A Generalized Higher-Order Chemical Computation Model with Infinite and Hybrid Multisets

Jean-Pierre Banâtre, Pascal Fradet, Yann Radenac

A Generalized Higher-Order Chemical Computation Model

J.-P. Banâtre [a], P. Fradet [b] and Y. Radenac [a]

[a] *INRIA/IRISA, Université de Rennes 1,*
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

Gamma is a programming model where computation is seen as chemical reactions between data represented as molecules floating in a chemical solution. Formally, this model is represented by the rewriting of a multiset where rewrite rules model the chemical reactions. Recently, we have proposed the γ-calculus, a higher-order extension, where the rewrite rules are first-class citizen. The work presented in this paper increases further the expressivity of the chemical model with generalized multisets: multiplicities of elements may be infinite and/or negative. Applications of these new notions are illustrated by some programming examples.

Key words: Chemical metaphor, computation model, higher-order, multiset, infinite and negative multiplicities.

1 Introduction

The Gamma formalism was proposed in [5] to capture the intuition of computation as the global evolution of a collection of atomic values interacting freely. Gamma can be introduced intuitively through the chemical reaction metaphor. The unique data structure in Gamma is the multiset which can be seen as a chemical solution. A simple program is made of a *reaction 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.

For example, the computation of the maximum element of a non empty set can be described by the reaction rule **replace** x, y **by** x **if** $x \geq y$ meaning that any couple of elements x and y of the multiset is replaced by x if the

This is a preliminary version. The final version will be published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

condition is fulfilled. This process goes on till a stable state is reached, that is to say, when only the maximum element remains. Note that, in this definition, nothing is said about the order of evaluation of the comparisons. If several disjoint pairs of elements satisfy the condition, the reactions can be performed in parallel.

Gamma can be formalized as a multiset rewriting language. The literature about Gamma as summarized in [1] is based on finite multisets (often called bags). However, one may think of extensions to this basic concepts by generalizing the multiplicity of elements in multisets to infinity (*infinite multisets*) and even multisets with elements possessing a negative multiplicity (*hybrid multisets*).

In this paper, we investigate these unconventional multiset structures (infinite and hybrid multisets) and show how they can be interpreted in a chemical programming framework. Section 2 presents the basic framework and introduces a higher-order chemical model [4]. Section 3 presents HOCL, a language based on the previous model which integrates infinite multisets and negative multiplicity. It sketches the semantics and implementation of the language using characteristic functions. We conclude in Section 4 with a short review of related work.

2 A higher-order chemical model

In this section, we introduce a higher-order chemical model called the γ-calculus [3,4]. In this chemical model of computation, every element (including reaction rules) is considered as a molecule. A program is a solution of mole-

M	::=	x	*; variable*
	|	$\gamma(P)\lfloor C\rfloor.M$	*; γ-abstraction (reaction rule)*
	|	M_1, M_2	*; multiset (AC)*
	|	$\langle M\rangle$	*; solution*
P	::=	x	*; matches any molecule*
	|	P_1, P_2	*; matches a compound molecule*
	|	$\langle P\rangle$	*; matches an inert solution*

Grammar 1: Syntax of molecules.

cules. A molecule can be (cf. Grammar 1) (1) a variable x that can represent any molecule, (2) a γ-abstraction $\gamma(P)\lfloor C\rfloor.M$ where P is the pattern which determines the format (or type) of the expected molecule, C is the reaction condition and M the result of the reaction, (3) a compound molecule (M_1, M_2)

built with the associative and commutative constructor ",", or (4) a solution denoted by $\langle M \rangle$ which isolates a molecule M from the others.

Molecules can be freely reorganized using the associativity and commutativity of the multiset constructor ",":

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

These rules can be seen as a formalization of the Brownian motion of chemical solutions.

Another distinctive feature of chemical models is the reaction concept. In our model, it is represented by a rewrite rule:

$$(\gamma(P)\lfloor C \rfloor.M),\ N \to \phi M \qquad \text{if } P \text{ match } N = \phi \text{ and } \phi C$$

If a γ-abstraction "meets" a closed molecule N that matches the pattern P (modulo a substitution ϕ) and satisfies the reaction condition C (a boolean expression), then they may react. The γ-abstraction $\gamma(P)\lfloor C \rfloor.M$ and the molecule N are replaced by the molecule ϕM (i.e. the body of the abstraction after substitution).

An execution consists in such rewritings ("chemical" reactions) until the solution becomes inert (no further rewriting is possible). There are two structural rules:

$$\textit{locality}\ \frac{M_1 \to M_2}{M, M_1 \to M, M_2} \qquad \textit{solution}\ \frac{M_1 \to M_2}{\langle M_1 \rangle \to \langle M_2 \rangle}$$

The *locality* rule states that if a molecule M_1 can react then it can do so whatever its context M. The *solution* rule states that reactions can occur within nested solutions.

This model of computation is intrinsically parallel and nondeterministic. As long as reactions involve different molecules, they can take place at the same time in a solution. Furthermore, if a molecule contains several elements, it is not know a priori how they will combine because of the Brownian motion. For example, consider the solution $\langle(\gamma(x, y).x), \mathbf{true}, \mathbf{false}\rangle$, it may reduce to two distinct stable terms ($\langle\mathbf{true}\rangle$ or $\langle\mathbf{false}\rangle$) depending on the application of AC rules and whether the x will match **true** or **false**.

Note that abstractions ($\gamma(P)\lfloor C \rfloor.M$) disappear in reactions: they are said to be *one-shot*. It is easy (using recursion) to define n-shot abstractions (denoted like in Gamma by $\mathbf{replace}\, P\, \mathbf{by}\, M\, \mathbf{if}\, C$) which do not disappear in reactions. For instance, the following program:

$$\langle 2, 10, 5, 8, 11, 8, \mathbf{replace}\, x, y\, \mathbf{by}\, x\, \mathbf{if}\, x \geq y \rangle$$

computes the maximum of some integers. The abstraction does not disappear and reacts as long as there are at least two integers in the solution. The resulting inert solution is $\langle 11, \mathbf{replace}\, x, y\, \mathbf{by}\, x\, \mathbf{if}\, x \geq y \rangle$.

$$
\begin{array}{l}
\text{largestPrime10} = \\
\quad \textbf{let}\ sieve = (sieve)\ \textbf{replace}\ x, y\ \textbf{by}\ x\ \textbf{if}\ x \operatorname{div} y\ \textbf{in} \\
\quad \textbf{let}\ max = \textbf{replace}\ x, y\ \textbf{by}\ x\ \textbf{if}\ x \geq y\ \textbf{in} \\
\quad \langle\langle 2, 3, 4, \ldots, 10, sieve\rangle, \gamma\langle (sieve)s, x\rangle.x, max\rangle
\end{array}
$$

Program 1: Computes the largest prime number lower than 10.

A solution can be matched only if it is inert (i.e. no more reaction can occur in it). This is an important property that permits the ordering of rewritings. For example, Program 1 uses this restriction to sequence the different steps of the computation. Program 1 computes the largest prime number lower than a given integer. First, only reactions inside the sub-solution may occur: the *sieve* abstraction computes all prime numbers lower than 10 by removing integers (y) for which a divider (x) is found. When the sub-solution becomes inert, the abstraction $\gamma\langle (sieve)s, x\rangle.x, max$ matches it s and extracts all the prime numbers x (i.e. suppresses the reaction *sieve*) and computes their maximum using the max reaction. Finally, the resulting solution is $\langle 7, max\rangle$.

3 HOCL: multiplets, infinite multiplets and hybrid multisets

HOCL (*Higher Order Chemical Language*) is a programming language based on the previous model extended with infinite and hybrid multisets. Grammar 2 gives its syntax.

3.1 Multiplets

A *multiplet* is a multiset of identical elements. We introduce an exponential notation to denote and manipulate them:

$$a^n \equiv a^{(n-1)}, a \quad \text{and} \quad a^1 \equiv a \qquad \text{for any } n > 1$$

We consider only multiplets of a basic element (constants, abstractions and solutions) and not of a multiset (for instance, we cannot write $(1, 5)^4$).

The exponential notation is also used for pattern-matching. Abstractions may select a fixed number of identical elements. For example, the abstraction matching three 1 is denoted by:

$$\gamma(x^3)\lfloor x = 1\rfloor.M \quad \equiv \quad \gamma(x, y, z)\lfloor x = y \wedge y = z \wedge x = 1\rfloor.M$$

Consider that an integer x is represented by a multiplet of x 1's, the integer division of x by y can be done by grouping y occurrences of 1's and replacing

Program

$S ::= M$; *a molecule*

$\mid \emptyset$; *nothing*

Molecules

$M ::= A$; *atom (basic molecule)*

$\mid M_1, M_2$; *compound molecule*

$\mid x$; *variable* ($x \in V$)

$\mid A^F$; *multiplet*

Atoms

$A ::= x{:}U$; *variable* ($x \in V$)

$\mid E$; *expression*

$\mid \gamma(P)\lfloor E \rfloor.S$; *one-shot reaction rule*

$\mid \langle M \rangle$; *solution*

$\mid (name)A$; *tagged molecule (*name *is a string)*

Expressions

$E ::= \ldots$; *usual integer and boolean expressions*

Exponent

$F ::= E \mid \infty \mid -\infty$; *integer expression or infinity*

Patterns

$P ::= x{:}T$; *matches any molecule of type* T

$\mid \omega$; *matches any molecule or nothing*

$\mid P_1, P_2$; *matches a compound molecule*

$\mid (name)P$; *matches a molecule tagged with name*

$\mid \langle P \rangle$; *matches an inert solution*

$\mid (P_1 | P_2)$; *matches a pair*

$\mid P^F$; *matches a multiplet*

Grammar 2: Syntax of programs.

$$
\begin{array}{l}
\text{intdiv} = \gamma(x, y). \\
\quad \textbf{let}\ \mathit{cluster} = \gamma(\overline{y}).\ \textbf{replace}\ 1^z\ \textbf{by}\ \widehat{1}\ \textbf{if}\ z = y\ \textbf{in} \\
\quad \textbf{let}\ \mathit{sumRemainder} = \textbf{replace}\ x, y\ \textbf{by}\ x + y\ \textbf{in} \\
\quad \textbf{let}\ \mathit{sumQuotient} = \textbf{replace}\ \widehat{x}, \widehat{y}\ \textbf{by}\ \widehat{x + y}\ \textbf{in} \\
\quad \langle\langle 1^x, \overline{y}, \mathit{cluster}, \mathit{sumQuotient}\rangle, \gamma\langle x\rangle.x, \mathit{sumRemainder}\rangle
\end{array}
$$

Program 2: Integer division.

$$
\begin{array}{l}
\text{power} = \gamma(x, y).\ \textbf{if}\ y = 0\ \textbf{then}\ 1 \\
\qquad\qquad\qquad\qquad \textbf{else if}\ y > 0\ \textbf{then}\ x^y, (\gamma(u, v).u * v)^{y-1} \\
\qquad\qquad\qquad\qquad \textbf{else}\ (\tfrac{1}{x})^{-y}, (\gamma(u, v).u * v)^{-y-1}
\end{array}
$$

Program 3: Power function.

them by a $\widehat{1}$ (a tagged integer denoted by a "hat" here) for the "quotient". Assuming that the denominator is tagged $\overline{y}$ to distinguish it from the 1's of the numerator, the following one-shot rule returns a reaction rule that transforms a multiplet of y 1's by one unit for the quotient.

$$\mathit{cluster} = \gamma(\overline{y}).\ \textbf{replace}\ 1^z\ \textbf{by}\ \widehat{1}\ \textbf{if}\ z = y$$

When the solution becomes inert, the multiplicity of $\widehat{1}$ is the quotient, and the multiplicity of the unmarked 1 is the remainder. For example, to divide 5 by 2 we write $\langle 1^5, \overline{2}, \mathit{cluster}\rangle$ which reduces to $\langle 1, \widehat{1}^2, \textbf{replace}\ 1^z\ \textbf{by}\ \widehat{1}\ \textbf{if}\ z = 2\rangle$. Program 2 is the complete program of that integer division.

The size of a multiplet may be determined only at runtime. For example, the power function (cf. Program 3) computes $\mathit{exp}(x, y)$ $(x, y \in \mathbb{Z})$ by generating two multiplets of variable size: y copies of x and $y-1$ products, if $y > 0$. Here is an example of a reduction to compute $\mathit{exp}(5, 7)$:

$$
\begin{array}{l}
\langle 5, 7, \text{power}\rangle \to \langle 5^7, (\gamma(u, v).u * v)^6\rangle \to \\
\qquad \langle 25, 5^5, (\gamma(u, v).u * v)^5\rangle \to \ldots \to \langle 78125\rangle
\end{array}
$$

Multiplets of solutions represent several copies of a multiset. Each copy is independent of the others, and so evolve on its own. For example, the multiplet $\langle 1, 2, \mathit{choose}\rangle^{42}$ denotes 42 solutions, that may converge, for example, to $\langle 1\rangle^{20}, \langle 2\rangle^{22}$.

In the Jackpot! program (Program 4), three solutions representing the three wheels of the machine evolve independently. When the three solutions become inert, the *win* abstraction checks if it finds 3 identical inert solutions, i.e. a multiplet of inert solutions of size 3.

jackpot =

$\quad$ **let** *choose* = **replace** x, y **by** x **in**

$\quad$ **let** *wheel* = $\langle$cherry, lemon, bell, bar, plum, orange, melon, seven, *choose*$\rangle$ **in**

$\quad$ **let** *win* = $\gamma(\langle x \rangle^3)$."wonJackpot" **in**

$\quad \langle wheel^3, win \rangle$

Program 4: Jackpot!

3.2 Infinite multiplets

An obvious generalization of multiplets is to allow infinite multiplets. An infinite multiplet is denoted by M^∞. It represents an infinity of copies of M, but it is also a molecule that can be produced or removed like any other molecule.

An application of infinite multiplets is to allow elements to take part in any number of reactions in parallel. For example, when a n-shot abstraction reacts, it is put back into the solution after the reaction where it may react again and so on. A parallel interpretation of a n-shot abstraction could be made using an infinite multiplet of the corresponding one-shot abstraction:

$$\textbf{replace}\, P\, \textbf{by}\, M\, \textbf{if}\, C \;\simeq\; (\gamma(P)\lfloor C \rfloor.M)^\infty$$

Instead of having one molecule taking part to one reaction at a time (sequential process), we consider having an infinity of abstractions that can react at the same time with distinct molecules.

Another example is the quicksort program where all integers must be compared to a predefined pivot. In the following solution all integers lower or equal to the pivot (represented here by an integer tagged by the string *pivot*) are removed:

$$\langle (pivot)5, 8, 3, 6, 4, 5, 3, \textbf{replace}(pivot)x, y\, \textbf{by}(pivot)x\, \textbf{if}\, x \geq y \rangle$$

Since the n-shot abstraction and the pivot are unique, only one reaction can occur at each reduction step. Considering the n-shot abstraction and the pivot as infinite multiplets, several comparisons can occur at the same time:

$$\langle ((pivot)5)^\infty, 8, 3, 6, 4, 5, 3, ((cmp)\gamma((pivot)x, y)\lfloor x \geq y \rfloor.\emptyset)^\infty \rangle$$

Here, the 6 comparisons can occur at the same time since we have at least 6 copies of the pivot and 6 copies of the comparator (rule *cmp*).

By extending patterns, infinite multiplets can be manipulated as a single molecule. For example, when the previous computation is finished, the infinite

multiplets can be removed in one reaction:

$$\gamma\langle((pivot)x)^\infty, ((cmp)y)^\infty, z\rangle.\langle z\rangle$$

Since we consider only multiplet of basic elements, the expression $(1,2)^\infty$ is illegal. This restriction is motivated by representation issues (see Section 3.4).

3.3 Negative multiplicities

Hybrid multisets [7,10] are a generalization of multisets. In a hybrid multiset, the multiplicity of an element can be negative. A molecule a^{-1} can be viewed as "antimatter"(e.g. an anti a): positive and negative multiplets of the same atom cannot cohabit in the same solution, they merge into one multiplet whose exponent is the sum of the multiplicities.

For example, assume that a rational number $\frac{p}{q}$ is represented by a molecule which contains the prime factorization of p and q but with negative multiplicities for the latter. Then, $\frac{20}{9}$ is represented by the molecule $2^2, 5, 3^{-2}$. By simply putting molecules representing rational numbers together in the same solution, we compute their product. For example, the product $\frac{20}{9} * \frac{15}{8}$ is represented by the following reaction $\langle 2^2, 5, 3^{-2}\rangle, \langle 3, 5, 2^{-3}\rangle, \gamma(\langle f\rangle, \langle g\rangle).\langle f, g\rangle \to \langle 5^2, 3^{-1}, 2^{-1}\rangle$.

Infinite negative multiplets can be seen as black holes. It can be used to remove all elements (present or to come) of a multiset. Let pi a reaction computing the product of a multiset of integers. Then, the integer 1, being the neutral element of the product, can be deleted prior to performing pi. The pi operator may be encoded by:

$$\text{pi} = \gamma\langle x\rangle.\langle 1^{-\infty}, x, (\textbf{replace}\, x, y\, \textbf{by}\, x * y)\rangle$$

Before considering any product, all 1's are annihilated, for example:

$$\langle 2, 9, 1, 5, 6, 1, 1, 2\rangle, \text{pi} \to \langle 1^{-\infty}, 2, 9, 5, 6, 2, (\textbf{replace}\, x, y\, \textbf{by}\, x * y)\rangle \to \ldots$$

After stabilization, $1^{-\infty}$ must be replaced by 1 (in case that the solution contained only 1's) and then the reaction can be removed.

Other examples that come to mind include a garbage collector that destroys useless molecules by generating their negative counterpart, or negative molecules used as anti-virus: a part of a system identifies a virus and generates an infinity of anti-virus (negative counterparts) that will spread in the whole system to clean it.

3.4 Representation with characteristic functions

Infinite multiplets cannot be represented by enumerating all their atoms. Representing them by generators (e.g. $M^\infty \equiv \textbf{replace}\, \emptyset\, \textbf{by}\, M$) makes it difficult to ensure that $M^\infty, M^\infty \equiv M^\infty$. Instead, we use the standard mathematical representation of a multiset, that is a function associating to each element of

the multiset its number of occurrences (multiplicity): A molecule M is represented by a *characteristic function* $\{\!|M|\!\} : \mathbf{Atoms} \to \mathbb{Z} \cup \{+\infty, -\infty\}$ such that for all $x \in \mathbf{Atoms}$, $\{\!|M|\!\}(x)$ represents the number of occurrences of x in M. The set of values **Atoms** is defined as:

$$\mathbf{Atoms} ::= e \mid \gamma(P)\lfloor B \rfloor.M \mid \langle M \rangle \mid (ident)A$$

Atoms are either an expression e in normal form (an integer, a boolean or a pair), a γ-abstraction, a solution (represented by its characteristic function) or a named molecule.

The representation of multisets in terms of functions depends on a notion of equality on **Atoms**. Since our multisets may contain programs, a true semantic equality cannot be implemented. A simple choice would be to use equality on basic values (integers, booleans and names), each composite value (γ-abstractions, sub-solutions, etc.) being only equal to themselves (structural equality). For example, the solution $\langle 1^{\infty}, 4, 5^{-2}, (\gamma(x,y).x), \langle 3, \mathbf{true} \rangle \rangle$ is represented by the function that for any $x \in \mathbf{Atoms}$:

$$f(x) = \begin{cases} \infty & \text{if } x = 1 \\ 1 & \text{if } x = 4 \\ -2 & \text{if } x = 5 \\ 1 & \text{if } x = \gamma(x,y).x \\ 1 & \text{if } \{\!|x|\!\} = g \\ 0 & \text{otherwise} \end{cases} \quad \text{with } g(x) = \begin{cases} 1 \text{ if } x = 3 \\ 1 \text{ if } x = \mathbf{true} \\ 0 \text{ otherwise} \end{cases}$$

4 Conclusion

In this short paper, we have tried to convey the main ideas and applications of infinite and hybrid multisets. The interested reader will be able to find more details in [2].

As far as the higher-order generalization is concerned, another higher-order extension of Gamma has already been proposed in [9]. However, reactions were not first-class citizens since they were kept separate from multisets of data. The hmm-calculus [8] (for higher-order multiset machines) is an extension of Gamma where reactions are one-shot and first-class citizens. Although, the hmm-calculus is an interesting attempt to generalize the Gamma model of computation, it has not been thought as a programming language as HOCL.

Other models, inspired by Gamma, are worth to be mentioned. The chemical abstract machine or Cham was proposed in [6] to describe the operational semantics of process calculi. P-systems [11] are computing devices inspired from biology. It consists in nested membranes in which molecules reacts. A set of partially ordered rewrite rules is associated to each membrane. These

rules describe possible reactions and communications between membranes of molecules. All these models are first-order.

Currently, we are investigating a complete semantics of infinite and negative multiplets with characteristic functions.

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., P. Fradet and Y. Radenac, *Chemical programming with infinite and hybrid multisets*, submitted for publication (available on request).

[3] Banâtre, J.-P., P. Fradet and Y. Radenac, *Higher-order programming style*, in: *Proc. of the workshop on Unconventional Programming Paradigms (UPP'04)*, LNCS **3566** (2005).

[4] Banâtre, J.-P., P. Fradet and Y. Radenac, *Principles of chemical programming*, in: S. Abdennadher and C. Ringeissen, editors, *Proceedings of the 5th International Workshop on Rule-Based Programming (RULE 2004)*, ENTCS **124** (2005), pp. 133–147.

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

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

[7] Blizard, W., *Negative membership*, Notre Dame Journal of Formal Logic **31** (1990), pp. 346–368.

[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] 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.

[10] Loeb, D., *Sets with a negative number of elements*, Advances in Mathematics **91** (1992), pp. 64–74.

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