b))} (a : \mathsf{nil} \parallel \mathsf{nil})$$

So the proof of the b-transition is $+_0(\|_1(b))$. On the other hand, once the a-transition has happened, the proof of the b-transition becomes $\|_1(b) = +_0(\|_1(b))/+_0(\|_0(a))$, and we have:

$$(\mathsf{nil} \parallel b : \mathsf{nil}) \xrightarrow{(\|_1(b))} (\mathsf{nil} \parallel \mathsf{nil})$$

This shows a difference between our framework and Mazurkiewicz theory of traces. In a trace, two independent actions may always be commuted as they stand, since the residual of one action after the other is the action itself. Our formalism is somehow more concrete, as we need to record choices in our proof terms. Note that without the constructors $+_i$ we would not be able to define the concurrency relation on our proofs. For consider the term:

$$(a : \mathsf{nil} \parallel b : \mathsf{nil}) \;+\; (a : \mathsf{nil} \parallel c : \mathsf{nil})$$

If we did not record the $+_i$ in our proofs, we would not be able to distinguish the two a-transitions and thus we would not know which is concurrent with the b-transition and which is concurrent with the c-transition.

We turn now to the main property of our concurrency relation. The following result, also known as the parallel moves lemma, states a "conditional Church-Rosser property", namely that any two concurrent transitions are confluent. This result is much simpler in CCS than in λ-calculus or term rewriting systems, since a proof of a transition cannot be duplicated or deleted by a concurrent one; it can only be consumed (by a communication), or left unchanged – up to the resolution of choices – when the two proofs are concurrent.

LEMMA (THE DIAMOND LEMMA). *Let* $t_0 = p \xrightarrow{\theta_0} p_0$ *and* $t_1 = p \xrightarrow{\theta_1} p_1$ *be two proved transitions. If they are concurrent then there exists a unique term* $\bar{p}$ *such that* $p_0 \xrightarrow{\theta_1/\theta_0} \bar{p}$ *and* $p_1 \xrightarrow{\theta_0/\theta_1} \bar{p}$.

This property is in fact much stronger than confluence: it says that a (proved) transition survives any concurrent one. We can then adopt the standard terminology of ([2,4,14,15]): the transition $t_1' = p_0 \xrightarrow{\theta_1/\theta_0} \bar{p}$ (with the notations of the diamond lemma) is the *residual* of t_1 by t_0, denoted t_1/t_0 and similarly $t_0/t_1 = p_1 \xrightarrow{\theta_0/\theta_1} \bar{p}$ is the residual of t_0 by t_1.

This allows us to define the equivalence by permutations on sequences of transitions of CCS terms. Each CCS term p determines a set $\mathcal{T}(p)$ of finite *sequences* of proved transitions of the form

$$p \xrightarrow{\theta_1} p_1 \cdots p_{n-1} \xrightarrow{\theta_n} p_n$$

which may equivalently be presented as sequences of steps:

$$t_1 \cdots t_n \quad \text{where} \quad t_i = p_{i-1} \xrightarrow{\theta_i} p_i, \quad 1 \leq i \leq n \quad (\text{and } p_0 = p)$$

We give next the definition of permutation equivalence on $\mathcal{T}(p)$: intuitively two sequences of proved transitions are equivalent if they are the same up to permutations of concurrent steps.
Let ss' denote the concatenation of $s \in \mathcal{T}(p)$ and $s' \in \mathcal{T}(q)$, which is only defined if s ends at q.

DEFINITION (PERMUTATION EQUIVALENCE). *Let* p *be a CCS term. The equivalence by permutations on* $\mathcal{T}(p)$ *is the least equivalence* $\simeq$ *such that*

$$s_0 t_0 (t_1/t_0) s_1 \;\simeq\; s_0 t_1 (t_0/t_1) s_1$$

(provided that $t_0 \smile t_1$ *and that concatenation is defined).*

An example of equivalent sequences of transitions is:

$$(a:p \parallel b:q) + c:r \xrightarrow{+_0(\|_0(a))} (p \parallel b:q) \xrightarrow{\|_1(b)} (p \parallel q)$$

$$(a:p \parallel b:q) + c:r \xrightarrow{+_0(\|_1(b))} (a:p \parallel q) \xrightarrow{\|_0(a)} (p \parallel q)$$

Here one can commute the two steps. There is another kind of sequences of transitions where this is not possible, because a step is *caused*, or created, by a previous one. The typical example is obviously:

$$a:b:\mathsf{nil} \xrightarrow{a} b:\mathsf{nil} \xrightarrow{b} \mathsf{nil}$$

We may finally proceed to the definition of the partial order semantics. The equivalence class of a sequence

$$p \xrightarrow{\theta_1} \cdots \xrightarrow{\theta_n} p'$$

may be represented as a *one step transition* $p \xrightarrow{P} p'$ where P is a *pomset* (partially ordered multiset [20]) of actions of A, – that is an isomorphism class of posets labelled in A. Such pomset transitions were introduced in [5] for a subset of CCS. Let us formalize this idea: we shall write $s \sim_\varsigma s'$ if s'

results from s by the transposition of the steps i and $i+1$, and ς is the corresponding transposition of $\{1,\ldots,n\}$, where n is the length of s (obviously $\simeq$ preserves the length of sequences). So $\varsigma(i) = i+1$ and $\varsigma(i+1) = i$. It should be clear that $s' \simeq s$ if and only if there is a sequence $\varsigma_1,\ldots,\varsigma_k$ of such transpositions from s to s'. Let us denote this fact by $s \sim_{\varsigma_1,\ldots,\varsigma_k} s'$. Then the equivalence class of $s = p \xrightarrow{\theta_1} \cdots \xrightarrow{\theta_n} p'$ determines a transition $p \xrightarrow{P} p'$, where $P = (E, l, \leq)$ is the labelled poset defined by

$$\begin{cases} E = \{e_1,\ldots,e_n\} \\ l(e_i) = \ell(\theta_i) \\ e_i \leq e_j \;\Leftrightarrow\; \forall s'.\ s' \sim_{\varsigma_1,\ldots,\varsigma_k} s \;\Rightarrow\; \eta(i) \leq \eta(j) \quad \text{where } \eta = \varsigma_k \circ \cdots \circ \varsigma_1 \end{cases}$$

Note that P is defined up to isomorphism, since the events e_i are taken arbitrarily. This construction is close to that of *dependency graph* corresponding to a trace – as defined by Mazurkiewicz in [16]. A similar definition is given in [13] for Petri nets.

Let us see an example. The equivalence class of the sequence

$$(a : p \parallel b : c : q) \xrightarrow{\|_0(a)} (p \parallel b : c : q) \xrightarrow{\|_1(b)} (p \parallel c : q) \xrightarrow{\|_1(c)} (p \parallel q)$$

may be represented as a transition whose label is a pomset consisting of events e_1, e_2 and e_3 labelled a, b and c respectively, where e_2 precedes e_3 and e_1 is incomparable with e_2 and e_3, that is:

$$(a : p \parallel b : b : q) \xrightarrow{\left\{\begin{array}{cc} a & b \\ & | \\ & c \end{array}\right\}} (p \parallel q)$$

We shall conclude this section with a short digression about fairness. It is well-known that to talk about fairness one needs a notion of *occurrence* of action. Thus our framework may be well-suited to deal with this issue. Moreover one must be able to reason about infinite sequences. The equivalence by permutations may be easily generalised to infinite sequences of proved transitions. In this case the equivalence is defined in terms of a *preorder*, which is just the prefix order up to permutations.

Let $\mathcal{T}^\infty(p)$ be the set of (finite and infinite) sequences of proved transitions of p, and $\ll$ be the usual prefix order:

$$s \ll s' \;\Leftrightarrow_{\text{def}}\; s = s' \text{ or } \exists s''\ ss'' = s'$$

where the concatenation ss'' is only defined if s is a finite sequence. The preorder $\lesssim$ on $\mathcal{T}^\infty(p)$ is then defined by:

$$s_0 \lesssim s_1 \;\Leftrightarrow_{\text{def}}\; \forall s_0' \in \mathcal{T}(p)\ s_0' \ll s_0 \;\Rightarrow\; \exists s_1', s_1'' \in \mathcal{T}(p) \;\; s.t. \;\; s_0' \ll s_1'' \simeq s_1' \ll s_1$$

It is easy to show that for finite sequences of transitions s and s' of the same term:

$$s \simeq s' \Leftrightarrow s \lesssim s' \;\&\; s' \lesssim s$$

Therefore we shall keep the notation $\simeq$ for the equivalence on $\mathcal{T}^\infty(p)$ induced by the preorder $\lesssim$.

We define the set of (partial order) *computations* of p to be the quotient $\mathcal{C} = \mathcal{T}^\infty(p)/\simeq$. This is a partially ordered set – the ordering on equivalence classes will be denoted $\sqsubseteq$. In a forthcoming paper we show that $\mathcal{C}$ is the domain of configurations of an event structure. This result is already delineated in [7].

In [4], the maximal computations (w.r.t. $\sqsubseteq$) were called *terminating* since, roughly speaking, it does not remain anything to do after a maximal computation. More precisely, if an action is possible at some point of a maximal computation, then after a finite amount of time, this possibility disappears – either because the action has been done or because it is no longer enabled.

Then for CCS the maximal computations set up a notion of *fairness*: these are the computations satisfying a *finite delay property*. For instance, if $(a^\omega \parallel b^\omega) = (\mu x.a : x \parallel \mu x.b : x)$, the sequence:

$$s = (a^\omega \parallel b^\omega) \xrightarrow{\parallel_0(a)} (a^\omega \parallel b^\omega) \cdots \xrightarrow{\parallel_0(a)} \cdots$$

is not maximal since the proved transition

$$s' = (a^\omega \parallel b^\omega) \xrightarrow{\parallel_1(b)} (a^\omega \parallel b^\omega)$$

has a residual along the whole computation. Indeed we have $s \lesssim s's$ and $s \not\simeq s's$, since – denoting by p the term $(a^\omega \parallel b^\omega)$, and writing for simplicity only the actions on top of transitions – we have:

$$p \xrightarrow{a} \cdots \xrightarrow{a} p \ll p \xrightarrow{a} \cdots \xrightarrow{a} \xrightarrow{b} p \simeq p \xrightarrow{b} \xrightarrow{a} \cdots \xrightarrow{a} p$$

On the other hand if $(a+b)^\omega = \mu x.(a : x + b : x)$, the sequence:

$$(a+b)^\omega \xrightarrow{+_0(a)} (a+b)^\omega \cdots \xrightarrow{+_0(a)} \cdots$$

is a maximal computation. Not too surprisingly, our proof terms are similar to the labels used by Costa and Stirling in [10] to define various notions of fairness. We should point out however that a maximal computation is not what is usually called (weakly or strongly) fair computation. This is so because our notion of proved transition is rather discriminating. For instance in $r = \mu x.(\alpha : x + \beta : \mathsf{nil})$, the action β has infinitely many distinct proofs (this is apparent in the infinite tree of this term): informally, at each point of choice in r, a "new" occurrence β is available. Then

$$(r \parallel \bar{\beta} : \mathsf{nil})\backslash\beta \xrightarrow{\parallel_0(+_0(\alpha))} (r \parallel \bar{\beta} : \mathsf{nil})\backslash\beta \cdots \xrightarrow{\parallel_0(+_0(\alpha))} \cdots$$

is a maximal computation: at each step the potential communication is different, and at each step it is discarded. There is no proved transition which is "infinitely often" or "almost always" enabled along this computation. Similarly if $q = \mu x.c : (\alpha : x + \beta : \mathsf{nil})$ then

$$(q \parallel \bar{\beta} : \mathsf{nil})\backslash\beta \xrightarrow{c} \xrightarrow{\parallel_0(+_0(\alpha))} (q \parallel \bar{\beta} : \mathsf{nil})\backslash\beta \cdots \xrightarrow{c} \xrightarrow{\parallel_0(+_0(\alpha))} \cdots$$

is a maximal computation.

4. Conclusion.

The technique presented here is very general and may be applied to any language whose semantics is given by a system of structural rules. Concerning CCS, further work has been undertaken in showing the relation between the semantics by permutations and the event structure [23] and Petri net semantics for this language. For Petri nets, we use a construction of nets from the operational semantics which is directly inspired from the work of Degano et al. [11] and of Olderog [19]. The relation with the *behaviour structures* of Trakhtenbrot et al. [22] seems also worth investigation.

REFERENCES

[1] M. A. BEDNARCZYK, *Categories of Asynchronous Systems*, Ph. D. Thesis, University of Sussex (1987).

[2] G. BERRY, J.-J. LÉVY, *Minimal and Optimal Computations of Recursive Programs*, JACM 26 (1979) 148-175.

[3] E. BEST, R. DEVILLERS, *Interleaving and Partial Orders in Concurrency: A Formal Comparison*, in Formal Description of Programming Concepts III, North-Holland (1987) 299-321.

[4] G. BOUDOL, *Computational Semantics of Term Rewriting Systems*, in Algebraic Methods in Semantics (M. Nivat, J.C. Reynolds, Eds), Cambridge University Press (1985) 169-236.

[5] G. BOUDOL, I. CASTELLANI, *On the Semantics of Concurrency: Partial Orders and Transition Systems*, TAPSOFT 87, Lecture Notes in Comput. Sci. 249 (1987) 123-137.

[6] G. BOUDOL, I. CASTELLANI, *Concurrency and Atomicity*, Theoretical Comput. Sci. 59 (1988) 1-60.

[7] G. BOUDOL, I. CASTELLANI, *Permutation of Transitions: An Event Structure Semantics for CCS and SCCS*, to appear in the Proc. of REX Workshop, Noordwijkerhout (1988).

[8] I. CASTELLANI, M. HENNESSY, *Distributed Bisimulations*, Comput. Sci. Rep. 5-87, University of Sussex (1987).

[9] I. CASTELLANI, *Bisimulations for Concurrency*, Ph. D. Thesis, University of Edinburgh (1988).

[10] G. COSTA, C. STIRLING, *Weak and Strong Fairness in* CCS, Information and Computation 73 (1987) 207-244.

[11] P. DEGANO, R. DE NICOLA, U. MONTANARI, *A New Operational Semantics for CCS Based on Condition/Event Systems*, Nota Interna B4-42, IEI, Pisa (1986) to appear.

[12] P. DEGANO, R. DE NICOLA, U. MONTANARI, *Concurrent Histories: a Basis for Observing Distributed Systems*, J. of Computer and Systems Sciences 34 (1987) 422-461.

[13] R. van GLABBEEK, F. VAANDRAGER, *Petri Net Models for Algebraic Theories of Concurrency*, Proceedings PARLE Conference, Eindhoven, Lecture Notes in Comput. Sci. 259 (1987) 224-242.

[14] G. HUET, J.-J. LÉVY, *Call-by-need Computations in Non-ambiguous Linear Term Rewriting Systems*, IRIA-LABORIA Report 359 (1979).

[15] J.-J. LÉVY, *Optimal Reductions in the Lambda Calculus*, in To H. B. CURRY: Essays on Combinatory Logic, Lambda Calculus and Formalism (J.P. Seldin, J.R. Hindley, Eds), Academic Press (1980) 159-191.

[16] A. MAZURKIEWICZ, *Concurrent Program Schemes and their Interpretations*, Aarhus Workshop on Verification of Parallel Programs, Daimi PB-78, Aarhus University (1977).

[17] R. MILNER, *A Calculus of Communicating Systems*, Lecture Notes in Comput. Sci. 92 (1980) reprinted in Report ECS-LFCS-86-7, Edinburgh University.

[18] M. NIELSEN, G. PLOTKIN, G. WINSKEL, *Petri Nets, Event Stuctures and Domains*, Theoret. Comput. Sci. 13 (1981) 85-108.

[19] E.-R. OLDEROG, *Operational Petri Net Semantics for CCSP*, Advances in Petri Nets 87, Lecture Notes in Comput. Sci. 266 (1987) 196-223.

[20] V.R. PRATT, *Modelling Concurrency with Partial Orders*, Intern. J. of Parallel Programming 15 (1986) 33-71.

[21] E. W. STARK, *Concurrent Transition Systems*, Tech. Report, State Univ. of New York at Stony Brook (1987).

[22] B. A. TRAKHTENBROT, A. RABINOVICH, J. HIRSHFELD, *Nets of Processes*, Tech. Rep. 97-88, Comput. Sci. Institute, Tel Aviv University (1988).

[23] G. WINSKEL, *Event Structures*, Advances in Petri Nets 86, Lecture Notes in Comput. Sci. 255 (1987) 325-392.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique