Process Algebras – Contribution for FMICS Handbook

Alessandro Fantechi[1] and Stefania Gnesi[2]

[1] Dipartimento di Sistemi e Informatica
Università degli Studi di Firenze
Firenze, Italy
fantechi@dsi.unifi.it

[2] ISTI-CNR
Pisa, Italy
gnesi@isti.cnr.it

1 Process Algebras

Process Algebras express the behaviour of a computing system in terms of processes. An elementary, non null, process is able at any time to perform an action and continue as another process. Hence, the simplest constructor of a Process Algebra is the action, or *action prefix* operator. Operators that combine two processes are also typically defined: nondeterministic choice between two processes, sequential composition and parallel composition (with or without some form of synchronization). Null or terminating processes are also defined, that constitute the basic elements on which structurally inductive definitions can be given. Process Algebras and languages derived from them, are endowed with an operational behavioural semantics, on top of which some equivalence is defined: two process terms are considered equivalent if according to the operational semantics, they can perform the same actions, and reconfigure into equivalent process terms. These semantics, and the associated equivalences, can be used in various ways: source to source transformations, proof of equivalence between programs, or direct analysis of the behaviour of a program. In particular, axiom systems can be defined for an equivalence on a Process Algebra, in order to inherit usual algebraic computation rules in the equivalence verification.

In the following we present first Milner's Calculus of Communicating Systems (CCS) [6], which can be considered as the first process formalism to which an algebraic form has been given. We discuss then LOTOS [5], which has been produced as an effort to make the expressing power of process algebra available in an international standard language. Other process algebras have been widely used, of which the most prominent are the Hoare's Concurrent Sequential Processes (CSP) [4], and the Basic Process Algebra (BPA) [1].

Evolutions of classical process algebras include formalisms to express mobility, such as the π-calculus[7], or formalisms that include stochastic attributes, such as Stochastic Process Algebra (SPA) [2].

2 CCS

We summarize the most relevant definitions regarding CCS, and refer to [6] for more details. The CCS syntax is the following:
$p ::= \sigma.p \mid nil \mid p + p \mid p|p \mid p\backslash A \mid x \mid p[f]$

Terms generated by p ($Terms$) are called *process terms* (called also *processes* or *terms*); x ranges over a set $\{X, Y, ..\}$ of process variables. A process variable is defined by a process definition $x \stackrel{def}{=} p$, (p is called the expansion of x). As usual, there is a set of visible actions $Act = \{a, \overline{a}, b, \overline{b}, ...\}$ over which α ranges, while σ ranges over $Act = Act \cup \{\tau\}$, where τ denotes the so-called *internal action*. We denote by $\overline{\alpha}$ the action complement. By nil we denote the empty process. The operators to build process terms are prefixing ($\sigma.p$), summation ($p+p$), parallel composition ($p|p$), restriction ($p\backslash A$) and relabeling ($p[f]$), where $A \subseteq Act$ and $f : Act \to Act$. Given a term p, an occurrence of a process variable x is *guarded in* p if it is within some sub-term of the form $\sigma.q$. We assume that (i) Act is finite; (ii) for each definition $x \stackrel{def}{=} p$, each occurrence of a process variable is guarded in p; (iii) all terms are closed, i.e. all variables occurring in a term are defined.

An operational semantics OP is a set of inference rules defining a relation $\to \subseteq Terms \times Act \times Terms$. The relation is the least relation satisfying the rules. If $(p, \sigma, q) \in \to$, we write $p \stackrel{\sigma}{\to} q$. The rules defining the semantics of CCS [6], from now on referred to as SOS, are recalled in Figure 1.

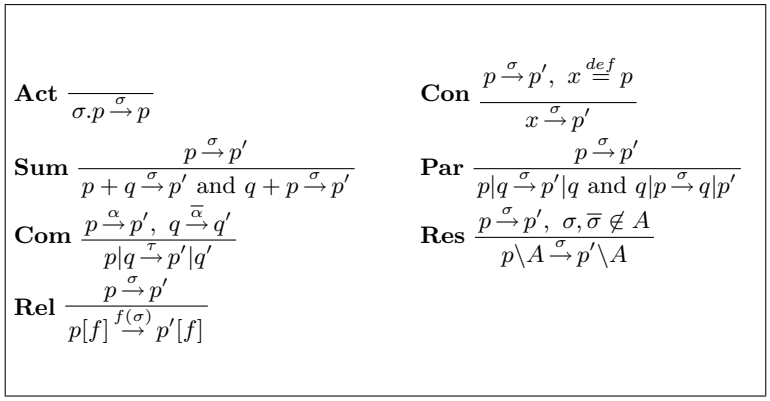

Fig. 1. The SOS rules

LTS definitions probably to be taken out since already present in the models part of the HB. Are also simulations and bisimulations already defined in the models part?

A *labeled transition system* (or simply *transition system*) TS is a quadruple $(S, T, \to, s_0)$, where S is a set of states, T is a set of transition labels, $s_0 \in S$ is

the initial state, and $\rightarrow \subseteq S \times T \times S$ is the transition relation. If $(s,t,s') \in \rightarrow$, we write $s \xrightarrow{t} s'$. A transition system is finite if $\rightarrow$ is finite.

For $A \subseteq Act$, we let $D_A(s)$ denote the set $\{s'$: there exists $\alpha \in A$ such that $s \xrightarrow{\alpha} s'\}$. We will also use the action name, instead of the corresponding singleton denotation, as subscript. Moreover, we let $D(s)$ denote in short $D_{Act}(s)$ and $D_{A_\tau}(s)$ denote $D_{A\cup\{\tau\}}(s)$.
We define:

- π is a path from $r_0 \in S$ if either $\pi = r_0$ (the empty path from r_0) or π is a (possibly infinite) sequence $(r_0,\sigma_1,r_1)(r_1,\sigma_2,r_2)\ldots$ such that $(r_i,\sigma_{i+1},r_{i+1}) \in \rightarrow$ for each $i \geq 0$. An initial path is a path starting from the initial state.
- A path π is called maximal if either it is infinite or it is finite and its last state r has no successor states ($D(r) = \emptyset$). The set of maximal paths from r_0 will be denoted by $\Pi(r_0)$. We intend that if $D(r_0) = \emptyset$ then the empty path from r is the only element of $\Pi(r_0)$.
- When $\pi = \rho\eta$, we say that ρ is a prefix of π; if $\pi = (r_0,\sigma_1,r_1)(r_1,\sigma_2,r_2)\ldots$, r_0 is the empty prefix of π; each path is a prefix of itself.
- We denote the *length* of a path π by $|\pi|$:
  If π is infinite, then $|\pi| = \omega$.
  If $\pi = r_0$, then $|\pi| = 0$.
  If $\pi = (r_0,\sigma_1,r_1)(r_1,\sigma_2,r_2)\ldots(r_n,\sigma_{n+1},r_{n+1})$, $n \geq 0$, then $|\pi| = n+1$.
  Moreover, we will denote the i^{th} state in the sequence, i.e. r_i, by $\pi(i)$.
- Given two transition systems TS_1 and TS_2, we say that a path π of TS_1 and a path π' of TS_2 correspond if the sequence of actions in π and π' is the same.

Given a term p (and a set of process variable definitions), and an operational semantics OP, $OP(p)$ is the transition system $(Terms, Act, \rightarrow, p)$, where $\rightarrow$ is the relation defined by OP. For example, $SOS(p)$ is the transition system defined by the SOS semantics for the term p.

In the following, we denote with $\mathcal{T}$ and TS or TS_i, the set of all LTSs and a generic LTS, respectively.

Definition 21 (Bisimulation, Simulation) *Let $TS_1 = (S_1,T_1,\rightarrow_1,s_{0_1})$ and $TS_2 = (S_2,T_2,\rightarrow_2,s_{0_2})$ be transition systems and let $s_1 \in S_1$ and $s_2 \in S_2$. We say that s_1 and s_2 are* strongly equivalent *(or simply* equivalent*) ($s_1 \sim s_2$) if there exists a* strong bisimulation *that relates s_1 and s_2. $\mathcal{B} \subseteq S_1 \times S_2$ is a strong bisimulation if $\forall(s_1,s_2) \in \mathcal{B}$ (where $\sigma \in T_1 \cup T_2$),*

- *$s_1 \xrightarrow{\sigma}_1 s_1'$ implies $\exists s_2' : s_2 \xrightarrow{\sigma}_2 s_2'$ and $(s_1',s_2') \in \mathcal{B}$; $s_2 \xrightarrow{\sigma}_2 s_2'$ implies $s_1 \xrightarrow{\sigma}_1 s_1'$ and $(s_1',s_2') \in \mathcal{B}$*

We say that s_2 simulates *s_1 ($s_1 \preceq_s s_2$) if there exists a* strong simulation *that relates s_1 and s_2. $\mathcal{R} \subseteq S_1 \times S_2$ is a strong simulation if $\forall(s_1,s_2) \in \mathcal{R}$ (where $\sigma \in T_1 \cup T_2$),*

- *$s_1 \xrightarrow{\sigma}_1 s_1'$ implies $\exists s_2' : s_2 \xrightarrow{\sigma}_2 s_2'$ and $(s_1',s_2') \in \mathcal{R}$.*

We say that s_1 *and* s_2 *are* weakly equivalent *(or observationally* equivalent*)* $(s_1 \approx s_2)$ *if there exists a* weak bisimulation *that relates* s_1 *and* s_2. $\mathcal{B} \subseteq S_1 \times S_2$ *is a weak bisimulation if* $\forall (s_1, s_2) \in \mathcal{B}$ *(where* $\sigma \in T_1 \cup T_2$*),*

- $s_1 \stackrel{\sigma}{\Longrightarrow}_1 s_1'$ *implies* $\exists s_2' : s_2 \stackrel{\sigma}{\Longrightarrow}_2 s_2'$ *and* $(s_1', s_2') \in \mathcal{B}$*;* $s_2 \stackrel{\sigma}{\Longrightarrow}_2 s_2'$ *implies* $s_1 \stackrel{\sigma}{\Longrightarrow}_1 s_1'$ *and* $(s_1', s_2') \in \mathcal{B}$ *(where* $s \stackrel{\tau}{\Longrightarrow} s'$ *iff* $s \stackrel{\tau}{\rightarrow} s_1 \stackrel{\tau}{\rightarrow} s_2 \ldots s_n \stackrel{\sigma}{\rightarrow} s'$*)*

TS_1 *and* TS_2 *are said to be* equivalent *(*$TS_1 \sim TS_2$*) if a strong bisimulation* $\mathcal{B}$ *exists such that* $(s_{0_1}, s_{0_2}) \in \mathcal{B}$. TS_2 simulates TS_1 *(*$TS_1 \preceq_s TS_2$*) if a strong simulation* $\mathcal{R}$ *exists such that* $(s_{01}, s_{02}) \in \mathcal{R}$. TS_1 *and* TS_2 *are said to be* observationally equivalent *(*$TS_1 \approx TS_2$*) if a weak bisimulation* $\mathcal{B}$ *exists such that* $(s_{0_1}, s_{0_2}) \in \mathcal{B}$.

Two CCS terms p and q are *equivalent*, or *strongly equivalent* $(p \sim q)$, if $SOS(p) \sim SOS(q)$; q simulates p $(p \preceq_s q)$ if $SOS(p) \preceq_s SOS(q)$; p and q are *observationally equivalent* $(p \sim q)$ if $SOS(p) \approx SOS(q)$;.

2.1 Axioms for strong equivalence

We show here the axioms that can be defined to equate CCS terms according to strong equivalence.
(Al) $x + (y + z) = (x + y) + z$
(A2) $x + y = y + x$
(A3) $x + x = x$
(A4) $x + NIL = x$
(A5) $x|(y|z) = (x|y)|z$
(A6) $x|y = y|x$
(A7) $x|NIL = x$
(A8) (expansion theorem)
if $u = \sum_i \mu_i.x_i$ and $v = \sum_i \nu_j.y_j$ then
$u|v = \sum_i \mu_i.(x_i|v) + \sum_j \nu_j(u|y_i) + \sum_{i,j:\mu_i=\overline{\nu}_j} \tau.(x_i|y_i)$
(A9) $\mu.x[f] = f(\mu).(x[f])$
(A10) $(x + y)[f] = x[f] + y[f]$
(A11) $NIL[f] = NIL$
(A12) $(\mu.x)\backslash A = \mu.(x\backslash A)$ if $\mu \notin A$
(A12) $(x + y)\backslash A = x\backslash A + y\backslash A$
(A13) $NIL\backslash A = NIL$

2.2 A CCS example

We show a simple CCS example describing a circular railway, divided into 6 track sections, numbered from 0 to 5. Two trains, namely train_a and train_b, can run in the same direction on this circular railway, with the constraint that it never happens that both are running on the same track section.
The CCS specification of the system can be chosen in a set of specifications $RAIL_{i,j}$, where i and j are respectively the track section numbers where $train_a$

and $train_b$ are positioned at the initial state of the circular railway; here we choose section 0 for $train_a$ enter in section 4 for $train_b$. The specification is made of a controller, $TRACK_i$ ($i = 0..5$), for every track section, which signals the preceding track when a train has left, and asks the following track for the permission to let a train enter (addition is modulo 6):

$TRACK_i = \overline{enter}_i.\overline{leave}_i.free_i.\overline{free}_{i+1}.TRACK_i$

Terms A_0 and B_4 below represent the behaviour of $train_a$ and $train_b$ respectively. When specifying these terms we make use of recursive calls to processes $TRAIN_{k,i}$ ($k = a, b$ and $i = 0..5$) which describe the behaviours of $train_k$ in the track section i. The action $moves_{k,i}$ represents the access of $train_k$ in section i while *enter*, *leave* and *free* are synchronization actions that regulate the accesses to the track sections:

$A_0 = enter_5.leave_5.\overline{free_5}.enter_0.TRAIN_{a,0}$
$B_4 = enter_3.leave_3.\overline{free_3}.enter_4.TRAIN_{b,4}$
$TRAIN_{k,i} = moves_{k,i}.leave_i.enter_{i+1}.TRAIN_{k,i+1}$

The specification of the system, $RAIL_{0,4}$, is obtained composing the track controllers in parallel with the behaviours of the two trains. Restrictions are placed on executable actions, in order to allow a higher degree of synchronization among the components and to reflect the point of view of an external observer (the only visible actions of the system are $moves_{k,i}$):

$RAIL_{0,4} =$
$(A_0|B_4|TRACK_0|TRACK_1|TRACK_2|TRACK_3|TRACK_4|TRACK_5)$
$\backslash enter_0 \backslash enter_1 \backslash enter_2 \backslash enter_3 \backslash enter_4 \backslash enter_5 \backslash leave_0 \backslash leave_1 \backslash leave_2 \backslash leave_3$
$\backslash leave_4 \backslash leave_5 \backslash free_0 \backslash free_1 \backslash free_2 \backslash free_3 \backslash free_4 \backslash free_5$

3 Basic LOTOS

Let us now recall the main behavioral concepts of LOTOS [5], which has been proposed as a standard formalisms for the specification of concurrent and distributed systems. Basic LOTOS is the version of LOTOS without value-passing; thus it is used to concentrate on the behavioral aspects of the systems. A Basic LOTOS program is defined as:

```
process ProcName := B
   where E
endproc
```

where `B` is a *behavior expression* (or *term*), `process ProcName := B` is a *process declaration* and `E` is a *process environment*, i.e. a set of process declarations. We can note that LOTOS processes share typical programming language constructs, in order to gain a better usability by the average engineer, with respect to the more math-looking syntax of CCS. Process algebraic operators are maintained in *behaviour expressions*. A behavior expression is the composition, by means of a set of operators, of a finite Alphabet $\mathcal{A}$=`{i,d, a,b, ...}` of atomic *actions*. Each occurrence of an action in $\mathcal{A}$ represents an event of the system. An occurrence of an action `a`$\in \mathcal{A}-$`{i,d}` represents a communication on the gate

`a`. The action `i` does not correspond to a communication and it is called the *unobservable action*; the action `d` does not correspond to a communication and can be considered as a termination signal.

The syntax of behavior expressions is the following:

```
B ::= stop | α;B | B[]B| X | B|[S]|B | B[f] | hide S in B | exit
                          | B>>B | B[>B
```

where X ranges over a set of process names, α ranges over $\mathcal{A}$, $S \subseteq \mathcal{A} - \{i\}$ and $f : \mathcal{A} \to \mathcal{A}$ is an action relabelling function, with the property that $f(i) = i$. We call $\mathcal{B}$ the processes generated from B. By `stop` we denote the empty process. The operators to build processes are *action prefix* $(a; B)$, *choice* $(B_1[]B_2)$, *parallel composition* $(B_1|[S]|B_2)$, *hiding* (`hide` S `in` B), *relabelling*, $(B[f])$,*process instantiation* (X), *successful termination* (**exit**), *sequential composition*, also called *enabling* $(B_1 >> B_2)$, and *disabling* $(B_1[> B_2)$.

In the following, we shall denote the set of all behavior expressions with $BEXP$. The meaning of the operators composing behavior expressions is the following:

The *action prefix* `a;B` means that the corresponding process executes the action `a` and then behaves as `B`.

The *choice* `B1 [] B2` composes the two alternative behavior descriptions `B1` and `B2`.

The expression `stop` cannot perform any move.

The *parallel composition* `B1|[S]|B2`, where `S` is a subset of $\mathcal{A}-\{$`i`$\}$, composes in parallel the two behaviors `B1` and `B2`. `B1` and `B2` interleave the actions not belonging to `S`, while they must synchronize at each gate in `S`. A synchronization at gate `a` is the simultaneous execution of an action `a` by both partners and produces the single event `a`. If `S`$=\emptyset$ or `S`$=\mathcal{A}$, the parallel composition means pure interleaving or complete synchronization. Cyclic behaviors are expressed by recursive process declarations.

The *relabeling* `B[f]`, where `f`: $\mathcal{A} \to \mathcal{A}$ is an action relabeling function, renames the actions occurring in the transition system of `B` as specified by the function `f`. `f` is syntactically defined as `a0 -> b0,...,an->bn`, meaning `f(a0)=b0,...,f(an)=bn`, and `f(a)=a` for each `a` not belonging to {`a0,...,an`}. Note that each relabeling function has the property that `f(i)` $=$ `i`, `f(d)` $=$ `d`. Actually, the relabeling operator is not part of the standard definition of LOTOS [5], but is used to define the semantics of other language constructs.

The *hiding* `hide S in B` renames the actions in `S`, occurring in the transition system of `B`, with the unobservable action `i`.

Successful termination is a process that has terminated, but still is able to emit a termination signal. It is used in conjunction with the *enabling* operator: if the left process terminates emitting the termination signal, the right process starts its execution consequently.

The disabling operator allows the right process to take the place of the left one, at any time of its execution.

As usual, the operational semantics of a behavior expression `B` is given in terms of a Labeled Transition System, whose states correspond to behavior expressions (the initial state corresponds to `B`) and whose transitions (arcs) are labeled by actions in $\mathcal{A}$. Given a set $\mathcal{E}$ of behavior expressions declarations, the standard *operational semantics* is given by a relation $\rightarrow_{\mathcal{E}} \subseteq \mathcal{B} \times \mathcal{A} \times \mathcal{B}$. $\rightarrow_{\mathcal{E}}$ ($\rightarrow$ for short) is the least relation defined by the rules in Table 1.

Table 1. Standard operational semantics of Basic LOTOS

$\alpha \in \mathcal{A},\ l \in \mathcal{A} - \{i\}, \beta \in \mathcal{A} - \{d\}$

pre $\dfrac{}{\alpha; B \xrightarrow{\alpha} B}$

choice $\dfrac{B_1 \xrightarrow{\alpha} B_1'}{B_1 \ [] \ B_2 \xrightarrow{\alpha} B_1'}$

inst $\dfrac{B \xrightarrow{\alpha} B'}{X \xrightarrow{\alpha} B'}\ X := B \in \mathcal{E}$

rel $\dfrac{B \xrightarrow{\alpha} B'}{B[f] \xrightarrow{f(\alpha)} B'[f]}$

par $\dfrac{B_1 \xrightarrow{\beta} B_1'}{B_1 \ |[S]| \ B_2 \xrightarrow{\beta} B_1' \ |[S]| \ B_2}\ \beta \notin S$

com $\dfrac{B_1 \xrightarrow{l} B_1',\ B_2 \xrightarrow{l} B_2'}{B_1 \ |[S]| \ B_2 \xrightarrow{l} B_1' \ |[S]| \ B_2'}\ l \in S \cup \{d\}$

hide$_1$ $\dfrac{B \xrightarrow{\alpha} B'}{\texttt{hide}\ S\ \texttt{in}\ B \xrightarrow{\alpha} \texttt{hide}\ S\ \texttt{in}\ B'}\ \alpha \notin S$

hide$_2$ $\dfrac{B \xrightarrow{l} B'}{\texttt{hide}\ S\ \texttt{in}\ B \xrightarrow{i} \texttt{hide}\ S\ \texttt{in}\ B'}\ l \in S$

enab$_1$ $\dfrac{B_1 \xrightarrow{d} B_1'}{B_1 >> B_2 \xrightarrow{i} B_2}$

enab$_2$ $\dfrac{B_1 \xrightarrow{\beta} B_1'}{B_1 >> B_2 \xrightarrow{\beta} B_1' >> B_2}$

exit $\dfrac{}{\texttt{exit} \xrightarrow{d} \texttt{stop}}$

dis$_1$ $\dfrac{B_1 \xrightarrow{d} B_1'}{B_1 [> B_2 \xrightarrow{i} B_1'}$

dis$_2$ $\dfrac{B_1 \xrightarrow{\beta} B_1'}{B_1 [> B_2 \xrightarrow{\beta} B_1' [> B_2}$

dis$_3$ $\dfrac{B_2 \xrightarrow{\alpha} B_2'}{B_1 [> B_2 \xrightarrow{\alpha} B_2'}$

In Table 1 the symmetric rules for choice and parallel composition are not shown.

Assume the precedence of the operators as specified by the following list, ordered in decreasing order:

```
;   [f]   hide   |[S]|   []
```

Let us consider as an example the description of a vending machine, giving hot coffee and two kinds of tea, with lemon or peach flavor:

```
process P :=
    coin;
    (coffee; boil; sugar; collectCoffee; P
     []
     tea; (peach; collectPeachTea; P
      []
      lemon; collectLemonTea; P
     )
    )
```

When a `coin` is inserted, either coffee or tea may be requested by pressing the buttons `coffee` or `tea`, respectively. If coffee has been requested, the machine performs the actions of boiling the coffee and put sugar into it and then the coffee can be collected (action `CollectCoffee`). If tea has been requested, by pressing button `lemon` or button `peach` tea with lemon or with peach can be collected, respectively (actions `collectLemonTea, collecPeachTea`). Finally, the process reverts to its initial state `P`.

4 Full LOTOS

The full LOTOS language [5], being aimed at a practical application for the specification of complex communication protocols, joins the process algebraic expression of behaviour with the description of values and types, therefore constituting an example of a fully developed specification language based on process algebraic principles; the most important effect of the inclusion of values from the behavioural point of view is that interprocess value communication can be expressed.
The syntax of a LOTOS specification and of a LOTOS process definition is:

```
specification Id [Gatelist](Varlist) : functionality
type definition
behaviour behaviour expression
where type definition , process definition
endspec

process Id [Gatelist](Varlist) : functionality :=
behaviour expression
where type definition , process definition
endproc
```

The representation of values, value expressions and data are derived from the specification language for abstract data types ACT ONE [3], an algebraic specification method to write non parameterized as well as parameterized specifications. In this context, we do not address the added complexity brought into the picture by the formal definition of types and expressions, but we maintain concentrated on the behavioural, process algebraic aspects. Indeed, the process

algebraic part of LOTOS can be seen as orthogonal to the type system, which could have been defined differently as well.
We need nevertheless to consider the effects that value passing has on communication and synchronization, but we can ignore the algebraic data definition part: in the following just two evaluation functions (β, ϵ) for value expressions will be considered as given.
The semantics of value passing is based on the concept of *matching offered values*: output is modeled as offering a single value, while input is modeled as offering any value of the type (possibly constrained by a predicate). $!E$ is a value declaration and represent the value expression offered by the process on a gate; $?x : t$ is a variable declaration and represent the set of values that the process offers on a gate (that is, it is ready to accept on the gate). In LOTOS variables are single assignment ones; scope rules for the variable declaration are straightforward: the scope of a variable x that is declared in an action is the behaviour expression following that action in an action prefix construct.
Communication occurs when both sides offer the same value; this means that two output actions may communicate as well, if matching occurs, and that two input action may communicate, nondeterministically choosing among the values offered by both sides. In Table 2 we summarize the different synchronizations that can occur between two processes. We assume in the following for simplicity that processes offer only one attribute, that is each action contains only an input or an output offer; actually an action can contain a list of mixed attributes, with the meaning that two actions match if all the attributes match. We indicate with: V the set of definable values of LOTOS, G the set of user definable gates, $\mathcal{A}$ the set {g<v> | g∈G,v∈V} ∪ {i} ∪ {d<v > | v∈V}.

In Table 3 we give the operational semantics for LOTOS, showing only those clauses that are different or additional w.r.t. Basic LOTOS ones. In particular, the clauses with primed names in Table 3 substitute the corresponding ones in Table 1, while the other clauses in Table 3 are additional. Moreover, all the non substituted clauses in Table 1 continue to hold, with the care of considering the names of gates referred in actions instead that the action itself, whenever an expression like $\alpha \in S$ occurs. Note that $[a/b]$ represents the syntactic substitution when b is a variable and a is a value (due to the single assignment assumption) and is a relabelling when a and b are (sets of) gates. Moreover, LOTOS allows lists of variables and lists of values wherever we have used single variables and values (that is, communication, successful termination and instantiation), with the obvious meaning.

We indicate with: V the set of definable values of LOTOS, G the set of user definable gates, *Act* the set {g<v^+> | g∈G,v^+⊂V} ∪ {i}. Act^+ is the set Act ∪ {d<v^*> | v^*⊂V}.

Table 2. LOTOS synchronization modalities

process A	process B	sync. condition	interaction type	effects
g!E1	g!E2	ϵ(E1) = ϵ(E2)	value matching synchronization	
g!E	g?x:t	ϵ(E)$\in$t	value passing after synchronization	x = ϵ(E)
g?x:t	g?y:u	t = u	value generation after synchroniz.	x=y=v for v$\in$t

Table 3. Operational semantics of full LOTOS

$$\textbf{pre'}\ \frac{}{i; B \xrightarrow{i} B}$$

$$\textbf{out}\ \frac{}{g!E; B \xrightarrow{g<\epsilon(E)>} B}$$

$$\textbf{in}_1\ \frac{}{g!x:t; B \xrightarrow{g<v>} B[v/x]}\ v \in t$$

$$\textbf{in}_2\ \frac{}{g!x:t[cond(x)]; B \xrightarrow{g<v>} B[v/x]}\ v \in t, cond(v)$$

$$\textbf{genchoice}_1\ \frac{B[v/x] \xrightarrow{\alpha} B'}{\textbf{choice}\ x:t[]B \xrightarrow{\alpha} B'}\ v{\in}t$$

$$\textbf{genchoice}_2\ \frac{B[g/g_i] \xrightarrow{\alpha} B'}{\textbf{choice}\ g\ \textbf{in}\ [g_1,...,g_n][]B \xrightarrow{\alpha} B'}\ , v{\in}t$$

$$\textbf{exit'}\ \frac{}{\texttt{exit(E)} \xrightarrow{d<\epsilon(E)>} \texttt{stop}}$$

$$\textbf{let}\ \frac{B[\epsilon(E)/x] \xrightarrow{\alpha} B'}{\textbf{let}\ x:t=E\ \textbf{in}\ B \xrightarrow{\alpha} B'}$$

$$\textbf{cond}\ \frac{B \xrightarrow{\alpha} B', cond}{[cond]{\rightarrow}B \xrightarrow{\alpha} B'}$$

$$\textbf{inst'}\ \frac{B[h_1,...,h_n/g_1,...,g_n][\epsilon(E)/x] \xrightarrow{\alpha} B'}{P[h1,...,hn](E) \xrightarrow{\alpha} B'}\ P[g1,...,gn](x) := B \in \mathcal{E}$$

$$\textbf{enab}_3\ \frac{B_1 \xrightarrow{\alpha} B_1'}{B_1\textbf{>>accept}\ x:t\ \textbf{in}\ B_2 \xrightarrow{\alpha} B_1'\textbf{>>accept}\ x:t\ \textbf{in}\ B_2}$$

$$\textbf{enab}_4\ \frac{B_1 \xrightarrow{d<v>} \textbf{stop}}{B_1\textbf{>>accept}\ x:t\ \textbf{in}\ B_2 \xrightarrow{i} B_2[v/x]}$$

References

1. J.A. Bergstra and J.W. Klop. Algebra of Communicating Processes with Abstraction. Theoretical Computer Science, 37:77–121, 1985.

2. E. Brinksma and J-P. Katoen and D. Latella and R. Langerak. A stochastic causality-based process algebra. The Computer Journal, 38, 7, 552–565,1995
3. I. Classen, H. Ehrig, D. Wolz, Algebraic Specification Techniques and Tools for Software Development: The Act Approach AMAST Series in Computing - Vol. 1. World Scientific. Nov. 1993.
4. C. A. R. Hoare. Communicating Sequential Processes. Prentice-Hall, 1985
5. ISO IS 8807. *Information Processing Systems, Open Systems Interconnection, LOTOS Formal Description Technique based on Temporal Ordering of Observational behavior*, June 1988.
6. R. Milner. Communication and Concurrency. Prentice Hall, 1989.
7. R. Milner, J. Parrow, D. Walker. A Calculus of Mobile Processes, Inf. Comput. 1 00(1): 1-40 (1992)
8. R. Milner, J. Parrow, D. Walker. A Calculus of Mobile Processes, II, Inf. Comput. 100(1): 41-77 (1992)