Higher-level synchronizing devices in Meije-SCCS

Robert de Simone

INRIA

CENTRE
SOPHIA ANTIPOLIS

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.: (3) 954 90 20

Rapports de Recherche

N° 360

HIGHER-LEVEL SYNCHRONIZING DEVICES IN MEIJE-SCCS

Robert de SIMONE

Janvier 1985

Higher-Level Synchronizing devices in Meije-SCCS.

Robert de Simone

INRIA
route des Lucioles
Sophia-Antipolis
06560 Valbonne
FRANCE

RESUME

Dans un cadre algébrique pour le parallèlisme et la synchronisation due à Robin Milner nous definissons (par des règles conditionnelles de sémantique opérationnelle) une classe d'opérateurs de synchronisation. Nous montrons qu'ils sont des opérateurs dérivés de l'algèbre SCCS d'origine, en y produisant leur réalisation. Les buts sont: étudier l'expressivité du calcul -soit absolue, calculatoire, soit relativement à d'autres formalismes- et définir techniquement une équivalence opérationnelle sur des expressions non closes. Mais avant tout ces opérateurs devraient permettre d'exprimer des fonctionnements parallèles de systèmes de transitions et des transformations sur ceux-ci de manière algébrique sans perdre l'intuition de leur action; en effet il est alors permis de s'affranchir de la réalisation "de base" en Meije-SCCS. Les règles sémantiques de comportements ont une syntaxe précise, à l'exception de relations liant les différentes actions simultanées des processus coopérants, qui elles sont "sémantiques". Cet aspect nous permet de traiter de la "puissance expressive" du calcul. Pour les termes clos, un résultat d'universalité est donné, pour l'expresssion des systèmes de transition calculable. La situation des expressions non-closes est encore floue.

ABSTRACT

In an algebraic setting for parallelism and synchronisation due to Robin Milner, we define operationally a variety of synchronizing operators on processes by the semantical conditional rules they obey. We prove them to be higher-level non-primitive operators from the original SCCS calculus, showing how to meet their behaviours with primitive expressions. Uses are: study of expressiveness -either absolute or relative to other formalisms through their translation in the calculus-, and technical shaping of equivalence proofs for expressions. But most of all this allows one to specify formally synchronisation settings dealing with (operational) products and transformations of transition systems, and still not loose their informal appealing intuition; for we then forget

the realisation working with Meije-SCCS elementary synchronisation mechanisms (the mechanics below the hood). The defining rules are syntactically given, except for the allowed relations on the components' actions monoïds. This "semantical" aspect allows us to treat many "calculability" issues. This is specially true of closed terms, where we may claim constructive "universality" amongst transition systems. Case of operators seems slightly more intricate.

Introduction

Meije [Bou1984a] and **SCCS** [Mil1983a] are two avatars of an algebraic calculus for dealing with parallelism and synchronisation. Their conceptual differences -universal time reference for the latter and local (even though equally granulous) times for the former- can easily be transcended by their technical similarities : algebraic syntax (most operators are shared [1]), then structural conditional rules defining an operational semantics, and structural recursive equivalences (bisimulations). They share a notion of global simultaneity recovered by a commutative monoïd of actions.

A potential user calculating in these algebras will feel little concern for their minimality in number of operators, and will probably gather both, as well as both time concepts they recover, into a unified frame. He can then use what is appropriate to ease specifying his private problem. What is to us there needed is a proof of mutual translatability [Bou1984a].

In this study we pursue this trend towards having more than strictly needed operators, for comfort of description. We provide a whole class of them, through a precise syntaxic pattern of allowed conditional semantical rules *à la* Plotkin [1981a] defining their operational behaviours. Transformations on transition systems transpare behind these rules. One can then design his own favorite process synchronisations, as soon as he can express them that way.

Apart from this obvious use letting one specify directly though rigourously his informal and intuitive synchronisation problem, the three main motivations we see in this work are:

* First, obtain results on the calculus expressive power, in the realm of transition systems labelled on an action structure (here monoïd). This is largely due to the fact that **actions** abilities are treated *semantically* in the rules schemes (under the form of specifically allowed recursively enumerable relations and sets of behaviours on the actions/signals monoïd). We retrieve here calculability concerns.

[1] with the notable exception of the operator setting two processes in parallel, either synchroneous or not.

* Second, provide a useful translation help, while attempting to express other formalisms (languages equipped with operational semantics [Bro1984a] or other algebraic devices [1979a] dealing with parallelism and synchronisation), into our theory. Additions of non-primitive operators amidst the object algebra, supposed each to represent a primitive operator from the source algebra, was the algebraic way to corner the problem, which then reduces to a trivial equivalence one at the *semantics* (in our case *transition systems*) level. First progress was made in [Bou1984b] with Labelled Petri Nets endowed with an algebraic structure.
* Third, the higher-level specification level might prove most convenient to conduct equivalence proofs, since there all behaviour deductions are straightforward (one-step deduction from process parameters). Thus going **back** from realisation to specification is of worth. Let's recall already now that unfortunately **not all** Meije-realized processes can be so specified, at least finitely. A typical counterexample is the file-process [Bera].

After a brief account on the Meije-SCCS theory (by now entirely published, see [Mil1983a] and [Bou1984a]), followed by the example of the file-process, we turn towards a specific type of open terms -or expressions- that are of special interest to us: these expressions in which occurring *meta*variables certainly cannot be duplicated (do not appear in right hand parts of recursive definitions in subterms). We will furthermore postulate these metavariables appear linearly in the expression. [1] The purpose of defining such a class of expressions is to allow devise on them a Park-like induction principle for an equivalence which is sound *w.r.t.* (thus included into) the strong congruence on open terms, but does **not** require checking equivalence for **every** instanciation for the variables. It is this enlarged principle which is put to use later in the paper, even though it goes unspoken. It makes the whole proofs "syntactic". We quickly face the problem of this principle completeness, which it fails for very specific reasons -this we hope to fix soon-.

We define a syntax of conditional rules for new operators. They allow to infer the system behaviour from the ones of its components. They generalize nicely the rules of the SCCS primitive operators. As we already mentioned, they use a "semantic" constituent: the relation (fulfilled predicate) that is to hold in between the immediate component behaviours and the compound resulting action is not given syntax for, but rather described directly as a subset of uples. This is again a strong feature to bear in mind. Bisimulation notions and so on are easily carried upon new calculi based on such defined operators. For instance the strong equivalence is still a congruence *w.r.t.* them.

A general theorem then asserts that for each such (finite) system of new operators so (finitely) specified there is a Meije-SCCS primitive expression

[1] It seems to us the latter restriction is unnecessary, but we did not carry out the proofs in the multiple case yet, and the gain in specification is not clear.

realizing it, **as soon** as the relations on uples of actions in the rules are recursively enumerable (or expressible). The specific study of realisation of such immediate-behaviour recursive relations was started in [Sim1984a]

The proof of the theorem is only sketched, appearing at length in [Sim1984b] It relies on our new "operational" equivalence principle for expressions.

We close on an important, directly connected, *"semantic"* corollary. It asserts the **Meije**-representiveness of any **"effective"** transition system, labelled in the action monoïd. Thus, operational *"meaning"* of our terms may range over as large a domain as **"possibly computable"**.

1. The SCCS and Meije calculi.

As mentioned, both were published [Mil1983a] , [Bou1984a] and several studies devoted to them by now. See bibliography.

Both are Σ-algebras, with interned recursive equations on first-order variables defining infinite terms. Operators are endowed with an intuitive architectural meaning that helps see the geometric settlement in a concurrent system of connected processes denoted by a term. This pattern may evolve dynamically (even without time consuming) with unwinding of recursive definitions. The first main goal was to attach to every such term an operational semantics, under the form of a transition system, whose possible transitions will represent the behaviours (possible **actions**) of the system. A structural (along the syntax) way of obtaining these behaviours for any term is certainly the calculus main advantage, for all the simple techniques it then allows to work on (huge) transition systems **through** the terms themselves. What one does is use a formal deduction system to produce possible next-step transitions.[1]

We will break down the presentation in four parts:

Actions Auxiliary sort of the algebra, labels of the transition systems.

Agents Terms of the algebra, states of transition systems (or transition systems themselves, confused with their initial state).

Processes Elements of the calculus, reduced transition systems w.r.t. the equivalence notion.

Expressions Open terms, on metavariables -or *parameters*- modelling synchronisation contexts, as protocols for instance. Proving expressions equivalence will be of great concern hereafter.

1.1. Actions

We start with *atomic* granuleous actions, whose semantics is volontary left unspecified, in the fashion of program schemes. We will always suppose a finite set A of such actions. Then this set is endowed with a simulteanity product, commutative and associative. Thus:

$a.b$ means a and b happening together.

Time being discretely modelled, a and b have therefore strongly comparable duration. See [1984a]. By now the action structure has become a full commutative semigroup (since as we will see there is no bound on the number of concurrent processes).

Communication (without explicit value-passing, since the model does not bother itself to model memories) is treated through new introduced **invertible** actions,

[1] Care should be taken in specific examples not to confuse too radically complexity used inside the deduction system with complexity of the resulting transition system. Of course on the average they are related.

called signals. To communicate is to emit together each the inverse signal than the other. The resulting common action is a neutral element, the **internal** action 1. The nature of the communication is thus not supplied to the exterior. The signal inverse from s is noted $\overline{s}$. No direction to the communication is provided in this model. Often we will prefer s to be an output and $\overline{s}$ to be an input, but without any effect on the model. Communication is not always forced and sometimes unmatched signals remain.

Finally the action structure is thus a commutative **monoïd**, free commutative product of a free commutative monoïd and of a free commutative group.

Sometimes we will allow signals to appear in overall behaviours (or transitions) of an agent, although they are mainly meant to communicate -and disappear- inside of systems. Then we can describe open systems, or parts of systems. We will feel free to create and introduce (finitely) many duplications of the set of atomic actions and signals a term can show the external world, as long as these copies consist of "internal" signals designed for private use. Their names will be mostly irrelevant as long as they are not confused together.

It is easily shown that, given an agent (next notion to define), one can find it a smallest set including all its possible atomic actions' and signals' occurrences in the future, which is finite, called its *Sort*. It is the agent's alphabet, call it A for a second, and M the generated monoïd. For such an A, every compound action of M can be identified with an n-uple of integers, n being the cardinal of A. Indeed by associative-commutative properties of the product, and elimination of inverse signals, every such action has a unique *normal* form as an ordered product of atomic actions and signals, each to a certain exponent. Given the vector of these integer exponents one can reconstruct the original action. For instance on the alphabet $\{a,b,c\}$, $a^3.c^2$ corresponds to (3,0,2).
If A contains two opposite signals then of course vectors have either corresponding component equal to 0.

This identification allows to introduce the classic recursive and computable notion on actions, by isomorphism with integer vectors. Recursive and primitive functions, unary or not, recursively enumerable sets and relations can be so defined. On the commutative monoïd notions of rational sets may be designed as well, after Parikh's results [Par1966a] being called *semi-linear sets*, for what they break down into: a semi-linear set is simply a finite union of linear sets, a linear set being of the form $a.b_1^*.\cdots.b_n^*$, where a and the b_i's are "commutative words" (not just letters). The result here is that in the commutative case the so-called *star-height* may be brought back to 1.

1.2. Agents

Agents are terms of a Σ-algebra, whose operators are sometimes parameterized by actions. They are provided an operational semantics structurally (that is, the inference of a behaviour/transition ability for a term having an operator as its root depends only from the behaviour abilities of its direct sons and the conditional deduction rule of the root operator). Applying recursively these deductions on resulting agents we come up with transition systems modelling the agent. We now give a list of operators and their rules for each of the two algebras, Meije and SCCS.

S.C.C.S operators		
names	**notations**	**behavioural rules**
inaction	**0**	–
precedence	**a**:T	$a:T \xrightarrow{a} T$
sum [1]	$S + T$	$\dfrac{S \xrightarrow{a} S'}{S + T \xrightarrow{a} S'}$ $\dfrac{T \xrightarrow{a} T'}{S + T \xrightarrow{a} T'}$
synchronous product	$S \times T$	$\dfrac{S \xrightarrow{a} S' \;,\; T \xrightarrow{b} T'}{S \times T \xrightarrow{a.b} S' \times T'}$
renaming [2]	$<\Phi> S$	$\dfrac{S \xrightarrow{a} S'}{<\Phi> S \xrightarrow{\Phi(a)} <\Phi> S'}$
restriction	$S \mid B$	$\dfrac{S \xrightarrow{a} S' \;,\; a \in B}{S \mid B \xrightarrow{a} S' \mid B}$
variables [3]	x, y, x_i, *etc* ...	–
recursive operators	x_i *where* R [4]	$\dfrac{[\forall j,\ x_j \text{ where } R / x_j]T_i \xrightarrow{a} S}{x_i \text{ where } R \xrightarrow{a} S}$ [5]

inaction, *precedence*, *sum*, *variables* and *recursive operators* may be used to build **sequential** processes [Mil1984a]. The three other operators are mainly used to configurate architectural structures, setting proper links and "wires" in between processes [Mil1979a]. These structures may be dynamically evolving with recursive unwindings.

---

[1] Non-deterministic choice, as shows the presence of two rules. One defines infinite sum $\sum_i S_i$ as well.

[2] Φ being a **morphism** on the action monoid (generally introducing private signals or setting *ports*).

[3] Syntactic variables are used to represent infinite terms, fixed-points of recursive equations contained in *where* operators. They have thus no "real" existence and could be represented by numbers. This feature is close to iterative theories of [Elg1975?a]. Here only recursively defined first-order (process) variables are defined. Defining second-order (operator) recursively defined variables does not bring increased power to the system. A proof exists in [Sim1984b]

[4] Here R is a finite list of equations $x_1 = T_1, \ldots, x_n = T_n$, where the T_j terms may contain the concerned variables.

[5] where $[***/**]*$ is the syntactic substitution.

Meije operators		
names	**notations**	**behavioural rules**
inaction	**0**	–
precedence	**a**:T	$a:T \xrightarrow{a} T$
parallel product [1]	$S \parallel T$	$\dfrac{S \xrightarrow{a} S'}{S \parallel T \xrightarrow{a} S' \parallel T}$
		$\dfrac{T \xrightarrow{a} T'}{S \parallel T \xrightarrow{a} S \parallel T'}$
		$\dfrac{S \xrightarrow{a} S' , T \xrightarrow{b} T'}{S \parallel T \xrightarrow{a.b} S' \parallel T'}$
triggering [2]	$s \Rightarrow S$	$\dfrac{S \xrightarrow{a} S'}{s \Rightarrow S \xrightarrow{s.a} S'}$
ticking [3]	$s^* S$	$\dfrac{S \xrightarrow{a} S'}{s^* S \xrightarrow{s.a} s^* S'}$
renaming	$<\Phi> S$	$\dfrac{S \xrightarrow{a} S'}{<\Phi> S \xrightarrow{\Phi(a)} <\Phi> S'}$
restriction	$S \backslash s$	$\dfrac{S \xrightarrow{a} S' , \text{ neither } s \text{ nor } \overline{s} \text{ appears in } a}{S \backslash s \xrightarrow{a} S' \backslash s}$
variables	x, y, x_i, *etc* $\cdots$	–
recursive operators	x_i *where* R [4]	$\dfrac{[\forall j, x_j \text{ where } R / x_j] T_i \xrightarrow{a} S}{x_i \text{ where } R \xrightarrow{a} S}$

[1] asynchroneous and non-deterministic.
[2] allow to synchronise on the starting of an agent, without prejudging on the nature of its behaviour. Can actually be simulated with the next operator and recursive definitions (so is not primitive).
[3] allow to synchronise on each behavioural step of an agent, without prejudging on the nature of its behaviour.
[4] We shall use the alternative notation $rec\ x.T$ meaning x *where* $x=T$ when there is onlyone variable.

Actually we will use an extended version of SCCS, where (in an earlier version) a de-synchronizing operator Δ is introduced, whose semantical rules are:

delaying	δT	$\delta T \xrightarrow{1} \delta T$
		$\dfrac{T \xrightarrow{a} T'}{\delta T \xrightarrow{a} T'}$
desynchronizing	ΔT	$\dfrac{T \xrightarrow{a} T'}{\Delta T \xrightarrow{a} \delta \Delta T'}$

The *delaying* operator δ is obviously non-primitive, but helps defining Δ. It was remarked that the presence of this latter operator made the + operator non-primitive (since it may be defined in **Meije**, and all **Meije** operators defined in **SCCS** without it).

Equivalence results in between both calculus will show the sameness of two apparently opposed visions of the world: one where processes work synchroneously ($\times$), and may be disregulated locally (Δ); the other where they work asynchroneouly ($\|$) and may be locally made time-controllable ($*$). Equally well we could use an algebra made out of $\|$, $\times$, and the usual $\mathbf{0}$, $a:$, $\backslash s$, $<\varphi>$ operators and get a resulting equivalent calculus (this since $*$ may be defined in terms of $\times$ and recursive definitions only).

Example

We informally describe a FIFO file. It uses an atomic input action a, as well as an atomic output action b. Generalisation to multiple such actions would be straightforward. One can thus "deposit" integers n by performing a^n, and withdraw them by b^n. The withdrawals must follow the deposits, and keep the same order. Otherwise they are relatively asynchronous.
Note that this philosophy of carrying an integer value encoded as the "intensity" of a signal is basically very primitive, as "sticks" in Tu"ring machines are. It is not due to a great future in non-virtual programming.

Definition

We shall call FIFO-agent the term :

$$\left[\alpha:\mathbf{0} \parallel \mathit{rec}\ x.\ \left[S \parallel \gamma:\left[\overline{\alpha}.\overline{\delta}:\beta:\mathbf{0} \parallel <\beta/\alpha>x\ \right]\backslash\beta\right]\right]\backslash\alpha$$

where

$$S = \mathit{rec}\ x.\ \left[(a:b:\mathbf{0})\times x\ \ +\ \overline{\gamma}:\delta:\mathbf{0}\right].$$

First remark S realizes the behaviour specification:

$$\sum_{n=1}^{\infty}\overline{\gamma}.a^n:\delta.b^n:\mathbf{0}$$

and thus acts as a cell swallowing a number of a actions, provoking altogether γ, and later letting a corresponding number of b actions out, at δ signal demands. The proper sequencing of moves amongst the successive S cells created is shaped through the α signals, which selectively allow the emission of next value, and β, which blocks others till their time comes. One may check that at any moment there is but one α signal susceptible from emission. There is at most one $\overline{\alpha}$ too, and if none the FIFO-agent is empty.

We shall not try here to prove any equivalence in between the FIFO-agent specification and its Meije-SCCS realization, the former being too vague. We wold just like to draw the reader's attention on the previously mentioned sequencing of mutually pertinent behaviours, and how it is realized by a recursive term of the algebra generating parallel components of life length **two**.

1.3. Processes

On agents a congruence is defined, called **strong** equivalence, whose classes we want to see as processes. Its main effect is to suppress redondancy in transition systems (when two alternative futures are alike) while conserving all characteristic features towards deadlock situations. It is a *bisimulation*, and as such recursively checkable (along the succession of reconfiguration). This is

nice since then an induction principle, known as Park's, can be devised. Unfortunately this equivalence in general is not decidable. Researches in that area on restricted subclasses of agents are still open.

Definition

An equivalence relation R, $R \subset Meije\text{-}SCCS \times Meije\text{-}SCCS$, is a **bisimulation** if

two agents S and T are such that $(S,T) \in R$ iff

$$\forall u \in M, \forall S' \text{ agent such that } S \xrightarrow{u} S', \exists T' \text{ agent }, (T',S') \in R, T \xrightarrow{u} T' \quad (1)$$

$$\forall u \in M, \forall T' \text{ agent such that } T \xrightarrow{u} T', \exists S' \text{ agent }, (S',T') \in R, S \xrightarrow{u} S' \quad (2)$$

Notice the definition depends on a certain alphabet A, through the monoïd M the possible behaviours are quantified upon. Proving an equivalence relation to be a bisimulation requires to check *all* possible behaviours, and this in fact amounts to build -and quantify upon- *all* "proof trees", calling this way the proof pattern that allowed infer a behaviour in our logical deduction system.

Definition

We call strong congruence, noted $\cong$, the largest bisimulation possible. Details on why it exists and turn out to be the union of all other bisimulations may be found in [Mil1983a]

Park's induction principle goes like this:

Definition

Let R be a relation (which we want to prove part of a bisimulation). Then if for every couple (S,T) of Meije-SCCS terms in R the following holds:

$$\forall u \in M, \forall S' \text{ agent such that } S \xrightarrow{u} S', \exists T' \text{ agent }, (T',S') \in (R \cup \cong)^*, T \xrightarrow{u} T'$$

$$\forall u \in M, \forall T' \text{ agent such that } T \xrightarrow{u} T', \exists S' \text{ agent }, (S',T') \in (R \cup \cong)^*, S \xrightarrow{u} S'$$

then $R \subset \cong$.

All this means is one is allowed to guess what part of the (huge) strong congruence one will need in a case exemple, then prove it formally -and by induction-, with possible use of already gained and established results.

1.4. Expressions

As usual for calculi, expressions (or **open terms**, or **contexts**) may be introduced in Meije-SCCS, using a second set of variables, different from the syntactic "with recursively defined behaviours" variables, and for this thereafter called **metavariables**. Contexts are said to be equivalent when all the closed terms obtained by instanciating their metavariables with closed terms of the algebra are so. Remember here that still an alphabet is always referred to, out of which no action exists, at least externally visible. There is a slight trick, alike an α-conversion or *bound variable renaming*, in the definition of substitution (*instanciation*), which helps validating equations that seem natural. Development of this may be found in [1984a]

The reason to introduce expressions in the calculus is that, although it may not be complete, equational deduction can often improve a great deal strong-equivalence proofs. Equations naturally deal with expressions. But, most of all, the calculus will be (hopefully) mainly used in the future to prove equivalence in between specifications and realisations, and variously abstracted versions of the same device, be it a protocol or or a VLSI chip. See [Bac1983a] and [Hen1984a]. These will generally be *open* towards the exterior, and so parameterized with "unknown" other processors. Thus proofs of equations in between expressions is at the very heart of the system. It becomes even more so if one introduces, as we will later, new operators with which to express more abstractly specifications of processes.

There was no equivalent to Park's principle in Meije-SCCS concerning (open) expressions. And effective proofs were far from obvious, because of that "..*iff for all substitutions*..." requirement. Elements of answers showed while inspecting how "human" proofs of equivalence and equations were handled. We will deal with this problem, in a restricted frame of particular (but every useful) expressions.

1.5. Architectural expressions

We want here to define a certain type of expressions in the calculus, allowing (meta)variables only in a very specific way, that is out of recursive agent definitions, so that the corresponding instanciating processes be clearly positioned into space. SeeMil1979a . The expressions will then be "finite" expressions, at least around the (meta)variables. These will appear linearly, being thought of as mere positional variables.

As usual, we will suppose the existence of a countable set of parameters -or rather metavariables-, noted $p, q, \ldots$. They can be apposed indexes.

Definition

We call **architectural expression** any Meije-SCCS expression containing parameters and belonging to the class $\mathcal{R}$ so defined:

1) *Meije−SCCS* $\subset$ $\mathcal{R}$. This so recursive definitions be allowed anywhere in closed terms, and closed terms only.

2) For all p metavariables, $p \in \mathcal{R}$.

3) For all $E, F \in \mathcal{R}$, such that $Metavar(E) \cap Metavar(F) = \phi$, $a:E$, $E\backslash s$, $s*E$, $s \Rightarrow E$, $E \| F$ (and other SCCS "finite" operators) belong to $\mathcal{R}$.

Metavariables indeed appear linearly inside such an expression.

The restriction on recursive definitions is certainly too strong for the desired effect, which is mostly not to allow duplication of metavariables. Nevertheless it has a nice simple formulation and corresponds to an intuitive notion of parameter processes appearing only in stable, static expressions, not evolving dynamically. The technical purpose of this choice is that, then, the multiplicity of "deduction trees" for the expression's behaviours will not come from a possible outburst of many parameter instances, each having -over the quantification- behaviours mutually related or not.

There is a very partial *normal form* result on architectural expressions:

Proposition

Every architectural expression $T[p_1, \ldots, p_n]$ may be put under the form

$$[\ (\overline{\alpha}_1 * <\Phi_1>p_1) \| \cdots \| (\overline{\alpha}_n * <\Phi_n>p_n) \| Synchronizer\ ] \setminus U_n$$

where for each i we let Φ_i be the morphism $\mathbf{M} \longrightarrow \mathbf{S_i}$ defined by :

$\forall\, u \in A$, $\Phi_i(u) = \alpha_{u,i}$ if u is an action or a positive (unbarred) signal, and

$\forall\, \overline{u} \in A$, $\Phi_i(\overline{u}) = \overline{\alpha}_{u,i}$ if $\overline{u}$ is a negative signal.
Here $S_i = \{\alpha_i\} \cup \{\alpha_{a,i} \,/\, a \in A\}$. We noted by U_n too the set
$U_n = \bigcup_{1 \le i \le n} S_i$ [1].

The proof of this result will be a corollary to the representability theorem of page ???. It shows to be sufficient (if not necessary) to face the problem of expressions equivalence to turn it into this of **closed terms** equivalence, checking only the *Synchronizers*. But these are closed processes on an extended alphabet -including the new *alpha* signals...-, that exceeds the permitted sorts. *Synchronizers* generally are able to present far more non-deterministic behaviours than is to check, the parameter processes providing the choice in "real" situations. This technique is thus certainly amenable to equational proofs through rewritings mostly.

[1] Notice U_n is a *finite* set of signals, and all the introduced signals being -by restriction on U_n- internal, the casualties of their names are irrelevant. More on terms being of that normal form may be found in [Sim1984b]

One way this proposition's result can be made intuitive is by thinking the resulting *Synchronizer* consists of the term itself, where real parameters were replaced by relays, which take orders (through the newly added signals) from the instanciating processes and propagate this information inside the system. Those relays are not "substituting" processes inside the term because their sort exceeds the one there allowed (the alphabet). They are non-deterministic clocks -see [Bou1984a] - *copying* sent orders back into real actions.

1.6. Process Calculi

The previous proposition brings forth an important remark on how equivalence proofs may be conducted in such normal forms. They will split into:

- checking fulfillment of hypothesis for the processes-arguments on one hand,
- devising (or discovering) the behaviour of the closed *Synchronizer* on the other,
- putting it all together to devise the behaviour at large.

If one keeps the metavariables un-instantiated, this processing appears as a transformation of the (variable) specifications of arguments into a behavioural specification of the compound term. One may then try to synthetize the *specification* of the expression, considered as a new higher-level derived Meije-SCCS operator. Specifications will be semantic rewrite rules of the very fashion of the early ones (see previous tables). The construction will be compositional, as one can devise more expressions on the just defined new operators. So architectural expressions become operators of a more abstract new calculus, in which architectural expressions may in turn be devised.

We are led to define a general notion of process calculus, mainly by **syntactically** providing a way to formulate semantic rules for new operators, already called *specifications*. Notice we shall not force the original Meije-SCCS operators to figure amongst the ones of a specific calculus. So one can define notions of sub- and super-calculus, calculi being respectively more or less expressive, and so on.

One can define recursive terms in these calculi the way it is done in SCCS. We shall still call architectural expression an expression where metavariables appear linearly outside of recursive definitions. Actually, keeping with the same trend we should replace recursive terms with recursive definitions, some expressions "feeding themselves" as premisses.

Definition

A Process Calculus is a triple $(F, M, Spec)$, where

F is an operator family

$\mathcal{A}$ is an action set. We shall here content with the simple action/signal monoïd over an alphabet. Same we will not consider typing operators with different action sets.

Spec

is a function assigning to an operator a specification. We shall deal next with what the universe of specifications is.

1.6.1. Specifications

If we consider how primitive operators see their operational semantics defined by *conditional rewrite rules* following their algebraic structure, we are struck by the fact that the three main aspects the calculus aimed to model are each very neatly cornered in some specific feature of the concerned inference rule. These three aspects are:

non-determinism (operator +.)

concurrency (operator ×.)

Those two notions being partially recovered by Meije operators || and s^*.

synchronisation/cooperation (restriction and renamming operators.)

For instance, **non-déterminism** is borne in the (finite) multiplicity of rules defining a single operator: 2 for +, 3 for ||. **Concurrency** shows in rules requiring in their condition to establish the functioning of more than one subprocess, and more generally in the fact that some operators do actually work on more than one subprocess, in a sustaining way (unlike +): × and || of course, but eventual non-primitive interleaving operators as well. **Synchronisation/cooperation** reflects in the presence of a required -or dually forbidden- relation holding
- in between actions performed by the subprocesses: *restriction*, and
- in between them and the resulting compound action: *renaming*.

Our proposed syntax for new operators rules will take advantage of these characteritics, and try to extend them to their full generality **inside** the calculus. Another important thing to retain is the agreement of the new operators with the congruence philosophy.

Definition

The general form of specification we will allow for each operator F will consist in **conditional behaviour rules**. A conditional behaviour rule rg is an object of the following shape:

$$\frac{(\ \forall\, i_j \in S,\ p_{i_j} \xrightarrow{u_j} p'_{i_j}\ ),\ \&\ \ Pr(u_1,\ldots,u_l,v)}{F(p_1,\ \cdots\ ,p_n) \xrightarrow{v} \mathrm{T}}$$

where $S = \{i_1, \ldots, i_l\} \subset \{1,\ldots,ar(F)\}$, Pr is a $(l+1)-ary$ relation on the action monoïd M, that is a subset of M^{l+1}, and T is an architectural expression of the process calculus. Indeed -due to the calculus- T may only contain **linearly** those

of the p_i's that were not required acting in the condition, and the p'_i's that were due to act in that same condition. This syntactic restriction seems reasonable enough in a lot of cases: if performance of a global action by a system requires performance of particular action by a subprocess, upon this move all we are left with is the resulting subprocess. There is no backtracking on the part of the components, no free-from-choice testing by the system of potential situations that may not occur.

Actually the triple $(S,Pr,\mathbf{T})$ is all there is needed to entirely characterize a rule. The graphic shape, despite all its redundancy, is still perhaps more talkative. In the future the parameters p's being merely positional, we shall replace p' by p so variables do not get duplicated. It should just be understood that were action is requested on a component, it is the resulting term that instantiate the metavariable in T. By convention, the last component in elements of Pr will **always** be the resulting compound action, denoted v, as other components will be the behaviours of the parameters.

As we previously mentioned, no particular syntax is provided for the "semantic" class in which Pr takes its values. Because of the clear cut in between the various satisfaction requirements involved in applying a rule, this relation can be checked slightly "*off the side*" of the structural deduction system for behaviours, and so accommodated to various syntactic computing systems as defined in general recursive theory, in case of "effective" relations. Still a general purpose propositional logic on the u_i's and v variables, with such atomic predicate as "equals in M" or "divides in M", with product and projections on subalphabets as operators, would certainly meet most needs -and at least the semi-linear relations-.

Our specifications make the strong equivalence a congruence with respect to the new operators. This is due to the fact that the compound behaviour only depends on the abilility of the subprocess to perform a specific action, and not on its syntax. So the condition works at the underlying process/transition systems level.

We now give instance of operators that (trivially) fall into our class:

* Let fib be the famous Fibbonacci function, and $A = \{a\}$. Then

$$\frac{p \xrightarrow{u} p' \,,\, \{(u = a^n, v = a^{fib(n)}) \,/\, n \in \mathbf{N}\}}{F(p) \xrightarrow{v} F(p')}$$

  is a specification of a context realizing a (parallel) fibbonacci transformation on behaviours of the processes to instanciate p.

* Let ack be the equally famous Ackermann function. Then

$$\frac{p \xrightarrow{u1} p' \,,\, q \xrightarrow{u2} q' \,,\, \{(u = a^m, a^n, v = a^{ack(m,n)}) \,/\, m,n \in \mathbf{N}\}}{F(p,q) \xrightarrow{v} F(p',q')}$$

This context synthesizes a signal of "*intensity*" $ack(m,n)$ in parallel from the input behaviours provided by the process put in place of p and q.

1.6.2. Building a specification from an architectural context.

Our next task will be to show how to build the specification of an architectural context (as operator of a higher-level calculus) in function of the (basic) specifications of the operators (of the more primitive calculus) that compose the expression. It will naturally proceeds structurally along the expression. The principal use of such a construction is that, in our opinion, showing equivalence of expressions is doomed to a major obstacle: one has to deal with the quantifying "iff for all process instantiations...". This may be released only through specification comparaison, if it may at all.

We shall note Sp the function that provide any architectural context a specification. For F an operator, we shall note Sf its specification $Sp(F)$. We shall index the three constituents $(S,Pred,T)$ of a rule of such an operator specification with an r, when it is clear what operator is dealt with. For specifications of the different sons of a term with F at its root, we shall index with numbers the corresponding components of their semantic rules. Then

Metavariables

$$Sp(p_i) = \left\{ \frac{p_i \xrightarrow{u} p_i \ , \ \{(a,a) \,/\, a \in M\}}{p_i \xrightarrow{v} p_i} \right\} \qquad \text{(axiom)}$$

Operators

Let $Arch(p_1,\cdots,p_n) = F(Arch_1(p_i,i\in I_1),\ldots,Arch_k(p_i,i\in I_k))$, where the $Arch_i$ are architectural expressions too and the I_j are disjoint sets whose union is $\{1,\ldots,n\}$. I_j may be empty in case of closed terms. Then

$$Sp(Arch) = \bigcup_{r\in Sf} Sp_r(\vec{S}p(Arch_j, j\in S_r))$$

This simply means that the construction of a specification for the term breaks down along the different rules of F. Now for the function Sp_r defined for each rule of F, it splits aong the different rules of the $Arch_i$ as well:

$$Sp_r(\vec{S}p(Arch_j, j\in S_r)) = \bigcup_{rj\in Sp(Arch_j), j\in S_r} \left\{ \frac{\bigcup S_j \ , \ Pred}{Arch \xrightarrow{v} T_r(T_j, j\in S_j)} \right\}$$

where, if we let $l_i = j$ if $i\in I_j$,

$$Pred = \left\{ (a_1^{l_1},\ldots,a_n^{l_n},a) \in M^{n+1},\ \exists b^j,\ (a^j_1,\ldots,a^j_{l_j},b_j)\in Pred_j,\ (b_1,\ldots,b_k)\in Pred_r \right\}$$

Notice this processing, mostly a composition -in a relational, not functional

case- is effective in case of **finite** specifications, letting aside calculation of specification for closed recursive terms, which we leave for further research.

1.7. Proving strong equivalences of architectural expressions through their specifications

We want to follow the same patterns as Milner, recalled shortly in (1.3 ?). First we need to define a "bisimulation under formal hypothesis", or *FH-bisimulation*, stating relations in between open architectural expressions.

Definition

Let R be a binary symetric relation in between architectural expressions. Then R is an *FH-bisimulation* iff:

$\forall\ T, U$ such that $R(T,U)$

1) $Metavar(T) = Metavar(U)$ [1]

2) $\forall\ (S,Pred,T')$ *specification rule* for T,
$\exists$ *specification rules* $(S,Pred_i,U_i')$ for U
such that $\forall\ i, R(T',U_i')$ and $Pred \subset \bigcup_i Pred_i$

One can then show there is a largest bisimulation, union from all the others. It is a congruence (for the operators, and for expression composition), and is closed by instanciation. We shall call this relation *strong congruence under formal hypothesis*, or *FH-stong congruence*. We shall note $T \overset{FH}{\cong} U$.

Definition (Park's Induction Principle for Architectural Contexts under formal hypothesis)

Let R be a symetric relation (which we want to prove part of a FH-bisimulation). Then if for every couple (S,T) of Meije-SCCS architectural expressions in R the following holds:

$$\forall\ (S,Pred,T')\ \textit{specification rule}\ T,$$
$$\exists\ \textit{specification rules}\ (S,Pred_i,U_i')\ \text{for}\ U$$
$$\textit{such that}\quad \forall\ i, (T',U_i') \in (R \cup \overset{FH}{\cong})^*\ \text{and}\ Pred \subset \bigcup_i Pred_i$$

[1] where *Metavar(T)* is the set of metavariables occurring in T. Note one can always reduce to that case by using an instantiation bringing all unshared metavariables down to 0, and keeping others untouched.

where we noted by R^{σ} the closure of R under instantiation of metavariables.

Then $R \subset \overset{FH}{\cong}$.

Note Neither the *FH*-bisimulation nor the induction principle associated stress finiteness requirements on specification rules.

But defining this new equivalence notion for expressions is of worth only if we can prove we are still dealing with the old one.

Theorem

$$\boxed{\begin{array}{c} T \overset{FH}{\cong} U \quad \textit{only if} \quad T \cong U, \\ \text{that is} \\ (\overset{FH}{\cong}) \subset (\cong). \end{array}}$$

Proof

Note We should keep in mind

$$\frac{S\ ,\ Pred}{T \overset{v}{\longrightarrow} T'} \Leftrightarrow$$

$$\left| \begin{array}{l} \forall\ (a_1,\ldots,a_l,a) \in Pred, \\ \\ \dfrac{p_{i_1} \overset{a_1}{\longrightarrow} p'_{i_1} \ldots, p_{i_l} \overset{a_l}{\longrightarrow} p'_{i_l}}{T \overset{a}{\longrightarrow} T'} \end{array} \right.$$

We shall prove our claim using the former Park's induction principle for the $\cong$ relation, and posing $R = (\overset{FH}{\cong})$ there in the hypothesis.

So let us suppose $T \overset{FH}{\cong} U$. Let n be T (or U)'s arity.

Let $[\sigma] = [T_1/p_1, \ldots, T_n/p_n]$ be any closed substitution.

Consider any particular deduction tree skeleton for a behaviour $[\sigma]T \overset{a}{\longrightarrow} T''$. It does necessarily contain sub -trees rooted at behavioural deductions of the form $T_{i_j} \overset{a_j}{\longrightarrow} T_{i_j}'$ (and indeed coming from T_{i_j}'s substituted for p_{i_j}'s).
Call S the (finite) set of i_j's for which such a deduction for T_{i_j} appears. One can build a new deduction tree by replacing each $T_{i_j} \overset{a_j}{\longrightarrow} T_{i_j}'$ by $p_{i_j} \overset{a_j}{\longrightarrow} q_{i_j}$ $(i_j \in S)$. The resulting deduction tree is a proof of the rule

$$\frac{p_{i_1} \overset{a_1}{\longrightarrow} q_{i_1} \ldots, p_{i_l} \overset{a_l}{\longrightarrow} q_{i_l}}{T \overset{a}{\longrightarrow} T'}$$

and we gather $T'' = [\sigma']T'$, where $\sigma'(p_i) = \sigma(p_i) = T_i$ and $\sigma'(q_i) = T_i'$.

We can now assert the existence of a specification rule $\frac{S \ , Pred}{T \xrightarrow{v} T'}$ such that $(a_1, \ldots, a_l, a) \in Pred$. *see note above.*

Since $T \overset{FH}{\cong} U$, there exists at least **one** specification rule for U of the form $\frac{S \ , Pred'}{U \xrightarrow{v} U'}$ with $(a_1, \ldots, a_l, a) \in Pred'$, and $T' \overset{FH}{\cong} U'$.

From the closure properties of $\overset{FH}{\cong}$ towards instanciation, we get

$$[\sigma']T' \overset{FH}{\cong} [\sigma']U' \qquad \text{(a)}$$

At this point we **do possess** the candidate resulting term, namely $[\sigma']U'$, to fulfill Park's principle requiremement -since (a)-. We do need to prove $[\sigma]U \xrightarrow{a} [\sigma']U'$. This is trivial since, from the specification rule (or rather the deduction tree it is obtained from) $\frac{S \ , Pred'}{U \xrightarrow{v} U'}$ one can construct a deduction tree for

$$\frac{p_{i_1} \xrightarrow{a_1} q_{i_1} \quad \ldots \quad p_{i_l} \xrightarrow{a_l} q_{i_l}}{U \xrightarrow{a} U'}$$

Then, replacing all axioms $p_{i_j} \xrightarrow{a_j} q_{i_j}$ by the sub-tree collected from the early deduction tree of $[\sigma]T$, and rooted there in $T_{i_j} \xrightarrow{a_j} T_{i_j}'$, one gets a proof tree for $[\sigma]U \xrightarrow{a} [\sigma']U'$.

▪ Q.E.D.

Once set the adequation of our new *operational* congruence definition for architectural expressions, one may wonder about its completeness. We provide below two counterexample why it does not hold.

$(\cong) \not\subset (\overset{FH}{\cong})$.

* Consider the (closed) term H_M, the clock over the whole monoïd of actions. It is a Meije agent as soon as the alphabet is finite. It is in a certain sense a "maximum" term in Meije, since it can perform **any** action and remain potentially the same. It "swallows" **any** term set in parallel with it:

$$p \,||\, H_M \ \cong \ H_M \ \cong \ H_M \,|\, p$$

the operator $|$ being classical interleaving (without simultaneous actions). Note that this result can be "intelligently" showed applying the earlier Park's induction system on all couples $\{(T||H_M \ , H_M), T \text{ closed Meije term}\}$ and $\{(T\,|H_M \ , H_M), T \text{ closed Meije term}\}$

But trivially $p \| H_M$ has a specification showing as premisses some hypothesis on both p and the clock, while $p \mid H_M$ does not. The basic equivalence comes from the fact that such a specification is here superfluous.

* Consider the case where the original alphabet for processes consists in 1 only. Then
$T_1 = (\alpha^* p \parallel 1 : \overline{\alpha} : \mathbf{0}) \backslash \alpha$ is strongly congruent to $T_2 = (\alpha^* p \parallel (1 : \overline{\alpha} : \mathbf{0} + \overline{\alpha} : 1 : \mathbf{0})) \backslash \alpha$ as may appear by bisimulation on instantiated expressions. Nevertheless

$$\frac{p \xrightarrow{1} p'}{T_2 \xrightarrow{1} (\alpha^* p \parallel 1 : \mathbf{0}) \backslash \alpha}$$

is informally a specification for T_2 while T_1 only obtain an equivalent behaviour by **not** running its argument, **but** while remembering it will be entitled to a 1-move. The latter has no specification using an hypothesis on p: specifications do not match.

We believe the two previous example to be highly generic, so a simple reduction procedure on specifications removing bad ones should allow completeness of equivalence on so processed sets of specifications.

2. A representability theorem.

We show how higher-level operators can be realized in the "basic" Meije-SCCS calculus, under finitary assumptions.

Theorem

Let A be a finite alphabet of actions.

Let $\{ F_1, \ldots, F_m \}$ be a finite set of new operators symbols.
We shall note ar(j) the (finite) arity from F_j.

Let *Spec* be a semantic specification for these operators. Suppose furthermore each operator is defined through a finite number of rules, in which appearing predicates (allowing combination of actions) always are **recursively enumerable relations on the action monoïd.** Let $\mathring{A}$ be the resulting calculus.

Then

$\mathring{A}$ may be realized in Meije-SCCS, that is
for any F_j , there exists a Meije-SCCS architectural expression C_j (with same arity) , realizing

$$C_j\ [p_1,\ldots,p_{ar(j)}] \overset{FH}{\cong} F_j\ (p_1,\ldots,p_{ar(j)})$$

Proof

This proof is the goal result from this paper. Recall $\overset{FH}{\cong} \subset \cong$. For sake of room we shall skip the proof itself, dealt at length with in [Sim1984b] , in an earlier version. We will rather focus on the **construction** of the candidate context. We shall enlight eventually which of its features may intuitively come for use in which aspect of the formal proof. Following the proof we'll provide the same construction, in the particular case of the exemples cited before, *fib* and *ack*. This will allow the reader unfamiliar with SCCS to notice a certain number of feature we shall make extensive use of in the general construction.

The C_j construction.

Let $n = ar(j)$.

We want to build an architectural context of our previous *normal form* nature, safely setting apart the parameters from the synchronisation and rule-verification mechanisms. While safe, general and *normal*, this form is often a less effective one for synchronisation, very centralized. Many syntactic restrictions concerning allowed rules may lead to more distributed classes of realizing contexts. We will not get here into this "optimisation" problem.

$$C_j\,[p_1,\ldots,p_n] = [\,(\overline{\alpha}_1 * <\Phi_1>p_1) \parallel \cdots \parallel (\overline{\alpha}_n * <\Phi_n>p_n) \parallel y_j \textit{ where } Synchr\,] \setminus U_n \quad [1]$$

$$\cong \mathbf{Arch}(p_1,\ldots,p_n,\, y_j \textit{ where } Synchr\,)$$

where all we mean by this is that C_j is a context architecture around the p_i's, renamed with internal new signals -for instance the *alphas*-, and conversing each with a central *synchronizer* which is made out of mutually recursive definitions on (syntactic) variables, as many variables as new F_j operators introduced. Our main duty will be to construct it right...

2.0.1. The synchronizer *Synchr*

We shall see it as a list of mutually recursive variable definitions, whose solution component we choose corresponds to the operator we want to simulate. This since we allow different operators, different synchronisation modes, to reproduce or lead to one another in time.

We did introduce one new recursive variable y_j for each symbol $\mathbf{F}_j$ (In fact, since we bind all of their occurrences, their names are irrelevant).

$$Synchr = (\, y_1 = Syn_1\,;\ \cdots\ ;\, y_m = Syn_m\,)$$

For each i, the term Syn_i (which may contain some y_k), will be built from $\mathbf{F}_i$ specifying rules: if $r(i)$ is the number of conditional behaviour rules defining $\mathbf{F}_i$, Syn_i consists externally of a global non-deterministic choice (+) in between terms each composed from a different rule.

$$Syn_i = \sum_{k=1}^{r(i)} Sy^{(i,k)}$$

We have broken the problem down to an extremity: each expression $Sy^{(i,k)}$ corresponds to an only rule.

---

[1] Definitions of the Φ_j morphisms and U_n as before.

2.0.2. The process Sy from a rule.

We shall omit the i and k indexes further on.
As mentioned earlier, each rule -in our sense- can be entirely described by three elements:

1. A subset S from $\{1,...,n\}$, where n is F_i's arity.
2. A $l+1$-ary recursively enumerable predicate Pr, where $l = Card(S)$.
3. A continuation term T

Starting from each of these elements we shall build a process -or a compound action-.

2.0.2.1. The compound firing signals (*from* S).

Let $\underline{\alpha} = \prod_{i \in S} \alpha_i$.

The action $\underline{\alpha}$, once performed by Sy and thus by the synchronizer, is entitled to verify the required effective participations on the part of the p_i processes, as mentioned in the rule. Obviously the right matching $\overline{\alpha_i}$'s have to be performed, and each of them imply the corresponding process to be active.

2.0.2.2. The process verifying the Pr relation property

We showed elsewhere [Sim1984b] how for **any** recursively enumerable predicate U, $U \subset M$, one could build in Meije-SCCS a process P_U realizing:

$$P_U \cong \sum_{u \in U} u:0 \ ,$$

in other words a process whose range of immediate behaviours is exactly comprised in U, and who dies afterwards. This can be easily extended, using l copies from our initial alphabet A, to the case where $\vec{U}$ is a subrelation (a predicate) from M^{l+1}. To this end, we use an obvious isomorphism separating the coordonates alphabets so they can safely be embedded in the same monoïd on the union of all alphabets:

$$M^{l+1} \longrightarrow B$$

$$\vec{u} = (u_1,...,u_{l+1}) \longrightarrow \overline{\Phi_1(u_1)}.\cdots.\overline{\Phi_l(u_l)}.u_{l+1}$$

where B is the desired commutative monoïd generated by A and its l repliquas, which we will suppose consist of *alpha bar* signals properly subsripted with j for the j^{th} copy, and with a for the signal miming a. Then the Φ_i morphisms are as before. The reason why we named the copying actions *alpha bars* should by now be obvious:

Given Pr, we build P_{Pr} as above. Then in order for this process to perform $\dot{u} = \overline{\Phi_1(u_1)}.\cdots.\overline{\Phi_l(u_l)}.u_{l+1}$ the processes instantiating the p_j, once renamed by their Φ_j's, should each perform $\Phi_j(u_j)$, and thus the processes themselves, unrenamed, perform u_j. Thus their behaviours fulfill the Pr requirement.

2.0.2.3. The resulting term (*from T*).

We shall build a new term $c(T)$ from T, by structural induction. c will therefore be a morphism of algebra ... from Å to Meije-SCCS.
Mainly, this operation will consist in replacing every occurrence of the F_j by the proper syntactic variables y_l, and to set *-through "relay-processes"-* the right arguments to their ruling. This is meant, in particular, for the parameters p_i appearing in T.

** If T is a metavariable $p_i, 1 \leq i \leq m$, then

$$c(p_i) = \alpha_i * H^{*}_{\{\overline{\alpha}_{ui}u \,/\, u \in A\}} \quad [1]$$

This process acts as a relay, using the *alpha*'s signals to receive the value of the action to perform (from the true "outside" parameter), and synthesing it then (for production "inside" c(T)).

** If $T = F_j\ (T_1, \ldots, T_n)$, then :
let $\varphi = (\gamma_i/\alpha_i\ ,\ \gamma_{u,i}/\alpha_{u,i}\ )$ be an isomorphism from the -large- monoïd generated by U_n into a disjoint copy, and $\psi = (\ \alpha_i/\gamma_i\ ,\ \alpha_{u,i}/\gamma_{u,i}\ )$ be the inverse morphism,

$$c(T) = <\psi>[\ (\overline{\alpha}_1 * <\Phi_1><\varphi>c(T_1)) \parallel \cdots \parallel (\overline{\alpha}_n * <\Phi_n><\varphi>c(T_n)) \parallel y_j\ ] \setminus U_n$$
$$\cong <\psi> \mathbf{Arch}\left(<\varphi>c(T_1),\ldots,<\varphi>c(T_n),\ y_j\right).$$

The reason for this trick, renaming *alpha*'s signals to make them "cross" the restriction, is that the inner *alpha*'s, requiring there a summoning value from the exterior metavariables, should not be confused with other potential ones, sent by the "interior" arguments, now considered as instanciating parameters, to *impose* then **their** own behaviours to deeper process listeners. As we mentioned earlier, the names of the used signals guaranteeing the restrictions are irrelevant, *as long as* they are new. For syntactic aspects we always use the same, being therefore forced to separate their different meanings through renaming.

This painful operation -dynamic renaming of the channels of an evolving,

[1] H^{*} here is a non-deterministic clock on the whole monoïd generated by the subscript set. See [Bou1984a]

perhaps without time consuming, recursive architecture- is an important feature of our constructions and the reader's attention is called up to it, as the realm of its implications inside the model is a critic issue for the latter's full credit. See [Hen1984a] for instance.

We are now able to define Sy, a closed term which may contain free *syntactic* variables (that is, purposefully recursive variables):

$$Sy = \left[s => (\underline{\alpha} \Rightarrow (P_{Pr})) \,||\, \overline{s} : c(T)\right]\backslash s \quad {}^{1}$$

This definition seems clear according to the previous ones. As promised we will skip any attempt of formal proof for the theorem. It may be found in [Sim1984b].

Notice recursive definitions in Å are being taken care of with our result: one may then translate the expressions appearing in recursive definitions into Meije-SCCS, then reset the recursive structure around them, instantiating the free metavariables into syntactic variables and binding them in recursive definitions afterwards. These expressions (appearing in recursive definitions) may always be trivially supposed to contain linearly their variables by introducing *aliases* amongst the syntactic variables.

Examples

We work in a manner *similar* to the preceding construction the exemples of the two classic recursive functions of *Ackermann* and *Fibonacci*. Specifications for operators based on them were given earlier. We realize them with Meije-SCCS contexts. The method is only similar to the general theorem since the usual recursive patterns of these functions do not follow primitive recursion shemata, and so on ...

Recall Fibonacci's function fib

$$\begin{cases} fib(0) = fib(1) = 1 \\ fib(n+2) = fib(n+1) + fib(n) \end{cases}$$

[1] Here the signal s only serves to shedule both sides of this term: the leftmost checking a particular conditional behaviour rule being applied; the rightmost composing with the future afterwards.

The corresponding Meije context will be

$$C_{fib}\ [p\,] = [\ <\alpha/a>p \times Fib\ ]\backslash a$$

where

$$Fib = rec\ y.\left[\overline{\alpha}^2 \Rightarrow y\ \&_{\alpha^*}\ \overline{\alpha} \Rightarrow y\ \right] + \overline{\alpha}.a : \mathbf{0} + a : \mathbf{0})$$

Here the auxiliary operator $\&_B$ is a *rendez-vous* operator, as defined in [Mil1983a] , and whose operational semantics clearly falls into our frame of derived operators.

Ackermann's function is defined by

$$\begin{cases} Ack(0,n) = n+1 \\ Ack(m,0) = Ack(m-1,1) \\ Ack(m+1,n+1) = Ack(m,Ack(m,n-1)) \end{cases}$$

Then the architectural context

$$C_{Ack}\ [p\,,q\,] = [\ <\alpha/a>p \times <\beta/a>q \times ACK\ ]\backslash a\,,\beta$$

where

$$ACK = rec\ z.\ \Big[\ \Big[<\beta/\gamma>\Big[\overline{\alpha} \Rightarrow z\ \&_{\alpha^*} <\beta/a>(\overline{\gamma} \Rightarrow\ (<\gamma/\beta>z))\ \Big]\backslash\beta\ \Big] + (\overline{\alpha}.\beta \Rightarrow\ z\ )\backslash\beta + a \Rightarrow\ \mathbf{T}_{\overline{\beta}.a}\ \Big]$$

realizes the specification, as the reader can check out.

3. A semantic theorem

Up till now, this work has mainly aimed at representing syntactically presented objects, namely higher-abstract operators. Yet there was an important "semantic" feature appearing in the class of rule allowed, the predicate setting relation between behaviours of components and compound resulting process. As remarked by G.Boudol, one can benefit from this aspect to produce a strong theorem on the calculus "universal" ability to represent any kind of "effective" transition system on a given monoïd M of our precedent kind for labels.

Definition

Let M be an "actions and signals" monoïd. A transition system (with labels in M) (Q, M, T) is called *recursively enumerable* iff:
the states set Q is denumerable, note it $\{q_0,\ldots,q_n,\cdots\}$, with q_0 as initial state.
The transitions set T is such that, if we introduce two new signals α and β : $U = \{\overline{\alpha}^i.u.\beta^j \;/\; (q_i,u,q_j) \in T\}$ is recursively enumerable. This last notion, calculability in the actions monoïd, has been defined precedently using Parikh's mapping.

Theorem

Every recursively enumerable transition system can be realized by a Meije-SCCS term, that for any such transition system there is a term having it for operational semantics.

Proof

It was said before that for such a set as U one could devise a process P_U specified by its only behaviours:

$$\forall\, u \in U,\ P_U \xrightarrow{u} 0.$$

Then the process R_U defined as x *where* $x = P_U \times 1 : x$ is operationaly specified by:

$$\forall\, u \in U,\ R_U \xrightarrow{u} R_U.$$

All is now needed is a synchronisation, involving only α and β signals, to ensure that transitions are applied only from the state we just arrived to at the last previous step. To this end, we will figure in R_U the exponent of the $\overline{\alpha}$ signal "means" the index of the state we leave and reciprocally the one from β "means" the state we arrive in. The desired synchronizing process shall be then such that anytime it performs an action, it is of the form $\alpha^i.\overline{\beta}^j$, and that i equals the j from the previous step.

$$Synchr = x \text{ where } x = \left[(\overline{\beta}:\alpha:0) \times x\right] + 1 : x$$

We shall leave as exercice to the reader to show that the presented term indeed fulfills the (informal) specification. Give formal specification of such a term is actually very hard, in our formalism -notice how the recursive expansion in space subsist for two successive instants-, and in any other as well -see about a file-process, of a (remotely) comparable form.

The process having (Q,M,T) as semantics is then

$$(R_U \parallel Synchr)\backslash \alpha, \beta \ ^{1}$$

---

[1] Note that the initial step may not contain any α or $\overline{\alpha}$ signal, which amounts to α^0. Thus we must start from q_0 indeed.

4. Conclusions

The new introduced operators should ease specification in the calculus, and provide a framework to define different calculi as well. Future researches should include: a release of the "architectural" condition -at least in the operational equivalence for expressions-, a study of its completeness -through a normalization of specifications-, and introduction of recursive specifications -held the same as simple ones-. We should explore the semantic "calculability" power of expressions, that is what class of transformations on multiples r.e. transition systems can be devised in the calculus.

Acknowledgments

The author would like to thank G. Boudol for his support.

1979a. *Net Theory and Applications*, Springer-Verlag (1979).

1984a. G. Boudol , "Notes on Algebraic Calculi of Processes," in *Advance Nato Study Institute on logics and models for verification and specification of concurrent systems*, Springer-Verlag (1984).

1981a. G. Plotkin , *A Structural Approach to Operational Semantics*, University of Aarhus (1981). Lecture Notes

Bac1983a. R. Backhouse, "Specification and Proof of a Regular Language Recogniser in SCCS," Research Report CSR-130-83, University of Edinburgh (1983).

Bera. J.A. Bergstra and J. Tiuryn, "Process Algebra Semantics for queues," Dep^t Comp. Sc. Techn. Rep^t IW 241/83, Mathematisch Centrum, Amsterdam. .

Bou1984a. G. Boudol and D. Austry, "Algèbre de Processus et Synchronisation," *T.C.S.* **30**(1) (1984).

Bou1984b. G. Boudol, G. Roucairol, and R. de Simone, "Petri Nets and Algebraic calculi of Processes," Rapport de Recherche I.N.R.I.A. $n°$ 292 (1984).

Bro1984a. S.D. Brookes, C.A.R. Hoare, and A.W. Roscoe, "a Theory of Communicating Sequential Processes," *J.A.C.M.* **31** (1984).

Elg1975?a. C.C. Elgot, "Monadic Computations and Iterative Algebraic Theories," in *Proceedings, Logic Colloq. Bristol 73*, North-Holland (1975?).

Hen1984a. M. Hennessy, "Proving Systolic Systems Correct," Research Report CSR-162-84, University of Edinburgh (1984).

Mil1979a. R. Milner, "Flowgraphs and Flow Algebras," *J. ACM* **26** (1979).

Mil1983a. R. Milner, "Calculi for Synchrony and Asynchrony," *T.C.S.* **25**(3) (1983).

Mil1984a. R. Milner, "A Complete Inference System for a Class of Regular Behaviours," *J.C.S.S.* **28** (1984).

Par1966a. R. Parikh, "Language Generating Devices," *M.I.T. Res. Lab. Electr. Quart. Prog. Rept. 1961*(60), réédité dans J. ACM 13 (1966).

Sim1984a. R. de Simone, "On Meije and SCCS: Infinite Sum Operators vs Non-Guarded Definitions," *T.C.S.* **30**(1), Note (1984).

Sim1984b. R. de Simone, *Calculabilité et expressivité dans l'algèbre de processus parallèles Meije*, Jussieu Paris 7 (1984). Thèse de 3 ième cycle

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique