The Chemical Abstract Machine

Gérard Berry

Ecole Nationale Supérieure des Mines de Paris
Centre de Mathématiques Appliquées
Sophia-Antipolis, 06560 Valbonne, France

Gérard Boudol

INRIA
Sophia-Antipolis, 06560 Valbonne, France

October 26, 1998

Abstract

We introduce a new kind of abstract machine based on the chemical metaphor used in the Γ language of Banâtre and Le Métayer. 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 and of the fragment of Milner, Parrow, and Walker's Calculus of Mobile Processes used by Milner to encode the lambda-calculus. We also give ideas on 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* or *cham*, suited to model concurrent computations. We show that chemical abstract machines can implement known models of concurrent computation such as algebraic process calculi [26, 8], Milner's Mobile Processes calculus [28, 27], and a concurrent λ-calculus similar to the one presented in [9].

Abstract Machines

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 [23] and the Categorical Abstract Machine [12] are used to study and implement the λ-calculus, while the SMC machine [25] 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 [30], Communicating Automata [3], or Data Flow Networks [22] can be considered as abstract machines, but certainly they lack expressive power. More expressive models such as Algebraic Process Calculi [26, 8] are intended to be specification formalisms for distributed systems rather than abstract machines. Implementation models of Concurrent Programming Languages such as CSP [21] are conceptually based on standard sequential machine models augmented with scheduling facilities, not on specific abstract machines.

The Γ Language

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 Métayer [4, 5]. 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. 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 [4, 11]. The solution transformation process is obviously inherently parallel: any number of reactions can be performed in parallel, provided that they involve disjoint sets of molecules.

Let us give a simple but striking example from [4, 5]. 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 [4, 5] 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.

Other authors have proposed models in which the flow of control is made completely implicit: see for example the set of assignments used in UNITY [11] or the tuple space of Linda [10]. These models are based on similar concepts and bear the same degree of potential parallelism. Another instance of multiset rewriting is the token game in Petri Nets: markings are multisets of places which play the role of molecules, and transitions are rules to transform markings.

The Chemical Abstract Machine

To define the chemical abstract machine, we elaborate on the original Γ language by specifying a syntax for molecules and refining the classification of rules.

Like when dealing with Turing machine, there are two description levels. The general chemical abstract machine level abstractly defines a syntactic framework and a simple set of structural behavior laws. An actual machine is given by adding a specific molecule syntax and a set of transformation rules that specify how to produce new molecules from old ones.

At the upper cham level, molecules are bound to be terms of some algebra. A general *membrane* construct transforms a solution into a single molecule, and an associated general *airlock* construct makes the membrane somewhat porous to permit communication between an encapsulated solution and its environment. The laws specify how reactions defined by specific transformation rules can take place and how membranes and airlocks behave.

A specific machine is defined by giving the molecule algebra and the rules. The rules have no premisses and are purely local, unlike the inference rules classically used in structural operational semantics [29].

In a given cham, we (informally) classify molecules and rules. Not all molecules directly exhibit interaction capabilities. Those which do are called *ions*. The interaction capability of an ion is generally determined only by a part of it that we call its *valence*. The rules that build new molecules from ions are called *reaction rules*. The non-ion molecules can be *heated* by heating rules to break them into simpler submolecules. Conversely, a set of molecules can cool down to a complex molecules using reverse *cooling rules*. In our examples, heating and cooling rules closely correspond to usual structural equivalence.

The strength of the cham 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 generalisations of the lambda-calculus.

Structure of the Paper

In this paper, we concentrate on the descriptive power of chemical abstract machines, by illustrating the use of the concept. The techniques needed to study chemical abstract machines and to compare them with more usual formalisms remain largely to be developed.

To make the reader familiar 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 [16] calculus and indicate how to handle other process calculi. Section 5 presents a cham for a subset of Milner's Calculus of Mobile Processes [28]. Section 6 is devoted to a concurrent lambda-calculus similar to that of [9]. We conclude in Section 7.

2 Handling a Subset of CCS

Our first illustrative example is a fragment CCS^- of Milner's process calculus CCS [26], containing the most basic operators $\mathtt{0}$ (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:

$$p ::= \mathtt{0} \;\mid\; \alpha.p \;\mid\; (p \,|\, p) \;\mid\; p \backslash a$$

2.1 Inference Rules Semantics

Process calculi semantics are usually defined by inference rules in Plotkin's structural operational semantics style [29], called SOS for short. Milner's original rules involve a special 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 [16] that define two kinds of transitions between agents: the *internal* transitions $p \to p'$ and the *labelled* transitions $p \xrightarrow{\alpha} p'$. Intuitively, $p \to p'$ 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 transition systems are defined in a structural way: the behaviour of an agent is deduced from the behaviours of its components. Since internal communications generate internal transitions, the inference system for $\to$ invokes the one for $\xrightarrow{\alpha}$:

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

$$\frac{p \to p'}{p \,|\, q \to p' \,|\, q \ \text{ and } \ q \,|\, p \to q \,|\, p'}$$

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

$$\frac{p \xrightarrow{\alpha} p' \quad q \xrightarrow{\overline{\alpha}} q'}{p \,|\, q \to p' \,|\, q'}$$

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

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

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 behaviours, we consider agents as *molecules* directly reacting with each other within a *solution*, that is a multiset $S = \{\!| p, q, \ldots |\!\}$. There are only two basic rules:

$$p \,|\, q \rightleftharpoons p,\ q \qquad (\textit{parallel})$$

$$a.p,\ \overline{a}.q \rightarrow p,\ q \qquad (\textit{reaction})$$

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 $\rightharpoonup$) to decompose it into its components p and q, and conversely that any pair p, q of molecules can be *cooled down* (symbol $\rightharpoondown$) 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. This is very similar to the decomposition of processes into sequential components used in the translation from CCS to Petri Nets presented in [18].

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.

Example

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

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

Cleaning Up Solutions

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

$$\mathtt{0} \rightharpoonup \qquad (\textit{inaction cleanup})$$

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

Non-Determinism

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

Cham versus SOS

The reader will appreciate the simplicity of the chemical executions compared to SOS executions. The rules for internal execution have no premisses and do not involve the labelled transitions $p \xrightarrow{\alpha} p'$, which represent external observation of the communication capabilities. In SOS, labelled transitions are necessary to overcome the rigidity of syntax when performing communication between two syntactically distant agents; in a term of the form $(\ldots a.p \ldots)|(\ldots \overline{a}.q \ldots)$ the inductive labelled transition system is used to report the a and $\overline{a}$ communication capabilities to the parallel operator and the communication is in fact realised by this operator. On the contrary, in the cham, we just make the syntactic distance vanish by putting molecules into contact when they want to communicate, and their communication is direct. Notice that the notion of a syntactic position disappears even for the standard parallel construct '$\,|\,$': it is impossible to know whether $\{\!|\, p, q \,|\!\}$ was obtained by heating $\{\!|\, p \,|\, q \,|\!\}$ or $\{\!|\, q \,|\, p \,|\!\}$ (unlike in [18]).

Chemical concurrency is *naturally* associative and commutative, since multisets are intrinsically unordered. On the contrary, the SOS semantics needs to first introduce behaviours to recover concurrency out of syntax, then to define what it means for processes to be equivalent, and finally to *prove* equivalences such as $p \,|\, q \sim q \,|\, p$.

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

Another advantage of the cham appears in the sequences of execution steps: one can directly chain reactions by keeping the solution hot, while SOS evaluation involves structural rules at *each* computation step. In other words, the use of the structural rules for '$\mid$' is factored throughout an execution by the heating process.

However, the cham can also spend its time looping heating molecules and cooling them back. To us, this is not really a drawback, but the immediate consequence of the abstract machine approach: the machine not only performs reactions but also searches for them. In the SOS semantics, the search for a proof is not part of the formalism, and the operational character is somewhat doubtful. It is of course possible to superimpose control mechanism or fairness constraints on a given cham, but we have no reason to do it by default and we shall not do it here.

Observation of a solution

So far, we have seen that the cham framework is well suited to deal with the *execution* of processes, as opposed to their *observation*. This is in the line of the standard notion of an operational semantics, where one uses unlabelled transitions for reduction, evaluation, rewriting, or machine runs, see [29]. However, classical process calculi semantics nicely define observation, and we must also do it if we want to make any use of chams and to define appropriate equivalence notions.

A solution should be able to perform an externally observable 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, we shall see that such a technique would violate Milner's most useful principle, which states that observing a process should not be different from communicating with it, using another process to describe the observer. The right solution is to make the observer *react* with the valence of any molecule of the solution. This requires a richer machinery developed 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:

$$(\alpha.q)\backslash a \rightleftharpoons \alpha.(q\backslash a) \quad \text{if } \alpha \notin \{a, \overline{a}\} \qquad (\textit{restriction ion})$$

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. We need a way to hierarchically structure solutions.

Membranes

To let p evolve on its own, we put it in a new local solution contained within a *membrane* $\{\!|\,.\,|\!\}$. Technically, we enrich our molecule syntax by considering any solution contained within a membrane as a *single molecule* to which operators such as '\' or '[]' (external choice, see [16]) can be applied. We can then construct complex molecules containing subsolutions, such as $\{\!|\, \mathsf{0},\ a.b.\mathsf{0} \,|\!\}\backslash a$. The rule that opens or closes a membrane below a restriction is:

$$p\backslash a \rightleftharpoons \{\!|\, p \,|\!\}\backslash a \qquad (\textit{restriction membrane})$$

Once created, a subsolution evolves by its own and obeys the same rules as the global solution. Therefore, reactions can now happen under the restriction operator. To realise global communications, we need to make the membrane *porous* to valences. A first simple idea would be to use a heavy ion formation rule such as:

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

Note that a heavy ion could emit the α-particle to the environment. However, 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 an arbitrary 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 $\{\!|\, a.\mathsf{0},\ \{\!|\, \overline{a}.\mathsf{0},\ b.\mathsf{0} \,|\!\}\backslash c \,|\!\}$ when choosing b: we are stuck with the inert solution $\{\!|\, a.\mathsf{0},\ b.(\{\!|\, \overline{a}.\mathsf{0},\ \mathsf{0} \,|\!\}\backslash c) \,|\!\}$.

Airlocks

The technique we propose involves two steps. First, we introduce a new mechanism at the general chemical machine level: the *airlock* mechanism. It uses a new molecule constructor '$\triangleleft$' that builds a molecule $m \triangleleft S$ out of a molecule m and a solution S. As well as the membrane constructor, the airlock constructor is generic and applicable to all sorts of chams, not only to the CCS one. The reversible airlock creation mechanism extracts any molecule from a solution (not necessarily an ion), and puts the rest of the solution within a membrane:

$$\{\!|\, m, m_1, m_2, \ldots, m_n \,|\!\} \rightleftharpoons \{\!|\, m \triangleleft \{\!|\, m_1, m_2, \ldots m_n \,|\!\} \,|\!\} \qquad (\mathit{airlock})$$

Since it is contained within a membrane, the new subsolution $\{\!|\, m_1, m_2, \ldots m_n \,|\!\}$ is allowed to freely continue internal reactions.

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

$$(\alpha.p) \triangleleft S \rightleftharpoons \alpha.(p \triangleleft S) \qquad (\mathit{heavy\ \ ion})$$

In this way, we obtain reversibility by preserving the attachment between α and p. By creating or removing airlocks, a restriction molecule can propose several valences in succession to its environment until a communication takes place.

Example

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

$$\begin{array}{cll}
 & \{\!|\, a.\mathsf{0} \mid (\overline{a}.p \mid q) \backslash b \,|\!\} & \\
\stackrel{*}{\rightharpoonup} & \{\!|\, a.\mathsf{0},\ \{\!|\, \overline{a}.p,\ q \,|\!\} \backslash b \,|\!\} & \text{(parallel, restriction membrane)} \\
\rightharpoonup & \{\!|\, a.\mathsf{0},\ \{\!|\, (\overline{a}.p) \triangleleft \{\!|\, q \,|\!\} \,|\!\} \backslash b \,|\!\} & \text{(airlock)} \\
\rightharpoonup & \{\!|\, a.\mathsf{0},\ \{\!|\, \overline{a}.(p \triangleleft \{\!|\, q \,|\!\}) \,|\!\} \backslash b \,|\!\} & \text{(heavy ion)} \\
\rightharpoondown & \{\!|\, a.\mathsf{0},\ (\overline{a}.(p \triangleleft \{\!|\, q \,|\!\})) \backslash b \,|\!\} & \text{(restriction membrane)} \\
\rightharpoonup & \{\!|\, a.\mathsf{0},\ \overline{a}.((p \triangleleft \{\!|\, q \,|\!\}) \backslash b) \,|\!\} & \text{(restriction ion)} \\
\rightarrow & \{\!|\, \mathsf{0},\ (p \triangleleft \{\!|\, q \,|\!\}) \backslash b \,|\!\} & \text{(reaction)} \\
\stackrel{*}{\rightharpoonup} & \{\!|\, \{\!|\, p \triangleleft \{\!|\, q \,|\!\} \,|\!\} \backslash b \,|\!\} & \text{(inaction cleanup, restriction membrane)} \\
\rightharpoondown & \{\!|\, \{\!|\, p,\ q \,|\!\} \backslash b \,|\!\} & \text{(airlock)}
\end{array}$$

Unlike in the simple case of the previous section, we cannot simply keep the solution hot. We must sometimes cleverly cool down the solution to remove membranes.

Reversibility is guaranteed by the usage of membranes. At the second step, if we choose to put q in the airlock instead of $\overline{a}.p$, there is no precipitate since we can put q back in the subsolution and build a new airlock with $\overline{a}.p$. Once the heavy ion $\overline{a}.((p \triangleleft \{\!|\, q \,|\!\})\backslash b)$ has been constructed, it is not possible to put p back into q's solution before an a-communication occurs, since the airlock is not any more contained within a membrane. It is not possible to build such a membrane, since the restriction membrane rule applies only to molecules and not inside them. This guarantees that we really emulate the CCS behaviour.

Defining Observations

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 $S \overset{\alpha}{\Rightarrow} S'$ if there exist a molecule m such that $S \overset{*}{\rightarrow} \{\!|\, \alpha.m \,|\!\}$ and $\{\!|\, m \,|\!\} \overset{*}{\rightarrow} S'$. For example, one has

$$\{\!|\, a.\mathsf{0},\ b.\mathsf{0} \,|\!\} \overset{a}{\Rightarrow} \{\!|\, b.\mathsf{0} \,|\!\}$$

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

The relation between this new kind of (weak) observational behaviour and the standard TCCS one will be precisely stated in section 4. Note that we could also define the strong labelled transitions $p \overset{\alpha}{\rightarrow} p'$ as $\exists m\ \{\!|\, p \,|\!\} \overset{*}{\rightharpoonup} \{\!|\, \alpha.m \,|\!\}\ \&\ \{\!|\, m \,|\!\} \overset{*}{\rightharpoonup} \{\!|\, p' \,|\!\}$.

3 Formal Definitions

3.1 Chemical Abstract Machines

A *chemical abstract machine* or *cham* $\mathcal{C}$ is specified by defining *molecules* m, m', etc., *solutions* S, S', etc., and *transformation rules* that determine a *transformation relation* $S \rightarrow S'$. The transformation rules are divided into two categories: general *laws* applicable to all chams, and specific rules that define a given cham. Only general laws involve premisses. Specific rules are bound to be elementary rewriting rules.

Molecule Syntax, Solutions

Molecules are terms of algebras, with specific operations for each cham. Solutions $S, S', \ldots$ are finite multisets of molecules, written $\{\!|\, m_1, m_2, \ldots, m_k \,|\!\}$. Furthermore, in each cham, any solution S can itself be considered as a single molecule and can therefore appear as a *subsolution* of another molecule. The corresponding $\{\!|\,.\,|\!\}$ operator is called the *membrane* operator. Some chams, but not all of them, use the additional *airlock* constructor '$\triangleleft$'. An airlock is a molecule of the form $m \triangleleft S$ where m is a molecule and S is a solution [1].

For instance, if $\mathtt{0}$ and $+$ are the molecule building operations, then $\mathtt{0}$, $\mathtt{0}+\mathtt{0}$, $\mathtt{0}+\{\!|\,\mathtt{0}\,|\!\}$, and $\{\!|\,\mathtt{0}, \mathtt{0} \triangleleft \{\!|\,\mathtt{0}+\mathtt{0}, \mathtt{0}\,|\!\}\,|\!\}$ are molecules, the latter also being a solution.

The multiset union of S and S' is written $S \uplus S'$. As in the λ-calculus [6], we use the context notation $C[\,]$ to denote a molecule with a hole $[\,]$ in which to place another molecule.

Specific Rules

The specific 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 specific 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 schema.

General Laws

All chams obey the following four laws:

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

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

  if M_1, M_2, …, M_k, M'_1, M'_2, …, M'_l, are instances of the m_i's and the m_j's by a common substitution, then

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

[1] Very precise algebraic definitions of these notions are given in [24].

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

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

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

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

- **The Airlock Law**

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

Remarks

The chemical and membrane laws are the only ones to involve premisses. They factor out what is usually called "structural rules" in particular calculi. All other laws and rules are purely local. Note that the transitions of a cham are always unlabelled ones.

A cham is an intrinsically parallel machine: one can simultaneously apply several rules to a solution provided that their premisses are not conflicting, i.e. that no molecule is involved in more than one rule; one can also transform subsolutions in parallel. In this paper, we only study the descriptive power of chams; it does not depend on using parallel evaluation, since a non-conflicting parallel application of rules is equivalent, up to permutations, to any sequence of the individual rules. See [4] 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* $\to$. The distinction is not enforced by the formalism. Pragmatically, we find it convenient to use the following conventions:

- Structural manipulation is performed by heating/cooling rule pairs: 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 $\rightleftharpoons$. In the sequel, we shall always assume that the transitions given by the airlock law are heating and cooling ones.

- Cleanup is performed by heating rules, with generally no associated cooling rule. The purpose is to remove useless molecules.

- Reaction rules really change the information in the solution in an irreversible way. Usually, they involve molecules that cannot be heated further and are called *ions*; the way an ion can react with another is determined by a portion of it that is called its *valence*.

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

Given the three kinds of rules, we say that a solution is *hot* (resp. *frozen*) if no heating (resp. cooling) rule applies to it. A solution is *inert* if no reaction rule applies to it, nor to any solution structurally equivalent to it. Since there is no default control mechanism ensuring fairness or distributed termination detection, these properties are semantic and observer-related. In particular, a machine has no way to detect that its solution is inert.

4 Process Calculi Chams

In this section, we finish the treatment of the TCCS process calculus, we relate the cham semantics with the original structural operational 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 [16] and of its SOS semantics. We have already seen the inaction '$\mathtt{0}$', parallel '$\,|\,$', prefixing '.', and restriction '$\backslash$' 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:

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

Relabelling

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:

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

Sums

Sums represent non-deterministic choices. There are several possible sums, see [16] for an extensive discussion. The simplest sum is the internal sum $\oplus$, which non-deterministically chooses a component:

$$p \oplus q \to p$$
$$p \oplus q \to q$$

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:

$$\frac{p \to p'}{p \| q \to p' \| q \quad \text{and} \quad q \| p \to q \| p'}$$
$$\frac{p \stackrel{\alpha}{\to} p'}{p \| q \stackrel{\alpha}{\to} p' \quad \text{and} \quad q \| p \stackrel{\alpha}{\to} p'}$$

Fixpoint

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:

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

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.

Simple Operators

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

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 a heavy ion $\alpha.(m \triangleleft S_1)$. To export the valence, we can use the rule:

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

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 recognise 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 realise the summand selection; we need a cooling rule of the form:

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

Here again, we must recognise 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:

$$
\begin{gathered}
< \{\!|\, \alpha.m \,|\!\}, S > \rightleftharpoons \alpha.(l :< m, S >) \\
l :< m, m' > \rightharpoonup m
\end{gathered}
$$

The rules for the right-hand-side choice are symmetric with a label r. Although it is not hard to implement, external sum is clearly far less natural than internal sum in our framework.

Example

Let us give a simple external sum evaluation example:

$$
\begin{array}{cc}
 & \{\!|\, a.\overline{b}.\mathsf{0} \mid ((\overline{a}.\mathsf{0} \mid b.\mathsf{0}) \| q) \,|\!\} \\
\overset{*}{\rightharpoonup} & \{\!|\, a.\overline{b}.\mathsf{0},\ \ < \{\!|\, \overline{a}.\mathsf{0},\ b.\mathsf{0} \,|\!\}, \{\!|\, q \,|\!\} > |\!\} \\
\overset{*}{\rightharpoonup} & \{\!|\, a.\overline{b}.\mathsf{0},\ \ < \{\!|\, \overline{a}.(\mathsf{0} \triangleleft \{\!|\, b.\mathsf{0} \,|\!\}) \,|\!\}, \{\!|\, q \,|\!\} > |\!\} \\
\overset{*}{\rightharpoonup} & \{\!|\, a.\overline{b}.\mathsf{0},\ \overline{a}.l :< \mathsf{0} \triangleleft \{\!|\, b.\mathsf{0} \,|\!\}, \{\!|\, q \,|\!\} > |\!\} \\
\longrightarrow & \{\!|\, \overline{b}.\mathsf{0},\ l :< \mathsf{0} \triangleleft \{\!|\, b.\mathsf{0} \,|\!\}, \{\!|\, q \,|\!\} > |\!\} \\
\rightharpoonup & \{\!|\, \overline{b}.\mathsf{0},\ \mathsf{0} \triangleleft \{\!|\, b.\mathsf{0} \,|\!\} \,|\!\} \\
\rightharpoonup & \{\!|\, \overline{b}.\mathsf{0},\ \mathsf{0},\ b.\mathsf{0} \,|\!\}
\end{array}
$$

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 solution. This requires to use both the airlock law and the chemical law.

4.3 The Complete TCCS Cham

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

Syntax

Agents:

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

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

$$\begin{array}{ll}
p \mid q \rightleftharpoons p,\ q & (\textit{parallel}) \\
a.m,\ \overline{a}.n \rightarrow m,\ n & (\textit{reaction}) \\
(\alpha.p)\triangleleft S \rightleftharpoons \alpha.(p \triangleleft S) & (\textit{heavy ion}) \\
m\backslash a \rightleftharpoons \{\!|\, m \,|\!\}\backslash a & (\textit{restriction membrane}) \\
(\alpha.m)\backslash a \rightleftharpoons \alpha.(m\backslash a) \quad \text{if } \alpha \notin \{a, \overline{a}\} & (\textit{restriction ion}) \\
m[\phi] \rightleftharpoons \{\!|\, m \,|\!\}[\phi] & (\textit{relabelling membrane}) \\
(\alpha.m)[\phi] \rightleftharpoons \phi(\alpha).(m[\phi]) & (\textit{relabelling ion}) \\
p \oplus q \rightarrow p & (\oplus\textit{-left}) \\
p \oplus q \rightarrow q & (\oplus\text{-right}) \\
p \| q \rightleftharpoons <\{\!|\, p \,|\!\}, \{\!|\, q \,|\!\}> & (\|\text{-expansion}) \\
<\{\!|\, \alpha.m \,|\!\}, S> \rightharpoonup \alpha.l:<m,S> & (\textit{left } \|\textit{-ion}) \\
<S, \{\!|\, \alpha.m \,|\!\}> \rightharpoonup \alpha.r:<S,m> & (\textit{right } \|\textit{-ion}) \\
l:<m,m'> \rightharpoonup m & (\textit{left projection}) \\
r:<m,m'> \rightharpoonup m' & (\textit{right projection}) \\
\mathtt{fix}_i(\vec{x} = \vec{p}) \rightharpoonup p_i[\vec{\mathtt{fix}}(\vec{x} = \vec{p})/\vec{x}] & (\textit{fixpoint})
\end{array}$$

Additional Cleanup Rules

$$\begin{array}{ll}
\mathtt{0} \rightharpoonup & (\textit{inaction cleanup}) \\
\{\!|\,|\!\}\backslash a \rightharpoonup & (\textit{restriction cleanup}) \\
\{\!|\,|\!\}[\phi] \rightharpoonup & (\textit{relabelling cleanup})
\end{array}$$

4.4 Comparing the Cham and SOS

We define weak observation as explained in section 2.

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

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

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

Case 1: Assume $p = p_1 \,|\, p_2 \rightarrow p'_1 \,|\, p'_2 = p'$ with $p_1 \overset{\alpha}{\rightarrow} p'_1$ and $p_2 \overset{\overline{\alpha}}{\rightarrow} p'_2$ for some α, by induction, there exist m'_1 and m'_2 such that $\{\!|\, p_1 \,|\!\} \overset{*}{\rightarrow} \{\!|\, \alpha.m'_1 \,|\!\}$, $\{\!|\, m'_1 \,|\!\} \overset{*}{\rightleftharpoons} \{\!|\, p'_1 \,|\!\}$, $\{\!|\, p_2 \,|\!\} \overset{*}{\rightarrow} \{\!|\, \alpha.m'_2 \,|\!\}$, and $\{\!|\, m'_2 \,|\!\} \overset{*}{\rightleftharpoons} \{\!|\, p'_2 \,|\!\}$. We build the following transformation sequence:

$$
\begin{array}{ccl}
 & \{\!|\, p \,|\!\} & \\
= & \{\!|\, p_1 \,|\, p_2 \,|\!\} & \\
\rightharpoonup & \{\!|\, p_1,\ p_2 \,|\!\} & \text{(parallel)} \\
\overset{*}{\rightarrow} & \{\!|\, \alpha.m'_1,\ \overline{\alpha}.m'_2 \,|\!\} & \text{(cham laws)} \\
\rightarrow & \{\!|\, m'_1,\ m'_2 \,|\!\} & \text{(reaction)} \\
\overset{*}{\rightleftharpoons} & \{\!|\, p'_1,\ p'_2 \,|\!\} & \text{(cham laws)} \\
\rightharpoondown & \{\!|\, p'_1 \,|\, p'_2 \,|\!\} & \text{(parallel)} \\
= & \{\!|\, p' \,|\!\} &
\end{array}
$$

which shows the required property of p.

Case 2: Assume $p = q\backslash a \xrightarrow{\alpha} q'\backslash a = p'$, 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 transformation sequence:

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

Furthermore, one has:

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

which shows the required property of p.

Proving 2) is harder and we just sketch the proof architecture. The properties of $\overset{*}{\rightarrow}$ and $\overset{\alpha}{\Rightarrow}$ 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 $\overset{*}{\rightarrow}$ is obvious with $p' = p$. To prove the property of $\overset{\alpha}{\Rightarrow}$, 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, let p be a TCCS term and assume $\{\!|\, p \,|\!\} \overset{*}{\rightleftharpoons} C[\alpha.m]$ where the ion $\alpha.m$ floats in some subsolution. Then one can structurally transform $C[\alpha.m]$ into $C'[\alpha.q]$ 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 $\{\!|\, C_1[q] \,|\!\} \stackrel{*}{\rightleftharpoons} \{\!|\, C'[q] \,|\!\}$.

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

Assume now that the number of irreversible transitions in a given derivation is strictly positive. The derivation can be written $S \xrightarrow{*} S_1 \rightarrow S_2 \xrightarrow{*} S'$ 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 $S_1 \stackrel{*}{\rightleftharpoons} C[\alpha.q][\overline{\alpha}.r]$ 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:

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

in 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:

$$a.m,\ \overline{a}.n \rightarrow \tau.(m \,|\, n) \qquad (\tau\!-\!\mathit{reaction})$$

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 $\stackrel{\tau}{\Rightarrow}$ 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 behaviour 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 compared to those of TCCS. This is actually quite well-known to CCS simulator implementors.

Handling other process calculi raises no particular problem. For example, one can define a cham for MEIJE [8], which is universal among the labelled process calculi [17]. One has to use a reaction rule similar to the one for CCS, and introduce two heating/cooling pairs for the *ticking* construct, similar to the ones for relabelling:

$$
\begin{array}{rcll}
\alpha * m & \rightleftharpoons & \alpha * \{\!|\, m \,|\!\} & \qquad (\textit{ticking membrane}) \\
\alpha * (\beta.m) & \rightleftharpoons & (\alpha \cdot \beta).(\alpha * m) & \qquad (\textit{ticking ion})
\end{array}
$$

5 Milner's Calculus of Mobile Processes

Milner's π-calculus of mobile processes is an extension of CCS that deals with *name passing*. Intuitively, channel names can be passed between processes through named channels. We only consider the restricted calculus studied in [27], which is powerful enough to simulate the lambda-calculus. The full π-calculus has other operators such as sums; they can be handled by chams just like the corresponding CCS operators.

Like CCS, the π-calculus deals with a set $\mathcal{N}$ of names and a set $\mathcal{L}$ of labels. We use the symbols $x, y, z, \ldots$ to range over names. The syntax of agents is as follows:

$$p ::= \mathsf{0} \;\mid\; x(y).p \;\mid\; \overline{x}y.p \;\mid\; (p \,|\, p) \;\mid\; (x)p \;\mid\; !p$$

There are two bindings operators: $x(y).p$ binds y in p and $(x)p$ binds x in p. Both bindings are subject to α-conversion. Intuitively, the agent $x(y).p$ waits to receive a name on channel x, and substitutes y by this name in p after reception. The agent $\overline{x}y.p$ sends the name y on channel x. Parallelism is interleaving, and replication conveniently replaces recursion: $!p$ generates $p \mid !p$.

5.1 The Original Semantics

In [27], Milner presents a semantics in a mixture of SOS and cham styles. Terms are considered modulo structural equalities, which include in particular associativity and commutativity of the parallel operator. This amounts to handle cham multisets in a purely algebraic framework. The exact structural equalities are:

$$
\begin{array}{cl}
p \mid q \equiv q \mid p, \quad p \mid (q \mid r) \equiv (p \mid q) \mid r & (\textit{multiset}) \\
p \equiv q \quad \text{if } p \text{ and } q \text{ are } \alpha\text{−convertible} & (\alpha\text{−}\textit{conversion}) \\
!p \equiv p \mid !p & (\textit{replication}) \\
(x)(p \mid q) \equiv p \mid (x)q \quad \text{if } x \text{ not free in } p & (\textit{scope extension}) \\
p \mid \mathsf{0} \equiv p, \quad (x)\mathsf{0} \equiv \mathsf{0} & (\textit{cleanup}) \\
(x)(y)p \equiv (y)(x)p & (\textit{restriction commutation})
\end{array}
$$

The inference rules are:

$$
\begin{array}{cl}
x(y).p \mid \overline{x}z.q \rightarrow p[z/y] \mid q & (\textit{communication}) \\
\dfrac{p \rightarrow p'}{p \mid q \rightarrow p' \mid q} & (\textit{parallel}) \\
\dfrac{p \rightarrow p'}{(y)p \rightarrow (y)p'} & (\textit{restriction}) \\
\dfrac{q \equiv p \quad p \rightarrow p' \quad p' \equiv q'}{q \rightarrow q'} & (\textit{structural equivalence})
\end{array}
$$

Scoping and Name Generation

The strength and difficulty of the calculus come from the dynamic character of name scoping. Reception binding is classical and raises no problem. Restriction binding is much more subtle, since it creates new names that can be exported outside their original scope. Consider for example the term $(x(y).\overline{y}u.\mathsf{0}) \mid ((z)(\overline{x}z.z(v).v))$. Initially, the scope of z is limited to the second branch of the parallel, and z is unknown in the first branch. By scope extension, the term is equivalent to $(z)((x(y).\overline{y}u.\mathsf{0}) \,|\, (\overline{x}z.z(v).v))$. One can then pass z to the first branch, obtaining the term $(z)((\overline{z}u.\mathsf{0}) \,|\, (z(v).v))$. From now on, z can be used as a communication channel between both branches. A last communication yields the term $(z)(\mathsf{0} \,|\, u)$, which is structurally equivalent to u.

5.2 A Cham Version of Mobile Processes

Milner's semantic rules are already in the spirit of the cham and are perfectly adequate to reason about term behaviors. However, they are fairly numerous and not very operational in character. Moreover, some of them are really naturally taken care of by the cham: the parallel and structural equivalence rules just express the chemical law, the other structural equalities correspond to simple heating/cooling rules, and the restriction rule is akin to the membrane law.

We present here two simple chams that perform all possible computations using only a small number of simple rules. Both chams share the following four elementary rules:

$$
\begin{array}{cl}
p \,|\, q \rightleftharpoons p,\ q & (\mathit{parallel}) \\
x(y).p,\ \overline{x}z.q \rightarrow p[z/y],\ q & (\mathit{reaction}) \\
!p \rightleftharpoons p,\ !p & (\mathit{replication}) \\
\mathsf{0} \rightharpoonup & (\mathit{inaction\ \ cleanup})
\end{array}
$$

The chams differ only in the way they handle restriction. The first one uses membranes and airlocks; it requires to implement α-conversion. The second one gets rid of α-conversion by using a *name server*, as do many operating systems mechanisms; it does not use membranes.

Mobile Processes with Membranes and Airlocks

In the first cham, we consider the restriction operation (x) as a molecule constructor that operates on solutions, just as we did in Section 4 for CCS. The specific rules are:

$$
(x)p \rightleftharpoons (y)p[y/x] \quad \text{if } y \text{ is not free in } p \qquad (\alpha - \mathit{conversion})
$$

$$(x)p \rightleftharpoons (x)\{\!|\, p \,|\!\} \qquad\qquad (\textit{restriction membrane})$$
$$(x)S,\ p \rightleftharpoons (x)\{\!|\, p \triangleleft S \,|\!\} \quad \text{if } x \text{ is not free in } p \qquad (\textit{scope extension})$$

A restriction agent $(x)p$ can be heated into a molecule $(x)\{\!|\ p\ |\!\}$ that can become $(x)S$ whenever $\{\!|\ p\ |\!\}$ can become S. To realize scope extension, we simply make the membrane permeable to agents q not having x as a free variable. If q meets $(x)S$, the restriction membrane rule puts in an airlock to be absorbed by S, yielding a molecule $(x)\{\!|\, q \triangleleft S \,|\!\}$ than can become $(x)(S \uplus \{\!|\, q \,|\!\})$. If x is free in q, then q cannot penetrate S; it is then necessary to cool the restricted solution down, to perform an α-conversion on the cold agent, and to heat up again to build an α-converted restricted solution into which q can enter. Notice than molecules can freely leave a restricted solution when the restricted name is not free in them; for this, it suffices to use the heating/cooling rules backwards.

Mobile Processes with Name Servers

The idea of our second cham is to forget about α-conversion and scope extension by actually generating names using a *name server*, as often done in operating systems. Technically, we enumerate the set of names, $\mathcal{N} = \{\underline{n}, n \in N\}$. The molecules are the π-calculus agents and the names themselves. There is only one specific rule:

$$(x)p,\ \underline{n} \rightarrow p[\underline{n}/x],\ \underline{n+1} \qquad (\textit{restriction})$$

To execute an agent p, we start with the solution $\{\!|\ p,\ \underline{n}\ |\!\}$ where n is any name of index bigger than those free in p. The initial molecule $\underline{n}$ plays the role of the name server that generates new names for restrictions. Each name generation reconfigurates the name server.

In fact, we have suppressed α-conversion only for restriction, and we may still have to perform it in substitutions. We can also suppress this remaining α-conversion if we don't stick to Milner's original view and introduce two separate name spaces for true constant names and for name variables, again as in operating systems. We then only allow name variables to be substituted by constant names, and we restrict communications to take place only on channels of constant names and to pass constant names. If we denote constant names by $\underline{m}, \underline{n}$ and name variables by x, y, the new syntax is:

$$u ::= \underline{m} \mid x$$
$$p ::= \mathsf{0} \mid u(x).p \mid \overline{u}u.p \mid (p \,|\, p) \mid (x)p \mid\ !p$$

Only the reaction rule needs to be adapted:

$$\underline{m}(y).p,\ \overline{\underline{m}}\,\underline{n}.q \rightarrow p[\underline{n}/y],\ q \qquad (\textit{reaction})$$

This technique could as well be applied to the original calculus.

We shall not give more details nor formally compare our chams with the original calculus. Clearly, it is easier to execute programs on the cham and transform then or reason about them using the algebraic presentation. Notice that real formal reasoning would actually require a precise definition of α-conversion, which will certainly involve computations on names such as those realised by using the name server.

6 A Concurrent λ-calculus

6.1 Generalising 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 [6, 7] and cannot handle even the weakest form of concurrency. Building new calculi that combine both abilities is a goal of primary importance [9, 31]. In [9], 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 [2]), that is the reflexive and transitive closure of the relation $M \triangleright M'$ inductively given by

$$(\lambda x.M)N \triangleright M[N/x]$$
$$M \triangleright M' \;\Rightarrow\; MN \triangleright M'N$$

β-Reduction as Communication

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 [31] 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.

Restricting Communication — A First Attempt

Another solution was presented in [9] using two concurrency operators: an interleaving operator '$\mid$' and a binary communication operator '$\odot$'. Communications arise as follows: in a term $(M \odot N)$, all '$\mid$' 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 $\mathtt{1}$. Then the $\odot$ operator disappears by application of the simplification rule $(M \odot \mathtt{1}) = (\mathtt{1} \odot M) = M$, 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):

$$((\lambda x.\lambda y.M \odot \overline{\lambda}(N)) \odot \overline{\lambda}(P))$$

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

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.

6.2 The γ-calculus

The key idea of our new higher-order concurrent calculus is to *internalise* 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 *localisation* 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 [9].

To formalise the execution mechanism, we need a syntactic notion of *stable* state, generalising that of weak head normal form. 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:

$$\begin{aligned} E &::= M^+ \mid (E \,|\, E) \\ R &::= x^- M \mid (R \,|\, R) \\ W &::= E \mid R \end{aligned}$$

Now we give the γ-cham describing the (lazy) evaluation of terms. The molecules are either terms written M, M', N or solutions written S, S'. The symbol W denotes a stable term.

$$\begin{array}{cl} M \,|\, N \rightleftharpoons M,\ N & (\textit{solution}) \\ \langle M \rangle \rightleftharpoons \{\!|\, M \,|\!\} & (\textit{membrane}) \\ \langle W \rangle \rightleftharpoons W & (\textit{hatching}) \\ x^- M,\ N^+ \rightarrow M[N/x] & (\beta\textit{-reaction}) \end{array}$$

Note that the reaction rule that embodies communication is the only irreversible rule. 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 [9] (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 $\{\!|\, M \,|\!\} \xrightarrow{*} \{\!|\, N \,|\!\}$.

Embedding the λ-Calculus

The γ-calculus contains the λ-calculus, since we can 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:

$$M \ ::= \ x \ \mid \ x^{-}M \ \mid \ \langle M \mid M^{+} \rangle$$

The translation is as follows:

$$\begin{aligned}
\theta(x) &= x \\
\theta(\lambda x.M) &= x^{-}\theta(M) \\
\theta(MN) &= \langle \theta(M) \mid \theta(N)^{+} \rangle
\end{aligned}$$

Then we can 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

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

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:

$$\begin{array}{cll}
 & \langle \langle x^{-}y^{-}M \mid N^{+} \rangle \mid P^{+} \rangle & \\
\overset{*}{\rightharpoonup} & \{\!|\, \{\!|\, x^{-}y^{-}M,\ N^{+} \,|\!\},\ P^{+} \,|\!\} & \text{(membrane, solution)} \\
\rightarrow & \{\!|\, \{\!|\, y^{-}M[N/x] \,|\!\},\ P^{+} \,|\!\} & \text{(reaction)} \\
\rightharpoondown & \{\!|\, \langle y^{-}M[N/x] \rangle,\ P^{+} \,|\!\} & \text{(membrane)} \\
\rightharpoonup & \{\!|\, y^{-}M[N/x],\ P^{+} \,|\!\} & \text{(hatching)} \\
\rightarrow & \{\!|\, M[N/x][P/y] \,|\!\} & \text{(reaction)}
\end{array}$$

Encoding the Full λ-calculus

As a matter of fact, we can also easily encode the full λ-calculus: we just have to extend the evaluation mechanism to deal with the ξ and μ rules. These rules allow to evaluate the body of an abstraction, i.e. M in $\lambda x.M$, and the operand of an application, i.e. N in MN. The corresponding cham for the extended γ-calculus has new molecule constructors $x^{-}U$ and U^{+} where

U is an arbitrary molecule, and two additional rules creating membranes encapsulating the subterms to evaluate:

$$x^- M \rightleftharpoons x^- \{\!|\, M \,|\!\}$$
$$M^+ \rightleftharpoons \{\!|\, M \,|\!\}^+$$

Notice that there is no interference between these new rules and β-reduction: once a membrane is opened within a term, this term cannot participate in a β-reduction, since such a reduction involves only terms and not arbitrary molecules.

Classical and Non-Deterministic Combinators

Since the λ-calculus is embedded in our γ-calculus, we 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 $(\mathrm{U}M) \xrightarrow{*} \mathrm{U}$. For example, let Y be Kleene's fixpoint combinator that satisfies $YM \triangleright M(YM)$. We can set $D = Y(\lambda f.\lambda x.(x \,|\, \langle f | x^+ \rangle))$.

Moreover, our concurrent γ-calculus is more powerful than the λ-calculus. Power is gained by making more than two molecules cooperate. 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 internal 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.

Concurrent Abstractions

As in [9], 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 values to be substituted for the x_i's in M in any order.

Obviously these generalised receptors can be incorporated in our calculus with an additional rule:

$$[x_1 \mid \cdots \mid x_n]^- M \to x_i^- [\cdots \mid x_{i-1} \mid x_{i+1} \mid \cdots]^- M \qquad (\mathit{choice})$$

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

$$[x \mid y]^- M =_{\mathrm{def}} x^- y^- M \oplus y^- x^- M$$

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

Parallel Or

We can also define a "parallel or", which is a parallel variant of the usual "left-sequential or" (cf. [9]). Let us see this point in some detail. It is known (see [6]) 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 can 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 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 [7, 6], one can show that there is no λ-definable combinator representing *parallel disjunction*, that is no combinator O such that both $\mathrm{OK}X$ and $\mathrm{O}X\mathrm{K}$ reduce to K without evaluating X and OFF reduces to F. This combinator does exist in the γ-calculus and is represented by:

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

It 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 [12].

Explicit Substitutions

The reader may have noted that we use ordinary substitution in our presentation of the γ-calculus, namely in the reaction rule. Then our set of rules does not really specify a *machine*: an abstract machine should not involve such a complex mechanism. We can remedy this deficiency using explicit substitutions, like in [1, 19]. The idea is to bind a formal substitution σ to a term M, building a new term denoted $M[\sigma]$ in [1]. Here we represent a substitution by a solution σ made out of molecules of the form $[N/x]$, and drop the substitution brackets, simply using molecules of the form:

$$M\sigma = M\{\!|\,[U_1/x_1], \ldots, [U_n/x_n]\,|\!\}$$

where the U_i's are molecules of the same shape (we omit the formal definition). The basic law concerning substitutions extracts the value of a variable from the given environment. The formulation of this law uses the airlock mechanism:

$$x\{\!|\,[U/x]\triangleleft\sigma\,|\!\} \rightarrow U \qquad (\mathit{fetch})$$

We can also add a "garbage collection" law:

$$x\{\!|\,[U/y]\triangleleft\sigma\,|\!\} \rightharpoonup x\sigma \qquad \text{if } y \neq x \qquad (gc)$$

The previous four laws of the γ-cham are modified as follows:

$$\begin{array}{cl}
(M \mid N)\sigma \rightleftharpoons M\sigma,\ N\sigma & \qquad (\mathit{solution}) \\
\langle M\rangle\sigma \rightleftharpoons \{\!|\,M\sigma\,|\!\} & \qquad (\mathit{membrane}) \\
\langle W\rangle\sigma \rightleftharpoons W\sigma & \qquad (\mathit{hatching}) \\
(x^{-}M)\sigma,\ N^{+}\rho \rightarrow M\{\!|\,[N\rho/x]\triangleleft\sigma\,|\!\} & \qquad (\beta\!-\!\mathit{reaction})
\end{array}$$

One can note that the new solution rule makes a full copy of the environment. Formally this should be allowed only for "molecular" substitutions. This means that we should use a syntax for substitutions, and apply reversible transformation rules allowing us to transform $\{\!|\,[U_1/x_1], \ldots, [U_n/x_n]\,|\!\}$ into $\langle [U_1/x_1] \mid \cdots \mid [U_n/x_n]\rangle$. The details are omitted.

To evaluate a γ-term M we now start with a solution consisting of a single molecule $N\{\!|\!|\!\}$, where N is obtained from M by α-conversion, distinguishing the nested abstractions. For instance $x^{-}x^{-}x$ has to be converted into $y^{-}x^{-}x$.

6.3 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:

$$\mathrm{OK}\Omega \simeq \mathrm{K} \simeq \mathrm{O}\Omega\mathrm{K}$$
$$\mathrm{OFF} \simeq \mathrm{F}$$
$$\mathrm{O}\Omega\Omega \simeq \Omega$$

In [9] it was proposed to adapt the notion of observational bisimulation $\approx$ of CCS [26, 31] 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 $\mathrm{OK}\Omega \not\approx \mathrm{K}$ since $\mathrm{OK}\Omega$ can 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 [14] and [20]). For instance Darondeau in [13] 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. [15]): 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 [13] and [15] for what concerns the result of experiments. To report the success of a test we shall use, as in [2], the simplest operational information, namely *convergence*, that is existence of a normal form: the agent M passes the test $C[]$ if $C[M]$ converges.

Convergence Testing

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. [6], exercise 16.5.5, and [2]), 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 in the lazy λ-calculus). We still use MN to abbreviate application, that is $\langle M \mid N^+\rangle$. As we saw, the typically divergent λ-term is $\Omega = \Delta\Delta$ where Δ is the duplicator $x^-(xx)$. It might be observed that $P = (\Delta \mid \Delta^+)$ 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 $Q = (\Upsilon \mid \Upsilon^+)$ is again a divergent term. Now there is a test separating P and Q, namely

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

(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 generalisation of the well-known "context lemma" (cf. [12]), showing that observers of the form

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

are enough to test the agents, that is

$$\begin{array}{l} M \sqsubseteq N \Leftrightarrow \forall k\ \forall R_1, \ldots, R_k. \\ \qquad \langle \cdots \langle M \mid R_1\rangle \cdots \mid R_k\rangle\Downarrow \;\Rightarrow\; \langle \cdots \langle N \mid R_1\rangle \cdots \mid R_k\rangle\Downarrow \end{array}$$

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

7 Conclusion

Unlike some other models, the Γ and cham models are operational in character and 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 implementation of TCCS, CCS, and mobile processes yield a simple operational semantics of these calculi, describing the execution mechanism. 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 [8] can be handled as well. Mobile processes can be very simply implemented. 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 [2].

Of course, this is still a preliminary work. Other concurrent computation applications should be modelled; we think in particular of process handling in operating systems. The theory of machine execution and observation should also be fully developed. The respective powers of devices such as membranes and name servers should be investigated.

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] Martin Abadi, Luca Cardelli, Pierre-Louis Curien, and Jean-Jacques Lévy. Explicit substitutions. In *Proc. 17th ACM Annual Symposium on Principles of Programming Languages*, pages 31–46, 1990.

[2] Samson Abramsky. The lazy λ-calculus. In D. Turner, editor, *Research Topics in Functional Programming*, pages 65–116. Addison Wesley, 1989.

[3] André Arnold. Sémantique des processus communicants. *RAIRO Informatique Théorique*, 15(2):103–139, 1981.

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

[5] Jean-Pierre Banâtre and Daniel Le Métayer. The gamma model and its discipline of programming. *Science of Computer Programming*, 15:55–77, 1990.

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

[7] 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.

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

[9] 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.

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

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

[12] 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.

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

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

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

[16] 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.

[17] Robert De Simone. Higher-level synchronising devices in meije-sccs. *Theoretical Computer Science*, 37:245–267, 1985.

[18] Pierpaolo Degano, Rocco De Nicola, and Ugo Montanari. A distributed operational semantics for CCS based on condition/events systems. *Acta Informatica*, 26:59–91, 1988.

[19] T. Hardin and J.-J. Lévy. A confluent calculus of substitutions. In *France-Japan Artificial Intelligence and Computer Science Symposium,*. Izu, 1989.

[20] 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.

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

[22] Gilles Kahn. The semantics of a simple language for parallel processing. In *IFIP Congress 1974*, 1974.

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

[24] José Meseguer. Rewriting as a unified model for concurrency. Technical report, SRI International Technical Report, 1990.

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

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

[27] Robin Milner. Functions as processes. In *ICALP 90, Lecture Notes in Computer Science 443*, pages 167–180. Springer-Verlag, 1990.

[28] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes. Technical Report ECS-LFCS-89-85, LFCS, Edinburgh University, 1989.

[29] Gordon Plotkin. A structural approach to operational semantics. Technical Report report DAIMI FN-19, University of Aarhus, 1981.

[30] Wolfgang Reisig. *Petri Nets: an Introduction.* EATCS Monographs on Theoretical Computer Science. Springer–Verlag, 1985.

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