Permutation of transitions: an event structure semantics for CCS

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° 798

PERMUTATION OF TRANSITIONS : AN EVENT STRUCTURE SEMANTICS FOR CCS

Gérard BOUDOL
Ilaria CASTELLANI

FEVRIER 1988

Permutation of Transitions:
An Event Structure Semantics for CCS

Permutations de Transitions:
Une Sémantique par Structures d'Evénements pour CCS

(Note)

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

Abstract.

We apply Berry & Lévy's notion of equivalence by permutations to CCS, thus obtaining a pomset transition semantics for this calculus. We show that this provides an operational counterpart for the event structure semantics given by Winskel for CCS.

Résumé.

Nous transposons à CCS la notion d'équivalence de calculs par permutations, due à Berry & Lévy. Nous obtenons ainsi un domaine ordonné de calculs, qui n'est autre que l'espace des configurations d'une structure d'événements. Ceci donne une signification opérationnelle à la sémantique donnée par Winskel pour CCS, avec une notion de transitions où l'action exécutée est un ordre partiel.

Permutation of Transitions: An Event Structure Semantics for CCS (Note)

G. Boudol & I. Castellani
INRIA Sophia-Antipolis
06560-VALBONNE FRANCE

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

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* of sequences for (elementary) computations.

This equivalence was first elaborated by Lévy in his thesis (*cf.* [9]) 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 [8], and to non-deterministic ones by Boudol in [2]. 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 [11] an equivalence that "*abstracts away from the ordering of concurrent firings of transitions*" (this is also used by van Glabbeek and Vaandrager in [7]). 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 calculus CCS [10], 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. As a matter of fact, our "operational" event structure semantics is the same as the "denotational" one given by Winskel in [12] (see also [13]). With the exception of communication, this also corresponds exactly to our semantics for "true concurrency" in [3].

2. Pure CCS: terms and transitions.

As in [10], 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 the name $\mathsf{nm}(\bar{\alpha})$ of $\bar{\alpha}$ is α, and so on. 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.

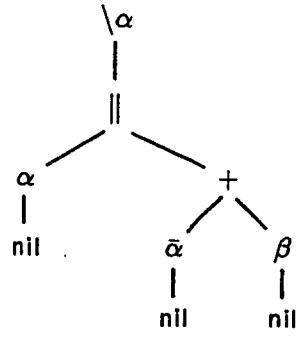
The *action* construct of CCS will be denoted by $a:p$, while the parallel composition is $(p \parallel q)$. We shall not use the relabelling operator, although it would not introduce any difficulty. The syntax of (pure) CCS terms is given by the following grammar:

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

We shall use p, q, r,... to range over terms; finite terms – built without fixpoint $\mu x.p$ – should be viewed as *finite trees*, with parallel composition and sum as binary operators, 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

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

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 $[q/x]p$. Terms involving fixpoint define *infinite trees*, obtained by unfolding $\mu x.p$ into $[\mu x.p/x]p$ 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 [10]). 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 $a:p$. But the action can also be a communication τ, in which case this indication is a path to a pair of complementary subterms $\lambda:p$ and $\bar{\lambda}:p'$. 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	$p \xrightarrow{a\,,\,\theta} p' \ \vdash \ (p \,\|\, q) \xrightarrow{a\,,\,\pi_0(\theta)} (p' \,\|\, q)$
parallel composition 2	$q \xrightarrow{b\,,\,\theta'} q' \ \vdash \ (p \,\|\, q) \xrightarrow{b\,,\,\pi_1(\theta')} (p \,\|\, q')$
communication	$p \xrightarrow{\lambda\,,\,\theta} p' \ , \ q \xrightarrow{\bar{\lambda}\,,\,\theta'} q' \ \vdash \ (p \,\|\, q) \xrightarrow{\tau\,,\,\delta(\theta,\theta')} (p' \,\|\, q')$
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	$p \xrightarrow{a\,,\,\theta} p' \ , \ \mathrm{nm}(a) \neq \alpha \ \vdash \ (p\backslash\alpha) \xrightarrow{a\,,\,\rho_\alpha(\theta)} (p'\backslash\alpha)$
fixpoint	$[\mu x.p/x]p \xrightarrow{a\,,\,\theta} p' \ \vdash \ \mu x.p \xrightarrow{a\,,\,\theta} p'$

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 [4,5] (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 should 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 of 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. We first proceed to define a notion of concurrency between proof terms, in notation $\theta \frown \theta'$. This relation is the least symmetric one which satisfies the following clauses:

(A1) $\quad i \neq j \;\Rightarrow\; \pi_i(\theta) \frown \pi_j(\theta')$

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

(A3) $\quad \theta \frown \theta' \;\Rightarrow\; \begin{cases} \pi_i(\theta) \frown \pi_i(\theta') \\ \sigma_i(\theta) \frown \sigma_i(\theta') \\ \rho_\alpha(\theta) \frown \rho_\alpha(\theta') \end{cases}$

(A4) $\quad \theta_0 \frown \theta_0' \;\&\; \theta_1 \frown \theta_1' \;\Rightarrow\; \delta(\theta_0, \theta_1) \frown \delta(\theta_0', \theta_1')$

For instance, considering the term $((\alpha : \mathsf{nil} \parallel \beta : \mathsf{nil}) \parallel \bar{\alpha} : \mathsf{nil})$, we have $\pi_0(\pi_1(\gamma_\beta)) \frown \delta(\pi_0(\gamma_\alpha), \gamma_{\bar{\alpha}})$. We can subsume clauses (A3) and (A4) by saying simply that $\frown$ is the least symmetric relation *compatible with the proof constructors* that satisfies (A1) and (A2). Note that $\theta \frown \theta' \Rightarrow \theta \neq \theta'$.

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* $t_0 \frown t_1$, *if and only if* $\theta_0 \frown \theta_1$.

Note that the concurrency relation between transitions is symmetric and irreflexive. For instance 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" $\pi_0(\gamma_\alpha)$. The *conflict* relation $\theta \# \theta'$ is the least symmetric relation which satisfies the following, where we denote by $\sharp$ the reflexive closure of #:

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

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

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

(B4) $\theta_0 \sharp \theta'_0$ or $\theta_1 \sharp \theta'_1$ (and $(\theta_0, \theta_1) \neq (\theta'_0, \theta'_1)$) $\Rightarrow \delta(\theta_0, \theta_1) \# \delta(\theta'_0, \theta'_1)$

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 modified by another one; it can only be eliminated (by making a choice in a sum), consumed (for instance by a communication), or left unchanged (when the two proofs are concurrent).

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, $t_0 \frown t_1$, then there exists a unique term $\bar{p}$ such that $p_0 \xrightarrow{b\,,\,\theta_1} \bar{p}$ and $p_1 \xrightarrow{a\,,\,\theta_0} \bar{p}$.*

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 ([1,2,8,9]): the transition $t'_1 = p_0 \xrightarrow{b\,,\,\theta_1} \bar{p}$ (with the notations of the diamond lemma) is the *residual* of t_1 by t_0, denoted t_1/t_0 and similarly $t_0/t_1 = p_1 \xrightarrow{a\,,\,\theta_0} \bar{p}$ 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 this makes sense). The preorder $\lesssim$ is given by $s_0 \lesssim s_1 \Leftrightarrow_{\text{def}} \exists s\ s_0 s \simeq s_1$.

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* of actions – that is a poset labelled in A. Such pomset transitions were introduced in [3] to deal with the semantics of "true concurrency". Let us formalize this idea: it should be clear that if $s' \simeq s$ then

$$s' = p \xrightarrow{a_{\eta(1)}\,,\,\theta_{\eta(1)}} \cdots \xrightarrow{a_{\eta(n)}\,,\,\theta_{\eta(n)}} p'$$

for some permutation η of $\{1,\ldots,n\}$. Let us denote this fact by $s' \simeq_\eta s$. Then the equivalence class of s determines a transition $p \xrightarrow{P} p'$ where $P = (X, \leq, \ell)$ is a labelled poset, defined by

$$X = \{(\theta_1, 1), \ldots, (\theta_n, n)\} \quad \text{and} \quad (\theta_i, i) \leq (\theta_j, j) \;\Leftrightarrow\; \forall s'.\; s' \simeq_\eta s \;\Rightarrow\; \eta(i) \leq \eta(j)$$

(a similar definition is given in [7] 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 action is a pomset consisting of a, b and c where b necessarily precedes c, and a is incomparable with b and c, that is

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

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 [2], the maximal computations 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

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

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

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

is a maximal computation. Not too surprisingly, our proof terms are similar to the labels used by Costa and Stirling in [6] 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 $r = \mu x.(\alpha : x + \beta : \mathsf{nil})$, 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" β is available. Then

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

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

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

is a maximal computation.

4. Event Structures.

It is known that the posets of computations $\mathcal{C}^\infty(p)$ are complete partial orders (*cf.* [1,2]). 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 [13] – 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 $(E, \prec, \#, \lambda)$ where*

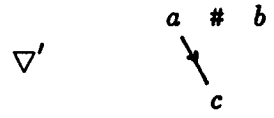
(i) E is the (denumerable) set of events,

(ii) $\prec \subseteq E \times E$ is the enabling relation;

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

(iv) $\lambda : E \to A$ is the labelling function.

Here too we denote $\sharp$ the reflexive closure of #. 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



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 [3] we have called computation of an event structure what Winskel calls configurations (configuration is a better word when dealing with data types), with some minor variations. To define the computations of an event structure $S = (E, \prec, \#, \lambda)$ we need to introduce an entailment 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:

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

Then $F \vdash e$ means that F is a maximal set of non-conflicting "immediate causes" of e, that is:
$F \vdash e \Leftrightarrow_{\text{def}}$ $e' \in F \Rightarrow e' \prec e \,\&\, e' \neq e$ and
$F \cup \{e\}$ is conflict-free: $F \cup \{e\} \in C_S$ and
F is closed under non-conflicting causes of e:
$e' \prec e \,\&\, e' \neq e \,\&\, e' \notin F \Rightarrow \exists e'' \in F \cup \{e\}\ e' \# e''$

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.* [13]) – if we relax the hypothesis that the consistent sets are finite.

DEFINITION (CONFIGURATIONS). *Given an A-labelled event structure $S = (E, \prec, \#, \lambda)$ 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 $e_n = e \,\&\, \forall i\ \exists G \subseteq \{e_1, \ldots, e_{i-1}\}\ \ G \vdash e_i$
(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.* [11,13] – 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 [13]. The ordering $\leq_F$ is the (local) *causality* relation in F. The restriction $\lambda\lceil F$ of the labelling λ to F is denoted λ_F.

DEFINITION (COMPUTATIONS). *Given an A-labelled event structure $S = (E, \prec, \#, \lambda)$ a computation of S is a labelled poset $(F, \leq_F, \lambda_F)$ 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 [3], 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 $F \subseteq E$ we define $F^\circ \subseteq E$ by $F^\circ = E - F^\circ$ where

$$e \in F^\circ \ \Leftrightarrow_{\text{def}}\ e \in F \text{ or } \forall G \in \mathcal{F}^\infty(S).\ e \in G \ \Rightarrow\ \exists e' \in F\, \exists e'' \in G.\ e' \# e''$$

If $P = (F, \leq_F, \lambda_F)$ is a computation of S, the *remainder* $S[P]$ of S by P is

$$S[P] =_{\text{def}} (F^\circ, \prec \cap (F^\circ \times F^\circ), \# \cap (F^\circ \times F^\circ), \lambda \lceil F^\circ)$$

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 $S(p)$ 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 I^∞. We shall take both ways; the constructions we use are adapted from those of Winskel [12,13].

4.1 Let us first define $S(p) = (\mathcal{E}(p), \prec, \#, \ell)$. The events are occurrences of possible future actions for a term, so we define the set 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 $a:b:\mathsf{nil}$ is $\gamma'_a(\gamma_b)$. Obviously not all occurrences may be legal events for a given term p. First of all they must be occurrences in the term p – moreover the restriction operator will preclude some of them. The set $O(p)$ of occurrences in p, as well as their labels $\ell(o)$, are defined inductively as follows:

(O1) $\gamma_a \in O(a:p)$ and $\ell(\gamma_a) = a$;
if $o \in O(p)$ then $\gamma'_a(o) \in O(a:p)$ and $\ell(\gamma'_a(o)) = \ell(o)$;

(O2) if $o \in O(p_i)$ then $\pi_i(o) \in O((p_0 \parallel p_1))$, and $\ell(\pi_i(o)) = \ell(o)$ for $i = 0$ or 1;
if $o \in O(p)$ and $o' \in O(p')$ and $\ell(o) = \overline{\ell(o')}$, then $\delta(o, o') \in O((p \parallel p'))$ and $\ell(\delta(o, o')) = \tau$;

(O3) if $o \in O(p_i)$ then $\sigma_i(o) \in O((p_0 + p_1))$, and $\ell(\sigma_i(o)) = \ell(o)$ for $i = 0$ or 1;

(O4) if $o \in O(p)$ and $\mathrm{nm}(\ell(o)) \neq \alpha$ then $\rho_\alpha(o) \in O((p \backslash \alpha))$ and $\ell(\rho_\alpha(o)) = \ell(o)$;

(O5) if $o \in O([\mu x.p/x]p)$ then $o \in O(\mu x.p)$.

Let us now define the notions of conflict and enabling between occurrences. We first extend the conflict relation to occurrences by adding to the clauses B1-B4 the following

(B5) $o \# o' \Rightarrow \gamma'_a(o) \# \gamma'_a(o')$

which states that # is still compatible with the occurrence constructors. In the structure $S(p)$, enabling will mean (immediate) *causality*. Quite obviously this relation $o \prec o'$ is brought out by the action construct $a:p$ – loosely speaking $\gamma_a \prec \gamma'_a(\theta)$. More precisely $\prec$ is the least relation on 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 $\begin{cases} \pi_0(o) \prec \delta(o', o'') \\ \pi_1(o) \prec \delta(o'', o') \end{cases}$

Note that this relation 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 $\pi_0(o_0) \prec \pi_1(o'_1)$. Note also that $\prec$ is asymmetric. Let us see some examples: in the term $r = (\alpha:c:\alpha:\mathsf{nil} \parallel \bar{\alpha}:\mathsf{nil})$ we have

$$\delta(\gamma_\alpha, \gamma_{\bar{\alpha}}) \prec \pi_0(\gamma'_\alpha(\gamma_c)) \prec \delta(\gamma'_\alpha(\gamma'_c(\gamma_\alpha)), \gamma_{\bar{\alpha}})$$
$$\delta(\gamma_\alpha, \gamma_{\bar{\alpha}}) \# \delta(\gamma'_\alpha(\gamma'_c(\gamma_\alpha)), \gamma_{\bar{\alpha}})$$

This shows that # and the transitive closure of $\prec$ are not necessarily disjoint. Similarly, in the term $q = (c:\alpha:\mathsf{nil} \parallel \bar{\alpha}:b:\mathsf{nil})$ we have

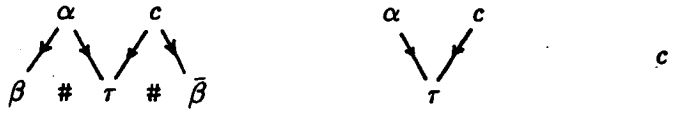
$$\pi_0(\gamma_c) \prec \delta(\gamma'_c(\gamma_\alpha), \gamma_{\bar{\alpha}}) \prec \pi_1(\gamma'_{\bar{\alpha}}(\gamma_b))$$
$$\pi_0(\gamma_c) \smile \pi_1(\gamma'_{\bar{\alpha}}(\gamma_b))$$

(with an obvious extension of concurrency to occurrences). To define the structure $S(p)$ it just remains to define the set $\mathcal{E}(p)$ of events. This set is the subset of $\mathcal{O}(p)$ defined inductively as follows:

(i) if $p \xrightarrow{a,\theta} p'$ then $\theta \in \mathcal{E}(p)$;

(ii) if $o \in \mathcal{O}(p)$ and there exists a finite subset F of $\mathcal{E}(p)$ such that $F \vdash o$ (where $\vdash$ is defined as above, with respect to $\prec$) then $o \in \mathcal{E}(p)$.

For instance if $r = (\alpha : \beta : \mathsf{nil} \,\|\, c : \bar{\beta} : \mathsf{nil})$ then $S(r)$, $S(r\backslash\beta)$ and $S(r\backslash\beta\backslash\alpha)$ may be drawn respectively (omitting concurrency)



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 $S = (E, \prec, \#, \lambda)$ then $a : S = (\{\varepsilon\} \cup E, \prec', \#, \lambda')$ where

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

(iii) if $S_i = (E_i, \prec_i, \#_i, \lambda_i)$ for $i = 0, 1$ then $S_0 \,\|\, S_1 = (E, \prec, \#, \lambda)$ where

- $E = \{(i, e) \mid e \in E_i\} \cup \{(e_0, e_1) \mid e_i \in E_i \;\&\; \lambda(e_0) = \overline{\lambda(e_1)}\}$
- $e \prec e' \;\Leftrightarrow\; \begin{cases} e = (i, e_i) \text{ or } e = (e_0, e_1) & \text{and} \\ e' = (i, e_i') \text{ or } e' = (e_0', e_1') & \text{and} \\ e_i \prec_i e_i' & \text{for each } i \end{cases}$
- $e \sharp e' \;\Leftrightarrow\; \begin{cases} e = (i, e_i) \text{ or } e = (e_0, e_1) & \text{and} \\ e' = (i, e_i') \text{ or } e' = (e_0', e_1') & \text{and} \\ e_i \sharp_i e_i' & \text{for some } i \end{cases}$
- $\lambda(i, e) = \lambda_i(e)$ and $\lambda(e_0, e_1) = \tau$

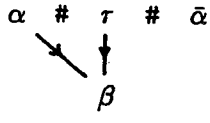
(iv) if $S_i = (E_i, \prec_i, \#_i, \lambda_i)$ for $i = 0, 1$ then $S_0 + S_1 = (E, \prec, \#, \lambda)$ where

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

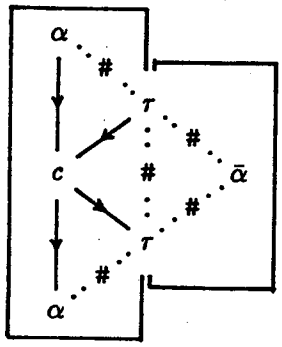
(v) if $S = (E, \prec, \#, \lambda)$ then $S\backslash\alpha = S\lceil H$ where $H = E - \bigcup\{E_n \mid n \geq 0\}$ and

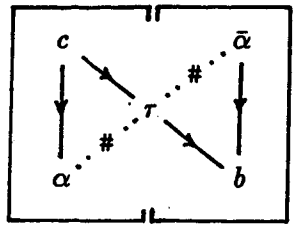
$$\begin{cases} E_0 = \{e \mid \mathsf{nm}(\lambda(e)) = \alpha\} \\ E_{n+1} = E_n \cup \{e \mid G \vdash e \;\Rightarrow\; G \cap E_n \neq \emptyset\} \end{cases}$$

The interpretation $I^\infty(p)$ is given by the unique continuous morphism I^∞ from the free algebra of infinite trees to the algebra of labelled event structures ($I^\infty(\mu x.p)$ is defined by a standard fixpoint construction, *cf.* [12,3]). Let us see some examples. The structure $I^\infty((\alpha : \beta : \mathsf{nil} \,\|\, \bar{\alpha} : \mathsf{nil}))$ may be drawn:

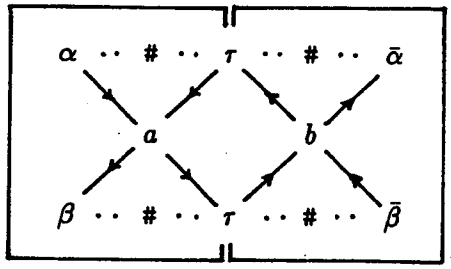


This example shows that $I^\infty(p)$ may contain a substructure $\triangledown'$ (*cf.* [3]). The interpretations of $(\alpha : c : \alpha : \mathsf{nil} \parallel \bar{\alpha} : \mathsf{nil})$ and $(c : \alpha : \mathsf{nil} \parallel \bar{\alpha} : b : \mathsf{nil})$ may be drawn respectively

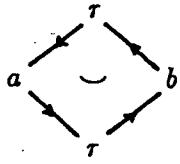




The second structure contains a substructure N and a substructure $\triangledown$, *cf.* [3]. An example, suggested by M. Nielsen, shows that $\prec^*$ is not an ordering: if we interpret $(\alpha : a : \beta : \mathsf{nil} \| \bar{\beta} : b : \bar{\alpha} : \mathsf{nil})$ we get



Note that if $r = (\alpha : a : \beta : \mathsf{nil} \parallel \bar{\beta} : b : \bar{\alpha} : \mathsf{nil}) \backslash \alpha \backslash \beta$ then $I^\infty(r)$ is



whereas $S(r)$ is the empty structure. In Winskel's terminology [12], $I^\infty(r)$ is not full.

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

We could prove moreover that there is an exact correspondence (as in [3]) 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

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

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" operationnal semantics for CCS, which generalizes the one we have given in [3] for a restricted language.

REFERENCES

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

[2] 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.

[3] 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.

[4] I. CASTELLANI, M. HENNESSY, *Distributed Bisimulations*, draft (1985) to appear.

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

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

[7] 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.

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

[9] 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.

[10] R. MILNER, *A Calculus of Communicating Systems*, Lecture Notes in Comput. Sci. 92 (1980).

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

[12] 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.

[13] 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