Permutation of Transitions: An Event Structure Semantics for CCS and SCCS

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

Abstract. We apply Berry & Lévy's notion of equivalence by permutations to CCS and MEIJE/SCCS, thus obtaining a pomset transition semantics for these calculi. We show that this provides an operational counterpart for an event structure semantics for CCS and SCCS similar to the one given by Winskel.

Keywords: process algebras, pomset-labelled transition systems, event structures.

Contents

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. When states are abstract programs one may extract from their syntactical structure some indication of *what* has been performed and *where* it has happened. In other words, one may decorate transitions with a label w, thus obtaining $s \xrightarrow{w} s'$, where w is an occurrence of action. Now assume that $s \xrightarrow{u} s_0$ and $s \xrightarrow{v} s_1$: in many cases we may have the intuition that these two moves are *compatible*, or independent. This means that we are 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 happen in state s_0, that is $s_0 \xrightarrow{v/u} s'$, and similarly $s_1 \xrightarrow{u/v} s''$. If u and v are really compatible, we should be able to perform them in any order, without affecting the result, that is: $s' = s''$. This is known as the *diamond property*, or the parallel moves property. Moreover, two sequences of transitions should 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 [1]. It was further extended to deterministic term rewriting systems by Huet and Lévy in [14], and to non-deterministic ones by Boudol in [3]. 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 commutations. A similar notion is used for Petri nets by Nielsen, Plotkin and Winskel, who define in [20] an equivalence that "*abstracts away from the ordering of concurrent firings of transitions*" (this is also used by van Glabbeek and Vaandrager in [13], and by Best and Devillers in [2]; a similar idea is that of *trace* of Mazurkiewicz [16]). Moreover they show that for nets the ordered space of computations has a nice characterization: it is the space of configurations of an *event structure*. As a matter of fact, the three basic connectives of event structures – causality, concurrency and conflict – are already present in computations. Roughly speaking, two occurrences of actions (events) u and v are *consistent* (non-conflicting), with respect to a state s if they can appear in the same computation of s:

$$s \cdots \xrightarrow{u} \cdots \xrightarrow{v} \cdots$$

In this case they are concurrent if they may be permuted:

$$s \cdots \xrightarrow{u} \cdots \xrightarrow{v} \cdots \simeq s \cdots \xrightarrow{v} \cdots \xrightarrow{u} \cdots$$

Otherwise they are causally related: one of them must precede the other.

In this note we propose an equivalence by permutations for Milner's calculi CCS and SCCS [17,18], and show that the ordered space of computations of a term is the poset of configurations of an event structure. The events are simply occurrences of actions, and, roughly speaking, they are compatible if they lie on different sides of a parallel system, though some complications arise from communication. We show that each equivalence class of computations (up to permutations) may be represented as a one step transition, where the action is a labelled poset of events. With the exception of communication, this corresponds exactly to our semantics for "true concurrency" in [6,7]. Our operational semantics for CCS is similar to the one given by Degano, De Nicola and Montanari in [11], who obtain a poset transition from a sequence of "atomic transitions" that they call atomic concurrent histories. The poset transition semantics provides us with an operational counterpart to the interpretation of CCS terms as event structures. However it remains to be checked that our constructions coincide, at least in interpreting CCS, with those given by Winskel in [22] (see also [23]).

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 $\bar{\alpha}$, and its name is $\mathsf{nm}(\alpha) = \alpha = \mathsf{nm}(\bar{\alpha})$. 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, $c, \ldots$ to range over A. We presuppose a collection X (disjoint from A) of *identifiers*, and use x, y, $z, \ldots$ to range over identifiers.

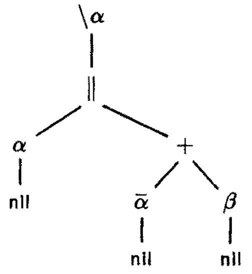
The *action* construct of CCS will be denoted by $a\colon p$, while the parallel composition is $(p \parallel q)$. We shall not use the relabelling operator, although it would not introduce any difficulty. The syntax of (pure) CCS terms is given by the following grammar:

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

We shall use p, q, $r, \ldots$ to range over terms; finite terms – built without fixpoint $\mu x.p$ – should 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 constant. However this representation will remain implicit throughout this paper. For instance the term

$$r = ((\alpha\colon \mathsf{nil} \parallel (\bar{\alpha}\colon \mathsf{nil} + \beta\colon \mathsf{nil}))\backslash \alpha)$$

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. As it is usual, we assume that there is a constant Ω, which is not a CCS term, in order to interpret diverging terms such as $\mu x.x$ for instance.

The semantics of CCS terms is given by means of *inference rules*, allowing one to prove *transitions* of the form $p \xrightarrow{a} p'$ for terms. We assume these rules to be known (see [17]). Modern proof theory shows that there are some advantages to reap from a *syntax* for proofs – if we think of inference rules as proof constructions. The case of CCS is very simple, since the validity of a "proposition" $p \xrightarrow{a} p'$ only depends on the structure of the term p. More precisely, a proof of such a transition is just an indication of how we get the action a from the term p. In the simplest case, this indication is a path which leads to an (outermost) subterm $a\colon q$. But the action can also be a communication τ, in which case this indication is a path to a pair of complementary subterms $\lambda\colon q$ and $\bar{\lambda}\colon q'$. Then we have to devise a syntax for these paths, which are some kind of initial subterms. Let F be a set of function symbols, which are symbols with arity, from which we build terms. Then with each $f \in F$ of arity n we can associate a collection of new symbols f_m, one for each $m \subseteq \{1, \ldots, n\}$, so that the arity of f_m is the cardinality of m. For instance the "split" term $f_{\{i_1,\ldots,i_k\}}(t_{i_1}, \ldots, t_{i_k})$ represents an initial subterm of $f(t_1, \ldots, t_n)$ obtained by deleting some arguments of f.

In the case of CCS, we only need some of these constructs, namely $a : \emptyset$, $\|_{\{1\}}$, $\|_{\{2\}}$ and $\|_{\{1,2\}}$, $+_{\{1\}}$ and $+_{\{2\}}$, $\backslash\alpha_{\{1\}}$. We shall use specific names for these, respectively γ_a, π_0, π_1, δ, σ_0, σ_1 and ρ_α. The syntax for proofs of CCS transitions is thus given by the grammar:

$$\theta \ ::= \ \gamma_a \mid \pi_0(\theta) \mid \pi_1(\theta) \mid \delta(\theta,\theta') \mid \sigma_0(\theta) \mid \sigma_1(\theta) \mid \rho_\alpha(\theta)$$

One should note that although we call them proof terms, the θ's will not in general represent *valid* proofs; for instance $\rho_\alpha(\gamma_\alpha)$ does not correspond to any CCS transition, and the reader should be able to find other kinds of examples. The valid proofs are those built by means of the formation rules below. Usually one denotes by $\theta : \Phi$ the fact that θ is a proof of the proposition Φ; since we shall use sequences of transitions, we prefer the notation $p \xrightarrow{a\,,\,\theta} p'$ for: *θ is a proof of the fact that p performs the action a and becomes p′ in doing so.* We call these enriched transitions *proved transitions.* The rules of inference (and formation of proofs) are the following:

action	$\vdash\ a : p \xrightarrow{a\,,\,\gamma_a} p$
parallel composition 1	$p \xrightarrow{a\,,\,\theta} p' \ \vdash\ (p \parallel q) \xrightarrow{a\,,\,\pi_0(\theta)} (p' \parallel q)$
parallel composition 2	$q \xrightarrow{b\,,\,\theta'} q' \ \vdash\ (p \parallel q) \xrightarrow{b\,,\,\pi_1(\theta')} (p \parallel q')$
communication	$p \xrightarrow{\lambda\,,\,\theta} p'\ ,\ q \xrightarrow{\bar\lambda\,,\,\theta'} q' \ \vdash\ (p \parallel q) \xrightarrow{\tau\,,\,\delta(\theta,\theta')} (p' \parallel q')$
sum 1	$p \xrightarrow{a\,,\,\theta} p' \ \vdash\ (p + q) \xrightarrow{a\,,\,\sigma_0(\theta)} p'$
sum 2	$q \xrightarrow{b\,,\,\theta'} q' \ \vdash\ (p + q) \xrightarrow{b\,,\,\sigma_1(\theta')} q'$
restriction	$p \xrightarrow{a\,,\,\theta} p'\ ,\ \mathsf{nm}(a) \neq \alpha \ \vdash\ (p\backslash\alpha) \xrightarrow{a\,,\,\rho_\alpha(\theta)} (p'\backslash\alpha)$
fixpoint	$p[\mu x.p/x] \xrightarrow{a\,,\,\theta} p' \ \vdash\ \mu x.p \xrightarrow{a\,,\,\theta} p'$

It should be clear that if we drop the proof terms these rules are exactly those 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 (meta) rule for fixpoint does not introduce any special proof constructor. Let us see an example: we have for the previous term $r = ((\alpha : \mathsf{nil} \parallel (\bar\alpha : \mathsf{nil} + \beta : \mathsf{nil}))\backslash\alpha)$

$$r \xrightarrow{\beta\,,\,\rho_\alpha(\pi_1(\sigma_1(\gamma_\beta)))} ((\alpha : \mathsf{nil} \parallel \mathsf{nil})\backslash\alpha)$$

and

$$r \xrightarrow{\tau\,,\,\rho_\alpha(\delta(\gamma_\alpha,\sigma_0(\gamma_{\bar\alpha})))} ((\mathsf{nil} \parallel \mathsf{nil})\backslash\alpha)$$

Decorating the transitions with their proofs provides us with a "maximal" concrete information. This can be weakened in various ways to obtain more abstract semantics. For instance we can extract from a proof θ of a transition $p \xrightarrow{a} p'$ the *local residual* associated with this proof, as defined by Castellani and Hennessy [8,9] (we omit the formal definition). Then one may consider decorated transitions of the form $p \xrightarrow{a\,,\,p''} p'$ where p'' is the local residual, and devise an enriched notion of bisimulation.

As a matter of fact, we could have used transitions $p \xrightarrow{\theta} p'$, since the action itself is determined by the proof: it is the *label* $\ell(\theta)$ of the proof, defined as:

(i) $\ell(\gamma_a) = a$;

(ii) $\ell(f(\theta)) = \ell(\theta)$ for all unary proof constructor f;

(iii) $\ell(\delta(\theta, \theta')) = \tau$.

3. Permutation of transitions.

In order to define the equivalence by permutations on sequences of transitions, we first need a notion of *concurrent* proved transitions. Roughly speaking, two transitions are concurrent if they occur on different sides of a parallel composition, whereas they are in conflict 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, denoted $\theta \smile \theta'$. The relation $\smile$ on proof terms is the least symmetric relation *compatible with the proof constructors* which satisfies the following clauses:

(A1) $\pi_0(\theta) \smile \pi_1(\theta')$

(A2) $\theta \smile \theta' \Rightarrow \begin{cases} \pi_0(\theta) \smile \delta(\theta', \theta'') \\ \pi_1(\theta) \smile \delta(\theta'', \theta') \end{cases}$

For instance, considering the term $((\alpha : \mathsf{nil} \parallel \beta : \mathsf{nil}) \parallel \bar{\alpha} : \mathsf{nil})$, we have $\pi_0(\pi_1(\gamma_\beta)) \smile \delta(\pi_0(\gamma_\alpha), \gamma_{\bar{\alpha}})$. As regards communication, compatibility of $\smile$ with the constructor δ amounts to requiring:

$$\theta_0 \smile \theta_0' \text{ and } \theta_1 \smile \theta_1' \Rightarrow \delta(\theta_0, \theta_1) \smile \delta(\theta_0', \theta_1')$$

Note that $\theta \smile \theta' \Rightarrow \theta \neq \theta'$.

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

Note that the concurrency relation between transitions is symmetric and irreflexive. The two transitions of the example above are not concurrent since they made two different choices at the subterm $(\bar{\alpha} : \mathsf{nil} + \beta : \mathsf{nil})$. Let us see another example of conflict, arising from communication: if q is the term $(\bar{\alpha} : \mathsf{nil} \parallel (\alpha : \mathsf{nil} \parallel \alpha : \mathsf{nil}))$ then the two transitions

$$q \xrightarrow{\tau,\, \delta(\gamma_{\bar{\alpha}}, \pi_0(\gamma_\alpha))} (\mathsf{nil} \parallel (\mathsf{nil} \parallel \alpha : \mathsf{nil})) \quad , \quad q \xrightarrow{\tau,\, \delta(\gamma_{\bar{\alpha}}, \pi_1(\gamma_\alpha))} (\mathsf{nil} \parallel (\alpha : \mathsf{nil} \parallel \mathsf{nil}))$$

are not concurrent, since they share the same "sub-transition" $\pi_0(\gamma_{\bar{\alpha}})$. The conflict relation will be formalized later.

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

$$i \neq j \;\Rightarrow\; \pi_i(\theta)\,/\pi_j(\theta') = \pi_i(\theta)$$

$$\theta \smile \theta' \;\Rightarrow\; \begin{cases} \pi_0(\theta)\,/\delta(\theta',\theta'') = \pi_0(\theta/\theta') \text{ and} \\ \delta(\theta',\theta'')/\pi_0(\theta) = \delta(\theta'/\theta,\theta'') \\ \pi_1(\theta)\,/\delta(\theta'',\theta') = \pi_1(\theta/\theta') \text{ and} \\ \delta(\theta'',\theta')/\pi_1(\theta) = \delta(\theta'',\theta'/\theta) \end{cases}$$

$$\theta \smile \theta' \;\Rightarrow\; \begin{cases} \pi_i(\theta)/\pi_i(\theta') = \pi_i(\theta/\theta') \\ \sigma_i(\theta)/\sigma_i(\theta') = \theta/\theta' \\ \rho_\alpha(\theta)/\rho_\alpha(\theta') = \rho_\alpha(\theta/\theta') \end{cases}$$

$$\theta_0 \smile \theta'_0 \text{ and } \theta_1 \smile \theta'_1 \;\Rightarrow\; \delta(\theta_0,\theta_1)/\delta(\theta'_0,\theta'_1) = \delta(\theta_0/\theta'_0,\theta_1/\theta'_1)$$

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

$$p \xrightarrow{\sigma_0(\pi_0(a))} (\mathsf{nil} \parallel b\colon \mathsf{nil}), \qquad p \xrightarrow{\sigma_0(\pi_1(b))} (a\colon \mathsf{nil} \parallel \mathsf{nil})$$

So the proof of the b-transition is $\sigma_0(\pi_1(b))$. On the other hand, once the a-transition has happened, the proof of the b-transition becomes $\pi_1(b) = \sigma_0(\pi_1(b))/\sigma_0(\pi_0(a))$, and we have:

$$(\mathsf{nil} \parallel b\colon \mathsf{nil}) \xrightarrow{(\pi_1(b))} (\mathsf{nil} \parallel \mathsf{nil})$$

The following result, also known as the parallel moves lemma, states a "conditional Church-Rosser property", namely that two transitions are confluent whenever they are concurrent. It 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 another concurrent one; it is always left unchanged, up to the resolution of choices.

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

This property is in fact much stronger than confluence: it says that a (proved) transition survives any concurrent one. Therefore we can adopt the standard terminology ([1,3,14,15]): the transition $t'_1 = p_0 \xrightarrow{b\,,\,\theta_1/\theta_0} \bar{p}$ (with the notations of the diamond lemma) is the *residual* of t_1 by t_0, denoted t_1/t_0 and similarly $t_0/t_1 = p_1 \xrightarrow{a\,,\,\theta_0/\theta_1} \bar{p}$ is the residual of t_0 by t_1. This is the basis of the equivalence by permutations.

Each CCS term p determines a set $\mathcal{T}^\infty(p)$ of finite or infinite sequences of proved transitions of the form

$$p \xrightarrow{a_1\,,\,\theta_1} p_1 \;\cdots\; p_{n-1} \xrightarrow{a_n\,,\,\theta_n} p_n \;\cdots$$

Equivalently we could have presented these as sequences of steps:

$$t_1 \cdots t_n \cdots \quad \text{where} \quad t_n = p_{n-1} \xrightarrow{a_n\,,\,\theta_n} p_n \quad (\text{and } p_0 = p)$$

The set of finite such sequences is denoted $\mathcal{T}(p)$, and we shall denote ss' the concatenation of $s \in \mathcal{T}(p)$ and $s' \in \mathcal{T}^\infty(q)$, which is only defined if s ends at q. We are now ready to define the permutation equivalence and the permutation preorder on $\mathcal{T}(p)$: intuitively two (finite) sequences of proved transitions are equivalent if they are the same up to permutations of concurrent steps; the preorder is just the prefix order up to permutations. We shall denote by $\ll$ the usual prefix order:

$$\forall s \in \mathcal{T}^{\infty}(p)\ \forall s' \in \mathcal{T}^{\infty}(p)\ s \ll s' \ \Leftrightarrow_{\text{def}}\ s = s' \text{ or } \exists s''\ ss'' = s'$$

DEFINITION (THE PERMUTATION EQUIVALENCE AND PREORDER). *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). The preorder $\lesssim$ is given by

$$s_0 \lesssim s_1 \Leftrightarrow_{\text{def}} \exists s\ s_0 \ll s\ \&\ s \simeq s_1$$

The typical example of equivalent sequences of transitions is (omitting the obvious proofs):

$$(a : p \parallel b : q) \xrightarrow{a} (p \parallel b : q) \xrightarrow{b} (p \parallel q) \simeq (a : p \parallel b : q) \xrightarrow{b} (a : p \parallel q) \xrightarrow{a} (p \parallel q)$$

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

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

The main idea of this note is that, if we only retain the actions and their possible permutations, we can represent the equivalence class of a sequence

$$s = p \xrightarrow{a_1\ ,\ \theta_1} \cdots \xrightarrow{a_n\ ,\ \theta_n} p'$$

as a *one step transition* $p \xrightarrow{P} p'$ where P is a *pomset* (partially ordered multiset [21]) of actions of A, – that is an isomorphism class of posets labelled in A. Such pomset transitions were introduced in [6] 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 $s = p \xrightarrow{a_1\ ,\ \theta_1} \cdots \xrightarrow{a_n\ ,\ \theta_n} p'$ determines a transition $p \xrightarrow{P} p'$, where $P = (E, l, \leq)$ is the labelled poset defined by

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

Note that P is defined up to isomorphism, since the events e_i are taken arbitrarily. A similar definition is given in [13] for Petri nets. For instance the equivalence class of

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

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:

$$(a\colon p \parallel b\colon c\colon q) \xrightarrow{\left\{\begin{matrix} a & b \\ & | \\ & c \end{matrix}\right\}} (p \parallel q)$$

As we shall see, we can interpret a term as an event structure, so that the pomsets of actions of the term are the configurations of this event structure.

The preorder $\lesssim$ is naturally extended to (possibly infinite) sequences of proved transitions $s \in \mathcal{T}^{\infty}(p)$:

$$s_0 \lesssim s_1 \Leftrightarrow_{\text{def}} \forall s \in \mathcal{T}(p)\ s \ll s_0 \Rightarrow \exists s' \in \mathcal{T}(p)\ s' \ll s_1\ \&\ s \lesssim s'$$

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 have

$$s_0 \simeq s_0'\ \&\ s_1 \simeq s_1' \Rightarrow s_0 \lesssim s_1 \Leftrightarrow s_0' \lesssim s_1'$$

Then the quotient $\mathcal{C}^{\infty}(p) = \mathcal{T}^{\infty}(p)/\simeq$, which is the set of *computations* of p, is a partially ordered set – the ordering on equivalence classes will be denoted $\sqsubseteq$.

In [3], 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

$$(a^{\omega} \parallel b^{\omega}) = (\mu x.a\colon x \parallel \mu x.b\colon x) \xrightarrow{a} (\mu x.a\colon x \parallel \mu x.b\colon x) \cdots \xrightarrow{a} \cdots$$

is not maximal since the proved transition

$$(a^{\omega} \parallel b^{\omega}) \xrightarrow{b,\ \pi_1(\gamma_b)} (a^{\omega} \parallel b^{\omega})$$

has a residual along the whole computation. On the other hand

$$(a+b)^{\omega} = \mu x.(a\colon x + b\colon x) \xrightarrow{a} \mu x.(a\colon x + b\colon x) \cdots \xrightarrow{a} \cdots$$

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. However 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 $r = \mu x.(\alpha\colon x + \beta\colon \mathsf{nil})$, 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" β is available. Then

$$(r \parallel \bar{\beta}\colon \mathsf{nil})\backslash\beta \xrightarrow{\alpha} (r \parallel \bar{\beta}\colon \mathsf{nil})\backslash\beta \cdots \xrightarrow{\alpha} \cdots$$

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 $q = \mu x.a\colon (\alpha\colon x + \beta\colon \mathsf{nil})$ then

$$(q \parallel \bar{\beta}\colon \mathsf{nil})\backslash\beta \xrightarrow{a} \xrightarrow{\alpha} (q \parallel \bar{\beta}\colon \mathsf{nil})\backslash\beta \;\cdots\; \xrightarrow{a} \xrightarrow{\alpha} \;\cdots$$

is a maximal computation.

4. Event Structures.

It is known that the posets of computations $\mathcal{C}^\infty(p)$ are complete partial orders (*cf.* [1,3]). But we can say much more: the main result of this note is that $\mathcal{C}^\infty(p)$ is the poset of configurations of an event structure. The following definition is a slight variation of Winskel's one [23] – the domain of configurations will be coherent, for inconsistency is given by a binary relation:

DEFINITION (LABELLED EVENT STRUCTURES). *An A-labelled event structure is a structure* $(E, \prec, \#, \ell)$ *where*

(i) E *is the (denumerable) set of events,*

(ii) $\prec\, \subseteq E \times E$ *is an irreflexive relation (i.e.* $e \prec e' \;\Rightarrow\; e \neq e'$*), the flow relation;*

(iii) $\# \subseteq E \times E$ *is a symmetric relation, the conflict relation;*

(iv) $\ell\colon E \to A$ *is the labelling function.*

Here too we denote $\sharp$ the reflexive closure of $\#$. Note that we do *not* assume that the conflict relation is irreflexive. We shall use *self-conflicting* (or inconsistent) events, which are the events $e \in E$ such that $e \# e$, in the interpretation of the restriction operator.

We shall always draw event structures up to isomorphism, that is omitting the name of events; moreover in the figures we shall represent $e \prec e'$ by $e \rightarrowtail e'$. For instance

$$\nabla' \qquad \begin{tikzcd} a \arrow[dr] \arrow[r, phantom, "\#"] & b \\ & c \end{tikzcd}$$

is a structure with three events e, e' and e'' respectively labelled a, b and c such that $e \prec e''$ and $e \# e'$.

In [6] we have introduced a notion of computation of an event structure, which is a labelled poset of events. This is just what Winskel calls a configuration, supplied with the causality ordering on events which holds in that configuration. To define the computations of an event structure $S = (E, \prec, \#, \ell)$ we need to introduce an *enabling* relation $F \vdash e$, for $e \in E$ and $F \subseteq E$. Let us denote by C_S the set of conflict-free subsets of E, that is:

$$F \in C_S \;\Leftrightarrow_{\text{def}}\; F \subseteq E \;\&\; \forall e, e' \in F\; \neg(e \# e')$$

We interpret $e' \prec e$ as meaning "e' is a condition for e". Then $F \vdash e$ means that F is a maximal set of non-conflicting conditions for e, that is:

$F \vdash e \quad \Leftrightarrow_{\text{def}} \quad e' \in F \;\Rightarrow\; e' \prec e$ and

$F \cup \{e\}$ is conflict-free: $F \cup \{e\} \in C_S$ and

F is closed under non-conflicting conditions for e:

$e' \prec e \;\&\; e' \notin F \;\Rightarrow\; \exists e'' \in F\; e' \# e''$

One can see that the structure $(E, C_S, \vdash)$ is what Winskel calls a *stable* event structure – where $\vdash$ is the minimal enabling relation (*cf.* [23]) – if we relax the hypothesis that the consistent sets are finite.

DEFINITION (CONFIGURATIONS). *Given an A-labelled event structure $S = (E, \prec, \#, \ell)$ a configuration of S is a set of events $F \subseteq E$ such that*

(i) F satisfies the finite causes property: for all $e \in F$ there exists $\{e_1, \ldots, e_n\} \subseteq F$ such that $e_n = e \;\&\; \forall i\, \exists G \subseteq \{e_1, \ldots, e_{i-1}\} \quad G \vdash e_i$

(ii) F is conflict-free (or consistent): $F \in C_S$

We denote by $\mathcal{F}^\infty(S)$ the set of configurations of the event structure S. This set, ordered by inclusion, has a nice property: the poset $(\mathcal{F}^\infty(S), \subseteq)$ is a finitary prime algebraic coherent poset, that is a coherent dI-domain, *cf.* [20,23] – in fact our event structures are just another concrete presentation of such domains. For F a configuration of S, we denote $\leq_F =_{\text{def}} (\prec \cap (F \times F))^*$, the reflexive and transitive closure of the restriction of $\prec$ to F. Then we have:

LEMMA. *For any configuration $F \in \mathcal{F}^\infty(S)$ of S the relation $\leq_F$ is an ordering such that*

$$e \leq_F e' \;\Leftrightarrow\; \forall G \in \mathcal{F}^\infty(S)\; G \subseteq F \;\&\; e' \in G \;\Rightarrow\; e \in G$$

Moreover $G \subseteq F$ is a configuration of S if and only if it is a left-closed subset of F:

$$\forall G \subseteq F.\; G \in \mathcal{F}^\infty(S) \;\Leftrightarrow\; e \in G \;\&\; e' \leq_F e \;\Rightarrow\; e' \in G$$

The proof is given in [23]. The ordering $\leq_F$ is the (local) *causality* relation in F. The restriction $\ell\lceil F$ of the labelling ℓ to F is denoted ℓ_F.

DEFINITION (COMPUTATIONS). *Given an A-labelled event structure $S = (E, \prec, \#, \ell)$ a computation of S is a labelled poset $(F, \leq_F, \ell_F)$ where F is a configuration of S.*

We shall denote by $\mathcal{G}^\infty(S)$ the set of computations of S. The previous result allows us to regard this set as ordered by inclusion, without ambiguity since for any configuration F there is only one ordering $\leq$ and only one labelling ℓ such that $(F, \leq, \ell)$ is a computation of S. As we have shown in [6], we can define a transition relation on event structures, where at each step the performed action is a computation – that is a labelled poset. For any computation $P = (F, \leq_F, \ell_F)$ of $S = (E, \prec, \#, \ell)$, let us define the *remainder* $S[P]$ of S after P by:

$$S[P] =_{\text{def}} (E', \prec', \#', \ell') \quad \text{where} \quad \begin{cases} E' = E - F \\ \prec' = \prec \cap (E' \times E') \\ e \;\#'\; e' \;\Leftrightarrow\; e \;\#\; e' \text{ or } e = e' \;\&\; \exists e'' \in F\; e \;\#\; e'' \\ \ell' = \ell\lceil E' \end{cases}$$

Then the transitions are

$$S \xrightarrow{P} S[P] \quad \text{for} \quad P \in \mathcal{G}^\infty(S)$$

There are two ways of interpreting a CCS term as an event structure: either we directly define from the syntactical materials a structure $\mathcal{S}(p)$ for each closed term p, or we define a construction on event structures for each CCS operator and interpret CCS by a morphism of algebra $\mathcal{I}^\infty$. We shall take both ways; the constructions we use are adapted from those of Winskel [22,23].

4.1 Let us first define $\mathcal{S}(p) = (\mathcal{E}(p), \prec, \#, \ell)$. The events are occurrences of possible future actions for a term, so we define the set $\mathcal{O}$ of *occurrences*. We just have to extend the syntax of proofs, allowing them to pass through a guard $a : p$, using the symbol $a : _{\{1\}}$, which will be denoted γ'_a. The syntax of occurrences is thus

$$o \;::=\; \gamma_a \mid \gamma'_a(o) \mid \pi_0(o) \mid \pi_1(o) \mid \delta(o, o') \mid \sigma_0(o) \mid \sigma_1(o) \mid \rho_\alpha(o)$$

For instance the occurrence of the action b in $a\colon b\colon \mathsf{nil}$ is $\gamma'_a(\gamma_b)$. We extend the labelling to occurrences, in an obvious way: $\ell(\gamma'_a(o)) = \ell(o)$. Let us now define the notions of conflict and flow on occurrences. The *conflict* relation $o \# o'$ is the least symmetric relation which satisfies the following, where we denote by $\sharp$ the reflexive closure of $\#$:

(B1) $i \neq j \Rightarrow \sigma_i(o) \# \sigma_j(o')$

(B2) $o \sharp o' \Rightarrow \begin{cases} \pi_0(o) \# \delta(o', o'') \\ \pi_1(o) \# \delta(o'', o') \end{cases}$

(B3) $o \# o' \Rightarrow \begin{cases} \pi_i(o) \# \pi_i(o') \\ \sigma_i(o) \# \sigma_i(o') \\ \rho_\alpha(o) \# \rho_\alpha(o') \\ \gamma'_a(o) \# \gamma'_a(o') \end{cases}$

(B4) $\mathsf{nm}(\ell(o)) = \alpha \Rightarrow \rho_\alpha(o) \# \rho_\alpha(o)$

(B5) $o_0 \sharp o'_0$ or $o_1 \sharp o'_1$ and $(o_0, o_1) \neq (o'_0, o'_1) \Rightarrow \delta(o_0, o_1) \# \delta(o'_0, o'_1)$
$o_0 \# o_0$ or $o_1 \# o_1 \Rightarrow \delta(o_0, o_1) \# \delta(o_0, o_1)$

In the structure $S(p)$, the flow represents possible immediate precedence. Quite obviously the relation $o \prec o'$ is brought out by the action construct $a\colon p$ – loosely speaking $\gamma_a \prec \gamma'_a(\theta)$. More precisely $\prec$ is the least relation on $\mathcal{O}$ compatible with the occurrence constructors that satisfies the following clauses:

(C1) $\gamma_a \prec \gamma'_a(\theta)$ where θ is any proof term

(C2) $o \prec o' \Rightarrow \begin{cases} \delta(o, o'') \prec \pi_0(o') \\ \delta(o'', o) \prec \pi_1(o') \end{cases}$ and $\begin{cases} \pi_0(o) \prec \delta(o', o'') \\ \pi_1(o) \prec \delta(o'', o') \end{cases}$

(C3) $o \prec o' \Rightarrow \begin{cases} \delta(o, o_1) \prec \delta(o', o'_1) \\ \delta(o_0, o) \prec \delta(o'_0, o') \end{cases}$

The relation $\prec$ is irreflexive; note on the other hand that it is not transitive: for instance if $o_0 \prec o'_0$ and $o_1 \prec o'_1$ then $\pi_0(o_0) \prec \delta(o'_0, o_1)$ and $\delta(o'_0, o_1) \prec \pi_1(o'_1)$ but we do not have $\pi_0(o_0) \prec \pi_1(o'_1)$. Let us see some examples: in the term $r = (\alpha\colon \alpha\colon \mathsf{nil} \parallel \bar{\alpha}\colon \mathsf{nil})$ we have

$$\delta(\gamma_\alpha, \gamma_{\bar{\alpha}}) \prec \delta(\gamma'_\alpha(\gamma_\alpha), \gamma_{\bar{\alpha}})$$
$$\delta(\gamma_\alpha, \gamma_{\bar{\alpha}}) \# \delta(\gamma'_\alpha(\gamma_\alpha), \gamma_{\bar{\alpha}})$$

This shows that $\#$ and $\prec$ are not necessarily disjoint. The following example shows that the transitive closure of $\prec$ is not disjoint from $\smile$ (extended to occurrences in the obvious way): in the term $q = (a\colon \alpha\colon \mathsf{nil} \parallel \bar{\alpha}\colon b\colon \mathsf{nil})$ we have

$$\pi_0(\gamma_a) \prec \delta(\gamma'_a(\gamma_\alpha), \gamma_{\bar{\alpha}}) \prec \pi_1(\gamma'_{\bar{\alpha}}(\gamma_b))$$
$$\pi_0(\gamma_a) \smile \pi_1(\gamma'_{\bar{\alpha}}(\gamma_b))$$

Note also that $\prec$ is not asymmetric; for instance in the term $(\alpha\colon \beta\colon \mathsf{nil} \parallel \bar{\beta}\colon \bar{\alpha}\colon \mathsf{nil})$ we have

$$\delta(\gamma_\alpha, \gamma'_{\bar{\beta}}(\gamma_{\bar{\alpha}})) \prec \delta(\gamma'_\alpha(\gamma_\beta), \gamma_{\bar{\beta}}) \prec \delta(\gamma_\alpha, \gamma'_{\bar{\beta}}(\gamma_{\bar{\alpha}}))$$

To define the structure $S(p)$ it just remains to define the set $\mathcal{E}(p)$ of events, which is a subset of the set $\mathcal{O}$ of occurrences – it should be understood that in $S(p) = (\mathcal{E}(p), \prec, \#, \ell)$ the flow and conflict relations are the restrictions to $\mathcal{E}(p)$ of the relations we just defined on occurrences. The set $\mathcal{E}(p)$ of events of p is defined inductively as follows:

(E1) $\gamma_a \in \mathcal{E}(a\colon p)$;
if $o \in \mathcal{E}(p)$ then $\gamma'_a(o) \in \mathcal{E}(a\colon p)$;

(E2) if $o \in \mathcal{E}(p_i)$ then $\pi_i(o) \in \mathcal{E}(p_0 \parallel p_1)$;
if $o \in \mathcal{E}(p_0)$ and $o' \in \mathcal{E}(p_1)$ and $\ell(o) = \overline{\ell(o')}$, then $\delta(o,o') \in \mathcal{E}(p_0 \parallel p_1)$;

(E3) if $o \in \mathcal{E}(p_i)$ then $\sigma_i(o) \in \mathcal{E}(p_0 + p_1)$;

(E4) if $o \in \mathcal{E}(p)$ then $\rho_\alpha(o) \in \mathcal{E}(p\backslash\alpha)$;

(E5) if $o \in \mathcal{E}(p[\mu x.p/x])$ then $o \in \mathcal{E}(\mu x.p)$.

For instance if $r = (a\colon \alpha\colon \mathsf{nil} \parallel \bar{\alpha}\colon b\colon \mathsf{nil})\backslash\alpha$ then $\mathcal{S}(r)$ may be drawn

$$\begin{tikzcd}
a \arrow[dd] \arrow[dr] & & \bar{\alpha} \arrow[dd] \\
& \tau \arrow[ur, dotted, dash, "\#"'] \arrow[dr] \arrow[dl, dotted, dash, "\#"] & \\
\alpha & & b
\end{tikzcd}$$

where the dotted circles around α and $\bar{\alpha}$ indicate that the corresponding events are self-conflicting. In this structure the enabling consists of (identifying the events with their labels) $\{a\} \vdash \tau$ and $\{\tau\} \vdash b$. Clearly α and $\bar{\alpha}$ cannot occur in a configuration since they are inconsistent.

4.2 On the other hand, the constructions on event structures corresponding to CCS operators are as follows:

(i) nil is the empty event structure;

(ii) if $S = (E, \prec, \#, \ell)$ then $a\colon S = (\{\varepsilon\} \cup E, \prec', \#, \ell')$ where

- $\varepsilon \notin E$
- $e \prec' e' \;\Leftrightarrow\; e \prec e'$ or $(e = \varepsilon \;\&\; \emptyset \vdash e')$
- $\ell'(\varepsilon) = a$ and $\ell'(e) = \ell(e)$ for $e \in E$;

(iii) if $S_i = (E_i, \prec_i, \#_i, \ell_i)$ for $i = 0,1$ then $S_0 \parallel S_1 = (E, \prec, \#, \ell)$ where

- $E = (E_0 \times \{*\}) \cup (\{*\} \times E_1) \cup \{(e_0, e_1) \mid e_i \in E_i \;\&\; \ell(e_0) = \overline{\ell(e_1)}\}$

where $* \notin E_0 \cup E_1$

- $e \prec e' \;\Leftrightarrow\; e = (x,y) \;\&\; e' = (x',y')$ and $x \prec_0 x'$ or $y \prec_1 y'$

where, by convention, $* \not\prec_i z$ and $z \not\prec_i *$ for any z.

- $e \# e' \;\Leftrightarrow\; \begin{cases} e = (x,y) \;\&\; e' = (x',y') \;\&\; e \neq e' \text{ and } x \sharp_0 x' \text{ or } y \sharp_1 y', \text{ or} \\ e = (x,y) = e' \;\&\; x \#_0 x \text{ or } y \#_1 y \end{cases}$

where, by convention, $\neg(* \sharp_i z)$ for any z.

- $\ell(e,*) = \ell_0(e)$, $\ell(*,e) = \ell_1(e)$ and $\ell(e_0,e_1) = \tau$

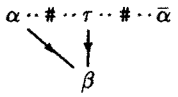
(iv) if $S_i = (E_i, \prec_i, \#_i, \ell_i)$ for $i = 0,1$ then $S_0 + S_1 = (E, \prec, \#, \ell)$ where

- $E = \{(i, e_i) \mid e_i \in E_i\}$
- $e \prec e' \;\Leftrightarrow\; e = (i, e_i) \;\&\; e' = (i, e'_i) \;\&\; e_i \prec_i e'_i$
- $e \# e' \;\Leftrightarrow\; \begin{cases} e = (i, e_i) \;\&\; e' = (i, e'_i) \;\&\; e_i \#_i e'_i \quad \text{or} \\ e = (i, e_i) \;\&\; e' = (j, e'_j) \;\&\; i \neq j \end{cases}$
- $\ell(i,e) = \ell_i(e)$.

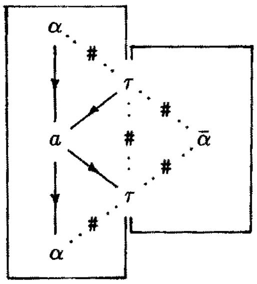
(v) if $S = (E, \prec, \#, \ell)$ then $S\backslash\alpha = (E, \prec, \#', \ell)$ where

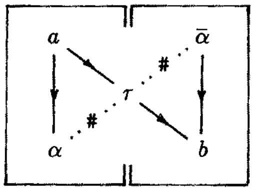
- $e \#' e' \;\Leftrightarrow\; e \# e'$ or $e = e' \;\&\; \mathsf{nm}(\ell(e)) = \alpha$

The interpretation $\mathcal{I}^\infty(p)$ is given by the unique continuous morphism $\mathcal{I}^\infty$ from the free algebra of infinite trees to the algebra of labelled event structures ($\mathcal{I}^\infty(\mu x.p)$ is defined by a standard fixpoint construction, *cf.* [22,6]). Let us see some examples. The structure $\mathcal{I}^\infty((\alpha:\beta:\mathsf{nil} \,||\, \bar{\alpha}:\mathsf{nil}))$ may be drawn:

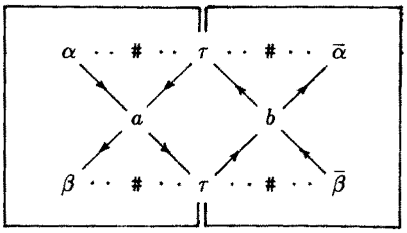


This example shows that $\mathcal{I}^\infty(p)$ may contain a substructure ∇' (*cf.* [6]). The interpretations of $(\alpha:a:\alpha:\mathsf{nil} \,||\, \bar{\alpha}:\mathsf{nil})$ and $(a:\alpha:\mathsf{nil} \,||\, \bar{\alpha}:b:\mathsf{nil})$ may be drawn respectively





The second structure contains a substructure N and a substructure ∇, *cf.* [6]. An example, suggested by M. Nielsen, shows that $\prec^*$ is not an ordering: if we interpret $(\alpha:a:\beta:\mathsf{nil} || \bar{\beta}:b:\bar{\alpha}:\mathsf{nil})$ we get



Note that if $r = (\alpha:a:\beta:\mathsf{nil} \,||\, \bar{\beta}:b:\bar{\alpha}:\mathsf{nil})\backslash\alpha\backslash\beta$ then in the structure $\mathcal{I}^\infty(r)$ the enabling is such that $F \vdash e \;\Rightarrow\; F \neq \emptyset$, so that it has no configuration. The same is true for the interpretation of $(\alpha:\beta:\mathsf{nil} \,||\, \bar{\beta}:\bar{\alpha}:\mathsf{nil})\backslash\alpha\backslash\beta$, where the enabling relation is empty. One can also see that in the interpretation of $((\alpha:\beta:\mathsf{nil} \,||\, \bar{\alpha}:\mathsf{nil}) \,||\, \alpha:\bar{\beta}:\mathsf{nil})\backslash\alpha,\beta$, there is no enabling for the β communication.

THEOREM 1. *For all closed* CCS *terms p the poset* $(\mathcal{C}^\infty(p),\sqsubseteq)$ *of computations of p is isomorphic to the poset* $(\mathcal{G}^\infty(\mathcal{S}(p)),\subseteq)$ *of computations of the labelled event structure* $\mathcal{S}(p)$. *Moreover the structure* $\mathcal{S}(p)$ *is isomorphic to* $\mathcal{I}^\infty(p)$.

We could prove moreover that there is an exact correspondence (as in [6]) between equivalence classes of finite computations and the pomset transitions on event structures, that is between the pomset transitions $p \xrightarrow{P} p'$ associated with the sequences of proved transitions

$$p \xrightarrow{a_1\,,\,\theta_1} p_1 \;\cdots\; p_{n-1} \xrightarrow{a_n\,,\,\theta_n} p'$$

up to permutations, and the transitions

$$\mathcal{S}(p) \xrightarrow{P} \mathcal{S}(p)[P]$$

Therefore we can say that we have given an operational meaning to the event structure semantics for CCS. This provides also for a "truly concurrent" operational semantics for CCS, which generalizes the one we have given in [6] for a restricted language. This "poset semantics" is similar to the one given, via concurrent histories, by Degano, De Nicola and Montanari in [11].

Its seems fair to say that, due to communication, the mathematical theory for CCS is not especially aesthetic. Communication may be achieved by other means, as proposed in [5,7] – but there, no mathematical theory is given. If we abandon communication, we get a more satisfactory theory: the syntax for concurrent – but non communicating – systems is

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

where $a \in A$, and the set A of actions does not need to have any particular structure. The operational semantics is the same, but obviously without the communication rule. Then the "permutations of transitions semantics" is exactly the semantics we have given in [6] for this calculus (if sequential composition $p\,;q$ of [6] is restricted to $p \in A$), and the corresponding constructions on event structures are much more pleasant than the ones presented here. However, as is noted in [5] and [7], the resulting calculus is too asynchronous. The next section presents a first attempt to remedy this deficiency.

5. Synchrony and Asynchrony.

In [18], Milner proposed the synchronous calculus SCCS, based on a structured set of actions – namely a commutative semigroup $(A\,,\cdot)$. The main new construction was the synchronous product $(p \times q)$. However, Milner noted in [19] that "*it was not made sufficiently clear that* SCCS *provides an asynchronous process model which stands in its own right*". To some extent, the calculus MEIJE of [4] – which is a variant of SCCS – does not suffer this defect. It was shown in [4] that, among several equivalent formulations, this calculus can be built upon two basic kinds of parallelism: interleaving $(p \mid q)$ and synchronous product $(p \times q)$. In this formulation of MEIJE, the sum is not primitive, but may be derived using communication. From now on we shall deal with this calculus, parameterized on a *free commutative semigroup* $(A^{\circ}\,,\cdot)$ of actions, generated by the set A of atomic actions. We let aside communication, as well as the restriction operator, which could be handled as in the previous sections, with respect to a commutative group of actions. Since we ignore communication, we need to regard sum as a primitive operator here. We shall give a so-called "non-interleaving" semantics for $(p \mid q)$, thus it is better to rename the two parallel operators *asynchronous* and *synchronous parallelism* respectively. Moreover we shall denote the synchronous parallelism by $(p \otimes q)$, in order to avoid confusion with the cartesian product, and we denote asynchronous parallelism by $(p \parallel q)$, as done so far in this paper. Therefore the syntax of our calculus for synchrony and asynchrony – let us call it **C** – is

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

where a belongs to the set A of atomic actions. The syntax for proofs of transitions is now

$$\theta \ ::= \ \gamma_a \mid \pi_0(\theta) \mid \pi_1(\theta) \mid \sigma_0(\theta) \mid \sigma_1(\theta) \mid \kappa(\theta, \theta')$$

and the inference rules are

action	$\vdash\ a : p \xrightarrow{a\ ,\ \gamma_a} p$
asynchronous parallelism 1	$p \xrightarrow{a\ ,\ \theta} p' \ \vdash\ (p \parallel q) \xrightarrow{a\ ,\ \pi_0(\theta)} (p' \parallel q)$
asynchronous parallelism 2	$q \xrightarrow{b\ ,\ \theta'} q' \ \vdash\ (p \parallel q) \xrightarrow{b\ ,\ \pi_1(\theta')} (p \parallel q')$
sum 1	$p \xrightarrow{a\ ,\ \theta} p' \ \vdash\ (p + q) \xrightarrow{a\ ,\ \sigma_0(\theta)} p'$
sum 2	$q \xrightarrow{b\ ,\ \theta'} q' \ \vdash\ (p + q) \xrightarrow{b\ ,\ \sigma_1(\theta')} q'$
synchronous parallelism	$p \xrightarrow{a\ ,\ \theta} p'\ ,\ q \xrightarrow{b\ ,\ \theta'} q' \ \vdash\ (p \otimes q) \xrightarrow{a \cdot b\ ,\ \kappa(\theta,\theta')} (p' \otimes q')$
fixpoint	$p[\mu x.p/x] \xrightarrow{a\ ,\ \theta} p' \ \vdash\ \mu x.p \xrightarrow{a\ ,\ \theta} p'$

We shall not repeat the definition of the equivalence by permutations and the associated preorder: it suffices to redefine the notion of concurrency – the diamond lemma will be the same as before. In **C**, the source of concurrency is asynchronous parallelism, therefore $\smile$ is the least relation compatible with the proof constructors such that

$$\forall\theta\,\forall\theta'\ i \neq j\ \Rightarrow\ \pi_i(\theta) \smile \pi_j(\theta')$$

We still denote by $\simeq$ the permutation equivalence on sequences of (proved) transitions, and by $(\mathcal{C}^\infty(p), \sqsubseteq)$ the ordered space of computations of p. We could define an event structure $\mathcal{S}(p)$ for each term p of **C**, as in the previous section, but it is more instructive to give the constructions on event structures which allow us to define $\mathcal{I}^\infty(p)$ for each closed term p of **C**.

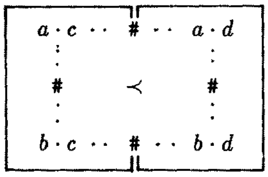
As noted by Girard ([12]) there are (at least) two natural ways to combine two sets of events E_0 and E_1: either we *juxtapose* these sets, by a disjoint union $E_0 \uplus E_1$, or we form their *cartesian product* $E_0 \times E_1$. In Girard's terminology, the first kind of construction defines *additive* operators whereas the second one defines *multiplicative* operators (as a matter of fact, Girard proposes his constructions for what he calls *coherent spaces*, which are event structures without causality). For what regards the additive operators on event structures, there are three natural constructions, according to which relation, among causality, concurrency and conflict, is set between the events of E_0 and E_1. These are the constructions we used in [6], where our calculus only stands on the additive (asynchronous) side. On the other hand, one of the additive operators, asynchronous parallelism, is missing in SCCS. The construction $a\colon S$ is a special case of sequential composition, namely *lifting*; it is the same as for CCS, and so is $S_0 + S_1$. For what regards the construction $S_0 \parallel S_1 = (E, \prec, \#, \ell)$, it is given as follows, assuming $S_i = (E_i, \prec_i, \#_i, \ell_i)$ for $i \in \{0, 1\}$:

- $E = E_0 \uplus E_1 = \{0\} \times E_0 \cup \{1\} \times E_1$
- $(i, e) \prec (j, e') \ \Leftrightarrow\ i = j \ \&\ e \prec_i e'$
- $(i, e)\ \#\ (j, e') \ \Leftrightarrow\ i = j \ \&\ e\ \#_i\ e'$
- $\ell(i, e) = \ell_i(e)$

The asynchronous parallel composition and sum are analogous to Girard's additive conjunction and additive disjunction respectively. On the other side, for what regards concurrency in the multiplicatives, Girard proposes three constructions: conjunction, disjunction and implication (*times*, *par* and *entail*, or linear implication, in Girard's terminology). We cannot see however what could be a diamond lemma if we allowed multiplicative disjunction or implication as process constructors. Moreover, our interpretation of the synchronous product, though a multiplicative one, is slightly different from Girard's multiplicative conjunction, since here the same event cannot be used twice in a computation. We define thus $S_0 \otimes S_1 = (E_0 \times E_1, \prec, \#, \ell)$ where

- $(e_0, e_1) \prec (e'_0, e'_1) \;\Leftrightarrow\; e_0 \prec_0 e'_0$ or $e_1 \prec_1 e'_1$
- $(e_0, e_1) \,\#\, (e'_0, e'_1) \;\Leftrightarrow\; \begin{cases} (e_0, e_1) \neq (e'_0, e'_1) \text{ \& } e_0 \sharp_0 e'_0 \text{ or } e_1 \sharp_1 e'_1, \text{ or} \\ (e_0, e_1) = (e'_0, e'_1) \text{ and } e_0 \#_0 e_0 \text{ or } e_1 \#_1 e_1 \end{cases}$
- $\ell(e_0, e_1) = \ell_0(e_0) \cdot \ell_1(e_1)$

The morphism $\mathcal{I}^\infty$ from **C** to event structures is now fully determined. For instance the interpretation of $(a{:}\, b{:}\, \mathsf{nil}) \otimes c{:}\, \mathsf{nil}$ is a structure with two events e and e' labelled $a \cdot c$ and $b \cdot c$ such that $e \prec e'$ and $e \,\#\, e'$, therefore e' cannot occur in a computation of this structure. Similarly the interpretation of $(a{:}\, \mathsf{nil} \parallel b{:}\, \mathsf{nil}) \otimes c{:}\, \mathsf{nil}$ consists of two events such that $e \,\#\, e'$, therefore the only two non-empty computations are $a \cdot c$ and $b \cdot c$. In both these examples, conflict arises from the *sharing* of a "sub-action". As another example, we can draw the interpretation of the term $(a{:}\, \mathsf{nil} \parallel b{:}\, \mathsf{nil}) \otimes (c{:}\, d{:}\, \mathsf{nil})$ as



The result we had for CCS, relating the event structure semantics to permutations of transitions, holds also for **C**:

THEOREM 2. *For any closed term p of **C** the poset $(\mathcal{C}^\infty(p), \sqsubseteq)$ of computations of p is isomorphic to the poset $(\mathcal{F}^\infty(\mathcal{I}^\infty(p)), \subseteq)$ of computations of the labelled event structure $\mathcal{I}^\infty(p)$.*

We have announced that this result remedies the lack of synchrony one finds in CCS without communication. Let us state this point more formally: in [6] we gave a *direct* operational meaning to the event structure semantics for the asynchronous part of the language **C** (without $\otimes$). This was achieved by giving rules to prove transitions $p \xrightarrow{u} p'$ where u is a pomset, but it was not clear how to describe such transitions for $p = (q \otimes r)$. The equivalence by permutations brings an answer to this question. Indeed, the problem is the same as for CCS communication: it lies in the fact that parallel composition (either CCS composition with communication, or SCCS synchronous product) does not preserve the computations, since it may introduce conflicts.

REFERENCES

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

[2] 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.

[3] 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.

[4] G. BOUDOL, *Notes on Algebraic Calculi of Processes*, in Logics and Models of Concurrent Systems (K. Apt, Ed.) NATO ASI Series F13 (1985) 261-303.

[5] G. BOUDOL, *Communication is an Abstraction*, Actes du Second Colloque C^3 (1987) 45-63, and INRIA Res. Rep. 636.

[6] 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.

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

[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, *Partial Ordering Derivations for CCS*, FCT 85, Lecture Notes in Comput. Sci. 199 (1985) 520-533.

[12] J.-Y. GIRARD, *Linear Logic*, Theoretical Comput. Sci. 50 (1987) 1-102.

[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] R. MILNER, *Calculi for Synchrony and Asynchrony*, Theoret. Comput. Sci. 25 (1983) 267-310.

[19] R. MILNER, *Process Constructors and Interpretations*, IFIP 86 (1986) 507-514.

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

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

[22] G. WINSKEL, *Event Structure Semantics for CCS and Related Languages*, Daimi PB-159, Aarhus University (1983) s.a. 9th ICALP, Lecture Notes in Comput. Sci. 140 (1982) 561-576.

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