The Chemical Abstract Machine

Gérard Berry*

Ecole des Mines
Sophia-Antipolis
06560 Valbonne, France

Gérard Boudol

INRIA
Sophia-Antipolis
06560 Valbonne, France

Abstract

We introduce a new kind of abstract machine based on the chemical metaphor used in the Γ language of Banâtre & *al.* States of a machine are chemical solutions where floating molecules can interact according to reaction rules. Solutions can be stratified by encapsulating subsolutions within membranes that force reactions to occur locally. We illustrate the use of this model by describing the operational semantics of the TCCS and CCS process calculi. We also show how to extract a higher-order concurrent λ-calculus out of the basic concepts of the chemical abstract machine.

1 Introduction

We present the notion of a *Chemical Abstract Machine*, suited to model asynchronous concurrent computations. We show that chemical abstract machines can "implement" known models of concurrent computation such as algebraic process calculi [18,6] and a concurrent λ-calculus similar to the one presented in [7].

Abstract machines are widely used in the classical theory of sequential computations. Turing Machines or Random Access Machines are primary tools within the theories of recursive functions and computational complexity. The SECD machine [17] and the Categorical Abstract Machine [10] are used to study and implement the λ-calculus, while the SMC machine [19] may be used to describe the semantics of usual imperative constructs.

The situation is much less clear in the field of concurrent programming. Models such as Petri Nets, Communicating Automata, or Data Flow Networks can be considered as abstract machines, but certainly they lack expressive power. More expressive models such as Algebraic Process Calculi [18,6] are intended to be specification formalisms for distributed systems rather than abstract machines. Implementation models of Concurrent Programming Languages such as CSP [16] are conceptually based on standard sequential machine models augmented with scheduling facilities, not on specific abstract machines.

Most available concurrency models are based on architectural concepts, e.g. networks of processes communicating by means of ports or channels. Such concepts convey a rigid geometrical vision of concurrency. Our chemical abstract machine model is based on a radically different paradigm, which originated in the Γ language of Banâtre and Le Metayer [2,3]. These authors pointed out that parallel programming with control threads is more difficult to manage than sequential programming, a fact that contrasts with the common expectation that parallelism should ease program design. They argued that a high-level parallel programming methodology should be liberated from control management. A similar idea motivates the UNITY model of Chandy and Misra [9]. Then they proposed a model where the concurrent components are freely "moving" in the system and communicate when they come in contact.

Intuitively, the state of a system is like a *chemical solution* in which floating *molecules* can interact with each other according to *reaction rules*; a *magical mechanism* stirs the solution, allowing for possible contacts between molecules. In chemistry, this is the result of Brownian motion, but we don't insist on any particular mechanism, this being an implementation matter not studied here, see [2,9]. The solution transformation process is obviously truly parallel: any number of reactions can be performed in

*presently at LIX, Ecole Polytechnique, 91 128 Palaiseau, France

Work supported by the french Programme de Recherches Coordonnées C^3.

parallel, provided that they involve disjoint sets of molecules. Notice that the tuple space model of Linda [8] is based on very similar concepts and bears the same degree of potential parallelism, as well as the sets of assignments used in UNITY [9].

Let us give a simple but striking example from [2,3]. Assume the solution is originally made of all integers from 2 to n, along with the rule that any integer destroys its multiples. Then the solution will end up containing the prime numbers between 2 and n. See [2,3] for more examples and for implementation techniques.

Technically, a Γ program is defined by the structure of the molecules it handles and by a set of reaction rules. Solutions are represented by *multisets* of molecules: this accounts for the associativity and commutativity of parallel composition, that is the implicit stirring mechanism. The reaction rules are multiset rewritings.

We keep the same basic notions for chemical abstract machines. We elaborate on the original Γ language by presenting molecules in a systematic way as terms of algebras and refining the classification of rules. Some molecules do not exhibit interaction capabilities; those which are ready to interact are called *ions*. A solution can be *heated* to break complex molecules into smaller ones up to ions. Conversely, a solution can be *cooled* to rebuild heavy molecules from components. Furthermore, to deal with abstraction and hierarchical programming, we allow a molecule to contain a subsolution enclosed in a *membrane*, which can be somewhat *porous* to allow communication between the encapsulated solution and its environment.

The chemical abstract machines all obey a simple set of structural laws. Each particular machine is given by adding a set of simple rules that specify how to produce new molecules from old ones. Unlike the inference rules classically used in structural operational semantics, the specific rules have no premisses and are purely local.

In this paper, we concentrate on the descriptive power of chemical abstract machines. The strength of the model lies in the membrane notion. Membranes make it possible to build chemical abstract machines that have the power of classical process calculi or that behave as concurrent generalizations of the lambda-calculus.

To familiarize the reader with our concepts, the next section presents a simple machine for a subset of CCS. Section 3 gives some formal definitions. In Section 4, we treat the full TCCS [13] calculus and indicate how to handle other process calculi. Section 5 is devoted to a concurrent lambda-calculus similar to that of [7]. We conclude in section 6.

2 Handling a Subset of CCS

Our first illustrative example is a fragment CCS^- of Milner's process calculus CCS [18], containing the most basic operators **0** (inaction), '.' (prefixing), and '|' (parallel), as well as the restriction operator '\' to make the example non-trivial.

Let $\mathcal{N} = \{a, b, \ldots\}$ be a set of *names* and $\mathcal{L} = \{a, \overline{a} \mid a \in \mathcal{N}\}$ be the set of *labels* built on $\mathcal{N}$. We use the symbols α, β, etc., to range over labels, with $\overline{\overline{\alpha}} = \alpha$. The CCS^- agents p, q, etc., are given by the syntax:

$$p ::= \mathbf{0} \mid \alpha.p \mid (p \mid p) \mid p\backslash a$$

2.1 The Semantics of TCCS

Process calculi semantics are usually defined by inference rules in Plotkin's structural operational semantics style, called SOS for short. Milner's original rules involve an additional τ label representing internal communication. This happens to be quite unnatural with respect to abstract machine executions, where internal transitions should not be visible to the user. We prefer to use the De Nicola – Hennessy TCCS rules [13] that define two kinds of transitions between agents: the *internal* transitions $p \rightarrow p'$ and the *labelled* transitions $p \xrightarrow{\alpha} p'$. Intuitively, $p \rightarrow p'$ means that p can become p' by executing an internal action, and $p \xrightarrow{\alpha} p'$ means that p can offer its environment to accept the action α and then become p'.

Both transitions systems are defined in a structural way: the behavior of an agent is deduced from the behaviors of its components. Since internal communications generate internal transitions, the inference systems for $\rightarrow$ invokes the one for $\xrightarrow{\alpha}$:

$$\alpha.p \xrightarrow{\alpha} p$$

$$\frac{p \rightarrow p'}{p \mid q \rightarrow p' \mid q \text{ and } q \mid p \rightarrow q \mid p'}$$

$$\frac{p \xrightarrow{\alpha} p'}{p \mid q \xrightarrow{\alpha} p' \mid q \text{ and } q \mid p \xrightarrow{\alpha} q \mid p'}$$

$$\frac{p \xrightarrow{\alpha} p' \quad q \xrightarrow{\overline{\alpha}} q'}{p \mid q \rightarrow p' \mid q'}$$

$$\frac{p \rightarrow p'}{p\backslash a \rightarrow p'\backslash a}$$

$$\frac{p \xrightarrow{\alpha} p' \quad \alpha \notin \{a, \overline{a}\}}{p\backslash a \xrightarrow{\alpha} p'\backslash a}$$

2.2 Basic Chemistry: Concurrency and Communication

We now take the chemical abstract machine point of view, limiting us to internal transitions of restriction-free agents in this section. Restriction and external communication will be treated in the next section.

Instead of composing their behaviors, we make the agents or *molecules* directly react with each other within a *solution*, that is a multiset $S = \{\!|\, p, q, \ldots \,|\!\}$. There are only two basic rules:

parallel:

$$p \mid q \rightleftharpoons p,\ q$$

reaction:

$$a.p,\ \overline{a}.q \rightarrow p,\ q$$

The rules apply to molecules present in the solution; they do not apply inside molecules.

The first rule is reversible. It says that any molecule of the form $p \mid q$ that floats in the solution can be *heated up* (symbol $\rightarrow$) to decompose it into its components p and q, and conversely that any pair p, q of molecules can be *cooled down* (symbol $\leftarrow$) to rebuild a compound molecule $p \mid q$. The comma ',' appearing in the right-hand side expresses that the heating and cooling rule respectively yield and take a pair of molecules.

The reaction rule deals with *ions*, i.e. molecules of the form $\alpha.p$. Since α is the ion's communication capability, we call it its *valence*. Whenever two complementary ions float in the solution, they can react with each other and release their bodies in the solution. The valences simply vanish. Unlike the parallel rule, the reaction rule is irreversible.

To execute an agent p, we start from the solution $S_0 = \{\!|\, p \,|\!\}$. Heating the solution exhibits the potential communications, which can then be performed using the reaction rule. Notice that a *hot* solution obtained by heating an agent as much as possible contains only ions. Conversely, any solution obtained by transitions from S_0 can be *frozen* by cooling rules into a solution $\{\!|\, q \,|\!\}$ consisting of a single CCS$^-$ term, or into the empty solution, analogous to the term 0.

To see the chemical abstract machine at work, let us consider an execution of the agent $a.b.0 \mid \overline{a}.0 \mid \overline{b}.0$.

$$\begin{array}{lll} & \{\!|\, a.b.0 \mid \overline{a}.0 \mid \overline{b}.0 \,|\!\} & \\ \stackrel{*}{\rightarrow} & \{\!|\, a.b.0,\ \overline{a}.0,\ \overline{b}.0 \,|\!\} & \text{(parallel)} \\ \rightarrow & \{\!|\, b.0,\ 0,\ \overline{b}.0 \,|\!\} & \text{(reaction)} \\ \rightarrow & \{\!|\, 0,\ 0,\ 0 \,|\!\} & \text{(reaction)} \end{array}$$

The final solution $\{\!|\, 0, 0, 0 \,|\!\}$ only contains the inert molecule 0. It is natural to clean it up by using the following rule, which says that 0 evaporates when heated:

inaction cleanup:

$$0 \rightarrow$$

A last cleaning step yields the empty solution $\{\!|\,|\!\}$.

Generally speaking, chemical executions are non-deterministic. For example, in the solution $\{\!|\, a.0,\ \overline{a}.b.0,\ \overline{a}.c.0 \,|\!\}$, the $a.0$ ion can react with any of the two others ions, yielding either $\{\!|\, b.0,\ \overline{a}.c.0 \,|\!\}$ or $\{\!|\, \overline{a}.b.0,\ c.0 \,|\!\}$ after cleanup.

The reader will appreciate the simplicity of the chemical executions compared to the sequence of proofs and simplifications involved in the SOS semantics. The use of the structural rules for '$\mid$' is factored throughout an execution by the heating process, since we directly chain reactions by keeping the solution hot. The SOS evaluation involves structural rules at *each* computation step.

In fact, the simplification comes from the abandon of the rigid algebraic syntax. Chemical concurrency is *naturally* associative and commutative, since multisets are intrinsically unordered. The notion of syntactic position disappears even for the standard syntactic parallel construct '$\mid$': it is impossible to know whether $\{\!|\, p, q \,|\!\}$ was obtained by heating $\{\!|\, p \mid q \,|\!\}$ or $\{\!|\, q \mid p \,|\!\}$. On the contrary, the SOS semantics need to first introduce behaviors to recover concurrency out of the fixed syntax, then to define what it means for processes to be equivalent, and finally to *prove* equivalences such as $p \mid q \sim q \mid p$. SOS also involves inference rules with non-trivial premisses, which are certainly more complex than the naive cham rewrite rules.

Furthermore, we treat structural simplifications in the same way as reactions: to suppress a 0, we simply evaporate it. In SOS semantics, one needs to prove that $p \mid 0$ is equivalent to p, and one performs transitions and simplifications in separate steps and by separate techniques.

However, the notion of behavior that underlies the SOS semantics has advantages. In particular, $p \stackrel{\alpha}{\rightarrow} p'$ also tells that p can perform an α when requested by some *external* observer, thus defining at once how an agent communicates with its environment. We must define the same notion in our setting.

A solution should be able to perform a visible α action whenever it contains an ion $\alpha.p$. This ion should then export the α valence and become p. One could imagine to let it disintegrate into p and emit an α-particle to the environment. However, such a technique would violate Milner's most useful principle,

which states that there should be no difference in nature between internal and external communications. The right solution is to make the observer *react* with the valence of any molecule of the solution. This requires a richer machinery developped in the next section.

2.3 More Advanced Chemistry: Membranes and Airlocks

Consider a restriction agent $p\backslash a$ floating in a solution. If p is already of the form $\alpha.q$, $\alpha \notin \{a, \overline{a}\}$, we can build a new ion by the following simple rule:

restriction ion:

$$(\alpha.p)\backslash a \rightleftharpoons \alpha.(p\backslash a) \quad \text{if } \alpha \notin \{a, \overline{a}\}$$

But this does not work if p is compound. In this case, p should be able to freely perform internal reactions and to also propose communications to other ions floating in the main solution, using its own ions of unrestricted valences. To let p evolve on its own, we put it in a new local solution contained within a *membrane* $\{\!|\, . \,|\!\}$:

restriction membrane:

$$p\backslash a \rightleftharpoons \{\!|\, p \,|\!\}\backslash a$$

The new subsolution obeys the same rules as the global solution. To realize global communications, we need to make the membrane *porous* to valences. A first simple idea would be to use an heavy ion formation rule such as:

$$\{\!|\, \alpha.p,\ p_1,\ p_2, \ldots p_n \,|\!\} \rightarrow \alpha.\{\!|\, p,\ p_1,\ p_2, \ldots p_n \,|\!\}$$

We reject such a rule for two reasons. First, it does not involve only simple molecules as did previous rules; on the contrary, it involves finding an ion within a solution, which is neither simple nor general. Second, it is irreversible, since the information of where α comes from is lost. If a wrong valence is chosen, the heavy ion can stay forever in the main solution, like a *precipitate*. Consider for example $\{\!|\, a.0,\ \{\!|\, \overline{a}.0,\ b.0 \,|\!\}\backslash c \,|\!\}$ when choosing b: we are stuck with the inert solution $\{\!|\, a.0,\ b.(\{\!|\, \overline{a}.0,\ 0 \,|\!\}\backslash c) \,|\!\}$

The technique we propose is in two steps. First, we introduce a new mechanism at the general chemical machine level: the *airlock* mechanism. It extracts any molecule from a solution (not necessarily an ion), puts the rest of the solution within a membrane, and isolates the extracted molecule within an airlock attached to the membrane. The airlock construct is written $m \triangleleft S$ where m is the isolated molecule and S the remaining solution; it is a *single* molecule. The reversible (meta) rule is:

airlock:

$$\{\!|\, m, m_1, m_2, \ldots, m_n \,|\!\} \rightleftharpoons \{\!|\, m \triangleleft \{\!|\, m_1, m_2, \ldots m_n \,|\!\} \,|\!\}$$

The solution $\{\!|\, m_1, m_2, \ldots m_n \,|\!\}$ contained in the new molecule is allowed to freely continue internal reactions (see the formal definitions in the next section).

Second, we build an heavy ion from any ion in the airlock, using the rule:

heavy ion:

$$(\alpha.p) \triangleleft S \rightleftharpoons \alpha.(p \triangleleft S)$$

In this way, we guarantee reversibility by preserving the attachment between α and p. A restriction molecule can propose several valences in succession to its environment until a communication takes place.

Let us give a simple example of communication involving an heavy ion:

$$\begin{array}{lll}
 & \{\!|\, a.0 \mid (\overline{a}.p \mid q)\backslash b \,|\!\} & \\
\xrightarrow{*} & \{\!|\, a.0,\ \{\!|\, \overline{a}.p,\ q \,|\!\}\backslash b \,|\!\} & \text{(par., restr. membr.)} \\
\rightarrow & \{\!|\, a.0,\ \{\!|\, (\overline{a}.p) \triangleleft \{\!|\, q \,|\!\} \,|\!\}\backslash b \,|\!\} & \text{(airlock)} \\
\rightarrow & \{\!|\, a.0,\ \{\!|\, \overline{a}.(p \triangleleft \{\!|\, q \,|\!\}) \,|\!\}\backslash b \,|\!\} & \text{(heavy ion)} \\
\rightarrow & \{\!|\, a.0,\ (\overline{a}.(p \triangleleft \{\!|\, q \,|\!\}))\backslash b \,|\!\} & \text{(restr. membrane)} \\
\rightarrow & \{\!|\, a.0,\ \overline{a}.((p \triangleleft \{\!|\, q \,|\!\})\backslash b) \,|\!\} & \text{(restriction ion)} \\
\rightarrow & \{\!|\, 0,\ (p \triangleleft \{\!|\, q \,|\!\})\backslash b \,|\!\} & \text{(reaction)} \\
\rightarrow & \{\!|\, \{\!|\, p \triangleleft \{\!|\, q \,|\!\} \,|\!\}\backslash c \,|\!\} & \text{(restr. membrane)} \\
\rightarrow & \{\!|\, \{\!|\, p,\ q \,|\!\}\backslash c \,|\!\} & \text{(airlock)}
\end{array}$$

Notice how we guarantee reversibility by putting or removing membranes. Once the heavy ion $\overline{a}.(p \triangleleft \{\!|\, q \,|\!\})$ has been constructed, it is not possible to put back p into q's solution, since the airlock molecule is not contained within a membrane. It is not possible to build such a membrane, since the membrane rule cannot be applied inside an ion.

The airlock technique makes it now easy to define what it means for an external observer to *observe* a solution. If the solution is reduced to a single ion, then the observer can pick up the ion's valence and release its body. More precisely, let S, S' denote solutions. We set $S \xrightarrow{\alpha} S'$ if there exist a molecule m such that $S \xrightarrow{*} \{\!|\, \alpha.m \,|\!\}$ and $\{\!|\, m \,|\!\} \xrightarrow{*} S'$. For example, one has

$$\{\!|\, a.0,\ b.0 \,|\!\} \xrightarrow{a} \{\!|\, b.0 \,|\!\}$$

taking $m = 0 \triangleleft \{\!|\, b.0 \,|\!\}$.

The relation between this new kind of behavior and the standard TCCS one will be precisely stated in section 4.

3 Formal Definitions

3.1 Chemical Abstract Machines

A *chemical abstract machine* or *cham* C is specified by defining *molecules* m, m', etc., *solutions* S, S', etc., and *transformation rules*. Molecules are terms of algebras, with specific operations for each cham. Solutions are finite multisets of molecules, written $\{\!|\ m_1, m_2, \ldots, m_k\ |\!\}$. Furthermore, in each cham, any solution S can itself be be considered as a single molecule and can therefore appear as a *subsolution* of another molecule. The corresponding $\{\!|\ .\ |\!\}$ operator is called the *membrane* operator. For instance, if 0 and $+$ are the molecule building operations, then 0, $0+0$, $0+\{\!|0|\!\}$, and $\{\!|0, \{\!|0+0, 0|\!\}|\!\}$ are molecules, the latter also being a solution.

The transformation rules have the form

$$m_1, m_2, \ldots, m_k \rightarrow m'_1, m'_2, \ldots m'_l$$

where the m_i and m'_j are molecules. As usual, the rules will be presented by means of *rule schemata*, the actual rules being the instances of these schemata. To avoid "multiset matching", we require the subsolutions appearing in rule schemata to be either a single solution variable S that generates all solutions, or of the form $\{\!|m|\!\}$ where m is a single molecule.

To state how reaction rules apply to solutions, we need some notation. The multiset union of S and S' is written $S \uplus S'$. As in the λ-calculus [4], we use the context notation $C[\,]$ to denote a molecule with a hole $[\,]$ in which to place other molecule.

The transformation rules determine a *transformation relation* $S \rightarrow S'$ between solutions, according to the following three laws:

- **The Reaction Law.** An instance of the right-hand-side of a rule can replace the corresponding instance of its left-hand-side. If

  $$m_1, m_2, \ldots, m_k \rightarrow m'_1, m'_2, \ldots m'_l$$

  is a rule and M_1, M_2, ..., M_k, M'_1, M'_2, ..., M'_l, are instances of the m_i's and the m_j's, then

  $$\{\!|M_1, M_2, \ldots, M_k|\!\} \rightarrow \{\!|M'_1, M'_2, \ldots M'_l|\!\}$$

- **The Chemical Law.** Reactions can be performed freely within any solution:

  $$\frac{S \rightarrow S'}{S \uplus S'' \rightarrow S' \uplus S''}$$

- **The Membrane Law.** A subsolution can evolve freely in any context:

  $$\frac{S \rightarrow S'}{\{\!|C[S]|\!\} \rightarrow \{\!|C[S']|\!\}}$$

The chemical and membrane laws are the only ones to involve premisses. They factor what is usually called "structural rules" in particular calculi. All other laws and rules are purely local.

Some chams, but not all of them, use the additional *airlock* construct. An airlock is a molecule of the form $m \triangleleft S$ where m is a molecule and S is a solution. Airlocks are built and suppressed by the following reversible law:

- **The Airlock Law**

  $$\{\!|m|\!\} \uplus S \leftrightarrow \{\!|m \triangleleft S|\!\}$$

A cham is an intrinsically parallel machine: one can simultaneously apply several rules to a solution provided that no molecule is involved in more than one rule; one can also transform subsolutions in parallel. In this paper, we only study the expressive power of chams; it does not depend on using parallel evaluation, since a non-conflicting parallel application of rules is equivalent to any sequence of the individual rules. See [2] for a practical use of parallel reductions.

3.2 A Classification of Rules

We usually distinguish between three kinds of rules: *heating rules* $\rightharpoonup$, *cooling rules* $\rightharpoondown$, and *reaction rules* $\rightarrow$. The distinction is merely a matter of taste. As a rule of thumb, we present all structural rules as heating rules, possibly paired with inverse cooling rules. Heating rules decompose a single molecule into simpler ones, and cooling rules recompose a compound molecule from its components. We generally write the heating and cooling rules together, using the symbol $\rightleftharpoons$. We say that a solution is *hot* (resp. *frozen*) if no heating (resp. cooling) rule applies to it. In the sequel, we shall always consider that the transitions given by the airlock law are heating and cooling ones.

The reflexive, symmetric, and transitive closure of $(\rightharpoonup \cup \rightharpoondown)$ is written $\overset{*}{\rightleftharpoons}$. According to our conventions, it usually represents structural equivalence.

The reaction rules usually involve several molecules. These molecules are often in a particular form in which they cannot be heated further; we call them *ions*. A solution is *inert* if no reaction rule applies to it, nor to any solution obtained by heating it.

4 Process Calculi Chams

In this section, we finish the treatment of the TCCS process calculus, we establish the semantical equivalence between the cham semantics and the original structural semantics, and we briefly indicate how to handle other process calculi.

4.1 The Full TCCS Calculus

We finish the description of the TCCS calculus [13] and of its SOS semantics. We have already seen the inaction '0', parallel '|', prefixing '.', and restriction '\' operators. We now add the remaining operators: the relabelling operator '[.]', the two sum operators '$\oplus$' (internal sum) and '$[]$' (external sum), and the fixpoint definition $\mathtt{fix}_i(\vec{x} = \vec{p})$, which is a shorthand for:

$$\mathtt{letrec}\ \ x_1 = p_1\ \mathtt{and}\ \ldots\ \mathtt{and}\ x_n = p_n\ \mathtt{in}\ x_i$$

The final syntax is as follows:

$$p ::= 0 \mid \alpha.p \mid (p \mid q) \mid p\backslash a \mid p[\phi] \\ \mid p \oplus q \mid p[]q \mid \mathtt{fix}_i(\vec{x} = \vec{p})$$

We give the semantics of the new operators. A relabelling is a mapping $\phi : \mathcal{N} \mapsto \mathcal{L}$, extended to labels by setting $\phi(\overline{\alpha}) = \overline{\phi(\alpha)}$. The relabelling operator takes an agent p and a relabelling ϕ and produces a new agent $p[\phi]$ that behaves like p except that all its visible actions are relabelled by ϕ:

$$\frac{p \rightarrow p'}{p[\phi] \rightarrow p'[\phi]}$$

$$\frac{p \xrightarrow{\alpha} p'}{p[\phi] \xrightarrow{\phi(\alpha)} p'[\phi]}$$

Sums represent non-deterministic choices. There are several possible sums, see [13] for an extensive discussion. The simplest sum is the internal sum $\oplus$, which non-deterministically chooses a component:

$$p \oplus q \rightarrow p$$
$$p \oplus q \rightarrow q$$

In an external sum $p[]q$, the agents p and q can freely perform internal actions and can also propose communications to the environment. The choice is made only when such a communication is performed:

$$\frac{p \rightarrow p'}{p[]q \rightarrow p'[]q \ \text{ and } \ q[]p \rightarrow q[]p'}$$

$$\frac{p \xrightarrow{\alpha} p'}{p[]q \xrightarrow{\alpha} p' \ \text{ and } \ q[]p \xrightarrow{\alpha} p'}$$

Finally, the fixpoint operation is a simple unfolding. Let $p[\vec{q}/\vec{x}]$ denote the result of the simultaneous substitution of the q_i to the x_i in p:

$$\mathtt{fix}_i(\vec{x} = \vec{p}) \rightarrow p_i[\vec{\mathtt{fix}}(\vec{x} = \vec{p})/\vec{x}]$$

4.2 Handling the New Operators

We first explain how to handle the new operators. Then we give the exact syntax of molecules and the complete set of rules of the TCCS cham.

The relabelling operator can be handled just as the restriction operator, by building a membrane and exporting relabelled names as heavy ion valences. Internal sum is handled by the same rules as in the SOS semantics; since the rules are not structural, we call them reaction rules and not heating rules. The fixpoint expansion rule is also as in the SOS semantics, and it is clearly a heating rule. See the exact rules in the rule summary below.

External sum needs more care. Since the summands p and q should each be able to freely perform internal transitions, we open one membrane for each of them. We therefore introduce a new *molecule pairing* operator $<.,.>$ and the expansion rule:

$$p[]q \rightleftharpoons < \{\!|p|\!\}, \{\!|q|\!\} >$$

Assume that the left subsolution S produces an ion $\alpha.m$ and that we want to export α. Then we can give S the form $\{\!|\,\alpha.n\,|\!\}$, either by taking $n = m$ if S only contains the given ion, or by building an airlock $(\alpha.m) \triangleleft S_1$ and then an heavy ion $\alpha.(m \triangleleft S_1)$. To export the valence, we can use the rule:

$$< \{\!|\alpha.n|\!\}, S' > \rightarrow \alpha.\ < n, S' >$$

However, as discussed in section 2, we must be careful to avoid precipitates and to make the above rule reversible. The reverse cooling rule must recognize that the valence belongs to n and not to S' when it is given a pair $< n, S' >$. Furthermore, once the α valence is consumed by some reaction, we are left with a pair $< n, S' >$ that we must transform into n to realize the summand selection; we need a cooling rule of the form:

$$< n, S' > \rightarrow n$$

Here again, we must recognize which is n and which is S'.

We can use several techniques to solve this problem. The one we choose is to tag the internal pair when the heavy ion is built to directly remember the valence attachment. The rules for the left-hand-side choice are:

$$< \{\!|\alpha.m|\!\}, S > \rightleftharpoons \alpha.(l :< m, S >)$$
$$l :< m, m' > \rightarrow m$$

The rules for the right-hand-side choice are symmetric with a label r.

An alternative technique would be to always force n to be an airlock, noticing that a solution $\{\!|\, \alpha.m \,|\!\}$ containing a single ion can always be heated into $\{\!|\, \alpha.(m \triangleleft \{\!|\, |\!\}) \,|\!\}$, and to use the following rules:

$$< \{\!|\, \alpha.(m \triangleleft S) \,|\!\}, S' > \rightleftharpoons \alpha. < m \triangleleft S, S' >$$
$$< m \triangleleft S, S' > \rightarrow m \triangleleft S$$

This technique would decrease the number of molecule constructors, but is rather *ad-hoc* and we don't use it here. This discussion might look a bit tedious, but it shows two things: first, designing a cham has much to do with standard programming; second, the external sum operator is not very natural in concurrent abstract machines.

Let us give a simple external sum evaluation example:

$$\begin{aligned}
& \{\!|\, a.\overline{b}.0 \mid ((\overline{a}.0 \mid b.0) \| q) \,|\!\} \\
\xrightarrow{*} \quad & \{\!|\, a.\overline{b}.0,\ < \{\!|\, \overline{a}.0,\ b.0 \,|\!\}, \{\!|\, q \,|\!\} > \,|\!\} \\
\xrightarrow{*} \quad & \{\!|\, a.\overline{b}.0,\ < \{\!|\, \overline{a}.(0 \triangleleft \{\!|\, b.0 \,|\!\}) \,|\!\}, \{\!|\, q \,|\!\} > \,|\!\} \\
\xrightarrow{*} \quad & \{\!|\, a.\overline{b}.0,\ \overline{a}.l :< 0 \triangleleft \{\!|\, b.0 \,|\!\}, \{\!|\, q \,|\!\} > \,|\!\} \\
\rightarrow \quad & \{\!|\, \overline{b}.0,\ l :< 0 \triangleleft \{\!|\, b.0 \,|\!\}, \{\!|\, q \,|\!\} > \,|\!\} \\
\rightarrow \quad & \{\!|\, \overline{b}.0,\ 0 \triangleleft \{\!|\, b.0 \,|\!\} \,|\!\} \\
\rightarrow \quad & \{\!|\, \overline{b}.0,\ 0,\ b.0 \,|\!\}
\end{aligned}$$

Notice the last step: when an airlock $m \triangleleft S$ floats in a solution, one can cool it down and release m and all the molecules of S in the containing solution. This requires to use both the airlock law and the chemical law.

4.3 The Complete TCCS Cham

We summarize the syntax and rules of the final TCCS cham.

Syntax

Agents:

$$\begin{aligned}
p ::= \ & 0 \mid \alpha.p \mid (p \mid q) \mid p \backslash a \mid p[\phi] \\
& \mid p \oplus q \mid p \| q \mid \mathtt{fix}_i(\vec{x} = \vec{p})
\end{aligned}$$

Molecules:

$$\begin{aligned}
m ::= \ & p \mid \alpha.m \mid m \backslash a \mid m[\phi] \mid S \mid m \triangleleft S \\
& \mid < m, m > \mid l :< m, m > \mid r :< m, m >
\end{aligned}$$

Notice that parallel and sums are agent operators, not molecules operators.

Rules

parallel:

$$p \mid q \rightleftharpoons p,\ q$$

reaction:

$$a.m,\ \overline{a}.n \rightarrow m,\ n$$

restriction membrane:

$$m \backslash a \rightleftharpoons \{\!|\, m \,|\!\} \backslash a$$

restriction ion:

$$(\alpha.m) \backslash a \rightleftharpoons \alpha.(m \backslash a) \quad \text{if } \alpha \notin \{a, \overline{a}\}$$

relabelling membrane:

$$m[\phi] \rightleftharpoons \{\!|\, m \,|\!\}[\phi]$$

rellabelling ion:

$$(\alpha.m)[\phi] \rightleftharpoons \phi(\alpha).(m[\phi])$$

$\oplus$-*left*:

$$p \oplus q \rightarrow p$$

$\oplus$-*right*:

$$p \oplus q \rightarrow q$$

$\|$-*expansion*:

$$p \| q \rightleftharpoons < \{\!|\, p \,|\!\}, \{\!|\, q \,|\!\} >$$

left $\|$-*ion*:

$$< \{\!|\, \alpha.m \,|\!\}, S > \rightarrow \alpha.l :< m, S >$$

right $\|$-*ion*:

$$< S, \{\!|\, \alpha.m \,|\!\} > \rightarrow \alpha.r :< S, m >$$

left projection:

$$l :< m, m' > \rightarrow m$$

right projection:

$$r :< m, m' > \rightarrow m'$$

fixpoint:

$$\mathtt{fix}_i(\vec{x} = \vec{p}) \rightarrow p_i[\mathtt{fix}(\vec{x} = \vec{p})/\vec{x}]$$

Additional Cleanup Rules

inaction cleanup:

$$0 \rightarrow$$

restriction cleanup:

$$\{\!|\, |\!\} \backslash a \rightarrow$$

relabelling cleanup:

$$\{\!|\, |\!\}[\phi] \rightarrow$$

4.4 Comparing the Cham and SOS

We define labeled transitions as explained in section 2.

Definition: Given a solution S, we write $S \xrightarrow{\alpha} S'$ if there exists a molecule m' such that $S \xrightarrow{*} \{\!|\, \alpha.m' \,|\!\}$ and $\{\!|\, m' \,|\!\} \xrightarrow{*} S'$.

Remember that the structural equivalence $\overset{*}{\rightleftharpoons}$ between solutions is the reflexive, symmetric, and transitive closure of the heating and cooling relations. In the sequel, we shall neglect the cleanup rules and consider only the reversible heating/cooling rules. Then $S \overset{*}{\rightleftharpoons} S'$ if and only if there exists a sequence of heating or cooling steps from S to S'.

The following result shows that the cham differs from the original TCCS calculus only in the number of internal steps involved in computations. As far as labelled transitions are concerned, the solution $\{\!|\, p \,|\!\}$ can do whatever the term p can do, and it cannot do more.

Theorem: Let p be a TCCS agent.

1) If $p \rightarrow p'$ in TCCS, then $\{\!|\, p \,|\!\} \xrightarrow{*} \{\!|\, p' \,|\!\}$ in the TCCS cham. If $p \xrightarrow{\alpha} p'$ in TCCS, then $\{\!|\, p \,|\!\} \xrightarrow{\alpha} \{\!|\, p' \,|\!\}$; more precisely, there exists a molecule m' such that $\{\!|\, p \,|\!\} \rightarrow \{\!|\, \alpha.m' \,|\!\}$ and $\{\!|\, m' \,|\!\} \overset{*}{\rightleftharpoons} \{\!|\, p' \,|\!\}$.

2) If $\{\!|\, p \,|\!\} \rightarrow S'$, then there exists a TCCS agent p' such that $S' \overset{*}{\rightleftharpoons} \{\!|\, p' \,|\!\}$. If $\{\!|\, p \,|\!\} \xrightarrow{\alpha} S'$, then there exists a TCCS agent p' such that $p \xrightarrow{\alpha} p'$ and $S' \overset{*}{\rightleftharpoons} \{\!|\, p' \,|\!\}$.

Sketch of proof: To prove 1), one shows how to perform given TCCS derivations by chaining cham transitions. The proof is by induction on the size of p and by cases on the form of the given TCCS transition. We show two typical cases.

First if $p = p_1 \,|\, p_2 \rightarrow p_1' \,|\, p_2' = p'$ with $p_1 \xrightarrow{\alpha} p_1'$ and $p_2 \xrightarrow{\overline{\alpha}} p_2'$ for some α, by induction, there exist m_1' and m_2' such that $\{\!|\, p_1 \,|\!\} \xrightarrow{*} \{\!|\, \alpha.m_1' \,|\!\}$, $\{\!|\, m_1' \,|\!\} \overset{*}{\rightleftharpoons} \{\!|\, p_1' \,|\!\}$, $\{\!|\, p_2 \,|\!\} \xrightarrow{*} \{\!|\, \alpha.m_2' \,|\!\}$, and $\{\!|\, m_2' \,|\!\} \overset{*}{\rightleftharpoons} \{\!|\, p_2' \,|\!\}$. We build the following transformation sequence:

$$
\begin{array}{lll}
 & \{\!|\, p \,|\!\} & \\
= & \{\!|\, p_1 \,|\, p_2 \,|\!\} & \\
\rightarrow & \{\!|\, p_1,\ p_2 \,|\!\} & (\textit{parallel}) \\
\xrightarrow{*} & \{\!|\, \alpha.m_1',\ \overline{\alpha}.m_2' \,|\!\} & (\textit{cham laws}) \\
\rightarrow & \{\!|\, m_1',\ m_2' \,|\!\} & (\textit{reaction}) \\
\overset{*}{\rightleftharpoons} & \{\!|\, p_1',\ p_2' \,|\!\} & (\textit{cham laws}) \\
\rightarrow & \{\!|\, p_1' \,|\, p_2' \,|\!\} & (\textit{parallel}) \\
= & \{\!|\, p' \,|\!\} &
\end{array}
$$

which shows the required property of p.

Assume now $p = q \setminus a \xrightarrow{\alpha} q' \setminus a = p'$, with $\alpha \notin \{a, \overline{a}\}$. By induction, there exists n' such that $\{\!|\, q \,|\!\} \rightarrow \{\!|\, \alpha.n' \,|\!\}$ and $\{\!|\, n' \,|\!\} \overset{*}{\rightleftharpoons} \{\!|\, q' \,|\!\}$. Let $m' = n' \backslash a$. We build the execution sequence:

$$
\begin{array}{lll}
 & \{\!|\, p \,|\!\} & \\
= & \{\!|\, q \backslash a \,|\!\} & \\
\rightarrow & \{\!|\, \{\!|\, q \,|\!\} \backslash a \,|\!\} & (\textit{restriction membrane}) \\
\xrightarrow{*} & \{\!|\, \{\!|\, \alpha.n' \,|\!\} \backslash a \,|\!\} & (\textit{cham laws}) \\
\rightarrow & \{\!|\, (\alpha.n') \backslash a \,|\!\} & (\textit{restriction membrane}) \\
\rightarrow & \{\!|\, \alpha.(n' \backslash a) \,|\!\} & (\textit{restriction ion}) \\
= & \{\!|\, \alpha.m' \,|\!\} &
\end{array}
$$

Furthermore, one has:

$$
\begin{array}{lll}
 & \{\!|\, m' \,|\!\} & \\
= & \{\!|\, n' \backslash a \,|\!\} & \\
\rightarrow & \{\!|\, \{\!|\, n' \,|\!\} \backslash a \,|\!\} & (\textit{restriction membrane}) \\
\overset{*}{\rightleftharpoons} & \{\!|\, \{\!|\, q' \,|\!\} \backslash a) \,|\!\} & (\textit{cham laws}) \\
\rightarrow & \{\!|\, q' \backslash a \,|\!\} & (\textit{restriction membrane}) \\
= & \{\!|\, p' \,|\!\} &
\end{array}
$$

which shows the required property of p.

Proving 2) is harder and we just sketch of the proof architecture. The properties of $\rightarrow$ and $\xrightarrow{\alpha}$ are proved together by induction on the number of irreversible rules applied in the given derivations.

If this number is 0, then the property of $\rightarrow$ is obvious with $p' = p$. To prove the property of $\xrightarrow{\alpha}$, we use a lemma about ion formation.

The lemma shows how ions $a.m$ can be formed in arbitrary subsolutions using only heating and cooling rules. The valences of such ions always come from label positions in TCCS terms that yield observable transitions. Furthermore, the ion bodies are kept untouched in the heating-cooling process. More formally, assume $\{\!|\, p \,|\!\} \overset{*}{\rightleftharpoons} C[\alpha.m]$ where the ion $\alpha.m$ floats in some subsolution. Then one can cool down $C[\alpha.m]$ into $C'[\alpha.q]$ by using only ion cooling rules, in such a way that $\alpha.q$ is exactly a subexpression of the original agent $p = C_1[\alpha.q]$, with the additional properties $p \xrightarrow{\alpha} C_1[q]$ in TCCS and $\{\!|\, C_1[q] \,|\!\} \overset{*}{\rightleftharpoons} \{\!|\, C'[q] \,|\!\}$.

Now if $\{\!|\, p \,|\!\} \overset{*}{\rightleftharpoons} \{\!|\, \alpha.m' \,|\!\}$, one can use the lemma to show that p has the form $C[\alpha.q]$ with $C[q] \overset{*}{\rightleftharpoons} m'$. This show the required property of $\xrightarrow{\alpha}$, taking $p' = C[q]$.

Assume now that the number of irreversible transitions in a given derivation is strictly positive. The

derivation can be put in the form $S \stackrel{*}{\rightarrow} S_1 \rightarrow S_2 \stackrel{*}{\rightarrow} S'$ with $S \stackrel{*}{\rightleftharpoons} S_1$ and where $S_1 \rightarrow S_2$ is irreversible. The only difficult case is the one where the transition form S_1 to S_2 is a reaction. By a slight extension of the lemma to two-hole contexts, one can show that $p = C[\alpha.q][\overline{\alpha}.r]$, $p \rightarrow p' = C[q][r]$ in TCCS, and $\{\!|\ p'\ |\!\} \stackrel{*}{\rightleftharpoons} C'[q][r]$ with $S_1 \stackrel{*}{\rightarrow} C[\alpha.q][\overline{\alpha}.r]$ and $S_2 \stackrel{*}{\rightleftharpoons} C[q][r]$. The global induction hypothesis applies to p' and gives the final result.

4.5 Handling Other Process Calculi

In Milner's original calculus CCS, there is no notion of an internal unlabelled transition. The special label τ is used to report transitions provoked by internal communications. The sum $p + q$ is defined by the following rule: i:

$$\frac{p \stackrel{\alpha}{\rightarrow} p'}{p+q \stackrel{\alpha}{\rightarrow} p' \text{ and } q+p \stackrel{\alpha}{\rightarrow} p'}$$

n which one can take $\alpha = \tau$. Therefore, a summand can be chosen either by an external communication or by an internal one.

To simulate CCS by a cham, we abandon the simple reaction rule of TCCS and replace it by the following rule:

τ-reaction:

$$a.m,\ \overline{a}.n \rightarrow \tau.(m \mid n)$$

Since a τ-ion can neither be heated nor interact with another molecule, the only thing it can do is to traverse all membranes up to the external observer. An observation by this observer consumes the τ valence, and frees the ion body that can be heated to release the parallel components. With this new definition of reaction, the rules of + are simply the above rules of $[\!]$.

Notice that performing an internal communication is more than just building a τ: the communication is really performed only when the final observer accepts it by *consuming* this τ. Therefore, the machine's behavior can no longer be defined independently of the observation process. Furthermore, the τ-reaction rule reduces the potential parallelism of the execution machine to a bare minimum. The simulation of CCS is rather unsatisfactory. We don't believe that CCS can be "implemented" in a more natural way, which is an indication that τ and + might not be good *programming* primitives. This is quite well-known to CCS simulator implementors.

Given a CCS agent p, we can show that the solution $\{\!|\ p\ |\!\}$ is in *weak bisimulation* with the agent p w.r.t. our definition of observation. Strong bisimulation can also be obtained by making the τ-reaction rule reversible, that is by allowing the machine to *undo* all its internal operations. The details are beyond the scope of this paper.

Handling other process calculi raises no particular problem. For example, the reader can easily write a natural cham for MEIJE [6], which is universal among the labelled process calculi. This tells that the cham formalism have basically the same power as the SOS one w.r.t. process calculi.

5 A Concurrent λ-calculus

5.1 Generalizing the λ-calculus

Algebraic process calculi model concurrency but have a limited expressive power compared to the λ-calculus, where one is able to express all possible combinators and to code many types of data. On the other hand, the λ-calculus is intrinsically sequential [4,5] and cannot handle even the weakest form of concurrency. Building new calculi that combine both abilities is a goal of primary importance [7,21]. In [7], we introduced such a tentative concurrent lambda-calculus called the γ-calculus. We could describe the (lazy) evaluation in this calculus by means of a cham. However, our formalism itself suggests a simpler, and perhaps better calculus of the same kind. To introduce this new calculus, let us first say a few words about the λ- and γ-calculi. Some familiarity with the λ-calculus will be assumed. We just recall the syntax:

$$M ::= x \mid (\lambda x.M) \mid (MM)$$

where x stands for any variable. We are interested here in the *lazy* evaluation of λ-terms (following [1]), that is the reflexive and transitive closure of the relation $M \triangleright M'$ inductively given by

$$(\lambda x.M)N \triangleright M[N/x]$$

$$M \triangleright M' \Rightarrow MN \triangleright M'N$$

Intuitively, a λ-calculus redex $(\lambda x.M)N$ is like a valued CCS communication of the form $\lambda x.M \mid \overline{\lambda}(N)$, since both yield $M[N/x]$ as a result. Hence one could imagine treating the lambda-calculus as a CCS-like process calculus where agents are communicable values, λ becoming a particular label. In such a calculus, functional application should appear as a particular parallel combination of two agents, the function and its argument, and β-reduction should be just a particular case of communication. However, the above simple redex translation would not take care of the non-associative character of application and would not treat double applications correctly. Consider, for

instance, the λ-term $((\lambda x.\lambda y.M)N)P$. The translation would be $\lambda x.\lambda y.M \mid \overline{\lambda}(N) \mid \overline{\lambda}(P)$. The associative/commutative character of concurrency would make the arguments N and P interchangeable, which is clearly wrong. Thomsen solved this problem in [21] using the CCS operators of restriction and renaming. However in his higher order calculus, β-reduction is performed in two steps, involving an intermediary state which does not represent a λ-term. Then the λ-calculus is not exactly a sub-calculus of Thomsen's CHOCS calculus.

Another solution was presented in [7] using two concurrency operators: an interleaving operator '|' and a binary communication operator '$\odot$'. Communications arise as follows: in a term $(M \odot N)$, all '|' concurrent components of M can communicate with all concurrent components of N, up to termination of M or N, termination being written as a special symbol 1. Then the $\odot$ operator disappears by application of the simplification rule $(M \odot 1) = (1 \odot M) = M$, and λ-application can be represented by $(M \odot \overline{\lambda}(N))$. For instance, the above double application works in the following way (assuming x, y not free in N):

$$
\begin{array}{lcl}
 & ((\lambda x.\lambda y.M \odot \overline{\lambda}(N)) \odot \overline{\lambda}(P)) & \\
\rightarrow & ((\lambda y.M[N/x] \odot 1) \odot \overline{\lambda}(P)) & \text{(communication)} \\
= & ((\lambda y.M[N/x]) \odot \overline{\lambda}(P)) & \text{(simplification)} \\
\rightarrow & (M[N/x][P/y] \odot 1) & \text{(communication)} \\
= & M[N/x][P/y] & \text{(simplification)}
\end{array}
$$

A cham describing this calculus would treat the terms $\lambda x.M$ and $\overline{\lambda}(N)$ as ions, but the interpretation of the concurrency operators of this calculus would be somewhat unnatural. In a cham, the parallelism is always commutative and associative and allows for communication, while $(M \mid N)$ disallows communication and $\odot$ is non-associative. As a matter of fact, the cham framework indicates another possibility for representing properly the λ-application, by means of an encapsulated parallel combination of the function and its argument.

5.2 The γ-calculus

The key idea of our new higher-order concurrent calculus is to *internalize* the concepts of the chemical abstract machine within the syntax. Let us review these concepts:

- *solutions*: these are built by heating a parallel combination of molecules. Therefore the corresponding syntactic construct is parallel composition $(M \mid N)$. Since solutions are multisets of possibly interacting processes, this operator allows communication.
- *membrane*: encapsulating a subsolution within a membrane forces reactions to occur locally. Here we will introduce a corresponding *localization* construct $\langle M \rangle$.
- *reactions*: basically, these occur when opposite ions float inside the same solution. We shall distinguish two kinds of reactive molecules, the *negative* ones, or receptors, and the *positive* ones, or emitters.

Typically, a receptor in the λ-calculus is an abstraction $\lambda x.M$. To emphasize the ion character, we shall denote such an atomic receptor $x^- M$, and an atomic emitter sending the value M will be denoted M^+. Therefore the syntax of our calculus is:

$$M \ ::= \ x \mid x^- M \mid (M)^+ \mid (M \mid M) \mid \langle M \rangle$$

where x stands for any variable. As usual we shall omit (or add) some parentheses in writing the terms, which will be called processes or sometimes agents. In what follows we shall call this concurrent calculus the γ-calculus, superseding the one proposed in [7].

To formalize the execution mechanism, we need a syntactic notion of *stable* state. Basically, a stable term is made out of ions of the same valence (either positive or negative), and will therefore represent an inert solution. Formally, the syntax for pure emitters or receptors and for stable terms is given by:

$$
\begin{array}{rcl}
E & ::= & M^+ \mid (E \mid E) \mid \langle E \rangle \\
R & ::= & x^- M \mid (R \mid R) \mid \langle R \rangle \\
W & ::= & E \mid R
\end{array}
$$

Now we give the γ-cham describing the (lazy) evaluation of terms. The molecules are given by the following grammar:

$$U \ ::= \ M \mid S \mid (U \mid U) \mid \langle U \rangle$$

where M stands for any term and S for any solution (i.e. finite multiset of molecules). The transformation rules are:

solution:

$$U \mid V \rightleftharpoons U,\ V$$

membrane:

$$\langle U \rangle \rightleftharpoons \{\!|U|\!\}$$

hatching:

$$\langle W \rangle \rightleftharpoons W$$

β-reaction:

$$x^- M,\ N^+ \to M[N/x]$$

where U, V stand for molecules, M, N for terms and W for any stable term. Note that the reaction rule, which is the only irreversible rule, embodies communication. The power of the calculus is essentially due to the rules concerning the membrane construct. This should not be confused with CCS restriction: if a membrane encloses a stable state (i.e. emitter or receptor), then it may vanish. The hatching rule conveniently replaces the termination equations concerning the cooperation operator of [7] (in our calculus, a "cooperation" operator would be $\langle M \,|\, N \rangle$). In what follows we shall use the notation $M \xrightarrow{*} N$ as an abbreviation for $\{\!|M|\!\} \xrightarrow{*} \{\!|N|\!\}$.

This γ-calculus contains the λ-calculus, since we may now define the application (MN) as the combination $\langle M \,|\, N^+ \rangle$. Let us see this point in some detail; we define a translation θ from the set of λ-terms to the set of terms given by the grammar:

$$M ::= x \mid x^- M \mid \langle M \,|\, M^+ \rangle$$

The translation is as follows:

$$\theta(x) = x$$

$$\theta(\lambda x.M) = x^- \theta(M)$$

$$\theta(MN) = \langle \theta(M) \,|\, \theta(N)^+ \rangle$$

Then we may show that there is a close correspondence between lazy evaluation of λ-terms and evaluation in the γ-cham of their translation. More precisely, it is easy to prove that

$$M \triangleright^* M' \ \Leftrightarrow\ \theta(M) \xrightarrow{*} \theta(M')$$

and, moreover, that each intermediate state in the evaluation of $\theta(M)$ cools down to a λ-term. For instance, the above double application works as follow:

$$\begin{array}{ll} & \langle \langle x^- y^- M \,|\, N^+ \rangle \,|\, P^+ \rangle \\ \xrightarrow{*} & \{\!|\, \{\!|x^- y^- M,\ N^+|\!\},\ P^+ |\!\} \quad \text{(membrane, solution)} \\ \to & \{\!|\, \{\!|y^- M[N/x]|\!\},\ P^+ |\!\} \quad \text{(reaction)} \\ \to & \{\!|y^- M[N/x],\ P^+|\!\} \quad \text{(hatching)} \\ \to & \{\!|M[N/x][P/y]|\!\} \quad \text{(reaction)} \end{array}$$

Since the λ-calculus is embedded in our γ-calculus, one can define arbitrary combinators such as a "replicator", D, that satisfies $(\mathrm{D}M) \xrightarrow{*} M \,|\, (\mathrm{D}M)$ for all M, or a "killer", U, that satisfies $(\mathrm{U}M) \xrightarrow{*} \mathrm{U}$. This is easy using standard fixpoint combinators. Moreover, our concurrent γ-calculus is more powerful than the λ-calculus. The most important non λ-definable object that can now be constructed is the *internal choice* (or more accurately *join*) operator. To see this, let us denote by K and F the two cancellators, i.e., respectively $\lambda x.\lambda y.x$ and $\lambda x.\lambda y.y$ (in our syntax $x^- y^- x$ and $x^- y^- y$). Then the choice operator is defined by

$$\oplus =_{\text{def}} \langle \mathrm{K} \,|\, \mathrm{K}^+ \,|\, \mathrm{F}^+ \rangle$$

This operator may be evaluated either into K, like (KK)F, or into F, like (KF)K, therefore one easily sees that $\oplus MN \xrightarrow{*} M$ and $\oplus MN \xrightarrow{*} N$. Clearly such a combinator is not λ-definable since it does not preserve the Church-Rosser property.

As in [7], we extend our syntax by defining *concurrent abstractions*, that is sets of negative valences. More precisely, we define receptors of the form $[x_1 \,|\, \cdots \,|\, x_n]^- M$ where $x_1, \ldots, x_n$ are distinct variables. Such a term is able to receive n unordered values, to be substituted for these variables in M. Obviously these terms can be incorporated in our calculus with an additional rule:

choice:

$$[x_1 \,|\, \cdots \,|\, x_n]^- M \to x_i^- [\cdots \,|\, x_{i-1} \,|\, x_{i+1} \,|\, \cdots]^- M$$

Concurrent abstractions do not add power to the original calculus, since we can also define an atomic receptor $[x_1 \,|\, \cdots \,|\, x_n]^- M$ as a choice among all possible permutations $x_{i_1}^- \cdots x_{i_n}^- M$. For instance, using an infix notation for the choice:

$$[x \,|\, y]^- M =_{\text{def}} x^- y^- M \oplus y^- x^- M$$

The concurrent abstraction feature allows us to define combinators in a very compact way. For instance, the choice operator can be redefined by $\oplus = [x \,|\, y]^- x$, that is a parallel variant of the usual cancellator K.

We can also define a "parallel or", which is a parallel variant of the usual "left-sequential or" (cf. [7]). Let us see this point in some detail. It is known (see [4]) that $\mathrm{K} = x^- y^- x$ and $\mathrm{F} = x^- y^- y$ can be regarded as the truth values, respectively true and false. Then one may define a combinator for disjunction, namely $\mathrm{V} = x^- y^- (x\mathrm{K})y$. This combinator is such that $\mathrm{VK}X$ reduces to K and $\mathrm{VF}X$ reduces to X. However, $\mathrm{V}X\mathrm{K}$ (that is "X or true") cannot be in general reduced to K without evaluating X. For instance if, as usual, Ω denotes the non-terminating term $\Delta\Delta$ (where $\Delta = x^-(xx)$ is the duplicator) then the evaluation of $\mathrm{V}\Omega\mathrm{K}$ does not terminate. This is why V is "left-sequential". Moreover from Berry's sequentiality theorem (see [4]), one can show that there is no λ-definable combinator representing *parallel disjunction*, that is a combinator O such that both $\mathrm{O}\mathrm{K}X$

and OXK reduce to K, without evaluating X (and obviously OFF reduces to F). This combinator exists in the γ-calculus and is represented by:

$$O = [x\,|\,y]^-(x\mathrm{K})y$$

that is a parallel variant of the left-sequential disjunction (or equivalently a choice between left-sequential disjunction V and right-sequential disjunction $y^-x^-(x\mathrm{K})y$, see [10]).

5.3 Semantics

It seems fair to say that we have not yet established that "parallel disjunction is γ-definable". This is a semantic statement, so we would have first to define an equivalence relation $\simeq$ on γ-terms such that (using an infix notation for the "parallel disjunction" combinator O):

$$\mathrm{K}\,O\,\Omega \simeq \mathrm{K} \simeq \Omega\,O\,\mathrm{K}$$

and

$$\Omega\,O\,\Omega \simeq \Omega$$

In [7] it was proposed to adapt the notion of observational bisimulation $\approx$ of CCS [18] (see also [21]), to serve as the *semantic* equivalence. We could define this notion here (with the idea that x^- is an input guard and M^+ an output action), but this does not seem to be a good choice. For instance we would have $(\mathrm{K}\,O\,\Omega) \not\approx \mathrm{K}$ since $(\mathrm{K}\,O\,\Omega)$ may be reduced to $\Omega\mathrm{KK}$, a term without any communication capability, which is certainly different from K.

As a matter of fact, observational bisimulation has often been criticized for being too discriminating, and weaker "extensional" equivalences have been proposed (for a survey, see [12] and [15]). For instance Darondeau in [11] argued that "a semantics which stems from more sophisticated observers [than programs] is not really extensional". In other words, the semantics of processes should be derived from their observation by means of *program contexts* $C[]$. These program contexts may be regarded as *tests* over processes, and there is a natural way to define an associated *testing equivalence* (cf. [14]): two process are equivalent if they pass the same tests. This is the kind of semantical equality we propose for our γ-calculus. However, we shall not follow [11] and [14] for what concerns the result of experiments. To report the success of a test we shall use, as in [1], the simplest operational information, namely *convergence*, that is existence of a normal form: the agent M pass the test $C[]$ if $C[M]$ converges.

Formally, an agent M is said to converge, in notation $M\Downarrow$, if and only if there exists an *inert* solution S such that

$$\{\!|\,M\,|\!\}\xrightarrow{*} S$$

Then the definition of the testing preorder (on closed terms) is exactly the one of Morris' preorder (cf. [4], exercise 16.5.5, and [1]), that is:

$$M \sqsubseteq N \;\Leftrightarrow_{\mathrm{def}}\; \forall C.\; C[M]\Downarrow \;\Rightarrow\; C[N]\Downarrow$$

As usual the associated equivalence $\simeq$ is given by

$$M \simeq N \;\Leftrightarrow_{\mathrm{def}}\; M \sqsubseteq N \;\&\; N \sqsubseteq M$$

Let us see an example, showing that testing allows to distinguish divergent terms in the γ-calculus (unlike the lazy λ-calculus). We still use MN to abbreviate application, that is $\langle M\,|\,N^+\rangle$. As we saw, the typically divergent λ-term is $\Omega = \Delta\Delta$ where Δ is the duplicator $x^-(xx)$. It might be observed that $P = (\Delta\,|\,\Delta^+)$ is also a divergent term, since it can only be evaluated into Ω. Similarly, we can define a "triplicator" $\Upsilon = x^-((xx)x)$, and it is easy to see that $Q = (\Upsilon\,|\,\Upsilon^+)$ is again a divergent term. Now there is a test separating P and Q, namely

$$C = \langle\langle [\cdot]\,|\,z^-(\mathrm{F})^+\rangle\,|\,\Omega^+\rangle$$

(recall that $\mathrm{F} = x^-y^-y$, hence $\mathrm{F}M \xrightarrow{*} I = y^-y$). It is not difficult to see that $C[P]$ diverges, whereas $C[Q]\xrightarrow{*} I$, therefore $P \not\simeq Q$.

We shall not investigate here the properties of the testing preorder. A first step would be to prove a generalization of the well-known "context lemma" (cf. [10]), showing that observers of the form

$$\langle\cdots\langle[\cdot]\,|\,R_1\rangle\cdots|\,R_k\rangle$$

are enough to test the agents, that is

$$M \sqsubseteq N \Leftrightarrow \forall k\;\forall R_1,\ldots,R_k.$$
$$\langle\cdots\langle M\,|\,R_1\rangle\cdots|\,R_k\rangle\Downarrow \;\Rightarrow\; \langle\cdots\langle N\,|\,R_1\rangle\cdots|\,R_k\rangle\Downarrow$$

Such a result would allow us to give a simple proof of the desired properties of the parallel-or combinator.

6 Conclusion

Unlike other models, the Γ [2] and cham models handle (true) concurrency as *the* primitive built-in notion. What the cham model adds to Γ is the structure of molecules as terms and the notion of a subsolution. The CCS and TCCS implementations give a simple operational vision of these calculi. Inference rules are replaced by standard rewrite rules. The difference between internal and external transitions is made obvious and so are the well-known difficulties with sums

considered as programming primitives. More powerful "universal" process calculi such as MEIJE [6] can be handled as well. The concurrent λ-calculus fully exploits the ability of going back and forth between terms and solutions. It can be viewed as a direct extension of the lazy λ-calculus of [1].

Of course, this is still a preliminary work. Other concurrent computation paradigms should also be modelled; we think in particular of modelling process handling in operating systems, and of providing a cham for the new calculus of mobile processes proposed by Milner & al. in [20]. The theory of machine executions should also be fully developed.

Acknowledgments

We are indebted to Ilaria Castellani, Philippe Darondeau, Matthew Hennessy, Robin Milner and Serge Yoccoz for helpful discussions about this work and previous versions of the paper.

References

[1] Samson Abramsky. The lazy λ-calculus. In D. Turner, editor, *Declarative Programming*, Addison Wesley, 1988.

[2] Jean-Pierre Banâtre, Anne Coutant, and Daniel Le Metayer. A parallel machine for multiset transformation and its programming style. In *Future Generation Computer Systems 4*, pages 133–144, North-Holland, 1988.

[3] Jean-Pierre Banâtre and Daniel Le Metayer. *A New Computational Model and Its Discipline of Programming.* Technical Report INRIA Report 566, 1986.

[4] Henk Barendregt. *The Type-Free Lambda-Calculus. Studies in Logic Volume 103*, North-Holland, 1981.

[5] Gérard Berry. Séquentialité de l'evaluation formelle des λ-expressions. In B. Robinet, editor, *Program Transformations 3rd International Colloquium on Programming*, pages 67–80, DUNOD, Paris, 1978.

[6] Gérard Boudol. Notes on algebraic calculi of processes. In K. Apt, editor, *Logic and Models of Concurrent Systems*, NATO ASI Series F13, 1985.

[7] Gérard Boudol. Towards a lambda-calculus for concurrent and communicating systems. In *TAPSOFT 1989, Lecture Notes in Computer Science 351*, pages 149–161, Springer–Verlag, 1989.

[8] Nicholas Carriero and David Gelerntner. Linda in context. *Communications of the ACM*, 32(4):444–458, 1989.

[9] Mani Chandy and Jayadev Misra. *Parallel Program Design, a Foundation.* Addison-Wesley, 1988.

[10] Pierre-Louis Curien. *Categorical Combinators, Sequential Algorithms, and Functional Programming. Research Notes in Theoretical Computer Science*, Pitman, London, John Wiley & Sons, New York, Toronto, 1986.

[11] Philippe Darondeau. About fair asynchrony. *Theoretical Computer Science*, 37:305–336, 1985.

[12] Rocco De Nicola. Extensional equivalences for transition systems. *Acta Informatica*, 24:211–237, 1987.

[13] Rocco De Nicola and Matthew Hennessy. CCS without τ's. In *TAPSOFT 87, Lecture Notes in Computer Science 249*, pages 138–152, Springer-Verlag, 1987.

[14] Rocco De Nicola and Matthew Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[15] Matthew Hennessy. Observing processes. In *Linear Time, Branching Time and Partial Orders in Logics and Models for Concurrency, Lecture Notes in Computer Science 354*, pages 173–200, Springer-Verlag, 1989.

[16] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[17] Peter Landin. The mechanical evaluation of expressions. *Computer Journal*, 6:308–320, 1964.

[18] Robin Milner. *Communication and Concurrency. International Series in Computer Science*, Prentice Hall, 1989.

[19] Robin Milner. *Program Semantics and Mechanized Proofs*, pages 3–44. Mathematical Center Tracts 82, Amsterdam, 1976.

[20] Robin Milner, Joachim Parrow, and David Walker. *A Calculus of Mobile Processes.* Technical Report ECS-LFCS-89-85, LFCS, Edinburgh University, 1989.

[21] Bengt Thomsen. A calculus of higher-order communicating systems. In *Proc. 16th ACM Annual Symposium on Principles of Programming Languages*, pages 143–154, 1989.