

Higher-Order Chemistry*

Jean-Pierre Banâtre, Pascal Fradet and Yann Radenac

IRISA
Campus Universitaire de Beaulieu
Avenue du Général Leclerc
35042 RENNES Cedex - France

email: {Jean-Pierre.Banatre,Pascal.Fradet,Yann.Radenac}@irisa.fr

Abstract

Gamma is a formalism in which programs are expressed in terms of multiset rewriting often referred as the Chemical Reaction Model. In this paper we are concerned with higher-order Gamma programming. First we review three proposals which introduce the notion of membrane and higher order facilities. Finally, we propose a higher-order Gamma which allows the definition of γ -abstractions (in the same sense as λ -abstractions in the λ -calculus) which are considered as first class citizens.

1 Introduction

Gamma was originally proposed as a formalism for the construction of programs without artificial sequentiality [2]. Computation is described as a kind of chemical reaction operating on a collection of data. A Gamma program is a pair (Reaction condition, Action). Execution proceeds by replacing in the multiset elements satisfying the reaction condition by the product of the action. The result is considered as obtained when a stable state is reached, meaning that no more reactions can occur. Let us take a traditional example which illustrates the Gamma style of programming:

$$\max : x, y \rightarrow y \Leftarrow x \leq y$$

$x \leq y$ specifies the reaction condition which must be fulfilled by the selected elements x and y . This program describes a computation where two arbitrary integers x and y are selected and rewritten into the greatest one. The computation terminates when only one element remains in the multiset. Nothing is said about the order of evaluation of the comparisons. If

*Prepared for the MolCoNet Workshop on Membrane Computing (WMC) – Tarragona – July 2003

several disjoint pairs of elements fulfill the condition, the reaction can be performed simultaneously, thus providing a lot of potential parallelism.

Much work has been devoted to the exploiting of Gamma in various directions ; overviews can be found in [3, 1].

In a first step, we review some contributions concerned with hierarchies of communicating multisets and higher-order multiset processing: (1) the Chemical abstract machine (or Cham) which was proposed by Berry and Boudol [4, 5] to describe operational semantics of process calculi extends the original Gamma model with the notion of membrane for the description of nested interacting multisets, and (2) two extensions of Gamma to higher-order multiset programming models. The first one, proposed by Le Métayer [7], introduces the notion of configuration which is a program operating on a record of named multisets. The second proposal [6] provides a unified view of data and computation within a multiset. In a second step, this paper describes ongoing work on higher-order Gamma. It investigates the way of incorporating “gamma programs” as potentially reacting data in a multiset. This necessitates the precise definition of a program as a first class citizen and leads to the exploration of a new Gamma which captures the notion of γ -abstraction and chemical reaction.

2 The chemical abstract machine

The chemical abstract machine (or *Cham*) was proposed by Berry and Boudol [4] to describe the operational semantics of process calculi. The most important additions to Gamma are the notions of *membrane* and *airlock mechanism*. Membranes are used to encapsulate solutions and to force reactions to occur locally. In terms of multisets, a membrane can be used to introduce multiset of molecules inside a multiset that is to say “*to transform a solution into a single molecule*” [5]. The airlock mechanism is used to describe communications between an encapsulated solution and its environment. The reversible airlock operator \triangleleft extracts an element m of a solution $\{m, m_1, \dots, m_n\}$:

$$\{m, m_1, \dots, m_n\} \rightleftharpoons \{m \triangleleft \{m_1, \dots, m_n\}\}$$

The new molecule can react as a whole while the sub-solution $\{m_1, \dots, m_n\}$ is allowed to continue its internal reactions. So the main rôle of the airlock is to allow one molecule to be visible from outside the membrane and thus to take part in a reaction in the embedding solution. The need for membranes and airlocks emerged from the description of CCS [8] in Cham and especially the treatment of the restriction operation (which limits the communication capabilities of a process to labels different from a particular value). The computation rules of the Cham are classified into general laws and two classes of rules:

- The general laws include the chemical law and the membrane law:

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

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

The former shows that reactions can be performed freely within any solution, which captures the locality principle. The latter allows reactions to take place within a membrane ($C[S]$ denotes any context of a solution S).

- The first class of rules corresponds to the proper reaction rules. The definition of a specific Cham requires the specification of a syntax for molecules and the associated reaction rules. As an example, molecules can be CCS processes and the rule corresponding to communication in CCS would be:

$$\alpha.P, \bar{\alpha}.Q \mapsto P, Q$$

- The second kind of rules are called structural and they are reversible. They can be decomposed into two inverse relations \rightarrow and \leftarrow called respectively heating and cooling rules. The first ones break complex molecules into smaller ones, preparing them for future reactions, and the second ones rebuild heavy molecules from light ones. Continuing the CCS example, we have the structural rule:

$$(P \mid Q) \rightleftharpoons P, Q$$

where \mid is the CCS parallel composition operator.

The Cham was used in [4] to define the semantics of various process calculi (TCCS, Milner's π -calculus of mobile processes) and a concurrent λ -calculus. A Cham for the call-by-need reduction strategy of λ -calculus is defined in [5].

3 Higher-Order Multiset Programming

An approach for the introduction of composition operators in a language consists in providing a way for the programmer to define them as higher-order programs. This is the traditional view in the functional programming area and it requires to be able to manipulate programs as ordinary data. This is the approach followed in [7] which proposes a higher-order version of Gamma. The definition of Gamma used so far involves two different kinds of terms: the programs and the multisets. The multiset is the only data structure and programs are described as collections of pairs (*Reaction Condition, Action*). The main extension of higher-order Gamma consists in unifying these two categories of expressions into a single notion of configuration. One important consequence of this approach is that active configurations may now occur inside multisets and reactions can take place (simultaneously) at different levels. Thus two conditions must be satisfied for a simple program to terminate: no tuple of elements satisfies the reaction condition and the multiset does not contain active elements.

A configuration is denoted:

$$[Prog, Var_1 = Multexp_1, \dots, Var_n = Multexp_n]$$

It consists of a (possibly empty) program *Prog* and a record of named multisets *Var_i*. A configuration with an empty program component is called

passive, otherwise it is active. The record component of the configuration can be seen as the environment of the program. Each component of the environment is a typed multiset. Simple programs extract elements from these multisets and produce new elements. A stable component $Multexp_i$ of a configuration C can be obtained as the result of $C.Var_i$.

The operational semantics is essentially extended with the following rules to capture the higher-order features:

$$\frac{X \rightarrow X'}{\{X\} \oplus M \rightarrow \{X'\} \oplus M}$$

$$\frac{M_k \rightarrow M'_k}{[P, \dots Var_k = M_k, \dots] \rightarrow [P, \dots Var_k = M'_k, \dots]}$$

The first and the second rule respectively account for the computation of active configurations inside multisets and for the transformation of multisets containing active configurations inside a configuration. Note that these rules are very similar to the chemical law and the membrane law of the Cham (section 2).

Let us take one example to illustrate the expressive power provided by this extension. The application of the sequential composition operator to simple programs can be defined in higher-order Gamma, and thus does not need to be included as a primitive. $(P_2 \circ P_1)(M_0)$ is defined by the following configuration:

$$[Q, E_1 = \{[P_1, M = M_0]\}, E_2 = \emptyset].E_2$$

where $Q = [\emptyset, M = M_1] : E_1 \rightarrow [P_2, M = M_1] : E_2$

E_1 is a multiset containing the active configuration $[P_1, M = M_0]$ initially. Note that Q reactions only apply to passive values of E_1 which means that M_1 must be a stable state for P_1 . Then the new active configuration $[P_2, M = M_1]$ is inserted into E_2 and the computation of P_2 can start. When a stable state is obtained, it is extracted from the top-level configuration through the access operation denoted by $.E_2$.

[7] shows how other useful combining forms can be defined in higher-order Gamma (including the chemical abstract machine). It is also possible to express more sophisticated control strategies such as the *scan vector model* suitable for execution on fine-grained parallel machines.

However, configurations still make a strong difference between data and computation. In reaction to [7], another proposal [6] does not distinguish data and programs within a multiset. Along these lines, Cohen and Muylaert-Filho define a higher-order calculus, called *hmm-calculus*, which considers the one-shot chemical reaction as its basic computational component. They investigate a translation of the λ -calculus into their model. In order to reach the expressing power of the original Gamma model, they introduce a \mathcal{Y} combinator which allows the “recursive” application of a reaction. This model is used to express Le Métayer’s higher-order calculus and different Gamma composition operators.

The interesting contribution of this work is the recognition of the one-shot chemical reaction as the basic constituent of the model.

4 Towards a Higher-Order Gamma

Our present work aims at providing a higher-order Gamma where the execution of a program can be seen as the evolution of a solution of molecules, which can be simple data or programs (γ -abstractions), which react until the solution becomes inert.

4.1 Syntax

Molecules (or γ -expressions) are constants, γ -abstractions, multisets or solutions of molecules. Their syntax is defined as follows:

$$\begin{array}{lcl}
 M ::= & 0 \mid 1 \mid \dots \mid 'a' \mid 'b' \mid \dots & ; \text{ constants} \\
 & \mid \gamma P[C].M & ; \gamma\text{-abstraction} \\
 & \mid M_1, M_2 & ; \text{ multiset} \\
 & \mid \langle M \rangle & ; \text{ solution}
 \end{array}$$

The multiset constructor “,” is associative and commutative. For example, the molecule $(1, (2, 1))$ represents a multiset of three integers and is equivalent by Brownian motion (i.e. Associative and Commutative rules) to $(2, (1, 1))$.

Solutions encapsulate molecules. Molecules can move (Brownian motion) within solutions but not across solutions. For example, $(1, \langle 2, 1 \rangle)$ is equivalent to $(1, \langle 1, 2 \rangle)$ but not to $(2, \langle 1, 1 \rangle)$ nor to $(1, (2, 1))$.

Just like integers and other constants, γ -abstractions are elements of multisets. They are specified using a pattern P , a condition C and a product M . For example, a γ -abstraction selecting the greatest of two integers is written

$$\gamma(x : \mathbf{Int}, y : \mathbf{Int})[x \geq y].x$$

4.2 Semantics

Equivalence of γ -expressions

Equivalence on γ -terms is defined using three axioms and one inference rule.

$$\begin{array}{lcl}
 (\gamma p[c].m_1), m_2 & = & \phi m_1 \quad \text{if } \text{match}(p/m_2) = \phi \wedge \phi c \quad ; \gamma\text{-conversion} \\
 m_1, m_2 & = & m_2, m_1 \quad ; \text{ commutativity} \\
 m_1, (m_2, m_3) & = & (m_1, m_2), m_3 \quad ; \text{ associativity} \\
 E_1 = E_2 & \Rightarrow & E[E_1] = E[E_2] \quad ; \text{ chemical law}
 \end{array}$$

The γ -conversion describes the reaction mechanism. A reaction takes place if the pattern p matches a molecule m_2 and yields a substitution ϕ such that the condition ϕc holds. The reaction consumes the reactive molecules $\gamma p[c].m_1$ and m_2 to produce a new molecule ϕm_1 .

The pattern has the following syntax:

$$p ::= x \mid p, p \mid \langle p \rangle$$

where x is any variable. If the matching succeeds, $\text{match}(p/m)$ returns the substitution corresponding to the unification of variables, otherwise it

returns **fail**. For example, $match(x, y/m_1, m_2) = \phi$ and then $\phi(x) = m_1$ and $\phi(y) = m_2$, but $match(x, y/a) = \mathbf{fail}$ and $match(\langle x : \mathbf{Int} \rangle / \langle 5, 2 \rangle) = \mathbf{fail}$.

The structural laws are summarized in the so-called chemical law which formalizes locality of reactions. It is defined using the notion of context $E[]$ which is a molecule with “holes” (denoted by $[]$) in it.

$$E[] ::= [] \mid constant \mid \gamma P[C].M \mid E_1[], E_2[] \mid \langle E[] \rangle$$

$E[E_1]$ denotes the molecule obtained by replacing holes in the context $E[]$ by the molecule E_1 .

A molecule is said to be *inert* if no reaction can be made within. Formally:

$$Inert(m) \Leftrightarrow (m \equiv m'[(\gamma p[c].m_1), m_2] \Rightarrow match(p/m_2) = \mathbf{fail})$$

Elements inside a solution can be matched only if the solution is inert. So a pattern $\langle p \rangle$ cannot match an active solution. Formally:

$$\forall p, m \neg Inert(m) \Rightarrow match(\langle p \rangle / m) = \mathbf{fail}$$

The fact that solutions cannot be decomposed before they reach their normal form permits to control (sequentialize) reactions.

Chemical reactions by reductions

Reactions are represented by γ -reductions:

$$(\gamma p[c].m_1), m_2 \rightarrow \phi m_1 \quad \text{if } match(p/m_2) = \phi \wedge \phi c \quad ; \gamma\text{-reduction}$$

Final results (or normal forms) of chemical reactions are inert γ -expressions. The structural rules are summarized into the following inference rule

$$\frac{E_1 \rightarrow E_2 \quad E \equiv C[E_1] \quad E' \equiv C[E_2]}{E \rightarrow E'}$$

For example, consider the γ -abstraction

$$inc = \gamma(f : (\bullet, \bullet) \rightarrow *, \langle x \rangle).f, f, \langle x + 1 \rangle$$

where the notation $(\bullet, \bullet) \rightarrow *$ represents the type of a γ -abstraction taking two basic elements (a basic element can be everything except a multiset) as argument and produces a molecule of any type. The program:

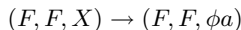
$$(\gamma(f, f, \langle n \rangle).n), inc, inc, \langle 0 \rangle$$

can be shown equivalent to any integer. The second γ -abstraction (*inc*) can produce any integer n (in n reaction steps). The first γ -abstraction can extract the integer and remove the other γ -abstractions at any step.

Repeated application of a γ -abstraction $\gamma p[c].a$ can be encoded as follows. The abstraction is written as

$$F = \gamma(f : * \rightarrow *, p)[c].f, f, a$$

Then assuming (at least) two instances of F (whose type is different from the other elements of the multiset) F will duplicate itself when it reacts. Let X be a molecule such that $match(p/X) = \phi$ then:



A classic way of extending a kernel language is to apply the so-called abstraction principle: *Every syntactic expression can be named*. This principle, that amounts to introducing declarations in the language, can be used to define recursive molecules. The syntax of molecules is extended with a **let** construct.

$$\begin{array}{l} M ::= \dots \\ | \quad \mathbf{let} \ m = M_1 \ \mathbf{in} \ M_2 \quad ; \textit{definition} \\ | \quad m \quad \textit{with} \ m \in \textit{Identifiers} \quad ; \textit{invocation} \end{array}$$

The recursive γ -abstraction *inc* above can now be written

$$\mathbf{let} \ inc = \gamma\langle x \rangle.inc, \langle x + 1 \rangle \ \mathbf{in} \ (inc, \langle 1 \rangle)$$

Declarations are just convenient syntactic sugar. It is easy to translate them into pure γ -expressions.

Yet other ways of computing the Maximum

There are several ways to compute the maximum of a multiset of integers:

- Each integer i can be represented by two elements : the integer and a function selecting the maximum out of two integers:

$$i \quad \text{and} \quad \max = \gamma(x : \mathbf{Int}, y : \mathbf{Int})[x \geq y].x$$

Let M a multiset of such pairs, the maximum is:

$$parsup = (\gamma\langle f, x : \mathbf{Int} \rangle.x), \langle M \rangle$$

The remaining function is extracted when the stable solution containing the maximum has been reached.

- We may also define the maximum using a recursive γ -abstraction. Let M a multiset of integers, the maximum is:

$$\mathbf{let} \ recmax = \gamma(x : \mathbf{Int}, y : \mathbf{Int})[x \geq y].recmax, x \ \mathbf{in} \\ (\gamma\langle f, x : \mathbf{int} \rangle.x), \langle recmax, M \rangle$$

This definition does not impose any specific order but is not explicitly parallel. However, it can be implemented in parallel. It can be shown that such replicating γ -abstractions are inherently parallel. They can be applied to any number of matching molecules at each reduction step.

- Sequences can be represented by nested solutions e.g. $\langle 1, \langle 2, \dots \rangle \rangle$. The maximum of such a sequence M can be implemented by a recursive definition

$$\mathbf{let} \ itmax = \gamma\langle x, \langle s \rangle \rangle.itmax, \langle max, x, s \rangle \ \mathbf{in} \ (\gamma\langle f, \langle x \rangle \rangle.x), \langle itmax, M \rangle$$

These various solutions illustrate the flexibility and the power of expression of the proposed model. It is possible to express solutions without introducing any artificial constraints; however, if required, it is always possible to constrain the execution flow by introducing a hierarchy of solutions.

5 Conclusion

The presented approach provides a new “vision” of Gamma programs which can be considered as pure multiset rewriting. There is no more distinction between the set of rewriting rules describing the computation and the data multiset. They are combined within a unique multiset containing only molecules which can be traditional data as well as computations (γ -abstractions). Control on the computation can be expressed by appropriate nesting of solutions.

References

- [1] Jean-Pierre Banâtre, Pascal Fradet, and Daniel Le Métayer. Gamma and the chemical reaction model: Fifteen years after. In *Multiset Processing*, LNCS 2235, pp. 17–44. Springer-Verlag, 2001.
- [2] Jean-Pierre Banâtre and Daniel Le Métayer, Programming by multiset transformation. *Communications of the ACM (CACM)*, 36(1):98–111, 1993.
- [3] Jean-Pierre Banâtre and Daniel Le Métayer, Gamma and the chemical reaction model: ten years after. In *Coordination programming: mechanisms, models and semantics*, pp. 1–9, 1996.
- [4] G. Berry and G. Boudol, *The chemical abstract machine*, Theoretical Computer Science, Vol. 96, pp. 217–248, 1992.
- [5] G. Boudol, *Some chemical abstract machines*, in *Proc. of the workshop on A decade of concurrency*, 1994, Springer Verlag, LNCS 803, pp. 92–123.
- [6] D. Cohen and J. Muylaert-Filho, *Introducing a calculus for higher-order multiset programming*, in *Proc. Coordination’96 Conference*, LNCS 1061, pp. 124–141, 1996.
- [7] D. Le Métayer, *Higher-order multiset programming*, in *Proc. of the DIMACS workshop on specifications of parallel algorithms*, American Mathematical Society, Dimacs series in Discrete Mathematics, Vol. 18, 1994.
- [8] R. Milner, *Communication and concurrency*, International Series in Computer Science, Prentice Hall, Englewood Cliffs, NJ, 1989.