# An Alternative Semantics for the Parallel Operator of the Calculus of Gamma Programs

P. Ciancarini R. Gorrieri G. Zavattaro

Department of Computer Science, University of Bologna
Piazza di Porta San Donato 5, I-40127 Bologna, Italy
e-mail: {cianca,gorrieri,zavattar}@cs.unibo.it

**Abstract**

The Gamma model is a minimal programming notation based on the chemical reaction metaphor. Hankin, Le Métayer, and Sands derived a Calculus of Gamma Programs adding two operators to the notation: the *parallel* and *sequential* composition operators. In this paper we address some compositional problems due to the semantics of the parallel operator for which we propose an alternative semantics as a solution to these problems. The semantics of the new version of the calculus is also defined by means of a *labeled* transition system. This permits the definition of some observational equivalences. We introduce two of them using the concept of bisimulation. The equivalences are proved to be congruences and are characterized by means of *sound* and *complete* axiomatizations.

## 1 Introduction

The Gamma model is a notation proposed by Banâtre and Le Métayer in [BM93] as a means for the high level description of parallel programs with minimum explicit control.

The model is based on a shared data structure represented by a multiset. A multiset is a bag containing items which are stored without any constraints or relationships among them. The items in the multiset can be seen as molecules that freely move in a sort of chemical solution. The molecules react when they come in contact provided that they satisfy certain constraints. The reaction rules are defined by a Gamma program. Each rule is essentially a *conditional multiset rewriting rule*, defined by a *reaction condition* and its related *rewriting* *action*. When a bag of molecules satisfy the *reaction condition*, then it can be rewritten in the way stated by its *rewriting action*.

For instance, in the Gamma program *max* defined as follows:

$$max \ : \ \{x, y\} \longrightarrow \{x\} \quad \Longleftarrow \quad x \geq y$$

$x \geq y$ is the *reaction condition* and $\{x, y\} \longrightarrow \{x\}$ is its *rewriting action*. Program *max* computes the maximum element of a non empty multiset of integers by eliminating each number of the multiset every time there exists another one greater than it.

## 1.1 The Calculus of Gamma Programs

For the sake of modularity it is desirable that a language offers a set of operators for combining programs. It is also fundamental that these operators enjoy a useful collection of algebraic laws. In [HMS92] and [HMS93] Hankin, Le Métayer and Sands defined CGP, a language obtained by adding two composition operators to Gamma. These operators are:

- **Parallel composition operator**: $P + Q$
  The reactions induced by the programs $P$ and $Q$ can be executed in any order, possibly in parallel. The program terminates only when no reactions can be generated by either programs.
- **Sequential composition operator**: $P \circ Q$
  The multiset reached after the execution of program $Q$ is given as argument to $P$.

Using parallel and sequential composition operators we can write programs that compute complex functions starting from elementary basic Gamma programs. For example, to compute the number of elements less than or equal to $N$ in a non empty set of positive integers, the following program can be used:
$\quad Occur \ : \ Add \circ (Ones + Del)$
where *Ones* transforms to 1 every element less than or equal to $N$:
$\quad Ones \ : \ \{x\} \longrightarrow \{1\} \ \Longleftarrow \ x \leq N \ \wedge \ x \neq 1$
*Del* eliminates every element greater than $N$:
$\quad Del \ : \ \{x\} \longrightarrow \{\} \ \Longleftarrow \ x > N$
and *Add* returns the sum of the elements of a multiset of integers:
$\quad Add \ : \ \{x, y\} \longrightarrow \{x + y\} \ \Longleftarrow \ true$

## 1.2 Why an alternative parallel operator ?

In this paper we focus on compositional problems related to the semantics of CGP. The problems are due to the parallel composition operator: a program obtained as the parallel composition of structured subprograms could give undesired results. The only way to eliminate the undesired computations is to decompose the subprograms. But, for the sake of modularity, such a decomposition is not desirable.

A possible solution to this kind of problems is to define an alternative semantics for the parallel operator,[1] that is what we are presenting.

### 1.3 Overview

In **Section 2** we present the syntax and semantics of CGP . We also recall one of the equivalences defined in [HMS92] and [San93]: the *transition trace equivalence* which is a congruence for the original CGP. In **Section 3** compositional problems of CGP are described. In **Section 4** an alternative parallel composition operator for CGP is proposed as a solution for the compositional problems. Moreover, the calculus with the new semantics is defined also by means of a *labeled* transition system which does not take into account the multiset representing the chemical solution. This new definition allows to introduce some observational equivalences for CGP. In **Section 5** two observational equivalences are defined: they are proved to be congruences and they are characterized by means of *sound* and *complete* axiomatizations. **Section 6** reports some concluding remarks. The proofs of Propositions are not reported in this extended abstract, and can be found in the full version of the paper.

## 2 Syntax and Semantics of CGP

In the following we use the notations:

| | |
|---|---|
| $Boolean$ : | set containing the boolean values *true* and $false$; |
| $Molecules$ : | set with the terms possibly contained in a chemical solution; |
| $\mathcal{M}(S)$ : | set containing all possible multisets of elements of $S$; |
| $\mathcal{P}(S)$ : | set containing all possible subsets of $S$; |
| $\mathcal{F}(A,B)$ : | set containing all possible functions defined from $A$ into $B$; |
| $\cup$,$\setminus$,$\cap$ : | union, difference, intersection operators for multisets. |

Let $Program$, ranged over by $P$, $Q$, etc., be the set of CGP programs defined by the rule:

$$P \quad ::= \quad (R, A) \ \mid \ P \circ P \ \mid \ P + P$$

where

$$\begin{array}{lcl} R & \in & \mathit{Reactions} \\ A & \in & \mathit{Actions} \end{array}$$

[1]This is equivalent to introduce an alternative parallel composition operator.

| | |
|---|---|
| (1) | $\dfrac{\mathcal{I}_R(R)(m_1) = true \ \wedge \ \mathcal{I}_A(A)(m_1) = m_2}{\langle (R,A), M \cup m_1 \rangle \longrightarrow \langle (R,A), M \cup m_2 \rangle}$ |
| (2) | $\dfrac{\nexists m_1 \subseteq M : \mathcal{I}_R(R)(m_1) = true}{\langle (R,A), M \rangle \longrightarrow M}$ |
| (3) | $\dfrac{\langle Q, M \rangle \longrightarrow M}{\langle P \circ Q, M \rangle \longrightarrow \langle P, M \rangle}$ |
| (4) | $\dfrac{\langle Q, M \rangle \longrightarrow \langle Q', M' \rangle}{\langle P \circ Q, M \rangle \longrightarrow \langle P \circ Q', M' \rangle}$ |
| (5) | $\dfrac{\langle P, M \rangle \longrightarrow M \ \wedge \ \langle Q, M \rangle \longrightarrow M}{\langle P + Q, M \rangle \longrightarrow M}$ |
| (6) | $\dfrac{\langle P, M \rangle \longrightarrow \langle P', M' \rangle}{\langle P + Q, M \rangle \longrightarrow \langle P' + Q, M' \rangle}$ |
| (7) | $\dfrac{\langle Q, M \rangle \longrightarrow \langle Q', M' \rangle}{\langle P + Q, M \rangle \longrightarrow \langle P + Q', M' \rangle}$ |

Table 1: Original operational semantics of CGP.

For the sake of generality we do not specify the syntax of *Reactions* and *Actions*; we only consider the existence of two functions which define their semantics:

$$\begin{array}{lcccl} \mathcal{I}_R & : & Reactions & \longrightarrow & \mathcal{F}(\mathcal{M}(Molecules), Boolean) \\ \mathcal{I}_A & : & Actions & \longrightarrow & \mathcal{F}(\mathcal{M}(Molecules), \mathcal{M}(Molecules)) \end{array}$$

In every basic program $(R, A)$, $\mathcal{I}_R(R)$ and $\mathcal{I}_A(A)$ are two functions: $\mathcal{I}_R(R)$ defines the condition that a multiset of molecules must satisfy to be able to react, $\mathcal{I}_A(A)$ associates to every bag of molecules the multiset in which it can be rewritten.

An alternative syntax for basic programs $(R, A)$ is the following:

$$name \ : \ \{x_1, \ldots, x_n\} \longrightarrow A(x_1, \ldots, x_n) \quad \Longleftarrow \quad R(x_1, \ldots, x_n)$$

where *name* is an identifier for the program, $R(x_1, \ldots, x_n)$ is the *Reaction* condition function and $A(x_1, \ldots, x_n)$ is the related *Action* function.

The semantics of CGP is given in the style of Plotkin's Structured Operational Semantics [Plo81]. Configurations consist of a program and a multiset;

transitions are of the form:

$$\langle P, M\rangle \longrightarrow \langle P', M'\rangle \quad \text{or}$$
$$\langle P, M\rangle \longrightarrow M$$

Configurations composed by multisets only are terminal. The semantics is presented in Table 1. Rules (1) and (2) define the possible computations for basic Gamma programs. Rule (1) says that if the multiset representing the chemical solution during the computation of $(R, A)$ contains a bag $m_1$ that satisfy the boolean function $\mathcal{I}_R(R)$, then it can be replaced by the result of the function $\mathcal{I}_A(A)$ applied to $m_1$. Instead, rule (2) says that the computation of $(R, A)$ terminates when the chemical solution does not contain bags of molecules that satisfy the reaction condition. The other rules define the computations for structured programs.

In order to identify programs with different operational behavior which can be considered semantically equivalent, in [HMS92] and [San93] a variety of equivalences on programs are defined. We recall the *transition trace equivalence* [San93] which is a congruence.

**Definition 2.1** *Transition traces* :

$$\begin{array}{rcl} \mathcal{TT}[P] & = & \{(M_0, N_0)(M_1, N_1)...(M_k, N_k) \mid \\ & & \langle P, M_0\rangle \longrightarrow^\star \langle P_1, N_0\rangle \ \& \ \langle P_1, M_1\rangle \longrightarrow^\star \langle P_2, N_1\rangle \ \& \\ & & .... \\ & & \langle P_{k-1}, M_{k-1}\rangle \longrightarrow^\star \langle P_k, N_{k-1}\rangle \ \& \ \langle P_k, M_k\rangle \longrightarrow^\star N_k\} \qquad \blacksquare \end{array}$$

Each transition trace of $P$ represents a terminating execution of program $P$ in some context starting with multiset $M_0$. Each of the pairs $(M_i, N_i)$ represents computation steps performed by (derivatives of) $P$ and the adjacent multisets $N_{i-1}, M_i$ represent possible "interfering" computation steps performed by the context.

**Definition 2.2** *Transition trace equivalence* :

$$P \simeq_T Q \Longleftrightarrow \mathcal{TT}[P] = \mathcal{TT}[Q] \qquad \blacksquare$$

# 3 Compositional Problems

We now show that the semantics of CGP, introduced in Section 2, presents some compositional problems. In fact, rules (3) and (6) of Table 1 imply the following property:

> *If the multiset $M$ is stable for the program $Q$ (i.e. $\langle Q, M\rangle \longrightarrow M$) then this transition is allowed: $\langle (P \circ Q) + R, M\rangle \longrightarrow \langle P + R, M\rangle$*

In other words, we have that program $P \circ Q$ can be reduced to term $P$ independently of the other programs composed in parallel with it. This can be called the *non context sensitive reduction property*. This kind of property implies the following problems.

## (i) Incoherence with final termination

Rule (5) of Table 1 defines the condition for the final termination of terms composed in parallel. We have that program $P$ in parallel with $Q$ could finish only when $Q$ is not able to perform further transformations. Thus for final termination we have a *context sensitive* condition while for partial termination (i.e. reduction) we have a *non context sensitive* condition.

## (ii) Undesired computations

The program $(P \circ Q) + R$ can be reduced to $P + R$ even if its subprogram $R$ could induce some reactions. But if $R$ is able to act a chemical reaction that changes the solution $M$ to a solution $M'$ in which $Q$ is active, then it is reasonable that the transformation of $M$ in $M'$ happens before $Q$ is eliminated. In fact, if $Q$ is removed before the transformation of $M$ into $M'$, then undesired results could be given as the following example shows.

**Example 3.1** Let $numMax$ be a program that computes the number of occurrences of the maximum element among the non negative elements of a set of integers:

$$numMax \;:\; sum \circ ones \circ max$$

where:

$$\begin{array}{lcllcl} max & : & \{x, y\} \longrightarrow \{x\} & \Longleftarrow & x > y \geq 0 \\ ones & : & \{x\} \longrightarrow \{1\} & \Longleftarrow & 1 \neq x \geq 0 \\ sum & : & \{x, y\} \longrightarrow \{x + y\} & \Longleftarrow & true \end{array}$$

Suppose to have a program $abs$ that returns the set with the absolute value of the elements of a set of integers:

$$abs \;:\; \{x\} \longrightarrow \{-x\} \;\Longleftarrow\; x < 0$$

If we want to compute the number of occurrences of the maximum absolute value among the positive and negative elements of a set of integers, we cannot use the program:

$$numMax + abs$$

because the program $max$ could terminate even if $abs$ is not finished. For example, if the program $numMax + abs$ starts on a multiset $M$ with only negative integers, then the following is allowed as the first step of computation:

$$\langle numMax + abs, M \rangle \longrightarrow \langle (sum \circ ones) + abs, M \rangle$$

and, if it happens, the final result of the computation is the number of elements of the multiset $M$. ■

With respect to the Example above, there are two possible ways to remove the undesired computations:

- **Use of the sequential composition operator :**

  $$numMax \circ abs$$

  Such a solution is not acceptable because it implies the loss of potential parallelism.

- **Decomposition of the subprograms :**

  $$sum \circ ones \circ (max + abs)$$

  Such a solution is not desirable for the sake of modularity; it is not acceptable to decompose structured programs every time we want to use them as components of more complex ones.

# 4 An Alternative Semantics

In the previous section we showed that the *non context sensitive reduction property* produces some compositional problems. These problems are due to the semantics of the parallel composition operator: that is why we propose a new version of the calculus with an alternative semantics for that operator.

## 4.1 Alternative semantics

Our idea for eliminating the *non context sensitive reduction property* is to synchronize the reduction of the programs:

> *Every time a program is not able to generate a computation step, it must waits until all the other parallel active programs are blocked. Only when no program can generate a transformation, the reductions of the programs can be executed all at the same time.*

| | |
|---|---|
| (1) | $\dfrac{\mathcal{I}_R(R)(m_1) = true \ \wedge \ \mathcal{I}_A(A)(m_1) = m_2}{\langle (R, A), M \cup m_1 \rangle \longrightarrow \langle (R, A), M \cup m_2 \rangle}$ |
| (2) | $\dfrac{\nexists m_1 \subseteq M : \mathcal{I}_R(R)(m_1) = true}{\langle (R, A), M \rangle \longrightarrow M}$ |
| (3) | $\dfrac{\langle Q, M \rangle \longrightarrow M}{\langle P \circ Q, M \rangle \longrightarrow \langle P, M \rangle}$ |
| (4) | $\dfrac{\langle Q, M \rangle \longrightarrow \langle Q', M' \rangle}{\langle P \circ Q, M \rangle \longrightarrow \langle P \circ Q', M' \rangle}$ |
| (5) | $\dfrac{\langle P, M \rangle \longrightarrow M \ \wedge \ \langle Q, M \rangle \longrightarrow M}{\langle P + Q, M \rangle \longrightarrow M}$ |
| (8) | $\dfrac{\langle P, M \rangle \longrightarrow \langle P, M' \rangle}{\langle P + Q, M \rangle \longrightarrow \langle P + Q, M' \rangle \ \ \& \ \ \langle Q + P, M \rangle \longrightarrow \langle Q + P, M' \rangle}$ |
| (9) | $\dfrac{\langle P, M \rangle \longrightarrow \langle P', M \rangle \ \wedge \ \langle Q, M \rangle \longrightarrow \langle Q', M \rangle \ \wedge \ P' \neq P \ \wedge \ Q' \neq Q}{\langle P + Q, M \rangle \longrightarrow \langle P' + Q', M \rangle}$ |
| (10) | $\dfrac{\langle P, M \rangle \longrightarrow M \ \wedge \ \langle Q, M \rangle \longrightarrow \langle Q', M \rangle \ \wedge \ Q' \neq Q}{\langle P + Q, M \rangle \longrightarrow \langle P + Q', M \rangle \ \ \& \ \ \langle Q + P, M \rangle \longrightarrow \langle Q' + P, M \rangle}$ |

Table 2: Alternative operational semantics of CGP.

Table 2 formally introduces this alternative semantics based on a *synchronized* program reduction. The alternative operational semantics is different from the original one only for rules (8), (9) and (10) which replace the original (6) and (7) (see Table 1). Rule (8) defines that in a program obtained using the parallel composition operator, both subprograms are able to generate multiset transformations. Rule (9) and (10) introduce the alternative semantics for the parallel composition operator: programs composed in parallel are reduced contemporarily only when no one of them is able to generate multiset transformations. That is why, we say that the alternative semantics that we propose satisfies a *context sensitive reduction property*.

It is not difficult to see that the semantics of Table 2 only removes some *non-determinism* from CGP: in fact, it only eliminates the computations which contain *non-synchronized* reductions. For example, if the alternative semantics is considered, the computations which do not return the number of occurrences of the maximum absolute value are removed from the possible behavior of the program $numMax + abs$ of Example 3.1.

After having removed the *non-synchronized* reductions from CGP, we analyze the problem of defining semantic equivalences for programs. First of all, we note that the *transition trace equivalence* is no more a congruence for the new semantics as the following example shows.

**Example 4.1** Consider a program $(R, A)$ not able to generate active *transformations*, that is a program $(R, A)$ such that $\mathcal{I}_R(R)(m) = false$ for every multiset $m$. Such a program is denoted by $nil$. For every generic program $P$:

$$P \simeq_T P \circ nil$$

If we consider two generic programs $Q$ and $R$ with $R \neq nil$ we have that:

$$P + (Q \circ R) \not\simeq_T (P \circ nil) + (Q \circ R)$$

because, if the alternative semantics is considered, in the program $(P \circ nil) + (Q \circ R)$ the subprogram $P$ becomes active only when $R$ is finished whereas in $P + (Q \circ R)$, $P$ and $R$ are contemporarily active. ■

Our idea for solving the problem of defining semantic congruences for CGP, is to introduce a new *labeled* transition system $(Program, Label, \longrightarrow)$ which describes all the possible transformations that a program can induce without taking into account a particular chemical solution. For example $P \stackrel{m_1 \overline{m_2}}{\longrightarrow} P$ could be used to mean that one of the possible chemical transformations induced by the program $P$ is the one which replaces the multiset $m_1$ with $m_2$. The advantages of having such a new definition of the operational semantics of CGP are several:

- The possible behavior of a program is completely described without taking into account the multiset which represents the chemical solution;
- General properties of CGP programs can be more easily verified: it is no more necessary to analyze all the computations obtained starting from every possible chemical solution;
- This new kind of definition for the semantics of CGP is very near to a process algebraic operational semantics definition: this allows to adapt to CGP the standard observational equivalences for process algebras (e.g. bisimulation, failures or traces).

## 4.2 New operational semantic definition

First of all we define a labeled transition system $(Program, Label, \longrightarrow)$ which represents the possible *transformations* generated by the programs of CGP.

There are two different kinds of *transformations* (i.e. two different kinds of label):

- **Active transformations**
  A program generates an active transformation when it executes a chemical reaction on the solution. We describe the fact that program $P$ is able to replace the multiset of molecules $m_1$ with the multiset $m_2$ by the following labeled transition:

$$P \stackrel{m_1\overline{m_2}}{\longrightarrow} P$$

- **Passive transformations**
  A program generates a passive transformation when it is not able to induce reactions on the solution. In this case the program $P$ must be reduced to a new program $P'$. We describe this by the following labeled transition:

$$P \stackrel{\nu}{\longrightarrow} P'$$

Let *Label*, ranged over by $\alpha$, $\beta$, etc., be the set defined in the following:

$$Label \;=\; \{m_1\overline{m_2} \mid m_i \in \mathcal{M}(Molecules)\} \cup \{\nu\}$$

The labeled relation $\longrightarrow \subseteq (Program \times Label \times Program)$ is the minimum one satisfying the axioms and rules of Table 3.

The first two rules define the possible *transformations* generated by a basic program. In order to define them, we have added the term $\underline{0}$ to CGP to represent a program at the end of the computation.[2] generate active transformations (it is able to generate only a passive transformation be Rules (3) and (4) describe the behavior of the *active transformations* for structured programs. The *passive transformations* are more difficult to define, in fact this kind of actions must be executed by all the parallel programs at the same time (in order to have a *synchronized* reduction). Rule (5) and rules (6) and (7) describe the termination and the possible non terminal reductions for programs obtained using the alternative semantics, respectively. Instead, rules (8) and (9) describes the *passive transformations* for programs composed using the sequential composition operator.

The possible *transformations* can be used to introduce a new definition for the alternative semantics introduced above: in fact, Table 4 shows how

[2] Term $\underline{0}$ is different from *nil* because $\underline{0}$ is not able to generate any kind of transformations (neither active nor passive) while *nil* is able to generate the following passive transformation: $nil \stackrel{\nu}{\longrightarrow} \underline{0}$.

$$(1)\quad \frac{\mathcal{I}_R(R)(m_1) = \mathit{true} \;\wedge\; \mathcal{I}_A(A)(m_1) = m_2}{(R,A) \stackrel{m_1\overline{m_2}}{\longrightarrow} (R,A)}$$

$$(2)\quad (R,A) \stackrel{\nu}{\longrightarrow} \underline{0}$$

$$(3)\quad \frac{P \stackrel{m_1\overline{m_2}}{\longrightarrow} P}{P+Q \stackrel{m_1\overline{m_2}}{\longrightarrow} P+Q \;\;\&\;\; Q+P \stackrel{m_1\overline{m_2}}{\longrightarrow} Q+P}$$

$$(4)\quad \frac{P \stackrel{m_1\overline{m_2}}{\longrightarrow} P}{Q \circ P \stackrel{m_1\overline{m_2}}{\longrightarrow} Q \circ P}$$

$$(5)\quad \frac{P \stackrel{\nu}{\longrightarrow} \underline{0} \;\wedge\; Q \stackrel{\nu}{\longrightarrow} \underline{0}}{P+Q \stackrel{\nu}{\longrightarrow} \underline{0}}$$

$$(6)\quad \frac{P \stackrel{\nu}{\longrightarrow} P' \neq \underline{0} \;\wedge\; Q \stackrel{\nu}{\longrightarrow} \underline{0}}{P+Q \stackrel{\nu}{\longrightarrow} P'+Q \;\;\&\;\; Q+P \stackrel{\nu}{\longrightarrow} Q+P'}$$

$$(7)\quad \frac{P \stackrel{\nu}{\longrightarrow} P' \neq \underline{0} \;\wedge\; Q \stackrel{\nu}{\longrightarrow} Q' \neq \underline{0}}{P+Q \stackrel{\nu}{\longrightarrow} P'+Q'}$$

$$(8)\quad \frac{P \stackrel{\nu}{\longrightarrow} P' \neq \underline{0}}{Q \circ P \stackrel{\nu}{\longrightarrow} Q \circ P'}$$

$$(9)\quad \frac{P \stackrel{\nu}{\longrightarrow} \underline{0}}{Q \circ P \stackrel{\nu}{\longrightarrow} Q}$$

Table 3: Possible *transformations*.

$$\frac{P \stackrel{m_1\overline{m_2}}{\longrightarrow} P}{\langle P, M \cup m_1 \rangle \longrightarrow \langle P, M \cup m_2 \rangle}$$

$$\frac{(\forall m_1 : P \stackrel{m_1\overline{m_2}}{\longrightarrow} P) : m_1 \not\subseteq M \quad\wedge\quad P \stackrel{\nu}{\longrightarrow} P'}{\langle P, M \rangle \longrightarrow \langle P', M \rangle}$$

Table 4: Alternative semantics definition using possible *transformations*.

the labeled transition system ($Program$,$Label$,$\longrightarrow$) can be used to obtain the operational semantics of Table 2.[3]

# 5 Observational Equivalences

In the previous section we have introduced a *labeled* transition system which describes the possible *transformations* that a program can generate: this system is used in this section to introduce some new observational equivalence relations.

## 5.1 Bisimulation equivalence

We introduce a semantic equivalence based on the concept of *bisimulation* :

> *The terms $P$ and $Q$ are equivalent if $Q$ is always able to reply with the same action to every one of $P$, and vice versa.*

**Definition 5.1** *Bisimulation equivalence* :
The *bisimulation equivalence* (denoted by $\sim$) is the maximum relation among the ones that satisfy the following condition:

$$
\begin{array}{lllllll}
P\mathcal{R}Q & \Longrightarrow & \forall \alpha \in Label : & & & & \\
 & (i) & P \xrightarrow{\alpha} P' & \Longrightarrow & \exists\, Q' & : & Q \xrightarrow{\alpha} Q' \ \wedge \ P'\mathcal{R}Q' \\
 & (ii) & Q \xrightarrow{\alpha} Q' & \Longrightarrow & \exists\, P' & : & P \xrightarrow{\alpha} P' \ \wedge \ P'\mathcal{R}Q' \qquad \blacksquare
\end{array}
$$

**Proposition 5.2** The bisimulation equivalence is a congruence:
$\forall\, P_1, P_2, Q_1, Q_2 \in Program \ : \ P_1 \sim Q_1 \ \wedge \ P_2 \sim Q_2 \ \Longrightarrow$

$$P_1 + P_2 \sim Q_1 + Q_2 \quad \wedge \quad P_1 \circ P_2 \sim Q_1 \circ Q_2 \qquad \blacksquare$$

The fact that bisimulation equivalence is a congruence allows to investigate an equational characterization for it. In Table 5 the *sound* set of axioms $\mathcal{A}$ is presented (it is sufficient to change in every axiom the symbol $=$ with $\sim$ to obtain a correct law). In the following $\mathcal{A} \vdash P = Q$ means that $P$ is proved equal to $Q$ with standard equational reasoning with the set of axioms $\mathcal{A}$.

Axiom (1) defines a sufficient condition for the equivalence of basic programs. Axioms (2) and (3) describe the commutative and associative property of the parallel composition operator. Axiom (4) represents the associative property of the sequential composition operator. Axiom (5) and (6) say that you the program $P$ can be composed in parallel with the program $nil$ or the

[3] The configuration $\langle \underline{0}, M\rangle$ has to be considered structural equivalent to the final configuration $M$ of Table 2.

$$
\begin{array}{lllll}
(1) & (R_1, A_1) & = & (R_2, A_2) & \\
 & & & \text{if } \mathcal{I}_R(R_1) = \mathcal{I}_R(R_2) \ \wedge \ \mathcal{I}_A(A_1) = \mathcal{I}_A(A_2) & \\
(2) & P + Q & = & Q + P & \\
(3) & P + (Q + R) & = & (P + Q) + R & \\
(4) & P \circ (Q \circ R) & = & (P \circ Q) \circ R & \\
(5) & P + nil & = & P & \\
(6) & P + P & = & P & \\
(7) & (R, A) + (Q \circ S) & = & ((R, A) + Q) \circ ((R, A) + S) & \\
(8) & (P \circ Q) + (S \circ T) & = & (P + S) \circ (Q + T) & \\
 & & & \text{if} \begin{cases} Q & = \ \sum_{i=1}^{n} (R_{Q_i}, A_{Q_i}) \\ T & = \ \sum_{j=1}^{m} (R_{T_j}, A_{T_j}) \end{cases} &
\end{array}
$$

Table 5: The *sound* set of axioms $\mathcal{A}$.

program $P$ without changing its semantics. Axioms (7) and (8) represent the basic properties that allows to transform a term in an equivalent *product-of-sum* form.

The axiomatization presented in Table 5 is not complete as the following example shows.

**Example 5.3** Consider the following programs:

$$
\begin{array}{lllll}
P & : & \{x\} \longrightarrow 1 & \Longleftarrow & (x = 2) \ \vee \ (x = 3) \\
Q & : & \{x\} \longrightarrow 1 & \Longleftarrow & x = 2 \\
S & : & \{x\} \longrightarrow 1 & \Longleftarrow & x = 3
\end{array}
$$

It is true that $P \sim Q + S$ but $\mathcal{A} \not\vdash P = Q + S$. ■

In order to have a complete axiomatization we extend the syntax of basic Gamma programs $(R, A)$. The extended syntax is obtained changing the meaning of *Actions*: *Actions* are no more functions but they become relations between multisets of *molecules*. Therefore it is possible to define basic programs $(R, A)$ able to generate different transformations on the same group of *molecules*. The transformation to apply is chosen non-deterministically. The extended syntax implies a new definition for the semantics functions $\mathcal{I}_R$ and $\mathcal{I}_A$.

**Definition 5.4** *Semantic functions for basic programs*:

$$\begin{array}{lcll} \mathcal{I}_R & : & Reactions & \longrightarrow \mathcal{F}(\mathcal{M}(Molecules), Boolean) \\ \mathcal{I}_A & : & Actions & \longrightarrow \mathcal{P}(\mathcal{M}(Molecules) \times \mathcal{M}(Molecules)) \end{array}$$

such that $\forall \ (R, A) \in Program$ :

$$\{m_1 \mid \exists \ (m_1, m_2) \in \mathcal{I}_A(A)\} \ = \ \{m \mid \mathcal{I}_R(R)(m) = true\}$$

where last property is necessary in order to combine basic programs composed in parallel in a single equivalent one as explained in the following. ■

The new definition of the semantic functions $\mathcal{I}_R$ and $\mathcal{I}_A$ implies the replacement of rule (1) of Table 3 with the following one:

$$(1) \qquad \frac{\mathcal{I}_R(R)(m_1) = true \ \wedge \ (m_1, m_2) \in \mathcal{I}_A(A)}{(R, A) \stackrel{m_1\overline{m_2}}{\longrightarrow} (R, A)}$$

We now define a new operator able to combine basic programs composed in parallel in a new equivalent one.

**Definition 5.5** *Programs union*: $\uplus$

$(R_1, A_1) \uplus (R_2, A_2) = (R, A)$

$$\text{where : } \begin{cases} \mathcal{I}_R(R)(m) = true \Leftrightarrow (\mathcal{I}_R(R_1)(m) = true) \vee (\mathcal{I}_R(R_2)(m) = true) \\ \mathcal{I}_A(A) = \mathcal{I}_A(A_1) \cup \mathcal{I}_A(A_2) \quad \blacksquare \end{cases}$$

The basic property related to the *programs union* operator is that the *union* of two basic programs is equivalent to the parallel composition of the same programs as the new axiom (9) explains:

$$(9) \qquad (R_1, A_1) + (R_2, A_2) = (R_1, A_1) \uplus (R_2, A_2)$$

The correctness of axiom (9) is ensured by the property introduced in Definition 5.4:

$$\forall \ (R, A) : \ \{m_1 \mid \exists \ (m_1, m_2) \in \mathcal{I}_A(A)\} \ = \ \{m \mid \mathcal{I}_R(R)(m) = true\}$$

In fact, if the domain of $\mathcal{I}_A(A)$ is not equal to $\{m \mid \mathcal{I}_R(R)(m) = true\}$, then new possible *transformations* could be introduced by the *programs union* operator. For example, if $(m_1, m_2) \in \mathcal{I}_A(A_1) \wedge \mathcal{I}_R(R_1)(m_1) = false$ and $(m_1, m_2) \notin \mathcal{I}_A(A_2) \wedge \mathcal{I}_R(R_2)(m_1) = true$, then the transition $\stackrel{m_1\overline{m_2}}{\longrightarrow}$ is generable by $(R_1, A_1) \uplus (R_2, A_2)$ but not by $(R_1, A_1) + (R_2, A_2)$. Adding axiom (9) to the set of axioms $\mathcal{A}$, a *sound* and *complete* set of axioms is obtained. With the new axiom (9), axioms (5) and (6) become unnecessary. In fact, it is possible to prove the correctness of the equivalences $P + nil \sim P$ and $P + P \sim P$ without using them. Moreover axiom (8) can be simplified. The new set of axioms is referred by $\mathcal{B}$ and it is presented in Table 6.

$$
\begin{array}{lllll}
(1) & (R_1, A_1) & = & (R_2, A_2) \\
 & & & \text{if } \ \mathcal{I}_R(R_1) = \mathcal{I}_R(R_2) \ \wedge \ \ \mathcal{I}_A(A_1) = \mathcal{I}_A(A_2) \\
(2) & P + Q & = & Q + P \\
(3) & P + (Q + R) & = & (P + Q) + R \\
(4) & P \circ (Q \circ R) & = & (P \circ Q) \circ R \\
(7) & (R, A) + (Q \circ S) & = & ((R, A) + Q) \circ ((R, A) + S) \\
(8') & (P \circ Q) + (S \circ T) & = & (P + S) \circ (Q + T) \\
 & & & \text{if} \begin{cases} Q & = & (R_Q, A_Q) \\ S & = & (R_T, A_T) \end{cases} \\
(9) & (R_1, A_1) + (R_2, A_2) & = & (R_1, A_1) \uplus (R_2, A_2)
\end{array}
$$

Table 6: The *sound* and *complete* set of axioms $\mathcal{B}$.

**Proposition 5.6** *Soundness* and *completeness* of $\mathcal{B}$:
$\forall \ P, Q \in \mathit{Program} : \qquad P \sim Q \iff \mathcal{B} \vdash P = Q$ ■

## 5.2 Jumping bisimulation equivalence

In the first part of this section an observational congruence for CGP has been introduced. The following example shows that the introduced *bisimulation equivalence* is not abstract enough to identify processes which are different only for their computational speed (the number of computation steps executed in order to obtain a certain transformation).

**Example 5.7** Consider the programs:

$$
\begin{array}{lllll}
R & : & \{x\} \longrightarrow \{2\} & \Longleftarrow & x = 1 \\
S & : & \{x\} \longrightarrow \{3\} & \Longleftarrow & x = 2 \\
T & : & \{x\} \longrightarrow \{3\} & \Longleftarrow & x = 1 \\
P & : & R + S & & \\
Q & : & R + S + T & &
\end{array}
$$

The agents $P$ and $Q$ are not bisimilar (i.e. $P \not\sim Q$) even though they can be considered observationally undistinguishable. In fact, if the computational speed is not observable (i.e. it is not possible to know the number of transformation steps that have been executed in order to obtain a certain transformation) an external observer is not able to distinguish between the behavior of the agents $P$ and $Q$. ■

In order to solve this problem we adapt to CGP the *jumping bisimulation equivalence* [DNM93]. This observational semantics is based on the concept of *observable transformations*.

**Definition 5.8** *Observable transformation*:
The *observable transformations* are denoted by the labeled transition $P \stackrel{\alpha}{\twoheadrightarrow} P'$ and they are defined by the following two rules:

$$\frac{P \stackrel{\alpha}{\longrightarrow} P'}{P \stackrel{\alpha}{\twoheadrightarrow} P'}$$

$$\frac{P \stackrel{m_1\overline{m_2}}{\longrightarrow} P \;\wedge\; P \stackrel{m_3\overline{m_4}}{\twoheadrightarrow} P}{P \stackrel{n_1\overline{n_2}}{\twoheadrightarrow} P}$$

where $\begin{cases} n_1 = m_1 \cup (m_3 \setminus m_2) \\ n_2 = m_4 \cup (m_2 \setminus m_3) \end{cases}$ ■

Observable transformations are used to see as a single step of computation the result of successive transformation steps.

**Definition 5.9** *Jumping bisimulation equivalence* :
The *jumping bisimulation equivalence* (denoted by $\approx$) is the maximum relation among the ones that satisfy the following condition:

$$\begin{array}{lllllllll} P\mathcal{R}Q & \Longrightarrow & \multicolumn{7}{l}{\forall \alpha \in Label:} \\ & (i) & P \stackrel{\alpha}{\longrightarrow} P' & \Longrightarrow & \exists\, Q' & : & Q \stackrel{\alpha}{\twoheadrightarrow} Q' & \wedge & P'\mathcal{R}Q' \\ & (ii) & Q \stackrel{\alpha}{\longrightarrow} Q' & \Longrightarrow & \exists\, P' & : & P \stackrel{\alpha}{\twoheadrightarrow} P' & \wedge & P'\mathcal{R}Q' \end{array}$$ ■

The bisimulation equivalence $\sim$ is a subset of the *jumping bisimulation equivalence*: in fact, the bisimulation equivalence satisfies the condition of Definition 5.9. The inverse relation it is not true, in fact the terms $P$ and $Q$ of Example 5.7 which are not bisimilar are identified by the *jumping bisimulation equivalence*. A first important property of $\approx$ is that it is a congruence.

**Proposition 5.10** The *jumping bisimulation equivalence* is a congruence:
$\forall\, P_1, P_2, Q_1, Q_2 \in Program: \;\; P_1 \approx Q_1 \;\wedge\; P_2 \approx Q_2 \Longrightarrow$

$$P_1 + P_2 \approx Q_1 + Q_2 \quad \wedge \quad P_1 \circ P_2 \approx Q_1 \circ Q_2$$ ■

As already stated, the jumping semantics is more abstract than the bisimulation one (i.e. $\sim \subset \approx$). This allows to say that the set of axioms $\mathcal{B}$ is

$$
\begin{array}{ll}
(10) & (R, A) = (R, A) \uplus (R', A') \\
\text{where:} & \left\{ \begin{array}{l} \mathcal{I}_R(R')(m) = true \overset{def}{\Longleftrightarrow} \exists\ m_1, m_2, m_3, m_4 : \\ ((m_1, m_2) \in \mathcal{I}_A(A) \wedge (m_3, m_4) \in \mathcal{I}_A(A)) \wedge \\ m = m_1 \cup (m_3 \setminus m_2) \\ \\ (m, n) \in \mathcal{I}_A(A') \overset{def}{\Longleftrightarrow} \exists\ m_1, m_2, m_3, m_4 : \\ ((m_1, m_2) \in \mathcal{I}_A(A) \wedge (m_3, m_4) \in \mathcal{I}_A(A)) \wedge \\ m = m_1 \cup (m_3 \setminus m_2)\ \wedge\ n = m_4 \cup (m_2 \setminus m_3) \end{array} \right.
\end{array}
$$

Table 7: Axiom (10) for the *jumping bisimulation equivalence*.

*sound* for the *jumping bisimulation equivalence* as well. Our idea for defining a complete axiomatization for $\approx$ is to add to the set of axioms $\mathcal{B}$ a new axiom presented in Table 7. The meaning of axiom (10) is that the *jumping bisimulation* semantics of a basic program is not changed by composing in parallel to it the program which is able to generate with only one step of computation the transformations which can be performed in two separated steps. The set of axioms $\mathcal{C} = \mathcal{B} \cup \{(10)\}$ is a *sound* and *complete* axiomatization for the *jumping bisimulation equivalence* as the following proposition explains.

**Proposition 5.11** *Soundness* and *completeness* of $\mathcal{C}$:
$\forall\ P, Q \in Program : \qquad P \approx Q \iff \mathcal{C} \vdash P = Q$ ■

# 6 Conclusions

In this paper we have highlighted some compositional problems for the Calculus of Gamma Programs. We have introduced an alternative semantics for the parallel composition operator as a solution for this kind of problems. The alternative Calculus of Gamma Programs loses all those computations in which parallel programs are not reduced contemporarily: the structured programs composed in parallel are reduced all at the same time when no one of them is able to generate a *chemical reaction*. At every reduction step, all the programs lose the basic Gamma subprograms at the right (i.e. the subprograms active before the reduction). This property is referred as the *context sensitive reduction* property.

Moreover, we have introduced a new way to define the operational semantics of CGP. Using a *labeled* transition system we completely describe the possible *transformations* that a program can generate without taking into account the multiset which it is working on. In this new framework, formal verification of properties becomes more easy: a general property of a program can be proved without testing all the computations starting from every possible multiset. Moreover, the new framework allows to adapt to CGP classical observational equivalences such as bisimulation, testing or traces. We have investigated two of them: we have proved that they are congruences and a *sound* and *complete* axiomatization is presented for both of them.

**Acknowledgements** We thank David Sands who inspired us Example 5.7 that shows the limits of the standard bisimulation semantics for CGP. The anonymous referees are also thanked for their helpful suggestions.

## References

[BM93] J.P. Banâtre and D. Le Métayer. Programming by Multiset Transformation. *Communications of the ACM*, 36(1):98–111, January 1993.

[DNM93] P. Degano, R. De Nicola, and U. Montanari. Universal axioms for bisimulation. *Theoretical Computer Science*, 114:63–91, 1993.

[HMS92] C. Hankin, D. Le Métayer, and D. Sands. A Calculus of Gamma Programs. Technical Report DOC 92/22, Department of Computing, Imperial College, 1992.

[HMS93] C. Hankin, D. Le Métayer, and D. Sands. *A parallel programming style and its algebra of programs*, volume 694 of *LNCS*, pages 367–378. Springer Verlag, 1993.

[Plo81] G. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, University of Aarhus, 1981.

[San93] D. Sands. A Compositional Semantics of Combining Forms for Gamma programs. In *Proc. of the International Conference on Formal Methods in Programming and Their Applications*, Novosibirsk, 1993. Springer Verlag.