Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 124 (2005) 133–147
www.elsevier.com/locate/entcs

Principles of Chemical Programming

Jean-Pierre Banâtre[a] Pascal Fradet[b] Yann Radenac[a]

[a] *IRISA, Université de Rennes1*
Campus de Beaulieu
35042 Rennes Cedex, France
{jbanatre,yradenac}@irisa.fr

[b] *INRIA Rhône-Alpes*
655, avenue de l'Europe
38330 Montbonnot, France
Pascal.Fradet@inria.fr

Abstract

The chemical reaction metaphor describes computation in terms of a chemical solution in which molecules interact freely according to reaction rules. Chemical models use the multiset as their basic data structure. Computation proceeds by rewritings of the multiset which consume elements according to reaction conditions and produce new elements according to specific transformation rules. Since the introduction of Gamma in the mid-eighties, many other chemical formalisms have been proposed such as the CHAM, the P-systems and various higher-order extensions. The main objective of this paper is to identify a basic calculus containing the very essence of the chemical paradigm and from which extensions can be derived and compared to existing chemical models.

Keywords: higher-order conditional multiset rewriting, chemical metaphor, formal calculi

1 Introduction

The chemical reaction metaphor has been discussed in various occasions in the literature. This metaphor describes computation in terms of a chemical solution in which molecules (representing data) interact freely according to reaction rules. Chemical models use the multiset as their basic data structure. Computation proceeds by rewritings of the multiset which consume elements according to reaction conditions and produce new elements according to specific transformation rules.

To the best of our knowledge, the Gamma formalism was the first "chemical model of computation" proposed as early as in 1986 [2] and later extended

doi:10.1016/j.entcs.2004.07.019

in [3]. A Gamma program is a collection of reaction rules acting on a multiset of basic elements. A reaction rule is made of a condition and an action. Execution proceeds by replacing elements satisfying the reaction condition by the elements specified by the action. The result of a Gamma program is obtained when a stable state is reached that is to say when no more reactions can take place. Figure 1 gives three short examples illustrating the Gamma style of programming. The reaction *max* computes the maximum element of

$$\begin{aligned} max &= \textbf{replace } x, y \textbf{ by } x \textbf{ if } x > y \\ primes &= \textbf{replace } x, y \textbf{ by } y \textbf{ if } multiple(x, y) \\ maj &= \textbf{replace } x, y \textbf{ by } \{\} \textbf{ if } x \neq y \end{aligned}$$

Figure 1. Examples of Gamma programs

a non empty set. The reaction replaces any couple of elements x and y such that $x > y$ by x. This process goes on till a stable state is reached, that is to say, when only the maximum element remains. The reaction *primes* computes the prime numbers lower or equal to a given number N when applied to the multiset of all numbers between 2 and N ($multiple(x, y)$ is true if and only if x is multiple of y). The majority element of a multiset is an element which occurs more than $card(M)/2$ times in the multiset. Assuming that such an element exists, the reaction *maj* yields a multiset which only contains instances of the majority element just by removing pairs of distinct elements. Let us emphasize the conciseness and elegance of these programs. Nothing had to be said about the order of evaluation of the reactions. If several disjoint pairs of elements satisfy the condition, the reactions can be performed in parallel.

Gamma makes it possible to express programs without artificial sequentiality. By artificial, we mean sequentiality only imposed by the computation model and unrelated to the logic of the program. This allows the programmer to describe programs in a very abstract way. In some sense, one can say that Gamma programs express the very idea of an algorithm without any unnecessary linguistic idiosyncrasies. The interested reader may find in [3] a long series of examples (string processing problems, graph problems, geometry problems, ...) illustrating the Gamma style of programming and in [1] a review of contributions related to the chemical reaction model.

Later, the idea was developed further into the CHAM [4], higher-order multiset rewriting [13], the hmm-calculus [8], the P-systems [14], etc. Although built on the same basic paradigm, these proposals have different properties and different expressive powers. This article is an attempt to identify the basic principles behind chemical models.

In Section 2, we exhibit a minimal chemical calculus, from which all other "chemical models" can be obtained by addition of well-chosen features. Basically, this minimal calculus, incorporates the γ-reduction which expresses the very essence of the chemical reaction and the associativity and commutativity rules which express the basic properties of chemical solutions. This calculus is then enriched in Section 3 with conditional reactions and, further, the possibility of rewriting atomically several molecules. These extensions give rise to four possible chemical calculi. Section 4 shows how existing chemical models relate and compare to the basic calculi presented previously. Section 5 suggests several research directions and concludes.

2 A minimal chemical calculus

In this section, we introduce a higher-order calculus, the γ_0-*calculus*, that can be seen as a formal and minimal basis for the chemical paradigm (in much the same way as the λ-calculus is the formal basis of the functional paradigm).

2.1 Syntax

The fundamental data structure of the γ_0-calculus is the multiset (a collection which may contain several copies of the same element). Elements can move freely inside the multiset and react together to produce new elements. Computation can be seen either intuitively, as chemical reactions of elements agitated by Brownian motion, or formally, as higher-order associative and commutative (AC) rewritings of multisets.

The syntax of γ_0-terms (also called *molecules*) is given in Figure 2. A γ-

$$
\begin{array}{rll}
M ::= & x & ; \textit{ variable} \\
| & (\gamma\langle x\rangle.M) & ; \gamma\textit{-abstraction} \\
| & (M_1, M_2) & ; \textit{ multiset} \\
| & \langle M\rangle & ; \textit{ solution}
\end{array}
$$

Figure 2. Syntax of γ_0-molecules

abstraction is a reactive molecule which consumes a molecule (its argument) and produces a new one (its body). Molecules are composed using the AC multiset constructor ",". A solution encapsulates molecules (*e.g.,* multiset) and keeps them separate. It serves to control and isolate reactions. To avoid notational clutter, we omit outermost parentheses, parentheses in multisets

and we assume that γ-abstractions associate to the right. For example, the γ-abstraction $(\gamma\langle x\rangle.(x,(x,(\gamma\langle y\rangle.y))))$ will be written $\gamma\langle x\rangle.x, x, \gamma\langle y\rangle.y$.

The γ_0-calculus bears clear similarities with the λ-calculus. They both rely on the notions of (free and bound) variable, abstraction and application. A λ-abstraction and a γ-abstraction both specify a higher-order rewrite rule. However, λ-terms are tree-like whereas the AC nature of the application operator "," makes γ_0-terms multiset-like. Associativity and commutativity formalizes Brownian motion and make the notion of solution necessary, if only to distinguish between a function and its argument.

2.2 Semantics

The conversion rules and the reduction rule of the γ_0-calculus are gathered in Figure 3. Chemical reactions are represented by a single rewrite rule,

γ-reduction:

$$(\gamma\langle x\rangle.M), \langle N\rangle \longrightarrow_\gamma M[x := N] \qquad \text{if } \mathit{Inert}(N) \vee \mathit{Hidden}(x, M)$$

α-conversion:

$$\gamma\langle x\rangle.M \equiv \gamma\langle y\rangle.M[x := y] \quad \text{with } y \text{ fresh}$$

commutativity:

$$M_1, M_2 \equiv M_2, M_1$$

associativity:

$$M_1, (M_2, M_3) \equiv (M_1, M_2), M_3$$

Figure 3. Rules of the γ_0-calculus

the γ-reduction, which applies a γ-abstraction to a solution. A molecule $(\gamma\langle x\rangle.M), \langle N\rangle$ can be reduced only if

$\mathit{Inert}(N)$: the content N of the solution argument is a closed term made exclusively of γ-abstractions or exclusively of solutions (which may be active),

or $\mathit{Hidden}(x, M)$: the variable x occurs in M only as $\langle x\rangle$. Therefore $\langle N\rangle$ can be active since no access is done to its contents.

So, a molecule can be extracted from its enclosing solution only when it has reached an inert state. This is an important restriction that permits the

ordering of rewritings. Without this restriction, the contents of a solution could be extracted in any state and the solution construct would lose its purpose.

Consider, for example, the following molecules:

$$\begin{aligned}\omega &\equiv \gamma\langle x\rangle.x, \langle x\rangle\\ \Omega &\equiv \omega, \langle\omega\rangle\\ I &\equiv \gamma\langle x\rangle.\langle x\rangle\end{aligned}$$

Clearly, Ω is an always active (non terminating) molecule and I an inert molecule (the identity function in normal form). The molecule $\langle\Omega\rangle, \langle I\rangle, \gamma\langle x\rangle.\gamma\langle y\rangle.x$ reduces as follows:

$$\langle\Omega\rangle, \langle I\rangle, \gamma\langle x\rangle.\gamma\langle y\rangle.x \longrightarrow \langle\Omega\rangle, \gamma\langle y\rangle.I \longrightarrow I$$

The first reduction is the only one possible: the γ-abstraction extracts x from its solution and $\langle I\rangle$ is the only inert molecule ($Inert(I) \wedge \neg Hidden(x, \gamma\langle y\rangle.x)$). The second reduction is possible only because the active solution $\langle\Omega\rangle$ is not extracted but removed ($\neg Inert(\Omega) \wedge Hidden(y, I)$)

A molecule is in normal form if all its molecules are inert. We say that two molecules M_1 and M_2 are syntactically equivalent (and we write $M_1 \equiv M_2$), if they can be rewritten into each other using α-conversion and AC rules.

As usual, rules can be applied in parallel as long as they apply to disjoint redexes. So, there can be several reactions at the same time for disjoint subterms and/or within nested solutions and/or inside γ-abstractions.

2.3 Expressivity

The γ_0-calculus is more expressive than the λ-calculus since it can easily express non-deterministic programs. For example, let A and B two distinct normal forms:

$$\begin{array}{ccc}(\gamma\langle x\rangle.\gamma\langle y\rangle.x), \langle A\rangle, \langle B\rangle & \equiv & (\gamma\langle x\rangle.\gamma\langle y\rangle.x), \langle B\rangle, \langle A\rangle\\ \downarrow_\gamma & & \downarrow_\gamma\\ (\gamma\langle y\rangle.A), \langle B\rangle & & (\gamma\langle y\rangle.B), \langle A\rangle\\ \downarrow_\gamma & & \downarrow_\gamma\\ A & \not\equiv & B\end{array}$$

On the other hand, the λ-calculus can easily be encoded within the γ_0-calculus. Figure 4 gives here a possible encoding for the strict λ-calculus using the function $[\![\cdot]\!]$ which takes a λ-term and returns its translation as a γ-term. The standard call-by-name λ-calculus can also be encoded but the

$$\begin{aligned} [\![x]\!] &\stackrel{\text{def}}{=} x \\ [\![\lambda x.E]\!] &\stackrel{\text{def}}{=} \gamma\langle x\rangle.[\![E]\!] \\ [\![E_1\ E_2]\!] &\stackrel{\text{def}}{=} \langle [\![E_1]\!]\rangle,\ \gamma\langle f\rangle.f, \langle [\![E_2]\!]\rangle \end{aligned}$$

Figure 4. Translating λ-terms into γ_0-molecules

translation is slightly more involved. This comes from the strict nature of the γ_0-calculus which enforces the argument to be inert/reduced before the reaction can take place.

As in the λ-calculus, recursion, integers, booleans, data structures, arithmetic, logical and comparison operators can be defined within the γ_0-calculus. We do not give their precise definitions in this article since they are similar to their definitions as λ-terms. From now on, we will give our examples assuming that these constructs have been defined. In particular, we will use pairs (written a:b) and recursive definitions to define n-shot abstractions (which re-introduce themselves after each reaction). For example, the molecule performing the product of all integers in its solution can be defined as:

$$pi = \gamma\langle x\rangle.\gamma\langle y\rangle.\langle x * y\rangle, pi$$

The reactive molecule pi takes two integers and replaces them by their product and a copy of itself. For example:

$$pi, \langle 2\rangle, \langle 3\rangle, \langle 2\rangle \longrightarrow_\gamma \ldots \longrightarrow_\gamma pi, \langle 12\rangle \longrightarrow_\gamma \gamma\langle y\rangle.\langle 12 * y\rangle, pi$$

3 Two fundamental extensions

The γ_0-calculus is a quite expressive higher-order calculus. However, compared to the original Gamma [3] and other chemical models [8,13,14], it lacks two fundamental features:

- *Reaction condition.* In Gamma, reactions are guarded by a condition that must be fulfilled in order to apply them. Compared to γ_0 where inertia and termination are described syntactically, conditional reactions give these notions a semantic nature.
- *Atomic capture.* In Gamma, any fixed number of elements can take part in a reaction. Compared to a γ_0-abstraction which reacts with one element at

a time, a n-ary reaction takes atomically n elements which cannot take part in any other reaction at the same time.

These two extensions are orthogonal and enhance greatly the expressivity of chemical calculi. Strictly speaking, these features do not permit to express a larger class of programs (the γ_0-calculus is Turing-complete). But, they do add expressivity in the sense that they are not syntactic sugar and can only be expressed using a global re-organization of programs.

3.1 Conditional reaction

In the γ_c-calculus, abstractions hold conditions. The condition of an abstraction must be satisfied before the reaction occurs. The syntax of the γ_c molecules is given in Figure 5. The reaction condition is modeled by M_0 which

$$\begin{array}{lll} M ::= x & & \textit{; variable} \\ \quad | \quad \gamma\langle x\rangle\lfloor M_0\rfloor.M_1 & & \textit{; conditional } \gamma\textit{-abstraction} \\ \quad | \quad (M_1, M_2) & & \textit{; multiset} \\ \quad | \quad \langle M\rangle & & \textit{; solution} \end{array}$$

Figure 5. Syntax of γ_c-molecules

must evaluate to a special constant **true** before the reaction occurs. The γ_c-reduction is formalized as follows:

$$\frac{\mathit{Inert}(N) \vee \mathit{Hidden}(x, (M_0, M_1)) \qquad M_0[x := N] \xrightarrow{*}_c \mathbf{true}}{(\gamma\langle x\rangle\lfloor M_0\rfloor.M_1),\ \langle N\rangle \longrightarrow_c M_1[x := N]}$$

where the molecule **true** is a given constant (*e.g.*, $\gamma\langle x\rangle\lfloor x\rfloor.x$).

Clearly, the γ_c-calculus embeds the γ_0-calculus: the abstractions of γ_0 correspond to the abstractions of γ_c with the condition **true**. Inert γ_c-molecules are molecules where no solution satisfies the reaction condition of any γ-abstraction. So, as opposed to γ_0, inert molecules in the γ_c-calculus can mix solutions and abstractions as long as no condition is satisfied. Inertia, as well as termination, becomes a semantic notion.

Consider the task of ceiling a collection of integers by 9. This can be expressed in γ_c by the following recursive molecule:

$$\mathit{ceil} = \gamma\langle x\rangle\lfloor x > 9\rfloor.\langle 9\rangle,\ \mathit{ceil}$$

and, for example,

$$ceil,\ \langle 10\rangle,\ \langle 3\rangle,\ \langle 11\rangle \xrightarrow{*}_{\gamma} ceil,\ \langle 9\rangle,\ \langle 3\rangle,\ \langle 9\rangle$$

Conditions can be used to encode type checking and pattern-matching. For example, assuming pairs $(x{:}y)$ (with the access functions *Fst* and *Snd*) and a tag (constant) *Int*, we can encode typed integers by $Int{:}x$. A γ-abstraction matching an integer as argument can be written as:

$$\gamma\langle i\rangle\lfloor Fst\ i = Int \wedge M_0\rfloor.M_1$$

It is easy to define a convenient and expressive pattern language to match integers, booleans, solutions, γ-abstractions, etc. For example, the γ-abstraction above could also be written:

$$\gamma(Int{:}x)\lfloor M_0\rfloor.M_1$$

There is no simple and local way to encode γ_c in γ_0. The encoding implies a global reorganization of γ_0 programs. A possible encoding consists in a γ_0 program interpreting γ_c programs. All γ_c elements are isolated in solutions with their description (type, value) and γ_0-abstractions simulate the semantics of γ_c (this can be done since γ_0 is Turing-complete). The algorithm must check that the reaction condition of all γ-abstractions is false before it terminates.

3.2 Atomic capture

In the γ_n-calculus, abstractions can capture several elements atomically. The syntax of the molecules is given in Figure 6. The n-ary abstraction can occur

$$\begin{array}{lll} M ::= x & & ;\ variable \\ \quad | \quad \gamma(\langle x_1\rangle,\ldots,\langle x_n\rangle).M & & ;\ n\text{-}ary\ \gamma\text{-}abstraction \\ \quad | \quad (M_1, M_2) & & ;\ multiset \\ \quad | \quad \langle M\rangle & & ;\ solution \end{array}$$

Figure 6. Syntax of γ_n-molecules

if it finds n solutions, otherwise no reaction takes place. Of course, an element cannot participate in several reactions simultaneously (mutual exclusion). The

γ_c-reduction is formalized as follows:

$$\frac{\forall\ 1 \leq i \leq n \quad Inert(N_i) \vee Hidden(x_i, M)}{(\gamma(\langle x_1\rangle, \ldots, \langle x_n\rangle).M),\ \langle N_1\rangle, \ldots, \langle N_n\rangle \longrightarrow_n M[x_i := N_i]}$$

For example, the addition and product of a collection of integers can be defined as binary recursive γ-abstractions:

$$sigma = \gamma(\langle x\rangle, \langle y\rangle).\langle x + y\rangle,\ sigma$$
$$pi = \gamma(\langle x\rangle, \langle y\rangle).\langle x * y\rangle,\ pi$$

The following example describes one possible execution where one addition and one multiplication have been performed (many other executions are possible):

$$sigma,\ pi,\ \langle 2\rangle,\ \langle 3\rangle,\ \langle 4\rangle \xrightarrow{*}_n \langle 20\rangle,\ sigma,\ pi$$

Consider the previous example but with *sigma* and *pi* defined as unary γ_0-abstractions. When there remains only two elements, *sigma* and *pi* could each take one element and would keep waiting for a second. These conflicts (which can also be seen as deadlocks) can only be avoided with the ability of taking several elements atomically.

This feature is not syntactic sugar. As with γ_c, a possible way to encode the atomic capture, is to isolate all γ_n elements in solutions and to emulate using γ_0 reactions the semantics of γ_n. Abstractions of γ_n are encoded with their arity and the emulator should test the presence of enough elements before triggering the reaction.

4 Chemical calculi and related chemical models

The previous extensions are orthogonal and can be combined. For example, the γ_0-calculus can be extended using reaction conditions and atomicity capture. We denote the resulting calculus γ_{cn}. The γ-calculi are depicted in Figure 7 where arrows stand for "can be extended into" or "is less expressive than". In the following, we relate well-known chemical models to γ_0 and γ_{cn}.

4.1 The γ_0-calculus and related models

Our minimal chemical calculus is quite close to Berry and Boudol's concurrent λ-calculus (referred to here as the γ_{lb}-calculus) introduced after the chemical

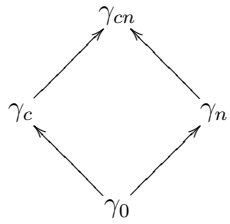

Figure 7. γ-calculi.

abstract machine (CHAM) in [4]. The γ_{lb}-calculus relies also on variables, abstractions, an AC application operator and solutions. However, to distinguish between the γ-abstraction and its argument, it adds the notion of positive ions (denoted M^+). The γ-abstractions are negative ions (denoted x^-M) which can react only with positive ions:

$$\beta\text{-reaction:}\ \ (x^-M), N^+ \to M[x := N]$$

In fact, no reaction can occur within a positive ion and so arguments are passed unchanged to abstractions. Furthermore, an additional reduction law, the *hatching rule*, extracts an inert molecule M from a solution $\langle M \rangle$:

$$\text{hatching:}\ \ \langle W \rangle \rightleftharpoons W \qquad \text{if } W \text{ is inert}$$

In the γ_0-calculus, these two notions are replaced by the strict γ-reduction. In particular, hatching can be written explicitly as

$$(\gamma\langle x \rangle.x), \langle M \rangle$$

which extracts M from its solution when it becomes inert. Even if the γ_0-calculus looks simpler than the γ_{lb}-calculus, it seems that they cannot be translated easily into each other (*e.g.,* by a translation defined on the syntax rules). They appear to be call-by-value (γ_0) and call-by-name (γ_{lb}) versions of similar ideas.

4.2 The γ_{cn}-calculus and related models

In the γ_{cn}-calculus, abstractions have a reaction condition and the ability to take several molecules atomically. Their syntax becomes:

$$\gamma(\langle x_1 \rangle, \ldots, \langle x_n \rangle)\lfloor M_0 \rfloor.M_1$$

The associated reduction rule mixes γ_c-reduction and γ_n-reduction :

$$\frac{\forall\ 1 \leq i \leq n \quad \mathit{Inert}(N_i) \vee \mathit{Hidden}(x_i, (C, M)) \quad C[x_i := N_i] \xrightarrow{*}_{cn} \mathbf{true}}{(\gamma(\langle x_1 \rangle, \ldots, \langle x_n \rangle)\lfloor C \rfloor.M), \langle N_1 \rangle, \ldots, \langle N_n \rangle \longrightarrow_{cn} M[x_i := N_i]}$$

The γ_{cn} model cumulates the expressive power of γ_c and γ_n. For example, the dining philosophers problem can be expressed in γ_{cn} as follows:

$$\mathit{Eat} = \gamma(\langle \mathit{Fork}{:}f_1 \rangle, \langle \mathit{Fork}{:}f_2 \rangle)\lfloor f_2 = f_1 + 1 \ \mathit{mod}\ N \rfloor.\langle \mathit{Phi}{:}f_1 \rangle, \mathit{Eat}$$

$$\mathit{Think} = \gamma(\langle \mathit{Phi}{:}f \rangle)\lfloor \mathbf{true} \rfloor.\langle \mathit{Fork}{:}f \rangle, \langle \mathit{Fork}{:}(f + 1 \ \mathit{mod}\ N) \rangle, \mathit{Think}$$

Initially the multiset contains only forks and the two recursive molecules. The *Eat* reaction looks for two adjacent forks $\langle \mathit{Fork}{:}f_i \rangle$ and "produces" an eating philosopher $\langle \mathit{Phi}{:}f \rangle$. This reaction needs the expressive powers of γ_n and γ_c: the two forks have to be adjacent (reaction condition of γ_c) and should be taken simultaneously (atomicity of γ_n) to prevent deadlocks. The *Think* reaction "transforms" an eating philosopher into two available forks.

Most of the existing chemical models have reaction conditions and the ability to take several molecules atomically. They are closely related to γ_{cn} even when they are first-order languages. We present here two first-order models (Gamma and the Cham) and higher-order extensions (higher-order multiset rewriting and the hmm-calculus).

4.2.1 Gamma

To the best of our knowledge, Gamma [2,3] is the first chemical model. It consists in a single multiset containing basic inactive molecules and external, conditional and n-ary reactions. Reactions are n-shot: they are applied until no reaction can take place. They are first-order: they are not part of the multiset and cannot be taken as argument or returned as result. Moreover, there is no nested solutions. Even if sub-solutions can be encoded, there is no notion of inertia in Gamma (only global termination). A standard Gamma program is easily expressed as a γ_{cn}-molecule made of a solution (inert because without any abstraction) representing the multiset and a collection of recursive γ-abstractions representing the reactions. Gamma has inspired many extensions (*e.g.,* composition operators [12]) and other chemical models. Most of these extensions and models remain related to γ_{cn}.

4.2.2 The Chemical Abstract Machines

The chemical abstract machine [4] (CHAM) is a chemical approach introduced to describe concurrent computations without explicit control. It started from Gamma and added many features such as membranes, (sub)solutions, inertia and airlocks. Like Gamma, reactions are n-ary and n-shot rewrite rules which are not part of the multisets. The selection pattern in the left-hand side of rewrite rules can include constants which is a form of reaction condition. For example, in [4], the description of the operational semantics of the TCCS and CCS calculi contains a cleanup rule ($0 \rightharpoonup$) which removes molecules equal to 0. The CHAM would be equivalent to γ_{cn} if it was higher-order.

4.2.3 Higher-order extensions

A first higher-order extension of Gamma has been proposed in [13]. The definition of Gamma involves two different kinds of terms: the program (set of rewrite rules) and multisets. The main extension of higher-order Gamma consists in unifying these two categories of expression into a single notion of configuration. A configuration contains a program and a list of named multisets. It is denoted by $[Prog,\ Var_1 = Multiset_1, \ldots, Var_n = Multiset_n]$. The program $Prog$ is a rewrite rule of the multisets (named Var_i) of the configuration. This model is an higher-order model because any configuration can handle other configurations through their program. It includes reaction conditions and n-ary rewrite rules. However, reactions are not first-class citizens since they are kept separate from multisets of data.

The hmm-calculus [8] (for *higher-order multiset machines*) is described as an extension of Gamma where reactions are one-shot and first-class citizens. An abstraction denoted by $\lambda\tilde{x}.M_1 \Leftarrow M_0$ describes a reaction rule: it takes several terms denoted by a tuple $\tilde{x}$, the term M_1 is the action and the term M_0 is the reaction condition. Like γ_{lb}, the hmm-calculus uses a call-by-name strategy. It needs an hatching rule to extract an inert molecule from its solution. Any reaction can occur within solutions and within abstractions. The hmm-calculus can be seen as a lazy version of the γ_{cn}-calculus, or as an extension of the γ_{lb}-calculus with conditional and n-ary reactions.

4.2.4 P-systems

P-systems [14] are computing devices inspired from biology. It consists in nested membranes in which molecules react. Molecules can cross and move between membranes. A set of partially ordered rewrite rules is associated to each membrane. These rules describe possible reactions and communications between membranes of molecules. These features can be expressed in γ_{cn} by

introducing two new notions. They do not add additional expressive power but they are convenient and interesting in themselves.

- The first needed notion is *universally quantified conditions*. Intuitively, a reaction condition C can be read "if it exists a solution that satisfies C ...". Another kind of condition could also be considered: "if all solutions satisfy C ...". This universally quantified condition can be expressed in γ_c. It amounts to testing the absence of a molecule satisfying $\neg C$. Using this mechanism, it is possible to specify a partial order between reactions as priorities. A high priority reaction should react before one with a lower priority. To encode priority, an abstraction should check that no abstraction with a higher priority can react, *i.e.,* that there is *no* elements in the solution that satisfy the conditions of the abstractions with a higher priority.
- The second notion is *porous solutions*. It is possible to define porous solutions which can be manipulated by γ-abstractions even when they are active. A porous solution made of the active molecule $X_1, \ldots, X_n$ can be encoded by $\langle \gamma \langle x \rangle . X_1, \ldots, X_n \rangle$. The body of the γ-abstraction is active but can be accessed by extracting it from its inert enclosing solution and by applying it to an argument. This feature can be used to represents the porous membranes of P-systems. This capability is also useful for example when modeling non-terminating reactive systems which interacts continuously with their environment. The reactive system is therefore represented by an always active porous solution and the environment by reactions adding (sending) and removing (receiving) elements in that solution.

4.2.5 Other models

Our list of comparisons is not exhaustive and other models could have been considered. For example, Linda and its variants (particularly Bauhaus Linda [7]) are close to Gamma. Other work has been carried out about concurrent λ-calculus according to a chemical metaphor such as [11], or, for example, models from [9].

5 Conclusion

In this article, we have studied the fundamental features of the chemical programming paradigm. The γ_0-calculus embodies the essential characteristics (AC multiset rewritings) in only four syntax rules. Terms are multisets (built with the AC application operator ",") which can be nested (inside solutions). This minimal calculus has been shown to be expressive enough to express the λ-calculus and a large class of non-deterministic programs. However, it does

not reflect closely existing chemical languages such as Gamma. Two extensions must be considered to achieve a comparable expressive power: reaction conditions and atomic capture. With appropriate syntactic sugar (recursion, constants, operators, pattern-matching, porous solutions, etc.), the γ_{cn}-calculus closely models most of the existing chemical programming models.

This work suggests several research directions. First, we should prove formally that our extensions really improve the expressive power of our minimal chemical calculus. The comparison of the expressive power of languages has been formally studied by Felleisen in [10]. He formalizes the intuitive notions of "syntactic sugar" and "expressive power". A new construct is considered as enhancing expressivity if its expression using the other constructs needs "a global reorganization of the entire program". A formal comparison of expressive power of different coordination languages has been carried out in [5]. This work compares different variants of Linda [6] with different models *à la* Gamma and with models featuring communication transactions. A similar approach could be taken to establish formally the pre-order of Figure 7. Our work could also be completed by providing formal translations of existing chemical models into the corresponding γ-calculus.

Another direction is to propose a realistic higher-order chemical programming language based on the γ_{cn}-calculus. It would consist in defining the already mentioned syntactic sugar, a type system, as well as expressive pattern and module languages.

References

[1] Banâtre, J.-P., P. Fradet and D. Le Métayer, *Gamma and the chemical reaction model: Fifteen years after*, in: *Multiset Processing*, LNCS **2235** (2001), pp. 17–44.

[2] Banâtre, J.-P. and D. Le Métayer, *A new computational model and its discipline of programming*, Technical Report RR0566, INRIA (1986).

[3] Banâtre, J.-P. and D. Le Métayer, *Programming by multiset transformation*, Communications of the ACM (CACM) **36** (1993), pp. 98–111.

[4] Berry, G. and G. Boudol, *The chemical abstract machine*, Theoretical Computer Science **96** (1992), pp. 217–248.

[5] Brogi, A. and J.-M. Jacquet, *On the expressiveness of coordination models*, in: P. Ciancarini and A. Wolf, editors, *Proc. 3rd Int. Conf. on Coordination Models and Languages*, LNCS **1594**, 1999, pp. 134–149.

[6] Carriero, N. and D. Gelernter, *Linda in Context*, Communications of the ACM **32** (1989), pp. 444–458.

[7] Carriero, N., D. Gelernter and L. Zuck, *Bauhaus Linda*, in: *Object-Based Models and Languages for Concurrent Systems*, LNCS **924** (1994), pp. 66–76.

[8] Cohen, D. and J. Muylaert-Filho, *Introducing a calculus for higher-order multiset programming*, in: *Coordination Languages and Models*, LNCS **1061**, 1996, pp. 124–141.

[9] Dittrich, P., J. Ziegler and W. Banzhaf, *Artificial chemistries – a review*, Artificial Life **7** (2001), pp. 225–275.

[10] Felleisen, M., *On the expressive power of programming languages*, in: *3rd European Symposium on Programming, ESOP'90*, LNCS **432**, Springer-Verlag, New York, N.Y., 1990 pp. 134–151.

[11] Fontana, W. and L. Buss, *The arrival of the fittest: Toward a theory of biological organization*, Bulletin of Mathematical Biology **56** (1994).

[12] Hankin, C., D. L. Métayer and D. Sands, *A calculus of Gamma programs*, in: *Languages and Compilers for Parallel Computing, 5th International Workshop*, LNCS **757** (1992), pp. 342–355.

[13] Le Métayer, D., *Higher-order multiset programming*, in: A. M. S. (AMS), editor, *Proc. of the DIMACS workshop on specifications of parallel algorithms*, Dimacs Series in Discrete Mathematics **18**, 1994.

[14] Păun, G., *Computing with membranes*, Journal of Computer and System Sciences **61** (2000), pp. 108–143.