Pipelining Transformation with a Multiset Logic

Francisco Hernández Quiroz

Department of Computer Science
IIMAS, UNAM
Ciudad Universitaria
D.F. 04510, MÉXICO
fhq@leibniz.iimas.unam.mx

Abstract. A common problem in refining programs in parallel languages is the pipelining transformation: the change of two sequentially composed programs into a version composed in parallel. [9] and [18] studied in depth this problem in the context of the Gamma programming language. The latter strengthened a series of requirements for a pipelining transformation to be safe. However, these requirements were stated in semi formal terms and therefore it is difficult to check their enforcement.
This issue can be re-stated in the multiset logic presented in [10, 11] and then a fully formal, (conditionally) decidable logic can be used to verify them.

1 Gamma Language

The Gamma language was proposed in [2] as a device for systematic program derivation in the spirit of a discipline of programming [7]. Interest in the language grew steadily and its theoretical aspects have been studied extensively. Gamma has also been used as a real programming language, and some of its applications are scheduling [3], image processing [6, 14] and distributed systems [15].

As their creators pointed out, one of Gamma's strengths is its ability to eliminate any artificial sequentiallity in a program. Additionally, solutions are not biased by the choice of a special (and many times arbitrary) data structure, as the multiset is the only one available in Gamma and, thanks to its lack of internal structure, it does not impose a particular way of thinking. Also, it is a very good formalism to express coordination between components of a given program and then complex relations can easily be stated and tested. In other words, Gamma use emphasizes correctness rather than efficency.

Let D be a given type and let $\mathbb{M}(D)$ be the set of finite multisets in D. Special curly braces denote multisets: $\{\!|x_1, \ldots, x_n|\!\}$. The syntax of Gamma programs is:

$$P ::= (x_1, \ldots, x_n) \rightarrow A(x_1, \ldots, x_n) \Leftarrow R(x_1, \ldots, x_n) \mid P \circ P \mid P + P,$$

where the first construct is an *atomic reaction* (also called *rewriting rule*). In this case, R is a predicate with n arguments and $A : D^n \rightarrow \mathbb{M}(D)$ is an action.

The effect of an atomic reaction in a multiset M is to take out a tuple satisfying R and replace it with the result of A applied to the same tuple. If there is not such a tuple in M then the multiset remains unchanged and the atomic reaction finishes.

$P_2 \circ P_1$ is the sequential composition of two programs, where P_2 is applied to a multiset M if and only if P_1 cannot react with M any longer. $P_1 + P_2$ is the parallel composition, where any of P_1 or P_2 can react with a multiset at a given time. To terminate, both P_1 and P_2 should simultaneously be unable to react any longer with the multiset. The set of all Gamma programs is $\mathbb{G}$.

To abbreviate programs, a reaction

$$(x_1, \ldots, x_n) \to A^n(x_1, \ldots, x_n) \Leftarrow R^n(x_1, \ldots, x_n)$$

can also be written as $\bar{x} \to A(\bar{x}) \Leftarrow R(\bar{x})$ (implying that R, A and $\bar{x}$ have the same cardinality) or even as $A \Leftarrow R$, when the context makes its meaning clear.

The semantics of Gamma programs is defined by the following structural operational semantics rules, where $\langle P, M\rangle$ is a pair made of the program P and the multiset M, to which the program is applied:

$$\frac{\{\!|a_1, \ldots, a_n|\!\} \subseteq M, \quad R(a_1, \ldots, a_n)}{\langle (A \Leftarrow R), M\rangle \to \langle (A \Leftarrow R), (M - \{\!|a_1, \ldots, a_n|\!\}) \uplus A(a_1, \ldots, a_n)\rangle},$$

$$\frac{\neg\exists \{\!|a_1, \ldots, a_n|\!\} \subseteq M.R(a_1, \ldots, a_n)}{\langle (A \Leftarrow R), M\rangle \to M},$$

$$\frac{\langle Q, M\rangle \to M}{\langle P \circ Q, M\rangle \to \langle P, M\rangle}, \quad \frac{\langle Q, M\rangle \to \langle Q', M'\rangle}{\langle P \circ Q, M\rangle \to \langle P \circ Q', M'\rangle},$$

$$\frac{\langle P, M\rangle \to \langle P', M'\rangle}{\langle P + Q, M\rangle \to \langle P' + Q, M'\rangle}, \quad \frac{\langle Q, M\rangle \to \langle Q', M'\rangle}{\langle P + Q, M\rangle \to \langle P + Q', M'\rangle},$$

$$\frac{\langle P, M\rangle \to M \quad \langle Q, M\rangle \to M}{\langle P + Q, M\rangle \to M}.$$

As usual, $\to^*$ is the transitive closure of $\to$.

2 The Pipelining Transformation

In many instances the task which a program has to perform can be decomposed into a sequence of subtasks. Then, a program for each subtask is designed and the total effect is achieved by the sequential composition of all the subprograms. However, in many cases a subprogram does not depend on the actions performed by its predecessor and then there is no need to execute the programs in sequence. Given a suitable implementation and a computer architecture, parallel composition may be more efficient. But then the question arises of when the transformation of a sequentially composed program into a parallel composed version

maintains the desired properties of the original program, that is, it is a correct refinement. [9] analyzed this problem in the context of Gamma programming and proposed conditions that would guarantee the correctness of the transformation. Weichert [18] proved that those conditions needed to be strengthened for the transformation to preserve some refinement properties. In the following we will summarize his proposal.

3 Weichert's Approach

Let us begin with the definition of an order for Gamma programs that is based on state-program pairs:

Definition 1. *A relation* $R \subseteq (\mathbb{G} \times \mathbb{M}(D)) \times (\mathbb{G} \times \mathbb{M}(D))$ *is called a state-based simulation if for all* P_1 *and* $P_2 \in \mathbb{G}$ *and* M, $N \in \mathbb{M}(D)$, *we have that* $(\langle P_1, M\rangle, \langle P_2, N\rangle) \in R$ *if and only if:*

1. $M = N$;
2. *if* $\langle P_1, M\rangle \rightarrow \langle P_1', M'\rangle$ *then there are* P_2', N' *such that* $\langle P_2, N\rangle \rightarrow^* \langle P_2', N'\rangle$ *and* $(\langle P_1', M'\rangle, \langle P_2', N'\rangle) \in R$;
3. *if* $\langle P_1, M\rangle \rightarrow M$ *then there are* P_2', N' *such that* $\langle P_2, N\rangle \rightarrow^* \langle P_2', N'\rangle$, $(\langle P_1, M\rangle, \langle P_2', N'\rangle) \in R$ *and* $\langle P_2', N'\rangle \rightarrow N'$.

Now $P_1 \sqsubseteq_{SB} P_2$ *if and only if there is a state-based simulation* R *such that* $(\langle P_1, M\rangle, \langle P_2, M\rangle) \in R$, *for all* $M \in \mathbb{M}(D)$. *The corresponding congruence relation is* $\equiv_{SB}$.

Additionally, $P_1 \sqsubseteq_{IO} P_2$ *if and only if* $\langle P_1, M\rangle \rightarrow^* N$ *implies* $\langle P_2, M\rangle \rightarrow^* N$. *Finally,* $\equiv_{SB}$ *and* $\equiv_{IO}$ *are the congruence relations of* $\sqsubseteq_{SB}$ *and* $\sqsubseteq_{IO}$.

Consider the following Gamma programs:

$$P : (x_1, \ldots, x_n) \rightarrow \{\!|f_1(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n)|\!\} \Leftarrow \Diamond\{\!|\phi_1, \ldots, \phi_n|\!\}$$
$$C : (y_1, \ldots, y_p) \rightarrow \{\!|g_1(y_1, \ldots, y_p), \ldots, g_k(y_1, \ldots, y_p)|\!\} \Leftarrow \Diamond\{\!|\psi_1, \ldots, \psi_p|\!\},$$

both of them applied to multisets in $\mathbb{M}(D)$.

Let us call P the producer program and C the consumer. When is it possible to transform $C \circ P$ into $P + C$? It depends partially on the notion of correct refinement we take (that is, which the program orders $\sqsubseteq_{SB}$ and $\sqsubseteq_{IO}$ is selected) but also and above all it depends on certain conditions both P and C must meet.

When producer and consumer are executed in sequence, we know that the output of the producer is going to be the input of the consumer. Parallel execution opens the possibility that:

- the consumer could use up elements "intended" for the producer (which will later be unable to use them with potentially important consequences); and

– the producer can use up the result of the consumer actions, again interfering in the original sequence of events with a possibly undesirable outcome.

In order to avoid these undesired effects, [18] set the following conditions:

(∗) the possible inputs of the consumer and the producer should be disjoint;

(∗∗) the possible input of the producer and the output of the consumer should also be disjoint.

When (∗) is met, we know that the consumer cannot disable the producer unintentionally. When (∗∗), we know that the consumer cannot re-enable the producer when the latter has used up all of its possible input. The conditions (∗) and (∗∗) together guarantee the *independence* of consumer and producer.

Now, if we take $\sqsubseteq_{SB}$ as the order against which we will test the correctness of a refinement step, [18] proved that (∗∗) implies:

$$C \circ P \sqsubseteq_{SB} P + C.$$

On the other hand, if we consider $\sqsubseteq_{IO}$ he also proved that (∗) and (∗∗) together imply:

$$C \circ P \equiv_{IO} P + C.$$

The first result can be generalized to arbitrary Gamma programs and not only atomic reactions acting as consumer and producer (as in our example), and the second one to parallel composed simple programs (further conditions are needed for sequentially composed programs). The generalizations are straightforward once we have a means of proving non-interference for atomic reactions. The problem is: how can we prove (∗) and (∗∗) for atomic reactions? [18] assumed implicitly there is a way of establishing both conditions, without going any further in that direction. Fortunately, the multiset logic of [10, 11] gives us such a method.

4 Verifying Transformations with a Multiset Logic

We present now the syntax and semantics of the multiset logic $L(\mathbb{M}(D))$ (for a full presentation see [10, 11]). This a logic is *geometric* [17], i.e. it has finite conjunctions and arbitrary disjunctions of formulae. Afterwards, we will restate the pipelining verification problem in the new terms.

4.1 Multiset Logic

Assume that D is a basic type in Gamma and that $L(D)$ is a language of assertions about elements in D. We will also have a proof system $\mathcal{L}(D)$ with axioms and inference rules such that it is possible to prove statements of the form

$$x \models \phi,$$

where $x \in D$ and $\phi \in L(D)$ and $\models$ is a satisfaction relation. t and f represent true and false in $L(D)$. For every element $x \in D$ we have:

$$x \models t \qquad \text{and} \qquad x \not\models f$$

Definition 2. *The language of $L(\mathcal{M}(D))$ is made of atomic and complex propositions. Atomic propositions are built in this way:*

$$\frac{\phi_1, \ldots, \phi_n \in L(D)}{\Box\{\!|\phi_1, \ldots, \phi_n|\!\} \in L(\mathcal{M}(D))},$$

where the order of the ϕ_i's is not relevant. Complex propositions can be built by finite conjunctions and arbitrary *disjunctions:*

$$\frac{\Phi, \Psi \in L(\mathcal{M}(D))}{\Phi \wedge \Psi \in L(\mathcal{M}(D))} \qquad \frac{\{\Phi_i\}_{i \in I} \subseteq L(\mathcal{M}(D)),}{\bigvee_{i \in I} \Phi_i \in L(\mathcal{M}(D))}.$$

True and false are defined as

$$t = \bigwedge \emptyset \qquad f = \bigvee \emptyset.$$

We will also use the following shorthands:

$$\text{if} \quad \phi \in L(D) \quad \text{then} \quad \{\!|\phi|\!\}^n \equiv_{def} \{\!|\underbrace{\phi, \ldots, \phi}_{n \text{ times}}|\!\}$$

$$\Diamond\{\!|\phi_1, \ldots, \phi_n|\!\} \equiv_{def} \bigvee_{m \in \mathbb{N}} \Box(\{\!|\phi_1, \ldots, \phi_n|\!\} \uplus \{\!|t|\!\}^m).$$

We will apply the convention that propositions in $L(D)$ are denoted by lower case Greek letters, while upper case Greek letters live in $L(\mathcal{M}(D))$.

On the semantic side, a satisfaction relation between multisets and formulae of $L(\mathbb{M}(D))$ is explained in the following:

Definition 3. *If we have that $\{\!|x_1, \ldots, x_n|\!\} \in \mathbb{M}(D)$ and $\phi_1, \ldots, \phi_n \in L(D)$, then $\{\!|x_1, \ldots, x_n|\!\} \models \Box\{\!|\phi_1, \ldots, \phi_n|\!\}$ if and only if there exists a permutation σ such that*

$$x_{\sigma(1)} \models \phi_1, \ldots, x_{\sigma(n)} \models \phi_n.$$

On the other hand, $\{\!|x_1, \ldots, x_n|\!\} \models \Phi \wedge \Psi$ if and only if

$$\{\!|x_1, \ldots, x_n|\!\} \models \Phi \quad \textit{and} \quad \{\!|x_1, \ldots, x_n|\!\} \models \Psi.$$

If $\{\Phi\}_{i \in I} \subseteq L(\mathcal{M}(D))$ then $\{\!|x_1, \ldots, x_n|\!\} \models \bigvee_{i \in I}\{\Phi_i\}$ if and only if

$$\{\!|x_1, \ldots, x_n|\!\} \models \Phi_j \quad \textit{for at least one} \quad \Phi_j \in \{\Phi\}_{i \in I}.$$

[10, 11] proved that the multiset logic $L(\mathbb{M}(D))$ (together with its axioms and rules of inference) is sound and complete, provided the logic $\mathcal{L}(D)$ is so. Additionally, through the equivalence of satisfaction tests in our multiset logic with the query language for bags discussed in [13], these tests are also decidable (perhaps even with a polynomial-time algorithm!), provided satisfaction is also decidable in the logic $\mathcal{L}(D)$, again.

4.2 Pipelining and Multiset Logic

Coming back to the example of programs P and C, disjointness of input would mean that the reaction conditions of the two programs cannot be satisfied simultaneously by any multiset $\{\!|x_1, \ldots, x_q|\!\}$, where $q = \max\{n, p\}$. Disjointness of the output of C and the input of P can be regarded as the inability of the result of the action in C to meet the reaction condition in P. Consider the following multiset of propositions:

$$\{\!|z \in g_1(D^p), \ldots, z \in g_k(D^p)|\!\},$$

i.e., each of the propositions in the multiset states that a certain element belongs to the image of the function g_i when applied to the whole of its domain. Then we have the following translations of $(*)$ and $(**)$:

$(*')$ there exists no M such that $M \models (\Diamond\{\!|\phi_1, \ldots, \phi_n|\!\} \wedge \Box\{\!|\psi_1, \ldots, \psi_p|\!\}) \vee (\Box\{\!|\phi_1, \ldots, \phi_n|\!\} \wedge \Diamond\{\!|\psi_1, \ldots, \psi_p|\!\})$;

$(**')$ there exists no M such that $M \models (\Diamond\{\!|\phi_1, \ldots, \phi_n|\!\} \wedge \Box\{\!|z \in g_1(D^p), \ldots, z \in g_k(D^p)|\!\}) \vee (\Box\{\!|\phi_1, \ldots, \phi_n|\!\} \wedge \Diamond\{\!|z \in g_1(D^p), \ldots, z \in g_k(D^p)|\!\})$.

Assuming that the logic of the elements in D is complete, the test of $(*')$ and $(**')$ can become a routine task. But they only apply to atomic rules. Suppose now that

$$P = P_1 + \cdots P_q \qquad \text{and} \qquad C = C_1 + \cdots C_r,$$

where

$$\begin{aligned} P_i = (\ x_1, \ldots, x_{n_i}) &\to \{\!|f_1^i(x_1, \ldots, x_{n_i}), \ldots, f_{m_i}^i(x_1, \ldots, x_{n_i})|\!\} \\ &\Leftarrow \Diamond\{\!|\phi_1^i, \ldots, \phi_{n_i}^i|\!\} \\ C_i = (\ \ y_1, \ldots, y_{p_i}) &\to \{\!|g_1^i(y_1, \ldots, y_{p_i}), \ldots, g_{k_i}^i(y_1, \ldots, y_{p_i})|\!\} \\ &\Leftarrow \Diamond\{\!|\psi_1^i, \ldots, \psi_{p_i}^i|\!\} \end{aligned}$$

Then the conditions $(*')$ and $(**')$ become:

$(*)_+$ There exists no M such that

$$\begin{aligned} M \models (\ &(\textstyle\bigvee_{i=1}^q \Diamond\{\!|\phi_1^i, \ldots, \phi_{n^i}^i|\!\}) \wedge (\bigvee_{j=1}^r \Box\{\!|\psi_1^j, \ldots, \psi_{p^j}^j|\!\})) \\ &\vee \\ &((\textstyle\bigvee_{i=1}^q \Box\{\!|\phi_1^i, \ldots, \phi_{n^i}^i|\!\}) \wedge (\bigvee_{j=1}^r \Diamond\{\!|\psi_1^j, \ldots, \psi_{p^j}^j|\!\})) \end{aligned}$$

$(**)_+$ There exists no M such that

$$\begin{aligned} M \models (\ &\textstyle\bigvee_{i=1}^q \Diamond\{\!|\phi_1^i, \ldots, \phi_{n^i}^i|\!\} \wedge \bigvee_{j=1}^r \Box\{\!|z \in g_1^j(D^{p^j}), \ldots, z \in g_{k_j}^j(D^{p^j})|\!\}) \\ &\vee \\ &(\ \textstyle\bigvee_{i=1}^q \Box\{\!|\phi_1^i, \ldots, \phi_{n^i}^i|\!\} \wedge \bigvee_{j=1}^r \Diamond\{\!|z \in g_1^j(D^{p^j}), \ldots, z \in g_{k_j}^j(D^{p^j})|\!\}) \end{aligned}$$

Now suppose that $P = P_q \circ \cdots \circ P_1$ and $C = C_r \circ \cdots C_1$. This time the conditions are

$(*)_\circ$ The same as in $(*)_+$.

$(**)_\circ$ There exist no M such that

$$M \models (\bigvee_{i=1}^{q} \Diamond\{\!|\phi_1^i, \ldots, \phi_{n_i}^i|\!\} \wedge \Box\{\!|z \in \bigcup_{j=1}^{k_{r-1}} (\bigcup_{l=1}^{k_{r-2}} \ldots (\bigcup_{s=1}^{k_1} g_1^r \circ g_j^{r-1} \circ \cdots \circ g_s^1(D^{p_1}))),$$
$$\ldots, z \in \bigcup_{i_{r-1}=1}^{k_{r-1}} (\bigcup_{i_{r-2}=1}^{k_{r-2}} \ldots (\bigcup_{i_1=1}^{k_1} g_{k_r}^r \circ g_{i_{r-1}}^{r-1} \circ \cdots \circ g_{i_1}^1(D^{p_1})))|\!\})$$
$$\vee$$
$$(\bigvee_{i=1}^{q} \Box\{\!|\phi_1^i, \ldots, \phi_{n_i}^i|\!\} \wedge \Diamond\{\!|z \in \bigcup_{j=1}^{k_{r-1}} (\bigcup_{l=1}^{k_{r-2}} \ldots (\bigcup_{s=1}^{k_1} g_1^r \circ g_j^{r-1} \circ \cdots \circ g_s^1(D^{p_1}))),$$
$$\ldots, z \in \bigcup_{i_{r-1}=1}^{k_{r-1}} (\bigcup_{i_{r-2}=1}^{k_{r-2}} \ldots (\bigcup_{i_1=1}^{k_1} g_{k_r}^r \circ g_{i_{r-1}}^{r-1} \circ \cdots \circ g_{i_1}^1(D^{p_1})))|\!\})$$

It looks quite complicated, but the intuitions behind these conditions are fairly simple. In both $(*)_+$ and $(*)_\circ$ we are checking that no multiset can meet simultaneously the reaction conditions of the consumers and producers (that is, that their input is disjoint). The condition $(**)_+$ takes the disjunction of the reaction conditions of the producer and checks that they cannot be met by a multiset which contains also elements that can be the output of the consumer. Finally, $(**)_\circ$ guarantees the same, but taking into account the sequential composition of the components of the consumer (that is, the image of the functions in the action of C_i are the input for the functions in the action of C_{i+1}).

5 A Small Application

It is time for an application now that the notation has been explained. Due to space limitations, we will present a very simple example just to illustrate the way the method works. [4] offered this example, also analyzed by [18]:

$$\begin{aligned} max &: x, y \to \{\!|y|\!\} \Leftarrow \Diamond\{\!|0 \leq x < y|\!\} \\ one &: x \to \{\!|1|\!\} \Leftarrow \Diamond\{\!|0 \leq x \neq 1|\!\} \\ add &: m, n \to \{\!|m+n|\!\} \Leftarrow \Diamond\{\!|t, t|\!\} \\ abs &: x \to -x \Leftarrow \Diamond\{\!|x < 0|\!\}. \end{aligned}$$

Weichert proved (in a long semi-formal proof) that

$$(add \circ one \circ \max) \circ abs \sqsubseteq_{SB} (add \circ one \circ \max) + abs,$$

is true, but, on the other hand,

$$(add \circ one \circ \max) + abs \sqsubseteq_{IO} (add \circ one \circ \max) \circ abs$$

does not hold. To verify his result, we need to prove $(**)$ and disprove $(*)$. Let us begin with $(*)$. There is a M such that

$$M \models (\Diamond\{\!|x < 0|\!\} \wedge \Box\{\!|t, t|\!\}) \vee (\Box\{\!|x < 0|\!\} \wedge \Diamond\{\!|t, t|\!\}),$$

namely, $M = \{\!|-1, 2|\!\}$, which invalidates $(*)$. Regarding $(**)$, we need to prove that no multiset satisfies the following proposition:

$$(\Diamond\{\!|x < 0|\!\} \wedge \Box\{\!|x \in \mathbb{Z}^+|\!\}) \vee (\Box\{\!|x < 0|\!\} \wedge \Diamond\{\!|x \in \mathbb{Z}^+|\!\}),$$

as $(x, y \to \max\{x, y\}) : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$, $(x \to 1) : \mathbb{Z} \to \{1\}$ and $(m, n \to m + n) : \{1\} \times \{1\} \to \mathbb{Z}^+$, for this last action is applied only to multisets in $\mathbb{M}(\{1\})$. Such a multiset would need to have an element x satisfying simultaneously the propositions $x < 0$ and $x \in \mathbb{Z}^+$, which is impossible.

In conclusion, both Weichert's claims follow also from our alternative rules based on multiset logic.

6 Conclusions and further work

We claim that our approach to verifying pipelining transformations in Gamma compares quite favourably with previous ones:

- It introduces a fully formal language, tailor-made to fit Gamma and its only data structure and (together with Weichert's conditions) can account for a proof of correctness with respect to the pipelining transformation of programs.
- It is also more general as it is not constrained to simple (i.e. parallelly composed) Gamma programs.
- Thirdly, and thanks to the equivalence between satisfaction tests in multiset logic and the query language of [13], it is a (conditionally) decidable and complete method, potentially automatic in polynomial time. See [10] for a proof of this claim.

An obvious extension of this work springs to mind: building an automatic or semiautomatic tool for verifying transformations. This would be a part of a bigger automatic proof system for Gamma programs.

Additionally, our method is being applied to the verification of a coherence protocol for bibliographic databases. As this algorithm is expressed in structured Gamma (see [8]) some adaptations will have to be made.

7 Acknowledgments

I wish to thank to the anonymous referees for their comments on the first version of the paper. Although I tried to follow their advice, there is no need to say that all mistakes remaining are entirely my responsability.

References

1. Andreoli, J.-M., Hankin, C., Le Métayer, D., *Coordination Programming: Mechanisms, Models and Semantics*, Imperial College Press, 1996.
2. Banâtre, J.-P., Le Métayer, D., "The GAMMA Model and its Discipline of Programming", *Science of Computer Programming*, 15:55–77, 1990.
3. Bourgois, M., "Advantage of Formal Specifications: a Case Study of Replication in Lotus Notes", in [16].
4. Ciancarini, P., Gorrieri, R., Zavattaro, G., "An Alternative Semantics for the Parallel Operator of the Calculus of Gamma Programs", in [1].
5. Ciancarini, P., Wolf, A.L., *Coordination Languages and Models*, 3rd International Conference Coordination '99, Amsterdam, The Netherlands, April 1999, Lecture Notes in Computer Science 1594, Springer, 1999.
6. Creveuil, C., Moguérou, G., "Développment systématique d'un algorithme de segmentation d'images à l'aide de Gamma", *Techniques et Science Informatiques*, 10(2):125-137, 1991.
7. Dijkstra, E.W., *A Discipline of Programming*, Prentice-Hall, 1976.
8. Fradet, P., Le Métayer, D., *Structured Gamma*, Technical report PI-989, Irisa, March, 1996.
9. Hankin, C., Le Métayer, D., Sands, D., "Refining Multiset Transformers", *Theoretical Computer Science*, 192:233–258, 1998.
10. Hernández Quiroz, F., *Semantics-Based Proof System for Gamma*, PhD thesis, Imperial College of Science, Technology and Medicine, 1999.
11. Hernández Quiroz, F., "A proof system for multisets", submitted to the *Journal of Logic and Computation*, 2001.
12. IEEE, *Software Engineering for Parallel and Distributed Systems, 2000*, IEEE, 2000.
13. Libkin, L., Wong, L., "On Representation and Querying Incomplete Information in Databases with Multisets", *Information Processing Letters*, 56:209-214, November, 1995.
14. McEvoy, H., "Gamma, Chromatic Typing and Vegetation", in [1].
15. Mentré, D., Le Métayer, D., Priol, T., "Formalization and verification of coherence protocols with the Gamma framework", in [12].
16. Najm, E., Stefani, J.-B. *Formal Methods for Open Object-based Distributed Systems*, International Workshop 1996, Paris, Chapman & Hall, 1997.
17. Vickers, S., *Topology via Logic*, Cambridge Tracts in Theoretical Computer Science 5, Cambridge University Press, 1989.
18. Weichert, M., "Pipelining the Molecule Soup: a Plumber's Approach to Gamma", in [5].