University of London
Imperial College of Science, Technology and Medicine
Department of Computing

A Semantics-Based Proof System for Gamma

Francisco Hernández Quiroz

A thesis submitted for the degree of
Doctor of Philosophy of the University of London
and for the
Diploma of Imperial College

September 1999

Abstract

Gamma is a simple yet powerful parallel programming language whose only data structure is the multiset or bag. Gamma programs can be composed either sequentially or in parallel. The language has been applied to graph algorithms, to scheduling problems and to image processing, among other areas.

Mathematical correctness is especially important for a language whose main use is to test and prove algorithms. In order to have a mathematically sound language there are at least two issues to be considered: a semantic model and a program logic.

Some attempts have been made to give Gamma a denotational semantics. A new approach, partially based on S.D. Brookes model for shared variable languages is used to propose a fully abstract model. This work extends an earlier proposal of Sands which failed to be fully abstract.

A logic for proving properties of programs is also presented. This logic is built upon: a) the denotational semantics previously introduced and b) a multiset logic, developed specifically for this purpose. Correctness and (conditional) completeness of the logic are also proved. The resulting system can be used to prove termination of programs and satisfaction of properties (useful in testing programs against specifications) and is also compared with previous (less general) approaches.

Some examples of the use of the program logic are given, which are taken both from previous proof systems and from non-formal sources.

Finally, the thesis discusses the possibility of extending the proof system and the semantics to higher-order Gamma.

Contents

Chapter 1

Introduction

Nowadays, the parallel computational paradigm is as firmly established as the old sequential model. From hardware and software architecture to programming languages and applications, parallelism is a normal feature of the landscape. There has been an explosion of parallel languages (both for specification and programming) and extensive research on mathematical models for them.

Another topic whose legitimacy nobody questions any longer is program verification. Since Dijkstra's (1976) famous book, proofs of correctness in programming—as opposed to just trial and error techniques—is an aim serious programmers and computers scientists should reach for. Proof systems for programming languages are the ultimate tool for this task.

This study of the Gamma programming language covers both areas: a mathematical model and a proof system for a parallel language.

1.1 Gamma

Parallelism and sequentiallity are just two sides of a coin. The problem of *coordinating* components of a system embraces both sequential and parallel solutions. Additionally, coordination and computation are distinct, albeit related, issues. In many cases it is natural to divide a task into different subtasks, assigning them to separate components which perform specific computations, and then coordinate the interaction between the com-

ponents according to the particular nature of the problem: the result of the computation of a given component can be the input for another and then they require a sequential execution, while other computations are independent and can be done in parallel.

On the other hand, many solutions in computer science have tended to be constrained not by the nature of the problem itself, but by the arbitrary data structures chosen for dealing with the problem. Take the common problem of finding the maximum value of a given set (with possible repetitions). Many solutions rely on a specific representation of the set, usually an array or list, and the resulting algorithm is normally sequential, as this is the most "natural" way of searching through an array or list.

The Gamma programming language addresses explicitly these questions. Its very simple syntax distinguishes clearly between computational and coordination issues and it helps to handle better their specific nuances. Its only data structure, namely the multiset or bag, imposes the least constraints on the way programs may access data.

Banâtre & Le Métayer (1990) proposed the language as a device for systematic program derivation in the spirit of Dijkstra's (1976) discipline of programming. 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 (Bourgois (1997)) and image processing (Creveuil & Moguérou (1991) and McEvoy (1996)).

The inspiration for the design of Gamma is the *chemical reaction metaphor*: a collection of atomic values reacting freely in a given medium (a multiset in this case). The atomic values change the medium until no further reaction is possible. (See chapter 2 for a full description of the language.) Gamma is then related to the chemical abstract machine of Berry & Boudol (1992). It is also an example of the composed reduction systems of Sands (1996).

Some extensions to Gamma have been proposed too. Le Métayer (1994) introduced higher-order Gamma (see chapter 6) and Fradet & Le Métayer (1996) proposed structured Gamma.

1.2 Semantics

Programming languages, in the same way as languages in general, can be studied from different points of view. From a syntactic outlook, programming languages are fairly well understood and we will say no more about this facet. Semantics, in its turn, offers different perspectives. *Operational semantics* defines the meaning of terms in a programming language according to the changes in the state of the variables modified by that term. In *denotational semantics*, programs are regarded, roughly, as mathematical functions acting on values and producing values. Normally, these values are elements in a particular *domain* (again, see chapter 2 for a definition of domain). Denotational semantics is important because it provides a more mathematically tractable way of explaining a language. A denotational model of a language can also be the basis upon which to build a proof system for verification of programs.

Of course, when two or more points of view exist, the question of their total or partial equivalence arises. There are different notions of equivalence between operational and denotational semantics, among them those of *adequacy* and *full abstraction*. The latter is a fairly strong equivalence and has the advantage of facilitating the generalization of results found in the operational view to the denotational one, and vice versa. We say that a denotational semantics is fully abstract if, given any two programs P and Q, they are equivalent in terms of the operational semantics if and only if they are also equivalent in denotational terms.

A simple syntax does not necessarily mean a simple semantics in programming languages and Gamma is not the exception. There are at least three competing operational semantics for Gamma (see chapter 3). Nevertheless, one can safely say that there is a good understanding of this topic, up to the point of characterizing where the competing models differ. The same cannot be said about Gamma denotational semantics.

Using ideas advanced by Brookes (1992, 1985, 1993) in the context of a shared-variable programming language, Gay & Hankin (1996*b*) produced a denotational semantics for Gamma. Sands (1993*a*) made another proposal, with the additional virtue of being compositional, that is, the

denotation of a program could be derived from its syntax (ie, the way its components are connected). But the two models lacked full abstraction, making it difficult to go from operational to denotational reasoning.

1.3 Gamma and program verification

When reasoning about correctness of programs, parallelism is inherently more difficult than sequentiality. Two programs can behave well in isolation, that is, they will terminate and produce a correct output, but when they act in parallel they may block each other's actions (deadlock is an instance of this problem) or fail to terminate. A *proof system* or *program logic* for correctness is the most powerful tool to deal with these undesired possibilities.

Gamma poses special challenges to the design of a proof system. To begin with, the program logic has to face the particular problems of parallelism. Although some help can be drawn from past experiences with parallel languages, there are some specific issues we shall explain in chapter 3 and 4. Secondly, we need a logic to reason about multisets, as it is the only data structure in the language. Contrary to what one might think on a first approach, the definition of a multiset logic is not a trivial problem. Thirdly, a complete proof system should have inference rules to deal with all the normal problems of program verification: total and partial correctness, termination etc. Finally, we have to prove soundness and at least a certain form of completeness of the system (see section 2.3).

Denotational semantics and proof systems are not unrelated fields. Denotational semantics appeared as a better way of understanding the behaviour of programs, making it easier to assess their correctness. Abramsky (1991) made an explicit connection between the two subjects explaining how a proof system for a programming language can be derived from its denotational semantics. Abramsky called his method *domain theory in logical form*, because it is based in domain theory and its purpose is to build program logics.

The relevant literature contains many examples of partial proof systems

for Gamma. Banâtre & Le Métayer (1990) already offered a method for designing correct Gamma programs. Errington, Hankin & Jensen (1993) presented a more complete program logic, although its design made it very difficult to apply. Gay & Hankin (1996*a*) presented a simpler system, but some of its inference rules were unproven. Chaudron (1998) and Reynolds (1996) are other proposals. Unfortunately, none of them constituted a full and integrated proof system.

1.4 Aims and plan of the thesis

We have already hinted at the two main topics of the present work: a denotational semantics and a proof system for Gamma. In looking for a answer to these questions, some other topics arose: the theoretical foundations of a multiset logic, its relationship with work on multisets in the context of databases and, finally, the application of the newly developed concepts to the study of program transformation.

The latter aims inform the structure of the thesis. Chapter 2 covers background issues, while chapters 3, 4 and 5 contain all of the original results. The last chapter is devoted to conclusions and future work. Each chapter opens with a brief survey of the situation in the field and the specific contributions of the present thesis. References to past or related work are given too. Here is a more detailed description of the content of each chapter:

Although domain theory is a staple component of theoretical computer science, domain theory in logical form is not so widely known. Thus, we present a brief summary of the theory in chapter 2, preceded by a formal introduction to Gamma and some notational issues.

Chapter 3 is devoted to Gamma semantics. It opens with Gamma operational semantics and some related concepts. A new denotational semantics comes afterwards. The next subject is the proof of full abstraction of the denotational semantics. Some other results appear also at this point. The chapter concludes with a functional version of the semantics.

Chapter 4 starts with the *multiset logic*, with some illustrative examples.

The Gamma proof system is then built upon the concepts introduced in chapter 2. Soundness and (conditional) completeness of the proof system are the next results. As with the multiset logic, some examples of correctness of programs are presented.

In chapter 5, related issues are dealt with. In the first place, the multiset logic is analyzed from a more theoretical point of view. Next, some relationships between the logic and the bag-language of Libkin & Wong (1993) are established. Finally, there is an example of application of the multiset logic to program transformation analysis.

Apart from two presentations in workshops—Hernández Quiroz (1998, 1999)—, the main results have not appeared before.

1.5 Acknowledgements

In the first place, I'd like to mention Chris Hankin, whose dedication and support towards my work exceeded by far what one can normally expect from a supervisor. His advice and direction made my PhD a very fruitful experience. Secondly, I want to thank Steve Vickers—also at Imperial College—for the many discussions and comments. I would have never achieved some results without his guidance.

Prahladavaradan Sampath was always a valuable source of ideas and references. Carlos Duarte and Russell Harmer made useful comments on previous versions of my work. David Sands discussed with me a preliminary version of the Gamma semantics. Simon Gay, Mauricio Álvarez Manilla and Philipp Sünderhauf offered some pointers to relevant literature.

I also want to thank the people who made my studies abroad possible. I start with Felipe Bracho: his encouragement and help were fundamental. Olbeth Hansberg, Adolfo García and Elisa Viso also help me with practical issues.

The work for a PhD is not only an academic pursuit, but also a personal challenge. Therefore, I'd like to name the people who played an important role during this period. Ana Rosa Pérez, apart from being my mentor during my years in Philosophy, influenced my decision to come to Britain. Ena

Lastra, Arturo G. Yáñez and Concepción Abellán also contributed in that direction. The friendship of Eugenia O'Reilly, Ángela Cenarro, Lei Yu and Virginia Aguirre was an unmissable part of the experience.

Finally, I thank my parents for their caring support and my brother, to whom this thesis is dedicated.

The DGAPA of the UNAM (National University of Mexico) sponsored my PhD. The ESPRIT Working Group "Coordina" funded an academic trip to the conference Coordination '99 in Amsterdam as part of my research.

Chapter 2

Gamma, program logics and domain theory

The purpose of this chapter is to introduce some notational devices and background theory which will be used in the rest of the thesis:

- We will start by presenting multisets and some other assorted concepts.
- Secondly, there will be a formal introduction to the Gamma programming language, including some examples, operations and relations useful to understand the behaviour of the language.
- Thirdly, comes a brief summary of what a program logic looks like, to clarify the goals of chapter 4 (and indeed of most of the thesis).
- Finally, domain theory in logical form (DTLF for short) will be explained. This theory is the basis for our Gamma proof system.

The Gamma programming language was first introduced by Banâtre & Le Métayer (1990). In a further paper, Banâtre & Le Métayer (1993) gave some examples of problem solving with Gamma including graph and string problems. More applications have appeared afterwards: scheduling (Bourgois (1997)), image processing (Creveuil & Moguérou (1991) and McEvoy (1996)), among others.

The field of program logics has a very long history. Among the first attempts we have Hoare (1969) and his logic of partial correctness, Dijk-

stra (1976) and his discipline of programming and weakest preconditions. Regarding parallel programming languages, Brookes (1985), Chandy & Misra (1988) and Lamport (1994) are some of the numerous examples. Banâtre & Le Métayer (1990) themselves offered a simple proof system for specifying and developing Gamma programs, taking some ideas about multiset orders from Dershowitz & Manna (1979).

One of the explicit aims of DTLF is precisely to address program logics. Abramsky (1991) intended to unify denotational semantics (in its domain-theoretical version as developed by Scott (1982) and Plotkin (1981) and many others) and program logics. The Stone duality theorems (Stone (1936)) are the mathematical foundation of Abramsky's work.

This approach has proved very fruitful: Abramsky (1987) generated a logic for labelled transition systems which subsumed Hennessy & Milner's (1985) logic. Jensen's (1992) strictness logic is an application of DTLF to the abstract interpretation of programs.

Zhang (1991) explored DTLF in the context of SFP and stable domains and their corresponding domain logics, and it is a very complete presentation of the subject.

Gay & Hankin (1996*b*, 1996*a*) have built proof systems for Gamma based on DTLF methods. Their results were the inspiration to apply DTLF to my own work.

2.1 Multisets and their operations

In plain language, a multiset is a collection of elements with repetitions. More formally, a multiset of elements in D is a function $M : D \to \mathbb{N}$. We will only be interested in finite multisets, that is functions where there is only a finite number of elements in D with $M(x) > 0$. We shall also use the following notation for multisets:

$$\{\!|x_1, \ldots, x_n|\!\},$$

where the x_i's are not necessarily distinct. The set of finite multisets of elements in D will be denoted by $\mathbb{M}(D)$. If M and $N \in \mathbb{M}(D)$ the following

functions define their union, difference and intersection, respectively:

$$(M \uplus N)(x) = M(x) + N(x)$$
$$(M - N)(x) = \begin{cases} M(x) - N(x) & \text{if } M(x) \geq N(x) \\ 0 & \text{otherwise} \end{cases}$$
$$(M \uplus N)(x) = \min\{M(x), N(x)\}.$$

In a way analogous to set inclusion, $M \subseteq N$ if and only if $M(x) \leq N(x)$ for every $x \in D$.

A bijection $\sigma : \{1, \ldots, n\} \to \{1, \ldots, n\}$ is called a *permutation*. It is clear that

$$\{\!|x_1, \ldots, x_n|\!\} = \{\!|x_{\sigma(1)}, \ldots, x_{\sigma(n)}|\!\},$$

although the labels of the elements in the multiset have been changed.

The set of booleans is $\mathbb{B} = \{true, false\}$. Given an arbitrary set T, a predicate of cardinality n is a function $R^n : T^n \to \mathbb{B}$. The set of predicates of cardinality n is $\mathbb{R}^n$ and the set of all predicates is

$$\mathbb{R} = \bigcup_{n \in \mathbb{N}} \mathbb{R}^n.$$

We will also call predicates *reaction conditions*.

Let $f_1 : T^n \to T$, $f_2 : T^n \to T$, $\ldots$, $f_m : T^n \to T$ be functions. The function $A^n : T^n \to \mathbb{M}(T)$ defined by

$$A^n(x_1, \ldots, x_n) = \{\!|f_1(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n)|\!\}$$

is called an *action*. The notation $(x_1, \ldots, x_n) \to A^n(x_1, \ldots, x_n)$ is also common for denoting actions. $\mathbb{A}^n = \{A^n : T^n \to \mathbb{M}(T)\}$ is the set of actions of cardinality n. Again,

$$\mathbb{A} = \bigcup_{n \in \mathbb{N}} \mathbb{A}^n.$$

Reaction conditions and actions are the basic ingredients of Gamma.

2.2 The Gamma language

The intentions of the creators of Gamma are clearly visible in the language's simplicity: coordination between components rather than computation is the main issue.

Intuitively, Gamma programs are made of *atomic reactions* (also called *rewriting rules*) which take a multiset, check if a certain reaction condition is met and then transform the multiset according to the rules of an action. Atomic reactions can be combined sequentially or in parallel. More formally, the syntax of a Gamma program is:

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

where R^n is a predicate and A^n an action. The set of all Gamma programs is $\mathbb{G}$.

The effect of an atomic reaction in a multiset M is to take out a tuple satisfying R^n and replace it with the result of A^n 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. A more mathematically precise way of describing the behaviour of Gamma programs will be given in chapter 3 by means of an operational semantics.

To abbreviate programs, a reaction

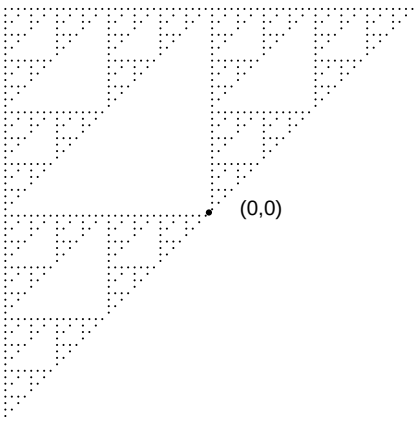
$$(x_1, \ldots, x_n) \to A^n(x_1, \ldots, x_n) \Leftarrow R^n(x_1, \ldots, x_n)$$

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 $A \Leftarrow R$.

A rewriting rule $A \Leftarrow R$ can be considered a function from multisets to sets of transformed multisets. To describe this view let us introduce yet another function $S : \mathbb{A}^n \times R^n \times \mathbb{M}(T) \to \mathcal{P}_{fin}(\mathbb{M}(T))$ defined as follows:

$$S(A^n, R^n, M) = \{N \mid N = (M - \{\!|x_1, \ldots, x_n|\!\}) \uplus A^n(x_1, \ldots, x_n)\}$$

where $\{\!|x_1, \ldots, x_n|\!\} \subseteq M$ and $R^n(x_1, \ldots, x_n)$ holds. According to this definition, when the rewriting rule cannot react with a multiset M (ie, it termi-



The Sierpinski triangle

nates) then $S(A^n, R^n, M) = \emptyset$. S can be generalized to arbitrary programs (of course, after changing its domain to $\mathbb{G}$):

$$S(P_2 \circ P_1, M) = \begin{cases} S(P_2, M) & \text{if } S(P_1, M) = \emptyset \\ S(P_1, M) & \text{otherwise} \end{cases}$$
$$S(P_1 + P_2, M) = S(P_1, M) \cup S(P_2, M).$$

To finish this section let us present some examples of Gamma programs:

2.2.1 *McEvoy (1996) offers a program for generating the Sierpinski triangle. The Sierpinski triangle is a fractal object (Barnsley (1993) gives a good introduction to the subject) and the program will produce a multiset with the points in the triangle up to k iterations. The elements in the multiset are tuples of the form (x, y, z), where x and y correspond to the coordinates of the point and z is the number of iterations which have produced that point:*

$$\begin{aligned} Sier = {} & ((x, y, z)) \rightarrow \\ & \{\!|(x/2 - 2/5, y/2 - 2/5, z + 1), (x/2 - 2/5, y/2 + 2/5, z + 1), \\ & (x/2 + 2/5, y/2 + 2/5, z + 1)|\!\} \\ & \Leftarrow z < k \end{aligned}$$

The execution of the program should start with the multiset $\{\!|(0, 0, 0)|\!\}$.

2.2.2 *A program for calculating the product of the two biggest elements in a multiset, taken from Reynolds (1996)'s paper. The program can be applied to any* $M \in \mathbb{M}(\mathbb{N})$.

$$\begin{aligned} Max &= (x, y, z) \to \{\!|x, y|\!\} \Leftarrow x \geq z \wedge y \geq z \\ Prod &= (x, y) \to \{\!|xy|\!\} \Leftarrow true \\ P &= Prod \circ Max. \end{aligned}$$

2.2.3 *A program to produce the n-th Fibonacci number when applied to the multiset* $\{\!|n|\!\}$. *This program was originally proposed by Hankin, Le Métayer & Sands (1993).*

$$\begin{aligned} Pred &= x \to \{\!|x-1, x-2|\!\} \Leftarrow (x > 1) \\ One &= x \to \{\!|1|\!\} \Leftarrow (x = 0) \\ Sum &= (x, y) \to \{\!|x + y|\!\} \Leftarrow true \\ Fib &= Sum \circ (Pred + One). \end{aligned}$$

The correctness of the three programs will be proved in chapter 4.

2.3 Program logics

In broad brush strokes a program logic consists of a language to make assertions about programs (which can be called specification) and axioms and rules to prove them (also known as verification). Hoare triples are expressions like $\{S\}P\{R\}$, where S and R are sets of assertions and P is a program. The meaning of the expression is: if S is true before the execution of P and P terminates, then R is true after the execution of P (Hoare (1969)).

In general, let us suppose we have a series of well-formed formulas $\mathcal{L}$, expressing properties of programs in a certain programming language $\mathcal{P}$. If $P \in \mathcal{P}$ and $\phi \in \mathcal{L}$, we want to prove statements of the form:

$$P \models \phi,$$

where '$\models$' means 'satisfies'. This is what Abramsky (1991) calls an *endogenous logic*, in which formulas describe properties of single programs.

We can also have *exogenous logics*: programs are embedded in formulas as modal operators. In this way, a Hoare triple like $\{\phi\}P\{\psi\}$ can be represented by

$$\phi \Rightarrow [P]\psi.$$

To build a proof system about every possible realm of application from scratch is clearly impossible. Rather, we will assume there are already working proof systems for actions and basic types (numbers, booleans, etc). Then those systems will be used as components of a multiset logic (namely a logic for making and proving assertions about multisets) and then a whole program logic. This plan will comply with Gamma's motto of separating coordination from computation. Our logic will be of the endogenous kind.

In designing a proof system we face a further problem: how do we know the logic itself is correct. Basically, we have two goals in mind:

- *Soundness*: if $P \models \phi$ according to the logic, then the execution of the program must meet the 'real-life' properties described by ϕ.
- *Completeness*: If a program P has a property p and the property is expressible in the language by the formula ϕ, can we always prove $P \models \phi$?

Soundness is paramount: we do not want spurious proofs of correctness. Completeness is beyond our reach as a consequence of Gödel's (1930, 1931) theorem of incompleteness, of course. Nevertheless, given a restricted (but still interesting) language of assertions, a form of completeness can be found. It is here that domain theory enters the picture.

2.4 Domain theory in logical form

Abramsky (1991) stated the research program of DTLF as follows:

1. A programming language can be seen as a set of types and type combinators. Programs are terms belonging to these types.
2. Types are assigned to domains and typed terms (programs) to elements in the domains through a *metalanguage*.

3. On the other hand, a logical interpretation of the metalanguage translates types into propositional theories and programs are interpreted via the satisfaction relation $\models$.

4. Correctness of this interpretation is guaranteed by the Stone duality theorems: the denotational (domain-theoretical) semantics of a language and the logical interpretation of the previous paragraph are Stone duals (ie, each determines the other up to isomorphism). Soundness and completeness of the program logic are direct consequences of Stone duality.

To fully present this programme we will start with some definitions regarding domains (mainly for notational clarity). Next, the metalanguage as well as its logical interpretation will be introduced. Lastly, (one of) the Stone duality theorem(s) will be given along with some soundness and completeness results.

2.4.1 Domains

It is not intended to give here an introduction to domain theory, but only to present the notation, so the concepts in this section appear in a rather concise fashion. Readers not familiar with the subject will find good introductions to posets and other ordered structures in Davey & Priestley (1990), Abramsky & Jung (1990) and Plotkin (1981).

Definition 2.4.1 *A* partially ordered set $\langle P, \sqsubseteq \rangle$ *(or* poset*, for short) consists of a set P and a reflexive, transitive and antisymmetric relation* $\sqsubseteq \subseteq P \times P$*. If* $X \subseteq P$*, then:*

i) u *is an upper bound of* X *iff* $x \sqsubseteq u$ *for every* $x \in X$*.* $m \in X$ *is a* minimal *element of* X *if there is no* $y \in X$ *such that* $y \neq m$ *and* $y \sqsubseteq m$*.*

ii) $\bigsqcup X$ *is the least upper bound of* X *(also known as supremum, lub or join) iff* $\bigsqcup X \sqsubseteq u$ *for every upper bound* u *of* X*. If* $X = \{a, b\}$ *then* $\bigsqcup X$ *can be written as* $a \sqcup b$*.*

iii) l *is a lower bound of* X *iff* $l \sqsubseteq x$ *for every* $x \in X$*.* $m \in X$ *is a* maximal *element of* X *if there is no* $y \in X$ *such that* $y \neq m$ *and* $m \sqsubseteq y$*.*

iv) $\bigsqcap X$*, defined in a similar way to* $\bigsqcup X$*, is the greatest lower bound (infimum, glb or meet) of* X*. If* $X = \{a, b\}$ *then* $\bigsqcap X$ *can be written as* $a \sqcap b$*.*

v) Let $\uparrow X = \{y \in P \mid \exists x \in X : x \sqsubseteq y\}$ *and* $\downarrow X = \{y \in P \mid \exists x \in X : y \sqsubseteq x\}$*.* $\uparrow x$ *is an abbreviation for* $\uparrow \{x\}$*.* $\downarrow x$ *is similar.*

vi) X *is* upper closed *iff* $X = \uparrow X$ *and* lower closed *iff* $X = \downarrow X$*.*

vii) The symbol $\bot$ *(called bottom) will represent an element in* P *such that* $\bot \sqsubseteq p$ *for every* $p \in P$*. The symbol* $\top$ *(top) will refer to an element with the property that* $p \sqsubseteq \top$ *for every* $p \in P$*. Note that* $\bot$ *and* $\top$ *do not always exists.*

viii) A relation $\sqsubseteq \subseteq P \times P$ *which is only transitive and reflexive (but not necessarily antisymmetric) is a* pre-order*.*

In studying posets the question of when a certain set has a supremum or an infimum is very important and the following definition allows us to talk about such elements in a general context.

Definition 2.4.2 *Suppose that* X *is a property of subsets of* P*. We will say that:*

- $\langle P, \sqsubseteq \rangle$ *is* X-complete *if* $S \subseteq P$ *and* $X(S)$ *imply that* $\bigsqcap S$ *exists.*
- $\langle P, \sqsubseteq \rangle$ *is* X-cocomplete *if* $S \subseteq P$ *and* $X(S)$ *imply that* $\bigsqcup S$ *exists.*

Being a *chain* and being *directed* are some of the properties of subsets we will be interested in:

- *Let* $X \subseteq P$*,* $X \neq \emptyset$*. If for every* $a, b \in X$ *either* $a \sqsubseteq b$ *or* $b \sqsubseteq a$ *then* X *is a* chain*. If for every* a *and* $b \in X$*,* $a \sqcup b$ *exists and* $a \sqcup b \in X$*, then* X *is* directed*.*
- *If for every directed set* $X \subseteq P$*,* $\bigsqcap X$ *(or* $\bigsqcup X$*) exists then the poset* P *is* directed-complete *(or* directed-cocomplete*).*

Lattices are finite-complete and finite-cocomplete posets. A lattice that is not only finite but arbitrarily complete and cocomplete is a *complete* lattice. A lattice in which

$$x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z)$$

holds is a *distributive* lattice. A *frame* is a finite-complete and arbitrary-cocomplete poset which is also distributive.

Ideals and filters are special subsets of a lattice:

Definition 2.4.3 *Let L be a lattice and X a non-empty subset of L. X is called*

i) an ideal *iff it is lower closed and* $x, y \in X$ *implies* $x \sqcup y \in X$*;*

ii) a filter *iff it is upper closed and* $x, y \in X$ *implies* $x \sqcap y \in X$*;*

iii) ideals and filters that do not coincide with L are proper. If $x \in L$*, ideals of the form* $\downarrow x$ *and filters of the form* $\uparrow x$ *are* principal*;*

iv) if X is a proper ideal and $x \sqcap y \in X$ *implies* $x \in X$ *or* $y \in X$*, then X is a* prime *ideal.*

v) Prime *filters are defined in a similar way. The set of prime filters of L ordered by inclusion is denoted by* $\mathsf{Spec}\, L$*.*

vi) When $X \subseteq L$ *is a proper ideal and the only ideal properly containing X is L itself, X is called a maximal ideal. A maximal filter (also known as an ultrafilter) is defined in an analogous way.*

vii) Given a pre-order P, the set of ideals of P ordered by $\subseteq$*, known as the ideal completion of P, is a lattice and it is represented by* $\mathsf{Idl}\, P$*. Lattices of this kind play an important role in the construction of powerdomains as we will see later.*

Elements in posets are frequently thought of as approximative values in the computation of a certain object (like a function). Some of these values are seen as finite steps in the computation process, while others can be regarded as limits of the computation. Hence it is useful to distinguish the first class of elements:

Definition 2.4.4 *Let* $c \in P$*. Then c is called* compact *or* finite *iff whenever* $X \subseteq P$ *is directed and* $c \sqsubseteq \bigsqcup X$*, there exists* $x \in X$ *such that* $c \sqsubseteq x$*. The set of compact elements of the poset P is denoted by* $K(P)$*.*

A poset P is algebraic *iff for every* $x \in P$*, we have that* $\{c \in K(P) \mid c \sqsubseteq x\}$ *is directed and* $x = \bigsqcup\{c \in K(P) \mid c \sqsubseteq x\}$*.*

We will be interested in some particular kinds of posets known generically as *domains*. However, while there is no confusion about the meaning

of *monotonic* and *continuous* functions in the literature, the term *domain* is used in different ways from one text to another. We shall use the following definition:

Definition 2.4.5 *A* domain *is an algebraic directed-cocomplete poset with a least element. A* Scott domain *is a domain which is also bounded cocomplete.*

As usual, a monotonic function between domains is an order-preserving function and a continuous function is a monotonic function which preserves joins. A *strict* function maps bottom to bottom in addition.

Given the set D, a *topology* of D is a set $T \subseteq \mathcal{P}(D)$ such that $\emptyset \in D$ and $D \in T$. Additionally, T should be closed under finite intersections and arbitrary unions. The elements of T are known as the *open sets* of D. The set $B \subseteq T$ is a *base* iff every open set in T is the union of elements of B. A set $S \subseteq D$ is *compact* if whenever $S \subseteq \bigcup C$ (with $C \subseteq T$), there exists a finite subset $C' \subseteq C$ such that $S \subseteq \bigcup C'$. Consider now the following topology:

Definition 2.4.6 *Suppose D is a domain. A subset $S \subseteq D$ is* Scott open *iff a) S is upper closed and b) if X is directed and $\bigsqcup X \in S$ then $S \cap X \neq \emptyset$. The topology formed by the Scott open sets of D is the* Scott topology *and is denoted by $\Omega(D)$.*

We have talked about directed-cocomplete and Scott domains. Some times it is preferable to work with a different kind of domain: the so-called *sequences of finite posets* or SFP domains. One strong argument in their favour is that the category of SFP domains is cartesian-closed (see Smyth 1983), which guarantees that it is also closed under the type operations we will introduce later.

Definition 2.4.7 *Suppose D is a domain. If $X \subseteq D$ then*

$$UB(X) = \{u \mid x \sqsubseteq u \text{ for all } x \in X\}$$
$$MUB(X) = \{m \in UB(X) \mid m \text{ is minimal in } UB(X)\}.$$

Let $U : \mathcal{P}_{fin}(D) \to \mathcal{P}_{fin}(D)$ *be a function as follows:*

$$\begin{aligned} U^0(X) &= X \\ U^{n+1}(X) &= \bigcup\{MUB(Y) \mid Y \subseteq U^n(X)\} \\ U(X) &= \bigcup_{n\in\mathbb{N}} U^n(X). \end{aligned}$$

A domain D *is a* SFP *domain if and only if:*

1. *If* $X \subseteq K(D)$ *is finite then: a)* $MUB(X)$ *is finite and b) for every* $u \in UB(X)$ *there is a* $m \in MUB(X)$ *such that* $m \sqsubseteq u$.
2. *If* $X \subseteq K(D)$ *is finite then* $U(X)$ *is also finite.*

When considering elements $X, Y \in \mathcal{P}(P)$ (P a poset) there are three commonly used orders:

$$\begin{aligned} X \sqsubseteq_l Y \quad &\text{iff} \quad \forall x \in X\, \exists y \in Y \text{ such that } x \sqsubseteq y \\ X \sqsubseteq_u Y \quad &\text{iff} \quad \forall y \in Y\, \exists x \in X \text{ such that } x \sqsubseteq y \\ X \sqsubseteq_c Y \quad &\text{iff} \quad X \sqsubseteq_l Y \text{ and } X \sqsubseteq_u Y \end{aligned}$$

These three pre-orders (they are not partial orders as antisymmetry does not necessarily hold) are known as the *lower* (or Hoare), the *upper* (or Smyth) and the *Egli-Milner* pre-orders, respectively.

If $\mathcal{P}^*_{fin}(D)$ represents the set of non-empty finite subsets of D, then we have three pre-orders over $\mathcal{P}^*_{fin}(D)$ using the previously defined orderings. The powerdomain construction is based on these pre-orders:

Definition 2.4.8 *Let* D *be a domain. Suppose now a pre-ordering of* $\mathcal{P}^*_{fin}(D)$ *is given. Then the ideal completion of* $\mathcal{P}^*_{fin}(D)$ *is called:*

i) the lower *or* Hoare powerdomain*—denoted by* $\mathcal{P}_l(D)$*—if the chosen pre-order for* $\mathcal{P}^*_{fin}(D)$ *is* $\sqsubseteq_l$*;*

ii) the upper *or* Smyth powerdomain*—represented by* $\mathcal{P}_u(D)$*—if the chosen pre-order for* $\mathcal{P}^*_{fin}(D)$ *is* $\sqsubseteq_u$*;*

iii) the convex *or* Plotkin powerdomain*—in symbols* $\mathcal{P}_c(D)$*—if the chosen pre-order for* $\mathcal{P}^*_{fin}(D)$ *is* $\sqsubseteq_c$.

2.4.2 A metalanguage for denotational semantics

Most programming languages have operations for constructing complex types from simpler ones. We can regard types as domains and type expressions as operations on domains. Let D, S range over type expressions and T over type variables (that is, basic types). Now we will consider these type expressions:

$$D ::= 1 \mid D \times S \mid D \to S \mid D \oplus S \mid D_\perp \mid \mathcal{P}(D) \mid T \mid \mathsf{rec}\, T.D,$$

- 1 is the one point lattice.
- $D \times S$ is the cartesian product of domains with coordinatewise ordering: $(a,b) \sqsubseteq (c,d)$ if and only if $a \sqsubseteq c$ and $b \sqsubseteq d$.
- $D \to S$ is the *space* of continuous functions from D to S. If f and $g \in D \to S$, then $f \sqsubseteq g$ if and only if $f(x) \sqsubseteq g(x)$ for every $x \in D$.
- $D \oplus S$ stands for the *coalesced sum*, ie the set

  $$((D - \{\perp_D\}) \times \{0\}) \cup ((S - \{\perp_S\}) \times \{1\}) \cup \{\perp_{D \oplus S}\}$$

  with the ordering: (a) $(x,m) \sqsubseteq (y,n)$ if and only if $m = n$ and $x \sqsubseteq y$; and (b) $\perp_{D\oplus S} \sqsubseteq x$ for all $x \in D \oplus S$.
- $D_\perp$ is the *lifting* of D, which is defined as the set $D \cup \{\perp'\}$, where $\perp' \notin D$. We have that $x \sqsubseteq y$ if and only if either $x = \perp'$ or x and $y \in D$ and $x \sqsubseteq_D y$.
- $\mathcal{P}(D)$ can be any of the powerdomain constructions already introduced.
- The domain $\mathsf{rec}\, t.D$ is the solution to the domain equation $t = D(t)$.

If we require the basic types to be SFP domains (and not necessarily Scott domains) the type expressions are closed under the above operations.

The function $\pi_i : D_1 \times \cdots \times D_n \to D_i$ is the ith-projection function and $\iota_i : D_i \to D_1 \oplus \cdots \oplus D_n$ is the ith-injection function as they are usually defined.

Although the type expressions are the first stage in giving a domain-theoretical semantics of a programming language, we still have the problem that a language is defined syntactically and we need to translate this

syntax-oriented presentation into a domain-theoretical one. This second step is implemented through a series of function definitions which map syntactical constructions in the language onto elements in a given domain.

Some of the required functions are recursive and, hence, the question of a least fixed point —the standard solution to a recursive equation— arises. But if we restrict the definitions to *continuous* functions the existence of least fixed points can be taken for granted.

To build the denotational functions, Plotkin (1983) proposed a metalanguage which preserves continuity. The metalanguage is a series of abbreviations of lambda-style definitions. We will use just a few of his metalanguage expressions, introduced in the next paragraphs. This presentation is based on Winskel (1993)'s account:

- Let $C : (D_1 \oplus \cdots \oplus D_n)$ and $E_1, \ldots, E_n : D$ be continuous. Then

  $$\begin{aligned} &\textbf{case } C \textbf{ of } \iota_1(c_1) \Leftarrow E_1; \\ &\qquad \cdots \\ &\qquad \iota_n(c_n) \Leftarrow E_n \\ &\textbf{end} \end{aligned}$$

  is defined as

  $$[\lambda c_1 \,.\, E_1, \ldots, \lambda c_n \,.\, E_n](C)$$

  and it is continuous, having type D.

- The following construction is a special but very important instance of the former. Let $C : \mathcal{B}$ and $E_1, E_2 : D$ be continuous (where $\mathcal{B}$ is the domain of booleans, to be defined in the next page). Then

  $$\textbf{if } C \textbf{ then } E_1 \textbf{ else } E_2 \textbf{fi} \equiv_{def} (\lambda x_1 \,.\, E_1, \lambda x_2 \,.\, E_2)(C)$$

  is continuous and has type D.

- If $M : \mathcal{P}(D_1)$ and $N : \mathcal{P}(D_2)$ then

  $$\textbf{over } M \textbf{ extend } x \,.\, N \equiv_{def} \bigcup \{(\lambda x \,.\, N)(m) \mid m \in M\}$$

  it is also continuous and has type $\mathcal{P}(D_2)$.

The Sierpinski space

- If D is a type and $e : D$ is continuous then

$$\mu x \,.\, e \equiv_{def} \bigsqcup_{n \in \mathbb{N}} (\lambda x.e)^n(\bot)$$

  is continuous (again) with type D. We shall call it the *least fixed point*.

The following examples illustrate how to build new domains and their operations from old ones:

1. Let $\mathbf{1}$ be the one point domain. Then the Sierpinsky space is $\mathbf{O} = \mathbf{1}_\bot$.
2. The domain of booleans is $\mathcal{B} = \mathbf{O} \oplus \mathbf{O}$. The following constructions are associated with this domain (where D is an arbitrary domain):

$$\frac{}{\mathit{true} : \mathcal{B}} \qquad \frac{}{\mathit{false} : \mathcal{B}} \qquad \frac{C : \mathcal{B} \quad E_1, E_2 : D}{\textbf{if } C \textbf{ then } E_1 \textbf{ else } E_2 \textbf{ fi} : D}$$

3. Now we have the powerdomain constructions. Let D be a domain. Then:

$$\frac{M : D}{\{M\} : \mathcal{P}(D)} \qquad \frac{M, N : \mathcal{P}(D)}{M \cup N : \mathcal{P}(D)} \qquad \frac{M : \mathcal{P}(D_1) \quad N : \mathcal{P}(D_2)}{\textbf{over } M \textbf{ extend } m \,.\, N \textbf{ end} : \mathcal{P}(D_2)}$$

   where $\mathcal{P}(D)$ can be replaced consistently in each of the rules by $\mathcal{P}_l(D)$, $\mathcal{P}_u(D)$ or $\mathcal{P}_c(D)$.

In section 3.5 we will use this metalanguage extensively.

2.4.3 The logical/semantic sides of a language

The originality of DTLF comes not from the concepts introduced in the last two sections, but from the construction of propositional theories out of domain expressions.

Definition 2.4.9 *Suppose D is a domain (corresponding to a certain type). Then*

$$\mathcal{L}(D) = \langle L(D), \leq_D, =_D, f_D, t_D, \vee_D, \wedge_D \rangle$$

is the propositional theory associated with D. $L(D)$ is a set of formulae; $\leq_D$ and $=_D$ are the relations of logical implication and logical equivalence between formulae, respectively; $\vee_D$ and $\wedge_D$ are logical disjunction and conjunction (in that order); and t_D and f_D are the constants true and false.

The subscript $_D$ is attached to the former symbols to make clear their type. In most contexts, however, it is clear what type an operation belongs to and then the subscript will be omitted.

The exact meaning of the abstract definition of $\mathcal{L}(D)$ will become more concrete when we introduce its formation rules and axioms. Let us start with the former:

2.4.10 *Formation rules for $L(D)$:*

F_1 $\quad t, f \in L(D)$ $\qquad$ F_2 $\quad \dfrac{\phi, \psi \in L(D)}{\phi \wedge \psi \in L(D)}$

F_3 $\quad \dfrac{\phi, \psi \in L(D)}{\phi \vee \psi \in L(D)}$ $\qquad$ F_4 $\quad \dfrac{\phi \in L(D_1),\ \psi \in L(D_2)}{(\phi \times \psi) \in L(D_1 \times D_2)}$

F_5 $\quad \dfrac{\phi \in L(D_1),\ \psi \in L(D_2)}{(\phi \to \psi) \in L(D_1 \to D_2)}$ $\qquad$ F_6 $\quad \dfrac{\phi \in L(D_1),\ \psi \in L(D_2)}{(\phi \oplus f),\ (f \oplus \psi) \in L(D_1 \oplus D_2)}$

F_7 $\quad \dfrac{\phi \in L(D)}{\phi_\perp \in L(D_\perp)}$ $\qquad$ F_8 $\quad \dfrac{\phi \in L(D)}{\blacksquare\phi,\ \blacklozenge\phi \in L(\mathcal{P}(D))}$

F_9 $\quad \dfrac{\phi \in L(D[\mathsf{rec}\, t.D/t])}{\phi \in L(\mathsf{rec}\, t.D)}$.

Observe that in rule F_6 the constant f does not belong to the same domain as the formulae ϕ and ψ. The meaning of the modalities $\blacksquare$ and $\blacklozenge$ will become clear when we explain the satisfaction relation $\models$. For the time being let us proceed with the axioms of $\mathcal{L}(D)$. They are divided between purely logical and type-specific axioms.

2.4.11 *Logical axioms and inference rules of $\mathcal{L}(D)$:*

$$(A_1)\quad \phi \leq \phi \qquad (A_2)\quad \frac{\phi \leq \psi,\ \psi \leq \chi}{\phi \leq \chi}$$

$$(A_3)\quad \frac{\phi \leq \psi,\ \psi \leq \phi}{\phi = \psi} \qquad (A_4)\quad \frac{\phi = \psi}{\phi \leq \psi,\ \psi \leq \phi}$$

$$(A_5)\quad \phi \leq t \qquad (A_6)\quad \frac{\phi \leq \psi,\ \phi \leq \gamma}{\phi \leq \psi \wedge \gamma}$$

$$(A_7)\quad \phi \wedge \psi \leq \phi \qquad (A_8)\quad \phi \wedge \psi \leq \psi$$

$$(A_9)\quad f \leq \phi \qquad (A_{10})\quad \frac{\phi \leq \psi,\ \gamma \leq \psi}{\phi \vee \gamma \leq \psi}$$

$$(A_{11})\quad \phi \leq \phi \vee \psi \qquad (A_{12})\quad \psi \leq \phi \vee \psi$$

$$(A_{13})\quad \phi \wedge (\psi \vee \gamma) \leq (\phi \wedge \psi) \vee (\phi \wedge \gamma).$$

These axioms give $\mathcal{L}(D)$ the structure of a distributive lattice.

Regarding the type-specific axioms, we need to introduce first the indexed conjunction and disjunction symbols. If I is a set then

$$\bigwedge_{i\in I} \phi_i \qquad \text{and} \qquad \bigvee_{i\in I} \phi_i$$

are the conjunction and disjunction, respectively, of a set of propositions labelled with the elements in I. We have now the second batch of axioms:

2.4.12 *Type-specific axioms and inference rules of $\mathcal{L}(D)$:*

$$AT_1\quad \bigwedge_{i\in I}(\phi_i \times \psi_i) = (\bigwedge_{i\in I} \phi_i \times \bigwedge_{i\in I} \psi_i) \qquad AT_2\quad \bigvee_{i\in I}(\phi_i \times \psi_i) = (\bigvee_{i\in I} \phi_i \times \bigvee_{i\in I} \psi_i)$$

$$AT_3\quad (\phi \times \bigvee_{i\in I} \psi_i) = \bigvee_{i\in I}(\phi \times \psi_i) \qquad AT_4\quad (\phi \to \bigwedge_{i\in I} \psi_i) = \bigwedge_{i\in I}(\phi \to \psi_i)$$

$$AT_5\quad (\bigvee_{i\in I} \phi_i \to \psi) = \bigvee_{i\in I}(\phi_i \to \psi) \qquad AT_6\quad (\bigwedge_{i\in I} \phi_i \oplus f) = \bigwedge_{i\in I}(\phi_i \oplus f)$$

$$AT_7\quad (f \oplus \bigwedge_{i\in I} \psi_i) = \bigwedge_{i\in I}(f \oplus \psi_i) \qquad AT_8\quad (\bigvee_{i\in I} \phi_i \oplus f) = \bigvee_{i\in I}(\phi_i \oplus f)$$

AT_9 $(f \oplus \bigvee_{i\in I} \psi_i) = \bigvee_{i\in I}(f \oplus \psi_i)$ AT_{10} $(\phi \wedge \psi)_\perp = \phi_\perp \wedge \psi_\perp$

AT_{11} $(\bigvee_{i\in I} \phi_i)_\perp = \bigvee_{i\in I}(\phi_i)_\perp$ AT_{12} $\blacksquare \bigwedge_{i\in I} \phi_i = \bigwedge_{i\in I} \blacksquare\phi_i$

AT_{13} $\blacklozenge \bigvee_{i\in I} \phi_i = \bigvee_{i\in I} \blacklozenge\phi_i$ AT_{14} $\blacksquare(\phi \vee \psi) = \blacksquare\phi \vee \blacksquare\psi$

AT_{15} $\blacklozenge\phi \wedge \blacklozenge\psi \leq \blacklozenge(\phi \wedge \psi)$ AT_{16} $\blacksquare f = f$

AT_{17} $\dfrac{\phi \leq \phi',\ \psi \leq \psi'}{(\phi \times \psi) \leq (\phi' \times \psi')}$ AT_{18} $\dfrac{\phi \leq \phi',\ \psi \leq \psi'}{(\phi \to \psi) \leq (\phi' \to \psi')}$

AT_{19} $\dfrac{\phi \leq \psi}{(\phi \oplus f) \leq (\psi \oplus f),\ (f \oplus \phi) \leq (f \oplus \psi)}$

AT_{20} $\dfrac{\phi \leq \psi}{\phi_\perp \leq \psi_\perp}$ AT_{21} $\dfrac{\phi \leq \psi}{\blacksquare\phi \leq \blacksquare\psi}$

AT_{22} $\dfrac{\phi \leq \psi}{\blacklozenge\phi \leq \blacklozenge\psi}$

Axioms AT_1–AT_{15} show how the type constructions distribute over (or interact with) conjunctions, disjunctions and modal operators. Axioms AT_{17}–AT_{22} assert the preservation of the implication order through the type and modal constructions. The motivation behind axiom AT_{16} is that we need only one false value and not many (one simple, the others prefixed with either $\blacksquare$ or $\blacklozenge$).

The road between domains and logics is two-way: we can also give a denotational interpretation of logical formulas. The denotational counterpart of a formula in $L(D)$ will be a compact open set of the Scott topology of D according to the next function:

Definition 2.4.13 *The interpretation function* $[\![\ ]\!]_D : L(D) \to K\Omega(D)$*:*

$$[\![\phi \wedge \psi]\!]_D = [\![\phi]\!]_D \cap [\![\psi]\!]_D$$

$$[\![t]\!]_D = D = 1_{K\Omega(D)}$$

$$[\![\phi \vee \psi]\!]_D = [\![\phi]\!]_D \cup [\![\psi]\!]_D$$

$$[\![f]\!]_D = \emptyset = 0_{K\Omega(\mathcal{D}(D))}$$

$$
\begin{aligned}
[\![\phi \times \psi]\!]_{D_1 \times D_2} &= \{(u,v) \mid u \in [\![\phi]\!]_{D_1}, v \in [\![\psi]\!]_{D_2}\} \\
[\![\phi \to \psi]\!]_{D_1 \to D_2} &= \{f \in (D_1 \to D_2) \mid f([\![\phi]\!]_{D_1}) \subseteq [\![\psi]\!]_{D_2}\} \\
[\![\phi \oplus f]\!]_{D_1 \oplus D_2} &= \{(0,u) \mid u \in [\![\phi]\!]_{D_1} - \{\bot_{D_1}\}\} \cup \\
&\quad \{d \in (D_1 \oplus D_2) \mid \bot_{D_1} \in [\![\phi]\!]_{D_1}\} \\
[\![f \oplus \phi]\!]_{D_1 \oplus D_2} &= \{(1,u) \mid u \in [\![\phi]\!]_{D_2} - \{\bot_{D_2}\}\} \cup \\
&\quad \{d \in (D_1 \oplus D_2) \mid \bot_{D_2} \in [\![\phi]\!]_{D_2}\} \\
[\![\phi_\bot]\!]_{D_\bot} &= \{(0,u) \mid u \in [\![\phi]\!]_D\} \\
[\![\blacksquare\phi]\!]_{\mathcal{P}(D)} &= \{S \in \mathcal{P}(D) \mid S \subseteq [\![\phi]\!]_D\} \\
[\![\blacklozenge\phi]\!]_{\mathcal{P}(D)} &= \{S \in \mathcal{P}(D) \mid S \cap [\![\phi]\!]_D \neq \emptyset\} \\
[\![\phi]\!]_{\mathsf{rec}\, t.D} &= \{\alpha_D(u) \mid u \in [\![\phi]\!]_{[\mathsf{rec}\, t.D]/t}\}
\end{aligned}
$$

With this last function we are ready to present a satisfaction relation:

Definition 2.4.14 *1. Let $\phi \in L(D)$. If ϕ follows from axioms A_1–A_{13}, AT_1–AT_{22}, we say that ϕ is a theorem of $\mathcal{L}(D)$. In symbols:*

$$\mathcal{L}(D) \vdash \phi.$$

2. Let ϕ, $\psi \in L(D)$. The satisfaction relation $\models$ is defined as

$$D \models \phi \leq \psi \qquad \textit{if and only if} \qquad [\![\phi]\!]_D \subseteq [\![\psi]\!]_D.$$

Let us come back to the modal operators. Suppose that $\phi \in L(D)$ and $S \subseteq D$. Then $S \models \blacksquare\phi$ if and only if for every $s \in S$ we have that $s \models \phi$ and $S \models \blacklozenge\phi$ if and only if there is a $s \in S$ such that $s \models \phi$.

We can now address the issues of soundness and completeness of $\mathcal{L}(D)$.

2.4.4 Stone duality

The Stone duality theorems are a family of isomorphisms between an ample variety of mathematical structures. To explain their exact meaning and importance is beyond the scope of the present thesis. Johnstone (1982) contains a thorough presentation of the subject for the interested reader.

We will use one of the duality theorems in the version offered by Abramsky (1991). This theorem takes the *Lindenbaum algebra* of the logic $\mathcal{L}(D)$, viz the partition of the formulas in $L(D)$ induced by the equivalence relation $=$ as defined in axioms A_1–A_{13}. The Lindenbaum algebra of $L(D)$ is denoted by $\mathcal{LA}(D)$.

Theorem 2.4.15 *Stone duality. Let D be a domain. Then we have the following isomorphisms:*

1. $D \simeq \mathsf{Spec}\,\mathcal{LA}(D)$*;*
2. $K(\Omega(D)) \simeq \mathcal{L}(D)$.

Finally we arrive to the next theorem (proved by Abramsky (1991)):

Theorem 2.4.16 *If* ϕ, $\psi \in L(D)$ *then*

$$\mathcal{L}(D) \vdash \phi \leq \psi \qquad \textit{if and only if} \qquad D \models \phi \leq \psi,$$

i.e., the logic of D is sound and complete.

A brief summary of the whole section is opportune: a) by means of a metalanguage, the constructions of a programming language are presented in a domain-theoretical basis; b) a propositional theory (formulae and axioms) arises from the domain version; c) thanks to Stone duality the logic is sound and complete with respect to the semantics (that is, the domain-theoretical presentation of the language).

This same programme will now be applied to Gamma. In Chapter 3 we will give a denotational semantics of the language. In Chapter 4, a full proof system will be derived from the semantics.

Chapter 3

Gamma semantics

Denotational semantics of programming languages has been a very important field for the past three decades. Not only does it give a mathematical model of programming languages, but it also provides a possible foundation for constructing a proof system for a language, as domain theory in logical form shows (see previous chapter).

However, denotational semantics of programming languages including parallel operators seems to be a more complex issue than semantics of sequential languages, at least at first sight. In the particular case of Gamma it has proved to be very hard to solve. Gay & Hankin (1996*b*) proposed a semantical model useful enough to serve as the basis of a proof system. Nonetheless, their proof system was full of intricate details making its use very complicated and, more importantly, the semantics was not fully abstract.

Sands (1993*a*) and Gay & Hankin (1996*a*) took a different approach based on ideas originally proposed by Brookes (1985) in a series of papers: the transition trace semantics. In spite of also failing to be fully abstract, its more natural and cleaner appearance made it a good starting point for an application of domain theory. The present proposal arose from the transition trace model.

Denotational models are not the only view, however. Reynolds (1996) proposed a semantics for Gamma based on temporal logic which also produced a proof system, briefly commented on at the beginning of chapter 4.

This chapter will deal first with the operational semantics of Gamma, including a derived program ordering. Secondly, it will introduce the denotational model from two points of view: abstract (that is, without considering the internal structure of programs) and compositional (ie, one in which structure is taken into account). A series of transformation rules (presented in Hankin et al. (1993) and Sands (1993*b*)) for Gamma programs is confirmed as valid in the new semantic setting. Then soundness and full abstraction are proved. Finally, as a preparation for the next chapter, some domain construction rules for the semantics are given.

The main result of this chapter, namely full abstraction of the denotational semantics, is important because it allows us to easily translate reasoning about the denotational behaviour of a program into statements about its operational behaviour. The utility of this will become clearer when the resulting proof system is discussed (chapter 4).

3.1 Operational semantics

The operational semantics for Gamma has been presented elsewhere (Hankin et al. (1993), for example). The following rules in structural operational semantics style are taken from that source, where $\langle P, M\rangle$ is a *configuration*, i.e., a pair made of a program (P) and a multiset (M), to which the program is applied:

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

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

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

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

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

These rules can be seen in the light of the function S defined in section 2.2. The first two correspond to the possible outcomes of $S(A \Leftarrow R, M)$. The following four rules referring to the sequential and parallel composition also correspond to S applied to the same kind of programs. The last one expresses the requirement of simultaneous termination of the two parallel components of a program if the latter is to terminate.

As can be noticed, $\rightarrow$ indicates a single step transition, thus defining a relation on $(\mathbb{G} \times \mathbb{M}(D)) \times ((\mathbb{G} \times \mathbb{M}(D)) \cup \mathbb{M}(D))$ (remember that $\mathbb{G}$ denotes the set of Gamma programs and $\mathbb{M}(D)$ is the set of finite multisets with elements in D). As usual, $\rightarrow^*$ refers to its reflexive and transitive closure.

This definition of Gamma operational semantics induces an interleaved approach to parallelism, that is, the effects of different atomic reactions parallelly composed take place only one at a time (non-determinism arises from a free choice of the actions). Chaudron (1998) called this interpretation *single-step semantics* as opposed to *multiple-step semantics* in which different reactions can act on mutually independent subsets of a given multiset at the same time. The single-step version was chosen for the present thesis because it makes reasoning about programs much easier. Chaudron (1998) pointed out that, when using simulation as a refinement notion, the single-step semantics does not model the parallelism of Gamma programs, justifying in this way his choice in favour of multiple-step semantics. Nevertheless, he also showed the equivalence of the two operational semantics in terms of input-output behaviour. The refinement notions referred to in this chapter are based on input-output issues and then we do not need to take into account Chaudron's argument.

On the other hand Ciancarini, Gorrieri & Zavattaro (1996) give another operational semantics which greatly differs from the one just presented in the way parallel composition is treated. Their idea is to synchronize the reduction of programs composed in parallel, so that some undesired computations can be avoided. This would alter deeply the definition of the denotation of sequential composition (see section 3.3). Although their proposal should be considered further, for the time being we will keep the more traditional option.

According to the rules just presented, a program is not static during its

execution. In particular, when the first component of a program made by sequential composition finishes its execution (i.e., there are no more possible reactions), the second component does not need to take into account the first component any longer. We can regard the second component as the residual part of the whole program. More formally:

Definition 3.1.1 *The residual part of a program P, denoted by* $\underline{\underline{P}}$ *is defined inductively:*

$$\underline{\underline{A \Leftarrow R}} \equiv_{def} A \Leftarrow R,$$
$$\underline{\underline{P_2 \circ P_1}} \equiv_{def} \underline{\underline{P_2}},$$
$$\underline{\underline{P_1 + P_2}} \equiv_{def} \underline{\underline{P_1}} + \underline{\underline{P_2}}.$$

Because some Gamma programs will appear repeatedly in the following, it will be useful to have some abbreviations to denote them:

$$Skip \equiv_{def} A \Leftarrow false,$$
$$Big\text{-}Bang \equiv_{def} \emptyset \rightarrow A \Leftarrow true,$$
$$Await \equiv_{def} x \rightarrow x \Leftarrow true.$$

The first one corresponds roughly to the ubiquitous program skip used in many correctness proof methods. The second is a never-ending program which will always add A to a multiset, regardless of the values it already contains. The last one is very similar in the sense that it can never end, but it will not act when the multiset is empty.

The traditional concept of program contexts will later help us to define a program order. A Gamma context C is:

$$C ::= [\ ] \mid P + C \mid C + P \mid P \circ C \mid C \circ P,$$

where $P \in \mathbb{G}$.

A special kind of contexts, *active contexts*, play a role in analysing the behaviour of programs:

$$A ::= [\ ] \mid P + A \mid A + P \mid P \circ A.$$

When dealing with transformations of programs a notion of order can be very useful. Let us suppose $\sqsubseteq$ is a reflexive and transitive relation on $\mathbb{G}$. If P, $Q \in \mathbb{G}$, the equivalence relation $P \equiv Q$ holds when $P \sqsubseteq Q$ and $Q \sqsubseteq P$. Consider now the following would-be laws:

3.1.2 *Program order laws for Gamma.*

The sequential laws

1. $P \circ (Q \circ R) \equiv (P \circ Q) \circ R$
2 $P \circ Skip \equiv P$
3. $P \equiv Skip \circ P$
4. $(A \Leftarrow R) \equiv (A \Leftarrow R) \circ (A \Leftarrow R)$.

The parallel laws

5. $P + (Q + R) \equiv (P + Q) + R$
6. $P + Q \equiv Q + P$
7. $P \equiv Skip + P$
8. $P \sqsubseteq P + P$
9. $(A \Leftarrow R) \equiv (A \Leftarrow R) + (A \Leftarrow R)$.

The parallel-sequential laws

10. $(P + Q) \circ R \sqsubseteq P + (Q \circ R)$
11. $(P_1 + P_2) \circ (Q_1 + Q_2) \sqsubseteq (P_1 \circ Q_1) + (P_2 \circ Q_2)$
12. $P \circ (Q + R) \sqsubseteq (P \circ Q) + (P \circ R)$

Residual program laws

13. $P \equiv \underline{\underline{P}} \circ P$
14. $(P + \underline{\underline{R}}) \circ (Q + R) \sqsubseteq (P \circ Q) + R$
15. $(P \equiv \underline{\underline{P}}) \Rightarrow (P \equiv P + P)$

Which of these laws are valid? That depends on the program order chosen. Considering just the Gamma operational semantics we have different options. Hankin et al. (1993) proposed the following:

Definition 3.1.3 *$P_1 \sqsubseteq_{IO} P_2$ if and only if $\langle P_1, M\rangle \rightarrow^* N$ implies $\langle P_2, M\rangle \rightarrow^* N$. As usual, $P_1 \equiv_{IO} P_2$ if and only if $P_1 \sqsubseteq_{IO} P_2$ and $P_2 \sqsubseteq_{IO} P_1$.*

As can be seen from the definition, $\sqsubseteq_{IO}$ addresses only partial correctness. Nevertheless, Hankin et al. (1993) proved the 15 laws for this order.

Program orders are a common tool for refinement of programs. One property that facilitates this process is *substitutivity*: if program P is a refinement of program Q we want to be able to replace any instance of P by an instance of Q in any program which includes the latter. Alas, $\sqsubseteq_{IO}$ is too weak to allow general substitutivity. In particular, $P \sqsubseteq_{IO} Q$ does not imply that $P + R \sqsubseteq_{IO} Q + R$. Sands (1993*a*) provided this counterexample:

$$Q : x \to 0 \Leftarrow x = 1$$
$$P : \mathit{Skip} \circ Q$$
$$R : x \to 1 \Leftarrow x = 0.$$

By 3.1.2 law 3, $P \sqsubseteq_{IO} Q$. Now $\langle Q + R, \{\!|1|\!\}\rangle$ never terminates, but $\langle P + R, \{\!|1|\!\}\rangle \to^* \{\!|1|\!\}$ and then $P + R \not\sqsubseteq_{IO} Q + R$.

The problem here is that when P and Q act in the context $[\ ] + R$, their behaviour differs. A solution is to enforce the order taking into account contexts:

Definition 3.1.4 *$P_1 \sqsubseteq_C P_2$ if and only if for all contexts C, $C[P_1] \sqsubseteq_{IO} C[P_2]$. $P_1 \equiv_C P_2$ if and only if $P_1 \sqsubseteq_C P_2$ and $P_2 \sqsubseteq_C P_1$.*

By definition $P \not\sqsubseteq_C Q$ and then the counterexample does not work any longer. Because of its stronger requirements we will use $\sqsubseteq_C$ as *the* operational order for Gamma.

Although he used the following weaker version of law 3, Sands (1993*b*) proved the 3.1.2 laws for a different order $\sqsubseteq_t$, based in denotational semantics (a predecessor of our own order):

3′. $P \sqsubseteq_t \mathit{Skip} \circ P$.

He also showed that for any P_1, $P_2 \in \mathbb{G}$, $P_1 \sqsubseteq_t P_2$ implies $P_1 \sqsubseteq_C P_2$, which in its turn means that the laws are also valid for $\sqsubseteq_C$.

Useful as they are, the laws are not a proof system, as they only establish some refining rules. The extra power will come from the denotational semantics of Gamma.

3.2 Denotational semantics

To move from operational to denotational semantics we require first a way of describing the behaviour of a program not as a series of steps but as a mathematical "object" which can be handled through suitable operations.

Definition 3.2.1 *Let D be an arbitrary set. Consider now the following set:*

$$\mathbb{T}(D) = \{(M_1, M_2)(M_3, M_4) \ldots | M_i \in D\}$$

whose elements will be called traces. $\mathbb{T}_{fin}(D)$ will denote the set of finite traces, while ϵ will be the empty trace.

Let us state now some operations and relations on traces that will be used extensively in the future:

Concatenation. $\mathbb{T}_{fin}(D) \times \mathbb{T}(D) \to \mathbb{T}(D)$. If α, $\beta \in \mathbb{T}(D)$ their concatenation is denoted simply by $\alpha\beta$.

Linking. $\odot : \mathbb{T}_{fin}(D) \times \mathbb{T}(D) \to \mathbb{T}(D)$. Linking differs from concatenation as shown in the following definition:

$$(M_1, M_2) \ldots (M_{n-1}, M_n) \odot (N_1, N_2) \ldots = \begin{cases} (M_1, M_2) \ldots (M_{n-1}, M_n)(N_1, N_2) \ldots & \text{if } M_n = N_1 \\ \text{undefined} & \text{otherwise} \end{cases}$$

Absorption. $A \subseteq \mathbb{T}(D) \times \mathbb{T}(D)$. Let $\alpha \in \mathbb{T}_{fin}(D)$ and $\beta \in \mathbb{T}(D)$. Then

$$A(\alpha(M, N)(N, P)\beta, \alpha(M, P)\beta)$$

holds. If $T \subseteq \mathbb{T}(D)$ then $\bar{A}(T)$ will be its closure under absorption.

Total absorption. $\_ : \mathbb{T}(D) \to \mathbb{T}(D)$. The result of total absorption is the last element in a chain $t_1, \ldots, t_n, \ldots t_\alpha$ such that $A(t_i, t_{i+1})$ and $t_i \neq t_{i+1}$. Needless to say, the chain can be infinite with a top element (the chain being equivalent to the ordinal number $\omega + k$, $k \in \mathbb{N}$). $\_$ is calculated recursively by the following rule:

$$\underline{\epsilon} = \epsilon$$

$$\underline{(M_1, M_2)(M_3, M_4)\alpha} = \begin{cases} \underline{(M_1, M_4)\alpha} & \text{if } M_2 = M_3 \\ (M_1, M_2)\underline{(M_3, M_4)\alpha} & \text{otherwise.} \end{cases}$$

Sets of traces can be regarded as denotations of programs in parallel languages with states. Each pair (M_i, M_{i+1}) in a trace denotes a *transition* from the state M_i to the state M_{i+1} performed by a certain program. Environment interference (ie, another program running in the background) plays a relevant role in parallel languages and this fact is reflected in traces where adjacent pairs such as (M_n, M_{n+1}) and (M_{n+2}, M_{n+3}) do not share contiguous elements, that is, $M_{n+1} \neq M_{n+2}$. In this example, the change from M_{n+1} to M_{n+2} is not performed by the program but by the environment in which it is executed. We will call a denotational model like this a *transition trace semantics*.

Applying this idea, Sands (1993*a*) and Gay & Hankin (1996*a*) produced the first transition trace semantics for Gamma. Consider a Gamma program P and a trace $(M_1, M_2)(M_3, M_4)\ldots$ such that $\langle P, M_1\rangle \rightarrow \langle P', M_2\rangle$, $\langle P', M_3\rangle \rightarrow \langle P'', M_4\rangle$, etc. The (possible) change from M_2 to M_3 can be explained by the interference from another Gamma program acting as the environment. No restriction on the behaviour of the environment was originally set.

Nevertheless, the fact that the environment interference was not regulated at all exposed the semantics to arbitrary environmental behaviour. This provided a basis for the proof that the semantics was not fully abstract, made by Sands (1993*a*) himself.

Sands first proved that $P \circ \mathit{Await} \sqsubseteq_C \mathit{Await}$, for every $P \in \mathbb{G}$. In particular, $\mathit{Skip} \circ \mathit{Await} \sqsubseteq_C \mathit{Await}$. On the other hand, according to his semantics, the denotation of *Await* is the set of traces $\{(\emptyset, \emptyset)\}$, while the denotation of $\mathit{Skip} \circ \mathit{Await}$ is the set $\{(\emptyset, \emptyset)(M, M) \mid M \in \mathbb{M}(D)\}$. It is clear that

$$\{(\emptyset, \emptyset)(M, M) \mid M \in \mathbb{M}(D)\} \not\subseteq \{(\emptyset, \emptyset)\}$$

and then $\mathit{Skip} \circ \mathit{Await} \not\sqsubseteq_t \mathit{Await}$, where $\sqsubseteq_t$ is Sands' denotational order. This fact ruled out full abstraction.

Sands suggested the elimination of *Big-Bang* and similar programs from Gamma as a potential solution, because only a program like *Big-Bang* acting as environment could make possible the change from $\emptyset$ to a non-empty M, as it happens in the traces of $\mathit{Skip} \circ \mathit{Await}$. This has the disadvantage of

ruling out a program that seems perfectly natural. In any case, his suggestion has been neither proven nor disproven.

An alternative approach is to restrict not the set of Gamma programs but the arbitrariness of the environment. Observe that the so-called environment is nothing but another Gamma program executed in parallel. Therefore, it is reasonable not to expect any behaviour which is impossible for a Gamma program.

Sands's (1993*a*) counterexample arose from considering the interference from a program acting as *Big-Bang* which, at the same time, would do something *Big-Bang* cannot: terminate and vanish into thin air. Hence, arbitrary environments are at the root of the problem.

Our own approach to the task of building the denotation of a program shall proceed in three steps:

1. We will begin with the transitions made by an isolated Gamma program.
2. The interference of all possible environments (ie, all Gamma programs and nothing but Gamma programs) will be considered afterwards.
3. The absorption closure of the resulting set of traces will be taken as the denotation of the program.

Let us start with a preliminary definition, which performs step 2 using step 1 as the basis:

Definition 3.2.2 *Given a program P and an environment program P^E, the function $T : \mathbb{G} \times \mathbb{G} \to \mathcal{P}(\mathbb{T}(\mathbb{M}(D)))$ produces the set of transition traces resulting from the interaction of the two programs:*

$$T(P, P^E) = \{(M_1, M_2) \ldots (M_{n-1}, M_n)(N_1, N_2)\alpha\} \mid \\ P = P_1, \langle P_i, M_i\rangle \to \langle P_{i+1}, M_{i+1}\rangle, \langle P^E, M_n\rangle \to^* \langle P^{E'}, N_1\rangle \\ \textit{and } (N_1, N_2)\alpha \in T(P_n, P^{E'}).$$

If $P^E = \mathit{Skip}$, we obtain the strict traces of Sands (1993*a*), that is, traces which correspond to the execution of an isolated program (as *Skip* does not perform any change in the multiset).

The definition has some of the ingredients we need, but also has a big shortcoming, namely, it does not show how to build the denotation of a program from the denotation of its components. For instance, if $P = (P_1 + P_2) \circ P_3$, there is no way of using the syntactic structure of P to calculate its traces from those of P_1, P_2 and P_3 (assuming we have already performed the easier task of calculating their traces). This fact makes it extremely difficult to reason about programs in refinement and correctness proofs (a problem we have already faced with the order $\sqsubseteq_{IO}$).

The ideal situation is to be able to *completely* derive the traces of a program from those of its components. This goal is achieved in Sands (1993*a*) but, as has been said, the resulting model is not fully abstract. A more modest aim is to use the traces of simpler program as pieces which, properly "processed", will be added to get the traces of less simple programs. And that is the purpose of the next section.

3.3 Compositionality of the semantics

With the previous target in mind, a new definition of the function T is offered:

Definition 3.3.1 *The function $T : \mathbb{G} \times \mathbb{G} \to \mathcal{P}(\mathbb{T}(\mathbb{M}(D)))$ calculates the traces resulting from the interaction of a program P and an environment P^E. T is defined by induction on the syntax of P. Also, two cases are taken into account in each instance of P: when $P^E = Skip$ and when $P^E \neq Skip$.*

$$
\begin{aligned}
T(A \Leftarrow R, Skip) = &\{(M,N)(N,N')\alpha \mid R(M) \text{ and } N \in S(A \Leftarrow R, M) \\
&\quad \text{and } (N,N')\alpha \in T(A \Leftarrow R, Skip)\} \\
&\cup \{(M,M) \mid R(M) \text{ does not hold}\}
\end{aligned}
$$

$$
\begin{aligned}
T(A \Leftarrow R, P^E) = &\{(M_1,M_2)\ldots(M_{n-1},M_n)(N_1,N_2)\alpha \mid \text{ there are } \gamma, \eta \text{ with} \\
&(M_1,M_2)\ldots(M_{n-1},M_n)\gamma \in T(A \Leftarrow R, Skip) \\
&\text{and } (M_n,N_1)\eta \in T(P^E, Skip),\ \eta \in T(P^{E'}, Skip) \\
&\text{and } (N_1,N_2)\alpha \in T(A \Leftarrow R, P^{E'})\} \\
&\cup \{(M,M) \mid \langle A \Leftarrow R, M\rangle \to M \text{ and } \langle P^E, M\rangle \to M\}
\end{aligned}
$$

$$
T(Q \circ P, Skip) = \{\alpha \odot \beta \mid \alpha \in T(P, Skip) \text{ and } \beta \in T(Q, Skip)\}
$$

$$
\begin{aligned}
T(Q \circ P, P^E) = &\{\alpha \odot \beta \mid \textit{there are } \gamma, P' \textit{ and } P^{E'} \textit{ such that} \\
&\alpha\gamma \in T(P, P^E) \textit{ and } \gamma \in T(P', P^{E'}) \textit{ and } \beta \in T(Q, P^{E'}) \\
&\textit{and } \alpha = (M_1, M_2) \ldots (M_{n-1}, M_n) \textit{ and } \langle P', M_n \rangle \to M_n\} \\
T(P + Q, Skip) = &\{\alpha \odot \beta \mid \textit{there are } \gamma, P' \textit{ such that } \alpha\gamma \in T(P, Skip) \\
&\textit{and } \gamma \in T(P', Skip) \textit{ and } \beta \in T(P' + Q, Skip)\} \\
&\cup \\
&\{\alpha \odot \beta \mid \textit{there are } \gamma, Q' \textit{ such that } \alpha\gamma \in T(Q, Skip) \\
&\textit{and } \gamma \in T(Q', Skip) \textit{ and } \beta \in T(P + Q', Skip)\} \\
T(P + Q, P^E) = &\{\alpha \odot \beta \mid \textit{there are } \gamma, P', P^{E'} \textit{ such that } \alpha\gamma \in T(P, P^E) \\
&\textit{and } \gamma \in T(P', P^{E'}) \textit{ and } \beta \in T(P' + Q, P^{E'})\} \\
&\cup \\
&\{\alpha \odot \beta \mid \textit{there are } \gamma, Q', P^{E'} \textit{ such that } \alpha\gamma \in T(Q, P^E) \\
&\textit{and } \gamma \in T(Q', P^{E'}) \textit{ and } \beta \in T(P + Q', P^{E'})\}.
\end{aligned}
$$

In the above definition (case $T(A \Leftarrow R, Skip)$), "$R(M)$ holds" means that there exist a subset $\{\!|x_1, \ldots, x_n|\!\} \subseteq M$ such that $R(x_1, \ldots, x_n)$ is true. In the case $T(A \Leftarrow R, P^E)$, the combined facts that $(M_n, N_1)\eta \in T(P^E, Skip)$ and $\eta \in T(P^{E'}, Skip)$ imply that $\langle P^E, M_n \rangle \to \langle P^{E'}, N_1 \rangle$.

The function T will serve as the basis for another program order (*the* denotational order):

Definition 3.3.2 *Let* $T_{fin}(P, P^E) = \{\alpha \in T(P, P^E) \mid \alpha \textit{ is finite}\}$ *and let* P_1, $P_2 \in \mathbb{G}$. *Then:*

$$P_1 \sqsubseteq_T P_2 \qquad \textit{iff} \qquad T_{fin}(P_1, P^E) \subseteq T_{fin}(P_2, P^E) \qquad \textit{for every } P^E \in \mathbb{G}.$$

The above definition needs three remarks. First of all, it is environment-dependent: the sets of traces compared for inclusion are built using the same environment. Secondly, only finite traces are taken into account, meaning that all non-terminating behaviour is made equivalent. While there are situations where this is not a proper strategy, all of our operational orders have the same limitation. In order to prove full abstraction later on, it is unreasonable to expect stronger properties from the denotational order than those required at operational level and then we are just equalizing the demands on both sides (operational and denotational). Thirdly, Sands'

counterexample to full abstraction no longer holds: $Skip \circ Await \sqsubseteq_T Await$ because there are no $P^E \in \mathbb{G}$ and $M \in \mathbb{M}(D)$ such that

$$(\emptyset, \emptyset)(M, M) \in T_{fin}(Skip \circ Await, P^E).$$

To conclude this section, the full denotation of a program is presented:

Definition 3.3.3 *If $P \in \mathbb{G}$, its denotation is defined by the function $[\![\ ]\!] : \mathbb{G} \to \mathcal{P}(\mathbb{T}(\mathbb{M}(D)))$:*

$$[\![P]\!] = \bar{A}(\bigcup_{P^E \in \mathbb{G}} T(P, P^E)).$$

The absorption closure is included because it allows us to take the reflexive and transitive closure of $\to$ instead of just $\to$ when a program (either the main one or the environment) is acting without temporary interference from the other.

3.4 Full abstraction

It is almost time to present the crucial theorem in this chapter: full abstraction of our transition trace semantics. Before proceeding we will need another definition and a short lemma.

When two programs P and P^E are executed in parallel, but with *Skip* as the environment, a strict trace is obtained. However, if we are interested only in the transitions made by the first program, the others should be eliminated from the trace. The next function performs this task:

$$E(P^E, \epsilon) = \epsilon$$

$$E(P^E, (M_1, M_2)\alpha) = \begin{cases} E(P^{E'}, \alpha) & \text{if } \langle P^E, M_1 \rangle \to^* \langle P^{E'}, M_2 \rangle \\ & \text{and } \alpha \in T(P, P^{E'}) \\ (M_1, M_2)E(P^E, \alpha) & \text{otherwise.} \end{cases}$$

If $\alpha = (M_1, M_2) \ldots (M_{n-1}, M_n) \ldots$ then $E(P^E, \alpha)$ looks like

$$(M_{k_1}, M_{k_1+1})(M_{k_1+k_2}, M_{k_1+k_2+1}) \ldots (M_{\sum_{i=1}^{m} k_i}, M_{1+\sum_{i=1}^{m} k_i}) \ldots$$

where $1 \leq k_1$. Moreover, there are P_1^E, P_2^E, ..., such that if

$$M_{1+\sum_{i=1}^{j} k_i} \neq M_{\sum_{i=1}^{j+1} k_i}$$

then

$$\langle P_j^E, M_{1+\sum_{i=1}^{j} k_i} \rangle \rightarrow^* \langle P_{j+1}^E, M_{\sum_{i=1}^{j+1} k_i} \rangle.$$

Now, if $k_1 = 1$ then $P_1^E = P^E$. On the other hand, if α is of finite length $n/2$ and $1 + \sum_{i=1}^{m} k_i < n$ then $\langle P_p^E, M_{1+\sum_{i=1}^{m} k_i} \rangle \rightarrow^* M_n$.

Additionally, there are $P = P_1, P_2, \ldots$ such that

$$\langle P_j, M_{\sum_{i=1}^{j} k_i} \rangle \rightarrow \langle P_{j+1}, M_{1+\sum_{i=1}^{j} k_i} \rangle.$$

Basically E converts a trace produced by $P + P^E$ into a (prefix of a) trace produced by P with P^E as the environment. The next lemma states and proves this fact formally:

Lemma 3.4.1 *Let $\alpha = (M_1, M_2) \ldots (M_{n-1}, M_n)$ and $E(P^E, \alpha) = (M_{k_1}, M_{k_1+1}) \ldots (M_{\sum_{i=1}^{m} k_i}, M_{1+\sum_{i=1}^{m} k_i})$. Let P_1^E be as defined by the function E in the previous paragraphs. Now*

1. *If $1 + \sum_{i=1}^{m} k_i = n$ then*

$$\alpha \in T(P + P^E) \qquad \textit{if and only if} \qquad E(P^E, \alpha) \in T(P, P_1^E).$$

2. *If $1 + \sum_{i=1}^{m} k_i < n$ then*

$$\alpha \in T(P + P^E) \qquad \textit{if and only if} \qquad E(P^E, \alpha)(M_n, M_n) \in T(P, P_1^E).$$

Proof. Case $1 + \sum_{i=1}^{m} k_i = n$. Assume $\alpha \in T(P + P^E, \mathit{Skip})$. The function E just eliminates all transitions made by P^E and its successors. That is, we are considering P^E as the environment. Nevertheless, as it is not necessarily the case that $k_1 = 1$, then our initial environment is P_1^E and not P^E itself. Then $E(P^E, \alpha) \in T(P, P_1^E)$.

Now suppose $E(P^E, \alpha) \in T(P, P_1^E)$. If $k_1 = 1$ then it is trivial that $\alpha \in T(P + P^E)$ as $P^E = P_1^E$ and α does contain the transitions made by P^E. If now $k_1 > 1$ then $\langle P^E, M_1 \rangle \rightarrow^* \langle P_1^E, M_{k_1} \rangle$ and $(M_1, M_2) \ldots (M_{k_1-2}, M_{k_1-1})\gamma \in T(P^E, \mathit{Skip})$ for a certain $\gamma \in T(P_1^E, \mathit{Skip})$. Again, all transitions made by P_1^E and its successors afterwards are also "restored" in α. Then, by definition of T (case $T(P + Q, \mathit{Skip})$) we conclude that $\alpha \in T(P + P^E, \mathit{Skip})$.

Case $1 + \sum_{i=1}^{m} k_i < n$. Let $\alpha \in T(P + P^E, Skip)$. Again, P^E has become the environment of P. But in this case, as the environment does not terminate with $M_{1+\sum_{i=1}^{m} k_i}$ (by hypothesis) we need to add (M_n, M_n) at the end of $E(P^E, \alpha)$ to get a terminating state. For the rest, the arguments of case 1 apply again and then $E(P^E, \alpha)(M_n, M_n) \in T(P, P_1^E)$.

Let $E(P^E, \alpha)(M_n, M_n) \in T(P, P_1^E)$. The same arguments as in case 1 holds here and so $\alpha \in T(P + P^E, Skip)$. ■

At long last we have the expected theorem:

Theorem 3.4.2 *Soundness and full abstraction. Let P, $Q \in \mathbb{G}$. Then*

$$P \sqsubseteq_T Q \qquad \textit{if and only if} \qquad P \sqsubseteq_C Q.$$

Proof. Suppose first that $P \sqsubseteq_T Q$. We need to prove that $C[P] \sqsubseteq_{IO} C[Q]$ holds for every context C. The proof will use induction on contexts.

$C = [\ ]$. If $\langle P, M_1\rangle \to^* M_n$ then there exists a trace $(M_1, M_2)\ldots(M_{n-1}, M_n) \in T(P, Skip)$. Then $(M_1, M_2)\ldots(M_{n-1}, M_n) \in T(Q, Skip)$, which implies that $\langle Q, M_1\rangle \to^* M_n$ and $P \sqsubseteq_{IO} Q$.

$C = [\ ] + R$. Now assume $\langle P + R, M_1\rangle \to^* M_n$. Again, there is a trace $\alpha = (M_1, M_2)\ldots(M_{n-1}, M_n) \in T(P+R, Skip)$. By lemma 3.4.1, $E(R, \alpha)(M_n, M_n) \in T(P, R_1)$ or $E(R, \alpha) \in T(P, R_1)$. Therefore we have that $E(R, \alpha)(M_n, M_n) \in T(Q, R_1)$ or $E(R, \alpha)(M_n, M_n) \in T(Q, R_1)$ (by hypothesis). In either case, it follows from lemma 3.4.1 that $\alpha \in T(Q + R, Skip)$ and then $\langle Q + R, M_1\rangle \to^* M_n$. In brief, $P + R \sqsubseteq_{IO} Q + R$.

$C = R + [\ ]$ is the same as $+$ is commutative.

$C = R \circ [\ ]$. If $\langle R \circ P, M_1\rangle \to^* M_n$ there is a trace $(M_1, M_2)\ldots(M_{n-1}, M_n) \in T(R \circ P, Skip)$. But by definition of T, there exists an $i < n$ such that

$$(M_1, M_2)\ldots(M_{i-1}, M_i) \in T(P, Skip) \qquad \text{and}$$
$$(M_i, M_{i+1})\ldots(M_{n-1}, M_n) \in T(R, Skip).$$

As $P \sqsubseteq_T Q$, then

$$(M_1, M_2)\ldots(M_{i-1}, M_i) \in T(Q, Skip) \qquad \text{and therefore}$$
$$(M_1, M_2)\ldots(M_{n-1}, M_n) \in T(R \circ Q, Skip),$$

which implies $\langle R \circ Q, M_1 \rangle \rightarrow^* M_n$ and then $R \circ P \sqsubseteq_{IO} R \circ Q$.

$C = [\ ] \circ R$. If $\langle P \circ R, M_1 \rangle \rightarrow^* M_n$ then $(M_1, M_2) \ldots (M_{n-1}, M_n) \in T(P \circ R, Skip)$. As in the previous case, there is an $i < n$ such that

$$(M_1, M_2) \ldots (M_{i-1}, M_i) \in T(R, Skip) \qquad \text{and}$$
$$(M_i, M_{i+1}) \ldots (M_{n-1}, M_n) \in T(P, Skip).$$

The rest of the proof is analogous to the previous case and then $P \circ R \sqsubseteq_{IO} Q \circ R$.

In conclusion, $P \sqsubseteq_C Q$.

Now let $P \sqsubseteq_C Q$ and $\alpha = (M_1, M_2) \ldots (M_{n-1}, M_n) \in T(P, R)$ for a certain R. By definition of E, there is a

$$\beta = (M_1, M_2)(M_1', M_3)(M_3, M_4) \ldots (M_m', M_{n-1})(M_{n-1}, M_n)$$

such that $E(R, \beta) = \alpha$. One more time, according to lemma 3.4.1 $\beta \in T(P + R, Skip)$. In other words $\langle P + R, M_1 \rangle \rightarrow^* M_n$ and by hypothesis $\langle Q + R \rangle \rightarrow^* M_n$. Hence $\beta \in T(Q + R, Skip)$. One last application of 3.4.1 produces $\alpha = E(R, \beta) \in T(Q, R)$ and therefore $P \sqsubseteq_T Q$. ■

Corollary 3.4.3 *The laws 3.1.2 1, 2, 3′ and 4–15 are valid for* $\sqsubseteq_T$.

Proof. The laws are valid for $\sqsubseteq_t$ (see Sands (1993*b*)) and for all $P\ Q \in \mathbb{G}$, $P \sqsubseteq_t Q$ implies $P \sqsubseteq_C Q$, which in its turn entails $P \sqsubseteq_T Q$. ■

3.5 Domain constructions for the semantics

The only remaining stage before introducing the proof system for Gamma is to give a domain-theoretical account of the denotational semantics (using the metalanguage of section 2.4.2). Let us start with some notation:

$\mathcal{M}(D)$ domain of finite multisets: It is $\mathbb{M}(D)$ with a domain order, although in chapter 4 we will be less demanding and will accept a poset of multisets

$\mathcal{A}$	domain of actions based on $\mathbb{A}$
$\mathcal{R}$	domain of reaction conditions built upon $\mathbb{R}$
$\mathcal{G}$	domain of Gamma programs (though its internal structure is not important we can think of it as a flat domain built on $\mathbb{G}$)
$\mathcal{T}(\mathcal{M}(D))$	domain of traces: again it is basically $\mathbb{T}(\mathbb{M}(D))$ with a domain order

$\mathcal{P}(\mathbb{T}(\mathbb{M}(D)))$ will also play a role, for the denotations of programs are sets of traces. Nevertheless, the power set lacks a particular order (other than inclusion) or, rather, it can have many. We will use the lower powerdomain of $\mathcal{T}(\mathcal{M}(D))$ (see definition 2.4.8). The reason to choose the lower powerdomain is that it comes with the proper modal operator we need for the logic (more about this in chapter 4).

We shall start the translation with the basic operations on traces (whose definition is taken for granted):

link : $\mathcal{T}(\mathcal{M}(D)) \times \mathcal{T}(\mathcal{M}(D)) \to \mathcal{T}(\mathcal{M}(D))$	the same as $\odot$
append : $\mathcal{T}(\mathcal{M}(D)) \times \mathcal{T}(\mathcal{M}(D)) \to \mathcal{T}(\mathcal{M}(D))$	as concatenation
absorb $\subseteq \mathcal{T}(\mathcal{M}(D)) \times \mathcal{T}(\mathcal{M}(D))$	as absorption
absorbset : $\mathcal{P}_l(\mathcal{T}(\mathcal{M}(D))) \to \mathcal{P}_l(\mathcal{T}(\mathcal{M}(D)))$	absorption closure of a set
length : $\mathcal{T}(\mathcal{M}(D)) \to \mathbb{N}$	length of traces.

The next ones are less trivial. Given a trace, prefix_n calculates its prefix of length n (or produces the zero-length trace ϵ in case the original trace is shorter than n):

$\mathsf{prefix}_n : \mathcal{T}(\mathcal{M}(D)) \to \mathcal{T}(\mathcal{M}(D))$
$\mathsf{prefix}_n((M_1, M_2)\ldots) =$
if $\mathsf{less}(\mathsf{length}((M_1, M_2)\ldots), n)$
then ϵ
else $(M_1, M_2)\ldots(M_{2n-1}, M_{2n})$
fi
end

allprefix$_n$ calculates the prefixes of length n for all the traces in a given set.

allprefix$_n$: $\mathcal{P}_l(\mathcal{T}(\mathcal{M}(D))) \to \mathcal{P}_l(\mathcal{T}(\mathcal{M}(D)))$
allprefix$_n(T)$ =
over T **extend** α.
  prefix$_n(\alpha)$
end

The next one does the same as allprefix$_n$, for all n.

allprefix : $\mathcal{P}_l(\mathcal{T}(\mathcal{M}(D))) \to \mathcal{P}_l(\mathcal{T}(\mathcal{M}(D)))$
allprefix(T) =
over $\mathbb{N}$ **extend** n . allprefix$_n(T)$
end

tabsorb is just total absorption in metalanguage presentation:

tabsorb : $\mathcal{T}(\mathcal{M}(D)) \to \mathcal{T}(\mathcal{M}(D))$
tabsorb$((M_1, M_2)(M_3, M_4)\alpha)$ =
μ tabsorb.
  if $M_2 = M_3$
    then
      tabsorb$((M_1, M_4)\alpha)$
    else
      append$((M_1, M_2),$ tabsorb$((M_3, M_4)\alpha))$
  fi
end

The third set of functions corresponds to the isolated execution of programs. We begin with some auxiliary functions. strict$_n$ (defined recursively) calculates the prefixes of length n of the strict traces of a program P, together with the residual part of P after the n-th transition. strict does the same for all n:

strict$_1$: $\mathcal{G} \to \mathcal{P}_l(\mathcal{T}(\mathcal{M}(D)) \times \mathcal{G})$
strict$_1(P_1)$ =
over $\mathcal{M}(D) \times \mathcal{M}(D) \times \mathcal{G}$ **extend** M_1, M_2, P_2.
  if $\langle P_1, M_1\rangle \to \langle P_2, M_2\rangle$ **then** $((M_1, M_2), P_2)$ **fi**
end

$\mathsf{strict}_{n+1} : \mathcal{G} \to \mathcal{P}_l(\mathcal{T}(\mathcal{M}(D)) \times \mathcal{G})$
$\mathsf{strict}_{n+1}(P_1) =$
over $\mathsf{strict}_n(P_1)$ **extend** $((M_1, M_2) \ldots (M_n, M_{n+1}), P_{n+1})$.
  if $\langle P_{n+1}, M_{n+1}\rangle \to \langle P_{n+2}, M_{n+2}\rangle$
    then $((M_1, M_2) \ldots (M_{n+1}, M_{n+2}), P_{n+2})$
  fi
end

$\mathsf{strict} : \mathcal{G} \to \mathcal{P}_l(\mathcal{T}(\mathcal{M}(D)) \times \mathcal{G})$
$\mathsf{strict}(P) =$
over $\mathbb{N}$ **extend** $n.\mathsf{strict}_n(P)$ **end**

Given a program P and a trace of length greater or equal to n, psucc_n checks if the n-th transition in the trace has been executed by P and, if so, produces the successor(s) of the program P after the n-th transition.

$\mathsf{psucc}_1 : \mathcal{T}(\mathcal{M}(D)) \times \mathcal{G} \to \mathcal{P}_l(\mathcal{G})$
$\mathsf{psucc}_1((M_1, M_2) \ldots, P_1) =$
over $\mathcal{G}$ **extend** P_2.
  if $\langle P_1, M_1\rangle \to \langle P_2, M_2\rangle$ **then** P_2 **fi**
end

$\mathsf{psucc}_{n+1} : \mathcal{T}(\mathcal{M}(D)) \times \mathcal{G} \to \mathcal{P}_l(\mathcal{G})$
$\mathsf{psucc}_{n+1}((M_1, M_2) \ldots, P_1) =$
if $\mathsf{less}(\mathsf{length}((M_1, M_2) \ldots), n)$
  then $\emptyset$ **else**
      over $\mathcal{G} \times \mathsf{psucc}_n((M_1, M_2) \ldots, P_1)$ **extend** P_{n+1}, P_n.
        if $\langle P_n, M_{2n-1}\rangle \to \langle P_{n+1}, M_{2n}\rangle$
          then P_{n+1}
        fi
  fi
end

The next function accounts for the evolution of the environment in the interaction with a given program. env_n calculates the value of the environment after the latter has performed its n-th transition:

$\mathsf{env}_1 : \mathcal{T}(\mathcal{M}(D)) \times \mathcal{G} \to \mathcal{P}(\mathcal{G})$
$\mathsf{env}_1((M_1, M_2) \ldots, P_1^E) =$
if less(length$((M_1, M_2) \ldots), 2)$
then $\emptyset$
else
if equal(M_2, M_3)
then $\{P_1^E\}$
else
over $\mathcal{G}$ **extend** P_2^E.
if $\langle P_1^E, M_2\rangle \to^* \langle P_2^E, M_3\rangle$ **then** P_2^E **fi**
fi
fi
end

$\mathsf{env}_{n+1} : \mathcal{T}(\mathcal{M}(D)) \times \mathcal{G} \to \mathcal{P}(\mathcal{G})$
$\mathsf{env}_{n+1}((M_1, M_2) \ldots, P_1^E) =$
if less(length$((M_1, M_2) \ldots), n+2)$
then $\emptyset$
else
if equal(M_{2n-2}, M_{2n-1})
then $\mathsf{env}_n((M_1, M_2) \ldots, P_1^E)$
else
over $\mathcal{G} \times \mathsf{env}_n((M_1, M_2) \ldots, P_1^E)$ **extend** P_{n+1}^E, P_{n+2}^E.
if $\langle P_{n+1}^E, M_{2n-2}\rangle \to^* \langle P_{n+2}^E, M_{2n-1}\rangle$
then P_{n+2}^E
fi
fi
fi
end

Now a functional version of the original $S : \mathbb{A} \times \mathbb{R} \times \mathbb{M}(D) \to \mathcal{P}(\mathbb{M}(D))$:

$\mathsf{successor} : \mathcal{A}^n \times \mathcal{R}^n \times \mathcal{M}(D) \to \mathcal{P}(\mathcal{M}(D))$
$\mathsf{successor}(A, R, M) =$
over $\mathcal{P}_n(M)$ **extend** $\bar{x}$.
  if $R(\bar{x})$ **then** $(M - \bar{x}) \uplus A(\bar{x})$ **else** $\emptyset$ **fi**
end

The next function produces all the traces of an atomic reaction starting with a given multiset:

$\mathsf{chain} : \mathcal{A} \times \mathcal{R} \times \mathcal{M}(D) \to$
$\mathcal{P}_l(\mathcal{T}(\mathcal{M}(D)))$
$\mathsf{chain}(A, R, M) =$
$\mu\,\mathsf{chain}.$
  if $R(M)$
    then
      over $\mathsf{successor}(A \Leftarrow R, M)$ **extend** N.
        $\mathsf{append}((M, N), \mathsf{chain}(A, R, N))$
    else (M, M)
  fi
end

The fourth set of functions relates programs and environments (including *Skip*).

Firstly, we have an atomic reaction with an empty environment using chain as the basic component:

$\mathsf{ARSkip} : \mathcal{A} \times \mathcal{R} \to \mathcal{P}_l(\mathcal{T}(\mathcal{M}(D)))$
$\mathsf{ARSkip}(A, R) =$
over $\mathcal{M}(D)$ **extend** M . $\mathsf{chain}(A, R, M)$
end

All of the following functions use the new function combine, whose definition will be given almost at the end of this section.

We consider now an atomic reaction with a non-trivial environment. Note how the strict traces are used to build non-strict traces:

$\mathsf{AROther}(A, R, P^E) =$
over $\mathsf{strict}(A \Leftarrow R) \times \mathsf{strict}(P^E)$ **extend** $((M_1, M_2) \ldots (M_{n-1}, M_n), P_n)$,
$((N_1, N_2) \ldots (N_{m-1}, N_m), P_m^E)$.
if $M_n = N_1$
then
over $\mathsf{combine}(A \Leftarrow R, P_m^E)$ **extend** $(P_1, P_2)\alpha$.
if $N_m = P_1$
then $\mathsf{append}((M_1, M_2) \ldots (M_{n-1}, M_n), (P_1, P_2)\alpha)$
fi
end

Next we consider the sequential composition in an empty environment:

$\mathsf{SCSkip} : \mathcal{G} \times \mathcal{G} \to \mathcal{P}_l(\mathcal{T}(\mathcal{M}(D)))$
$\mathsf{SCSkip}(P, Q) =$
over $\mathsf{combine}(P, skip) \times \mathsf{combine}(Q, skip)$ **extend** α, β.
$\mathsf{link}(\alpha, \beta)$
end

Then sequential composition with an environment different from *Skip*. This is the point where the auxiliary functions $\mathsf{allprefix}_n$, psucc_n and env_n justify their existence:

$\mathsf{SCOther} : \mathcal{G} \times \mathcal{G} \times \mathcal{G} \to \mathcal{P}_l(\mathcal{T}(\mathcal{M}(D)))$
$\mathsf{SCOther}(P, Q, P^E) =$
over $\mathbb{N}$ **extend** n.
over $\mathsf{allprefix}_n(\mathsf{combine}(P, P^E))$ **extend** $(M_1, M_2) \ldots (M_{2n-1}, M_{2n})$.
over $\mathsf{psucc}_{n+1}((M_1, M_2) \ldots (M_{2n-1}, M_{2n}), P_1) \times$
$\mathsf{env}_{n-1}((M_1, M_2) \ldots (M_{2n-1}, M_{2n}), P^E)$ **extend** $P', P^{E'}$.
over $\mathsf{combine}(Q, P^{E'})$ **extend** β.
if $\langle P', M_{2n} \rangle \to M_{2n}$
then $\mathsf{link}((M_1, M_2) \ldots (M_{2n-1}, M_{2n}), \beta)$
fi
end

Next we consider parallel composition with *Skip*:

PCSkip : $\mathcal{G} \times \mathcal{G} \rightarrow \mathcal{P}_l(\mathcal{T}(\mathcal{M}(D)))$
PCSkip$(P, Q) =$
over allprefix(combine$(P, Skip)$) **extend** α.
  over psucc$_{\text{length}(\alpha)}(\alpha, P)$ **extend** P'.
    over combine$(P' + Q, Skip)$ **extend** β.
      link(α, β)
$\bigcup$
over allprefix(combine$(Q, Skip)$) **extend** α.
  over psucc$_{\text{length}(\alpha)}(\alpha, Q)$ **extend** Q'.
    over combine$(P + Q', Skip)$ **extend** β.
      link(α, β)
end

We now present parallel composition in a more complex environment:

PCOther : $\mathcal{G} \times \mathcal{G} \rightarrow \mathcal{P}_l(\mathcal{T}(\mathcal{M}(D)))$
PCOther$(P, Q, P^E) =$
over allprefix(combine(P, P^E)) **extend** α.
  over psucc$_{\text{length}(\alpha)}(\alpha, P) \times$ env$_{\text{length}(\alpha)}(\alpha, P^E)$ **extend** $P', P^{E'}$.
    over combine$(P' + Q, P^{E'})$ **extend** β.
      link(α, β)
  $\bigcup$
  over allprefix(combine(Q, P^E)) **extend** α.
    over psucc$_{\text{length}(\alpha)}(\alpha, Q) \times$ env$_{\text{length}(\alpha)}(\alpha, P^E)$ **extend** $Q', P^{E'}$.
      over combine$(P + Q', P^{E'})$ **extend** β.
        link(α, β)
      end

Finally, the most important function: combine.

combine : $\mathcal{G} \times \mathcal{G} \rightarrow \mathcal{P}_l(\mathcal{T}(\mathcal{M}(D)))$
combine$(P, P^E) =$
μ combine.
case P **of** $(A \Leftarrow R) \Rightarrow$
if equal$(P^E, Skip)$
then ARSkip(A, R)
else AROther(A, R, P^E)
fi;
$Q \circ P \Rightarrow$
if equal$(P^E, Skip)$
then SCSkip(P, Q)
else SCOther(P, Q, P^E)
fi;
$P + Q \Rightarrow$
if equal$(P^E, Skip)$
then PCSkip(P, Q)
else PCOther(P, Q, P^E)
fi;
end
end

Our presentation concludes with the following:

Definition 3.5.1 *The denotation of a program P is:*

$$[\![P]\!] = \text{absorbset}(\bigcup_{P^E \in \mathcal{G}} \text{combine}(P, P^E)).$$

We conclude this chapter with a short summary of results: starting with the operational semantics of Gamma, some program orders were introduced. Then, a Gamma denotational semantics based in the transition

trace model was defined and proved fully abstract. The laws 3.1.2, originally proved for an operational order, were extended to the denotational order thanks to full abstraction. Finally, using the metalanguage of section 2.4.2, we presented a domain theoretical version of the semantics. We are now prepared for the Gamma logic.

Chapter 4

Gamma logic

All the concepts needed to present the Gamma proof system have been introduced in the the last two chapters: the (general) framework of DTLF and a denotational semantics of Gamma.

As we said in chapter 2, the Gamma logic will be built in a hierarchical fashion:

- At the bottom there will be a series of proof systems corresponding to basic types: numbers, tuples, booleans, etc. The existence of these systems will be taken for granted.
- A multiset logic with formation rules, axioms and inference rules will be the second level.
- The next layer will consist of a logic about the execution of Gamma programs. Given that the denotation of a Gamma program is a set of traces, these are also the basic objects in the language of assertions and, therefore, the whole system will be called the *transition trace logic*.
- At the top there will be a couple of theorems connecting the logic with more traditional approaches in the field of program logics (namely, the use of invariants and termination conditions).

The content of this chapter reflects these goals. The multiset logic is introduced in the first section. Soundness and (conditional) completeness

of the logic are proved afterwards. Reaction conditions in atomic rules are reformulated in the new language (giving a unified framework to a hitherto heterogeneous way of expressing reactions). At the end of the section, the examples of Gamma programs of chapter 2 will be restated in terms of the multiset logic.

The next section will deal with the transition trace logic. Soundness and completeness are also proved, this time extending the results up to the operational semantics.

The third section will discuss a basic approach to prove termination of the execution of Gamma programs. It is basically a generalization of the technique advocated by Banâtre & Le Métayer (1990).

The chapter concludes with some examples of correctness proofs for Gamma programs.

There have not been many examples of multiset logics. Libkin & Wong (1995) proposed a language to express properties of multisets in the context of database representations, although it was not a proof system. Hernández Quiroz (1998) offered a first version of the multiset logic developed in this thesis.

As has been mentioned, Banâtre & Le Métayer (1990) brought the use of invariants, termination conditions and postconditions to Gamma programming. At the core of their method was the multiset order of Dershowitz & Manna (1979). Nevertheless, their proposal was restricted to atomic rules and there was no systematic way to generalize it to sequentially composed programs.

Errington et al. (1993) presented a logic based on an axiomatic semantics in the style of Hoare. Using a denotational semantics of resumptions and DTLF, Gay & Hankin (1996*b*) restated the former logic, this time with a proof of soundness. Unfortunately, the resulting language of assertions and proof rules were full of intricate details and they were too closely coupled to execution of the programs, making the use of the logic very difficult.

Gay & Hankin (1996*a*) made a further attempt, now using a transition trace semantics as the basis. Their new proof system was more manageable, but due to the lack of a proper multiset logic, two of the most important inference rules remained unproven. It also needed an extension to deal

with more abstract properties of the execution of programs. Our transition trace logic arose as a complement to their system: the remaining rules have been proven, other more abstract rules have been added and a method for proving termination is also included (see section 4.2 and 4.3).

Chaudron (1998) followed a very different route. He took the Unity logic of Chandy & Misra (1988) as the basis for his system. It included a basic logical language for multisets (although not in an axiomatic presentation) and a termination condition (again, not very easy to generalize to sequentially composed programs).

On the other hand, the temporal semantics for Gamma of Reynolds (1996) was intended also as a way of reasoning about program properties (including total correctness). It did not have an axiomatic presentation either, making it difficult to know where a particular proof rule, axiom or theorem came from. It also lacked a formal proof system for multisets.

4.1 Multiset logic

Consider the reaction condition of the atomic rules in the examples 2.2.1, 2.2.2 and 2.2.3:

$$z < k, \qquad x \geq z \wedge y \geq z, \qquad \mathit{true}, \qquad x > 1, \qquad x = 0.$$

They are typical of Gamma programs. Reaction conditions refer to properties of elements in a multiset, not to properties of the multiset itself, except when a minimal cardinality is required (as in the second example). A multiset meets a reaction condition when one of its subsets makes the reaction true, namely, when its elements satisfy the corresponding predicate. This suggests a strategy for designing a multiset logic.

Assume that D is a basic type in Gamma and that $L(D)$ is language of assertions about elements in D. As in DTLF, 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 $\phi \in L(D)$. t and f represent true and false in $L(D)$. For every element $x \in D$ we have:

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

With these ingredients we can present the logic of $\mathbb{M}(D)$.

Definition 4.1.1 *The language of assertions $L(\mathcal{M}(D))$ 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 $L(\mathcal{M}(D))$.

Observe an important difference with respect to domain-theoretical logics: arbitrary (and not just finite) disjunctions are allowed.

The next step would be to introduce logical implication and equivalence relations, together with axioms and inference rules for proving them. But the motivations for the axioms of the logic will be clearer if the satisfaction relation for multisets is introduced before.

Definition 4.1.2 *(Satisfaction). If* $\{\!|x_1, \ldots, x_n|\!\} \in \mathbb{M}(D)$ *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*

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

Just with this definition, we can prove some easy properties of the satisfaction of propositions by multisets.

Theorem 4.1.3 *For every* $M \in \mathbb{M}(D)$*:*

a) If $\{\!|x_1, \ldots, x_n|\!\} \models \Box\{\!|\phi_1, \ldots, \phi_n|\!\}$ *then* $\{\!|x_1, \ldots, x_n|\!\} \uplus M \models \Diamond\{\!|\phi_1, \ldots, \phi_n|\!\}$.

b) $\{\!|x_1, \ldots, x_n|\!\} \models \Diamond\{\!|\phi_1, \ldots, \phi_m|\!\}$ *if and only if there is a* $\{\!|y_1, \ldots, y_m|\!\} \subseteq \{\!|x_1, \ldots, x_n|\!\}$ *such that* $\{\!|y_1, \ldots, y_m|\!\} \models \Box\{\!|\phi_1, \ldots, \phi_m|\!\}$.

c) $M \models \Box\{\!|t|\!\}^m$ *if and only if* $|M| = m$.

d) $M \models \Diamond\{\!|t|\!\}^m$ *if and only if* $|M| \geq m$.

Proof. It is enough to observe that $\{\!|x_1, \ldots, x_n|\!\} \uplus M \models \Box\{\!|\phi_1, \ldots, \phi_n|\!\} \uplus \{\!|t|\!\}^k$, where $|M| = k$. Then $M \models \bigvee_m \Box(\{\!|\phi_1, \ldots, \phi_n|\!\} \uplus \{\!|t|\!\}^m = \Diamond\{\!|\phi_1, \ldots, \phi_n|\!\}$. The other properties follow trivially from this and the definition of $\models$. ■

We now can carry on with the presentation of the proof system.

Definition 4.1.4 *Given a type D, the multiset logic* $\mathcal{L}(\mathcal{M}(D))$ *consists of the well-formed formulas of* $L(\mathcal{M}(D))$ *and the relations of logical implication and equivalence* $\leq$ *and* $=$*, respectively. The following axioms define the way implication and equivalence behave:*

$$(A_1) \qquad \Phi \leq \Phi, \qquad\qquad (A_2) \qquad \frac{\Phi \leq \Psi, \Psi \leq \mathrm{X}}{\Phi \leq \mathrm{X}},$$

$$(A_3)\quad \frac{\Phi \leq \Psi, \Psi \leq \Phi}{\Phi = \Psi}, \qquad (A_4)\quad \frac{\Phi = \Psi}{\Phi \leq \Psi, \Psi \leq \Phi},$$

$$(A_5)\quad \frac{\Phi \leq \Psi_1, \Phi \leq \Psi_2}{\Phi \leq \Psi_1 \wedge \Psi_2}, \qquad (A_6)\quad \Phi \wedge \Psi \leq \Phi,$$

$$(A_7)\quad \Phi \wedge \Psi \leq \Psi, \qquad (A_8)\quad \frac{\forall \Phi \in \{\Phi_i\}_{i\in I}.\ \Phi \leq \Psi}{\bigvee_{i\in I} \Phi_i \leq \Psi},$$

$$(A_9)\quad \frac{\Phi \in \{\Phi_i\}_{i\in I}}{\Phi \leq \bigvee_{i\in I} \Phi_i}, \qquad (A_{10})\quad \Phi \wedge \bigvee_{i\in I} \Psi_i \leq \bigvee_{i\in I} (\Phi \wedge \Psi_i)$$

$$(A_{11})\quad \Box S \wedge \Box T \leq f \qquad \text{if } |S| \neq |T|,$$

$$(A_{12})\quad \Box\{\!|\phi_1, \ldots, \phi_n|\!\} \wedge \Box\{\!|\psi_1, \ldots, \psi_n|\!\} \leq \bigvee_{\sigma\in\Sigma(n)} \Box\{\!|\phi_1 \wedge \psi_{\sigma(1)}, \ldots, \phi_n \wedge \psi_{\sigma(n)}|\!\},$$

$$(A_{13})\quad \Box(S \uplus \{\!|\phi|\!\}) \leq \Box(S \uplus \{\!|\psi|\!\}) \qquad \text{if } \phi \leq \psi,$$

$$(A_{14})\quad \Box(S \uplus \{\!|\bigvee_{i\in I} \phi_i|\!\}) \leq \bigvee_{i\in I} \Box(S \uplus \{\!|\phi_i|\!\}).$$

In spite of some similarity, the axioms A_1–A_{11} are not the same as those found in domain theoretical logics. A very important difference is the inclusion of arbitrary disjunctions, which give $\mathcal{L}(\mathbb{M}(D))$ the structure of a frame and not just that of a lattice.

As an illustrative example of how the axioms work, we have the next theorem and its proof:

Theorem 4.1.5 *The following statements are true:*

a) $\Box\{\!|t|\!\}^m \wedge \Diamond\{\!|\phi_1, \ldots, \phi_n|\!\} \leq f$ *if* $m < n$.

b) $\Box(S \uplus \{\!|\bigvee_{i\in I} \phi_i|\!\}) = \bigvee_{i\in I} \Box(S \uplus \{\!|\phi_i|\!\})$.

c) $\Diamond(S \uplus \{\!|\bigvee_{i\in I} \phi_i|\!\}) = \bigvee_{i\in I} \Diamond(S \uplus \{\!|\phi_i|\!\})$.

Proof. For a) we have:

$$\begin{aligned}
\Box\{\!|t|\!\}^m \wedge \Diamond\{\!|\phi_1, \ldots, \phi_n|\!\} &= \Box\{\!|t|\!\}^m \wedge \bigvee_k \Box(\{\!|\phi_1, \ldots, \phi_n|\!\} \uplus \{\!|t|\!\}^k) && \text{definition}\\
&\leq \bigvee_k \Box\{\!|t|\!\}^m \wedge \Box(\{\!|\phi_1, \ldots, \phi_n|\!\} \uplus \{\!|t|\!\}^k) && A_{10}\\
&\leq f && \text{by } A_{11} \text{ and hypothesis.}
\end{aligned}$$

From A_{13} and the fact that for every i, $\phi_i \leq \bigvee_{i\in I} \phi_i$, we have $\Box(S \uplus \{\!|\phi_i|\!\}) \leq \Box(S \uplus \{\!|\bigvee_{i\in I} \phi_i|\!\})$, also for every i. Then $\bigvee_{i\in I} \Box(S \uplus \{\!|\phi_i|\!\}) \leq \Box(S \uplus \{\!|\bigvee_{i\in I} \phi_i|\!\})$. The other direction of the inequality is A_{14} and we get b). c) follows from b) and the definition of $\Diamond$. ■

The reader must have noticed by now that the multiset logic is very similar to the domain-theoretical logics. To go beyond formal similarity, we would need to give a domain presentation of multisets. Alas, this has proved to be a very difficult question and it is still open. In chapter 5 we will deal with the subject in more detail. In the meantime, one consequence of not having a domain of multisets is that soundness and completeness of the logic do not come "for free". Fortunately, it is possible to prove them even without resorting to DTLF methods.

Theorem 4.1.6 *(Soundness and completeness). For every Φ and Ψ, $\Phi \leq \Psi$ if and only if for every $M \in \mathbb{M}(T)$, $M \models \Phi$ implies $M \models \Psi$.*

Proof. Assume $\Phi \leq \Psi$ by virtue of one of the axioms A_1–A_{14}.

If it is because of A_1–A_5, then it is trivial that $M \models \Phi$ implies $M \models \Psi$.

In the case of A_6, we have that $\Phi = \mathrm{A} \wedge \mathrm{B}$ (for certain A and B) and $\Psi = \mathrm{A}$. But $M \models \mathrm{A} \wedge \mathrm{B}$ implies $M \models \mathrm{A} = \Psi$. The case A_7 is analogous.

In A_8, $\Phi = \bigvee_{i\in I} \Phi_i$ and $M \models \bigvee_{i\in I} \Phi_i$ implies that there is a Φ_j such that $M \models \Phi_j$. Now, by hypothesis, $\Phi_j \leq \Psi$ and then the case is reduced to one of the other axioms.

A_9 is clear, as $M \models \Phi_j$ implies that $M \models \bigvee_{i\in I} \Phi_i$ by definition.

In A_{10}, $\Phi = \mathrm{A} \wedge \bigvee_{i\in I} \Psi_i$. Then $M \models \mathrm{A}$ and $M \models \bigvee_{i\in I} \Psi_i$ implying that there is a Ψ_j such that $M \models \Psi_j$. Then $M \models \mathrm{A} \wedge \Psi_j$ and, by definition, $M \models \bigvee_{i\in I}(A \wedge \Psi_i)$.

For axiom A_{11} we have $\Phi = \Box S \wedge \Box T$ and $\Psi = f$. But no $M \in \mathbb{M}(D)$ can satisfy simultaneously $\Box S$ and $\Box T$ if $|S| \neq |T|$, and the theorem holds by vacuity.

With A_{12}, now $\Phi = \Box\{\!|\phi_1, \ldots, \phi_n|\!\} \wedge \Box\{\!|\psi_1, \ldots, \psi_n|\!\}$ and $\Psi = \bigvee_{\sigma\in\Sigma(n)} \Box\{\!|\phi_1 \wedge \psi_{\sigma(1)}, \ldots, \phi_n \wedge \psi_{\sigma(n)}|\!\}$. Let us suppose that $\{\!|x_1, \ldots, x_n|\!\} \models \phi$, ie, $x_1 \models \phi_{\sigma_1(1)}$, $\ldots x_n \models \phi_{\sigma_1(1)}$ and $x_1 \models \psi_{\sigma_2(1)}$, $\ldots x_n \models \psi_{\sigma_2(n)}$. Then $x_1 \models \phi_{\sigma_1(1)} \wedge \psi_{\sigma_2(1)}$, $\ldots$, $x_n \models \phi_{\sigma_1(n)} \wedge \psi_{\sigma_2(n)}$. In other words $\{\!|x_1, \ldots, x_n|\!\} \models \psi$.

Regarding A_{13}, if $M \models \Box(S \uplus \{\!|\phi|\!\})$ then $M = \{\!|x_1, \ldots, x_n|\!\}$ such that $\{\!|x_1, \ldots, x_{n-1}|\!\} \models \Box S$ and $x_n \models \phi$. Hence $x_n \models \psi$. Consequently $M \models \Box(S \uplus \{\!|\psi|\!\})$.

Finally if $M \models \Box(S \uplus \{\!|\bigvee_i \phi_i|\!\})$ then again $M = \{\!|x_1, \ldots, x_n|\!\}$, with $\{\!|x_1, \ldots, x_{n-1}|\!\} \models \Box S$ and $x_n \models \bigvee_i \phi_i$. Therefore $x_n \models \phi_i$ for some i and then $M \models \Box(S \uplus \{\!|\phi_i|\!\})$ for the same i, which leads directly to the desired conclusion.

Now assume that $M \models \Phi$ implies $M \models \Psi$ for every $M \in \mathbb{M}(D)$. Can we have $\Phi = \Diamond\{\!|\phi_1, \ldots, \phi_n|\!\}$ and $\Psi = \Box\{\!|\psi_1, \ldots, \psi_m|\!\}$? Consider the multiset $\{\!|m_1, \ldots, m_p|\!\}$ and $p = \max(n, m) + 1$. then $M \not\models \Psi$ (definition of $\models$), so this case is not possible.

Now $\Phi = \Box\{\!|\phi_1, \ldots, \phi_n|\!\}$ and $\Psi = \Diamond\{\!|\psi_1, \ldots, \psi_m|\!\}$. Clearly $m \leq n$. Let $\Psi' = \Box(\{\!|\psi_1, \ldots, \psi_m|\!\} \uplus \{\!|t|\!\}^{n-m})$. If $M \models \Phi$ by hypothesis $M \models \Psi$ and, by definition of $\models$, $M \models \Psi'$. That is $m_1 \models \phi_{\sigma_\phi(1)}, \ldots, m_n \models \phi_{\sigma_\phi(m)}$ implies $m_1 \models \psi_{\sigma_\psi(1)}, \ldots, m_n \models \psi_{\sigma_\psi(m)}$. As $\mathcal{L}(D)$ is complete $\phi_{\sigma_\phi(1)} \leq \psi_{\sigma_\psi(1)}, \ldots, \phi_{\sigma_\phi(m)} \leq \psi_{\sigma_\psi(m)}$. For the cases when $i \geq m$ just remember that $\gamma \leq t$ for every γ. Then, applying axiom A_{13} $\Phi \leq \Psi' \leq \Psi$.

Let $\Phi = \Box\{\!|\phi_1, \ldots, \phi_n|\!\}$ and $\Psi = \Box\{\!|\psi_1, \ldots, \psi_m|\!\}$. Because of axiom A_{11} $m = n$. As for every M, $M \models \Phi$ implies $M \models \Psi$, if $M \models \Phi$, that is $m_1 \models \phi_{\sigma_\phi(1)} \ldots, m_n \models \phi_{\sigma_\phi(n)}$ then $m_1 \models \psi_{\sigma_\psi(1)}, \ldots, m_n \models \psi_{\sigma_\psi(n)}$. As $\mathcal{L}(D)$ is complete, this implies $\phi_{\sigma_\phi(1)} \leq \psi_{\sigma_\psi(1)}, \ldots, \phi_{\sigma_\phi(n)} \leq \psi_{\sigma_\psi(n)}$. By A_{13} $\Phi \leq \Psi$.

In the case of $\Phi = \Diamond\{\!|\phi_1, \ldots, \phi_n|\!\}$ and $\Psi = \Diamond\{\!|\psi_1, \ldots, \psi_m|\!\}$ just take any M satisfying $M \models \Phi$, build $M' \subseteq M$ of cardinality n with $M' \models \Phi$ still valid and the argument for the previous case can be repeated. ■

A logic like $\mathcal{L}(D)$ refers only to the satisfaction of a proposition by a single element in D. But predicates in reaction conditions also refer to tuples. How can we extend our multiset logic in this direction?

Let us start assuming the existence of logics for tuples of elements in D. The logic of pairs will be $\mathcal{L}(D \times D)$ and the logic of tuples of length n will be $\mathcal{L}(D^n)$. Now atomic formulae in the multiset logic are built according to this rule:

$$\frac{\phi_1^{m_1} \in L(D^{m_1}), \ldots, \phi_n^{m_n} \in L(D^{m_n})}{\Box\{\!|\phi_1^{m_1}, \ldots, \phi_n^{m_n}|\!\} \in L(\mathcal{M}(D))}.$$

Note that the index on top of the ϕ's does not mean repetition of the same formula as before, but the cardinality of the domain product. Satisfaction of formulas in $L(\mathcal{M}(D))$ by multisets needs to be redefined:

Definition 4.1.7 *Let* $\Phi = \Box\{\!|\phi_1^{m_1}, \ldots, \phi_n^{m_n}|\!\}$, $M = \{\!|m_1, \ldots, m_p|\!\}$ *and* $p = \sum_{i=1}^{n} m_i$. $M \models \Phi$ *if and only if there exist a permutation* σ *such that*

$$(m_{\sigma(1)}, \ldots, m_{\sigma(m_1)}) \models \phi_1^{m_1}, \ldots, (m_{\sigma(p-m_n+1)}, \ldots, m_{\sigma(p)}) \models \phi_n^{m_n}.$$

The proof of soundness and completeness of the multiset logic relies on our ability to prove the satisfaction of a proposition in $\mathcal{L}(D)$ (and now in $\mathcal{L}(D^n)$). This comes from completeness and soundness of $\mathcal{L}(D)$, which holds for DTLF theories. But many interesting reaction conditions cannot be proved to hold with DTLF methods. If $\mathcal{L}(D)$ goes beyond DTLF then completeness is lost (of course, we knew this must be the case) and the loss will be carried to the whole Gamma proof system. Nevertheless, soundness is not affected by this restriction.

Now it is time for some examples of the multiset logic at work. To begin with, all reaction conditions can be formulated in the language $L(\mathcal{M}(D))$ and therefore the examples of chapter 2 will be restated. To save space, a predicate like '$x = a$' will be written just as 'a'.

4.1.8 *Modified version of example 2.2.1:*

$$\begin{aligned} Sier = ((x,y,z)) \rightarrow\ & \{\!|(x/2-2/5, y/2-2/5, z+1), (x/2-2/5, y/2+2/5, z+1), \\ & (x/2+2/5, y/2+2/5, z+1)|\!\} \\ & \Leftarrow \Diamond\{\!|z < k|\!\} \end{aligned}$$

4.1.9 *New version of example 2.2.2:*

$$\begin{aligned} Max &= (x,y,z) \rightarrow \{\!|\max\{x,y,z\}, \max\{\{x,y,z\} - \max\{x,y,z\}\}|\!\} \\ &\quad \Leftarrow \Diamond\{\!|t,t,t|\!\} \\ Prod &= (x,y) \rightarrow \{\!|xy|\!\} \Leftarrow \Diamond\{\!|t,t|\!\} \\ P &= Prod \circ Max. \end{aligned}$$

4.1.10 *Restatement of example 2.2.3:*

$$\begin{aligned} Pred &= x \to \{\!|x-1, x-2|\!\} \Leftarrow \Diamond\{\!|x > 1|\!\} \\ One &= x \to \{\!|1|\!\} \Leftarrow \Diamond\{\!|0|\!\} \\ Sum &= (x,y) \to \{\!|x+y|\!\} \Leftarrow \Diamond\{\!|t,t|\!\} \\ Fib &= Sum \circ (Pred + One). \end{aligned}$$

Some previous ambiguities in the reaction conditions have disappeared. For example, in the example 4.1.9, the program *Prod* used to have *true* as the reaction condition, implying that the program could be executed with any multiset. But this is obviously not true, for the action of *Prod* requires two elements and a multiset with a smaller cardinality would be unsuitable. The new condition $\Diamond\{\!|t,t|\!\}$ clearly establishes this additional requirement. In the case of *Max*, additionally, the reaction condition was mixed with the action. As long as there are three elements in the multiset (whatever their value) *Max* can be applied so it is better to move the calculation of the two maximal elements to the action.

As an illustration of the expressive power of the multiset logic, consider the following propositions:

- Suppose that $m, n \in \mathbb{N}$. We can have the formula $\Box\{\!|m \leq z \wedge n \leq z|\!\}$, which is satisfied by any multiset with a single element p greater than both m and n. This definition can be generalized: let $\{\!|m_1, \ldots, m_n|\!\} \in \mathbb{M}(\mathbb{N})$. The formula
$$\Box\{\!|\bigwedge_{i=1}^{n} m_i \leq z|\!\}$$
is satisfied by any multiset $\{\!|p|\!\}$ such that p is greater than each $m_i \in \{\!|m_1, \ldots, m_n|\!\}$.
- The formula $\Box\{\!|\bigvee_{i=1}^{n} m_i \leq z|\!\}$ is satisfied by a multiset $\{\!|p|\!\}$ such that there is at least one $m_i \in \{\!|m_1, \ldots, m_n|\!\}$ with $m_i \leq p$.
- If $m, n \in \mathbb{N}$, then the formula $\Box\{\!|m+n = z|\!\}$ is satisfied by any multiset $\{\!|p|\!\}$ such that $p = m + n$. Again, if $\{\!|m_1, \ldots, m_n|\!\} \in \mathbb{M}(\mathbb{N})$, then
$$\Box\{\!|\sum_{i=1}^{n} m_i = z|\!\}$$

is satisfied by any multiset $\{\!|p|\!\}$ with $p = \sum_{i=1}^{n} m_i$. Obviously, we can also have the formula

$$\Box\{\!|\prod_{i=1}^{n} m_i = z|\!\}.$$

Further concrete examples of propositions and proofs in the multiset logic shall be offered when the correctness of these programs is proved in the last section.

4.2 Transition trace logic

The relation $\models \subseteq D \times \mathcal{L}(D)$, where D is a domain, was defined in chapter 2 in terms of the denotation of elements in D (which are open sets). Then $x \models \phi$ means that ϕ is a point belonging to the open set denoted by x. According to the semantics of chapter 3, the denotation of a Gamma program is a set of traces, that is, an element of $\mathcal{P}(\mathbb{T}(\mathbb{M}(D)))$. While presenting the metalanguage version of the semantics, we said that the lower powerdomain was the chosen order for sets of traces. The modal operator $\blacklozenge$ arises with the lower powerdomain (see section 2.4.3). In more formal terms:

1 Let D be a domain corresponding to a basic type in Gamma. As we saw in section 3.5, the domain of traces of multisets in D is $\mathcal{T}(\mathcal{M}(D))$. The language of assertions associated with this domain is $L(\mathcal{T}(\mathcal{M}(D)))$. Its elements will be denoted by overlined lower case Greek letters: $\bar{\alpha}$, $\bar{\beta}$, etc.

2 The operations on traces of section 3.2 have their logical counterparts:

$$\frac{\bar{\alpha}, \bar{\beta} \in L(\mathcal{T}(\mathcal{M}(D)))}{\bar{\alpha}\bar{\beta} \in L(\mathcal{T}(\mathcal{M}(D)))} \qquad \frac{\bar{\alpha}, \bar{\beta} \in L(\mathcal{T}(\mathcal{M}(D)))}{\bar{\alpha} \odot \bar{\beta} \in L(\mathcal{T}(\mathcal{M}(D)))} \qquad \frac{\bar{\alpha} \in L(\mathcal{T}(\mathcal{M}(D)))}{\underline{\bar{\alpha}} \in L(\mathcal{T}(\mathcal{M}(D))),}$$

in addition to the formation rules in 2.4.10.

3 Let $t \in \mathcal{T}(\mathcal{M}(D))$ and $\bar{\phi} \in L(\mathcal{T}(\mathcal{M}(D)))$. As usual, $t \models \bar{\phi}$ means that the trace t satisfies the property expressed by $\bar{\phi}$.

4 The denotation of a Gamma program was defined by the function $[\![\ ]\!] : \mathcal{G} \to \mathcal{P}_l(\mathcal{T}(\mathcal{M}(D)))$ in section 3.5. According to the rules of section 2.4.3, if

$$\bar{\phi} \in L(\mathcal{T}(\mathcal{M}(D)))$$

then

$$\blacklozenge\bar{\phi} \in L(\mathcal{P}_l(\mathcal{T}(\mathcal{M}(D)))).$$

5 Let $P \in \mathcal{G}$ and let $\bar{\phi} \in L(\mathcal{T}(\mathcal{M}(D)))$. Then

$$P \models \blacklozenge\bar{\phi}$$

means that there is a $t \in [\![P]\!]$ (that is, there is an execution of P) such that $t \models \bar{\phi}$.

6 Let P, $P^E \in \mathbb{G}$. $P \oplus P^E$ means that the program P is executed in the environment P^E. Needless to say, $\oplus$ is not commutative. Now

$$P \oplus P^E \models \blacklozenge\bar{\phi}$$

means that there is a $t \in T(P, P^E)$ such that $t \models \bar{\phi}$.

7 The *transition trace logic* $\mathcal{L}(\mathcal{P}_l(\mathcal{T}(\mathcal{M}(D))))$ arises from the language $L(\mathcal{P}_l(\mathcal{T}(\mathcal{M}(D))))$ together with the implication and equivalence relations $\leq$ and $=$, and the axioms A_1–A_{13} in 2.4.11.

The intuitive meaning of the satisfaction relation $\models$ for Gamma programs is now clear. But how can we formally prove a statement of the form $P \models \blacklozenge\bar{\phi}$? Some inference rules suitable for this task are derived from the denotational definitions. For example, the denotational function T says that if $\alpha \in T(P, \mathit{Skip})$ and $\beta \in T(Q, \mathit{Skip})$, then $\alpha \odot \beta \in T(Q \circ P, \mathit{Skip})$. On the logical side, we would like to have $Q \circ P \models \blacklozenge\bar{\alpha} \odot \bar{\beta}$ from the fact that $P \models \blacklozenge\bar{\alpha}$ and $Q \models \blacklozenge\bar{\beta}$. In the same way, given that

$$T(P + Q, P^E) = \{\alpha \odot \beta \mid \text{ there are } \gamma, P', P^{E'} \text{ such that } \alpha\gamma \in T(P, P^E) \\ \text{and } \gamma \in T(P', P^{E'}) \text{ and } \beta \in T(P' + Q, P^{E'})\}$$

we want to prove that $(P+Q) \oplus P^E \models \blacklozenge\bar{\alpha} \odot \bar{\beta}$ from $P \oplus P^E \models \blacklozenge\bar{\alpha}\bar{\gamma}$, $P' \oplus P^{E'} \models \blacklozenge\bar{\gamma}$ and $(P' + Q) \oplus P^{E'} \models \bar{\beta}$. All these deductions are summarized in the following theorem:

Theorem 4.2.1 *The following are valid inference rules in the transition trace logic:*

Environment-free rules

Mediator

$$\frac{M \models \Phi,\ M' \in S(A \Leftarrow \Phi, M),\ (A \Leftarrow \Phi) \models \blacklozenge\bar{\alpha}}{(A \Leftarrow \Phi) \models \blacklozenge(M, M') \odot \bar{\alpha}}$$

Terminal

$$\frac{M \not\models \Phi}{(A \Leftarrow \Phi) \models \blacklozenge(M, M)}$$

Sequential composition

$$\frac{P \models \blacklozenge\bar{\alpha},\ Q \models \blacklozenge\bar{\beta}}{Q \circ P \models \blacklozenge\bar{\alpha} \odot \bar{\beta}}$$

Parallel composition I

$$\frac{P \models \blacklozenge\bar{\alpha}\bar{\beta},\ P' \models \blacklozenge\bar{\beta},\ P' + Q \models \blacklozenge\bar{\gamma}}{P + Q \models \blacklozenge\bar{\alpha} \odot \bar{\gamma}}$$

Parallel composition II

$$\frac{Q \models \blacklozenge\bar{\alpha}\bar{\beta},\ Q' \models \blacklozenge\bar{\beta},\ P + Q' \models \blacklozenge\bar{\gamma}}{P + Q \models \blacklozenge\bar{\alpha} \odot \bar{\gamma}}$$

Environment-sensitive rules

E-mediator

$$\frac{\begin{array}{c}A \Leftarrow \Phi \models \blacklozenge\bar{\alpha}(M, N)\bar{\beta},\ \ P^E \models \blacklozenge(N, O)\bar{\gamma},\\ P^{E'} \models \blacklozenge\bar{\gamma},\ (A \Leftarrow \Phi) \oplus P^{E'} \models \blacklozenge(O, P)\bar{\eta}\end{array}}{(A \Leftarrow \Phi) \oplus P^E \models \blacklozenge\bar{\alpha}(M, N)(O, P)\bar{\eta}}$$

E-terminal

$$\frac{A \Leftarrow \Phi \models \blacklozenge(M, M),\ P^E \models \blacklozenge(M, M)}{(A \Leftarrow \Phi) \oplus P^E \models \blacklozenge(M, M)}$$

E-sequential composition

$$\frac{P \oplus P^E \models \blacklozenge\bar{\alpha}(M, N)\bar{\beta},\ P' \oplus P^{E'} \models \blacklozenge\bar{\beta},\ P' \models \blacklozenge(N, N),\ Q \oplus P^{E'} \models \blacklozenge\bar{\gamma}}{(Q \circ P) \oplus P^E \models \blacklozenge\bar{\alpha}(M, N) \odot \bar{\gamma}}$$

E-parallel composition I

$$\frac{P \oplus P^E \models \blacklozenge\bar{\alpha}\bar{\beta},\ P' \oplus P^{E'} \models \blacklozenge\bar{\beta},\ (P' + Q) \oplus P^{E'} \models \blacklozenge\bar{\gamma}}{(P + Q) \oplus P^E \models \blacklozenge\bar{\alpha} \odot \bar{\gamma}}$$

E-parallel composition II

$$\frac{Q \oplus P^E \models \blacklozenge\bar{\alpha}\bar{\beta},\ Q' \oplus P^{E'} \models \blacklozenge\bar{\beta},\ (P + Q') \oplus P^{E'} \models \blacklozenge\bar{\gamma}}{(P + Q) \oplus P^E \models \blacklozenge\bar{\alpha} \odot \bar{\gamma}}$$

Absorption

$$\frac{P \models \blacklozenge\bar{\alpha}(M,N)(N,O)\bar{\beta}}{P \models \blacklozenge\bar{\alpha}(M,O)\bar{\beta}}$$

Proof. *Terminal*. The denotation of $A \Leftarrow \Phi$ includes all the pairs (M,M) such that $M \not\models \Phi$. But then there is a $p \in [\![A \Leftarrow \Phi]\!]$ such that $p \models (M,M)$, namely (M,M) itself. By definition of $\models$ in $\mathcal{L}(\mathcal{P}_l(\mathcal{T}(\mathcal{M}(D))))$ then $A \Leftarrow \Phi \models \blacklozenge(M,M)$.

Mediator. If $p \in [\![A \Leftarrow \Phi]\!] \models \bar{\alpha}$, $M \models \Phi$ and $M' \in S(A \Leftarrow \Phi, M)$, then $(M,M') \odot p \in [\![A \Leftarrow \Phi]\!]$ and $(M,M') \odot p \models (M,M') \odot \bar{\alpha}$. The rule follows as a consequence.

Sequential composition. If $P \models \blacklozenge\bar{\alpha}$ and $Q \models \blacklozenge\bar{\beta}$ then $p \models \bar{\alpha}$ and $q \models \bar{\beta}$ for certain $p \in [\![P]\!]$ and $q \in [\![Q]\!]$. Hence, $p \odot q \models \bar{\alpha} \odot \bar{\beta}$ and the rule follows.

Parallel composition. By hypothesis, $p \models \bar{\alpha}\bar{\beta}$, $p' \models \bar{\beta}$ and $r \models \bar{\gamma}$ for some $p \in [\![P]\!]$, $p' \in [\![P']\!]$ and $r \in [\![P' + Q]\!]$. Now, $p \odot r \models \bar{\alpha} \odot \bar{\gamma}$. Again, by definition of $[\![\ ]\!]$ we have that $p \odot r \in [\![P + Q]\!]$ and then $P + Q \models \blacklozenge\bar{\alpha} \odot \bar{\gamma}$.

Absorption. Suppose that $P \models \bar{\alpha}(M,N)(N,O)\bar{\beta}$. This means that $p \models \bar{\alpha}(M,N)(N,O)\bar{\beta}$ for a $p \in [\![P]\!]$. As $[\![P]\!]$ is closed under absorption, there must be a $p' \in [\![P]\!]$ such that $p' \models \bar{\alpha}(M,O)\bar{\beta}$ and then $P \models \blacklozenge\bar{\alpha}(M,O)\bar{\beta}$.

The environment sensitive rules are proved in an analogous manner. ■

A very important advantage of a DTLF setting for the transition trace logic is that, in contrast to the multiset logic, soundness and completeness are direct consequences of (one of the) Stone dualities:

Theorem 4.2.2 *Soundness and completeness of the transition trace logic. For every* $\blacklozenge\bar{\alpha}$, $\blacklozenge\bar{\beta} \in \mathcal{L}(\mathcal{P}_l(\mathcal{T}(\mathcal{M}(D))))$,

$$\blacklozenge\bar{\alpha} \leq \blacklozenge\bar{\beta}$$

if and only if for every $P \in \mathbb{G}$,

$$P \models \blacklozenge\bar{\alpha} \qquad \textit{implies} \qquad P \models \blacklozenge\bar{\beta}.$$

Proof. By theorem 2.4.16 and definition 2.4.14 we have that

$$[\![\blacklozenge\bar{\alpha}]\!]_{\mathcal{P}_l(\mathcal{T}(\mathbb{M}(D)))} \subseteq [\![\blacklozenge\bar{\beta}]\!]_{\mathcal{P}_l(\mathcal{T}(\mathbb{M}(D)))}.$$

But $P \models \blacklozenge\bar{\alpha}$ means that $[\![P]\!] \in [\![\blacklozenge\bar{\alpha}]\!]_{\mathcal{P}_l(\mathcal{T}(\mathbb{M}(D)))}$ and then $[\![P]\!] \in [\![\blacklozenge\bar{\beta}]\!]_{\mathcal{P}_l(\mathcal{T}(\mathbb{M}(D)))}$ and finally $P \models \blacklozenge\bar{\beta}$. ■

Soundness and completeness can be considered from the point of view of the denotational order $\sqsubseteq_T$ too:

Theorem 4.2.3 *For every P, $Q \in \mathbb{G}$*

$$P \sqsubseteq_T Q$$

if and only if for every $\blacklozenge\bar{\alpha} \in \mathcal{L}(\mathcal{P}_l(\mathcal{T}(\mathcal{M}(D))))$

$$P \oplus P^E \models \blacklozenge\bar{\alpha} \qquad \textit{implies} \qquad Q \oplus P^E \models \blacklozenge\bar{\alpha}.$$

This is also valid if we replace $\sqsubseteq_C$ for $\sqsubseteq_T$.

Proof. Assume $P \sqsubseteq_T Q$. Now $P \oplus P^E \models \blacklozenge\bar{\alpha}$ implies that $p \models \bar{\alpha}$ for a certain $p \in [\![P]\!]$. But by definition of $\sqsubseteq_T$, $p \in T(Q, P^E)$ and then $p \in [\![Q]\!]$. Hence

$$Q \oplus P^E \models \blacklozenge\bar{\alpha}.$$

Suppose now that $P \oplus P^E \models \blacklozenge\bar{\alpha}$ implies $Q \oplus P^E \models \blacklozenge\bar{\alpha}$. Take the trace $t \in T(P, P^E)$. There is a $\bar{\tau} \in L(\mathcal{T}(\mathcal{M}(D)))$ such that $x \models \bar{\tau}$ if and only if $x = t$ (that is, $\bar{\tau}$ expresses the property of being identical to t). It follows that $P \oplus P^E \models \blacklozenge\bar{\tau}$ and by hypothesis

$$Q \oplus P^E \models \blacklozenge\bar{\tau},$$

i.e. there is a $q \in T(Q, P^E)$ such that $q \models \bar{\tau}$ and by construction, $q = t$. Therefore $P \sqsubseteq_T Q$.

As for the last statement in the present theorem, full abstraction (3.4.2) allows us to replace $\sqsubseteq_C$ for $\sqsubseteq_T$. ■

Corollary 4.2.4 *If $P \sqsubseteq Q$ according to one of the laws 3.1.2 then $P \models \Diamond\bar{\alpha}$ implies $Q \models \Diamond\bar{\alpha}$.*

Proof. Make $P^E = Skip$ and then $P \oplus P^E = P$. The corollary is an immediate consequence of the previous theorem. ■

How powerful is the transition trace logic? As an example, consider the program:

$$Pred = x \rightarrow \{\!|x-1, x-2|\!\} \Leftarrow \Diamond\{\!|x > 1|\!\},$$

one of the components in example 4.1.10. Assuming that $n \geq 2$ and $Pred \models \blacklozenge(\{\!|n-1, n-2|\!\}, N_1)(N_1, N_2)\ldots$ then

$$Pred \models \blacklozenge(\{\!|n|\!\}, \{\!|n-1, n-2|\!\})(\{\!|n-1, n-2|\!\}, N_1)(N_1, N_2)\ldots$$

by mediator, as $\{\!|n|\!\} \models \Diamond\{\!|x > 1|\!\}$. The next step is finding out the value of N_1.

If $n = 3$ then $N_1 = \{\!|2, 1, 0|\!\}$ and we can perform *Pred* again to get N_2. If $n = 2$ then the terminal rule can be applied. But what if $n > 3$? We have different possibilities for applying the action $x \rightarrow \{\!|x-1, x-2|\!\}$. In short, it is very difficult to know what final multiset, if any, we will get. Some form of induction on multisets seems necessary here. We will start by reviewing a former proposal for dealing with the problem.

4.3 Termination

We just need to remember the halting problem to realize that determining termination of programs is in general an undecidable problem. Nevertheless, in practice there are ways of having an acceptable method of handling termination. The most common one is through the so-called *well-founded induction*: if we can prove that a certain well-founded order is kept during the execution of a program, then the program will terminate.[1]

As Gamma programs can be seen as rewriting rules for multisets (in the sense of term rewriting systems), termination of a program is guaranteed if the rules respect a well-founded order for multisets. Dershowitz & Manna (1979) proposed what is the best-known multiset order:

[1] Just for the sake of clarity, let us state that a well-founded order on a set P is a transitive and non-reflexive relation $\succ \subseteq P \times P$ which contains no infinite descending chain.

Definition 4.3.1 *If $M, N \in \mathbb{M}(D)$ and $\succ$ is a transitive relation on D, then*

$$M \succ_{DM} N$$

if and only if there are $X, Y \in \mathbb{M}(D)$ such that

$$\begin{aligned} X &\neq \emptyset \\ X &\subseteq M \\ N &= (M - X) \uplus Y \\ X &\sqsubseteq_u Y \end{aligned}$$

ie, for all $y \in Y$ there is a $x \in X$ such that $x \succ y$.

They also proved that $\succ_{DM}$ is well-founded if $\succ$ is so. With these ideas in mind, Banâtre & Le Métayer (1990) proposed a method for developing Gamma programs from a specification, following the approach of Dijkstra (1976). Their original proposal was restricted to atomic reactions. The steps of the method are:

(1) To express the specification of a program in first order predicate calculus.

(2) To split the specification in two parts: invariant and terminating condition.

(3) To state the reaction condition of the rewriting rule as the negation of the terminating condition.

(4) To design the action in the rewriting rule as a function that validates the terminating condition locally, that is, in the subset satisfying the reaction condition.

(5) To prove that the execution of the program respects the well-founded order $\succ_{DM}$.

Although we are interested in proving correctness of programs rather than developing them, their method can be adapted for correctness proofs. For instance, we could take for granted the existence of a specification (step 1), albeit it could be written in informal language. In the case of step 3,

we would have to verify (rather than state) that the reaction condition is the negation of the terminating condition. Step 4 would also involve verification rather than design.

But we would also need to include more general Gamma programs and not just atomic reactions. On the other hand, $\succ_{DM}$ is not the only possible order for multisets, and the next theorem is valid for any well-founded order.

Theorem 4.3.2 *Let $P \in \mathbb{G}$ and consider the traces $(M_1, M_2) \ldots \in T(P, Skip)$. If $\succ$ is a well-founded relation on $\mathbb{M}(D)$ and for every i, $M_i \succ M_{i+1}$, then:*

(a) all the considered traces are finite, that is, they have the form

$$(M_1, M_2) \ldots (M_{n-1}, M_n);$$

(b) $P \models \blacklozenge(M_1, M_n)$.

Proof. If a trace $(M_1, M_2) \ldots$ were infinite, by hypothesis we would have an infinite descending chain $M_1 \succ M_2 \cdots$ which is not possible as $\succ$ is well founded. Hence, all the traces are finite.

The proof of (b) is made by induction on Gamma syntax:

1. Base case. $P = A \Leftarrow \Phi$, where $A = \{\!|f_1(x_1, \ldots, x_i), \ldots, f_k(x_1, \ldots, x_i)|\!\}$ and $\Phi = \Diamond\{\!|\phi_1, \ldots, \phi_i|\!\}$. If $M_n \models \Phi$ then there is a σ such that $m_{\sigma(1)} \models \phi_1$, $\ldots$, $m_{\sigma(i)} \models \phi_i$. Then consider

$$M' = (M_n - \{\!|m_{\sigma(1)}, \ldots, m_{\sigma(i)}|\!\}) \uplus A(m_{\sigma(1)}, \ldots, m_{\sigma(i)}).$$

Clearly $M' \in S(A \Leftarrow \Phi, M_n)$ so (M_{n-1}, M_n) was not the last pair. Therefore $M_n \not\models \Phi$.

Using the terminal rule we have $A \Leftarrow \Phi \models \blacklozenge(M_n, M_n)$. By the mediator rule then

$$A \Leftarrow \Phi \models \blacklozenge(M_{n-1}, M_n)(M_n, M_n).$$

A finite number of applications of the mediator rule give us

$$A \Leftarrow \Phi \models \blacklozenge(M_1, M_2) \ldots (M_{n-1}, M_n)(M_n, M_n),$$

and performing the absorption rule n times renders

$$A \Leftarrow \Phi \models \blacklozenge(M_1, M_n).$$

2. $P = Q \circ P$. Then by hypothesis there is a $i \leq n$ such that

$$(M_1, M_2) \ldots (M_{i-1}, M_i) \in T(P, Skip)$$
$$(M_{i+1}, M_{i+2}) \ldots (M_{n-1}, M_n) \in T(Q, Skip),$$

with $M_i = M_{i+1}$. We apply now the inductive hypothesis and then

$$P \models \blacklozenge(M_1, M_i) \qquad \text{and} \qquad Q \models \blacklozenge(M_{i+1}, Mn),$$

and by the sequential rule

$$Q \circ P \models \blacklozenge(M_1, M_i) \odot (M_{i+1}, M_n),$$

from which it follows that $Q \circ P \models \blacklozenge(M_1, M_n)$ by definition of $\odot$ and the absorption rule.

3. $P = P + Q$. We will use here mathematical induction on the length of the trace too (namely, $n/2$):

Base case $n = 2$. By definition of T, $M_1 = M_2$ and $\langle P, M_1\rangle \rightarrow M_1$ and $\langle Q, M_1\rangle \rightarrow M_1$. Then $(M_1, M_1) \in T(P, Skip)$ and $(M_1, M_1) \in T(Q, Skip)$. By inductive hypothesis (on the syntax of Gamma):

$$P + Q \models (M_1, M_2).$$

Inductive case. Suppose that for every $k < n/2$ it is true that for all P, $Q \in \mathbb{G}$

$$(M_1, M_2) \ldots (M_{2k-1}, M_{2k}) \in T(P + Q, Skip)$$

implies

$$P + Q \models \blacklozenge(M_1, M_{2k}).$$

Now suppose $(M_1, M_2) \ldots (M_{n-1}, M_n) \in T(P + Q, Skip)$. Without loss of generality, we can assume that there are γ and P' such that

$$(M_1, M_2) \ldots, (M_{i-1}, M_i)\gamma \in T(P, Skip)$$
$$\gamma \in T(P', Skip) \qquad \text{and}$$
$$(M_{i+1}, M_{i+2}) \ldots (M_{n-1}, M_n) \in T(P' + Q, Skip).$$

with $M_i = M_{i+1}$. By inductive hypothesis on n:

$$P' + Q \models \blacklozenge(M_{i+1}, M_n)$$

and by inductive hypothesis on Gamma syntax:

$$P \models \blacklozenge(M_1, M_i) \qquad \text{and} \qquad P' \models \blacklozenge\gamma.$$

Applying the parallel rule we obtain:

$$P + Q \models \blacklozenge(M_1, M_i) \odot (M_{i+1}, M_n),$$

and the absorption rule gives $P + Q \models \blacklozenge(M_1, M_n)$. ■

The termination part of Banâtre & Le Métayer's (1990) method has now been extended from atomic reactions to general Gamma programs. Now it is the turn of the invariant side.

Theorem 4.3.3 *Consider a Gamma program P whose strict traces meet the hypothesis in theorem 4.3.2, i.e. they have the form $(M_1, M_2)\ldots(M_{n-1}, M_n)$, and let I_P, T_P and $\Pi_P \in L(\mathcal{M}(D))$. If*

(a) For every $i \leq n$ we have $M_i \models \mathrm{I}_P$;

(b) $\mathrm{I}_P \leq \mathrm{T}_P \vee \Pi_P$, and

(c) $M_n \not\models \mathrm{T}_P$;

then $M_n \models \Pi_P$. The formulas I_P, T_P and Π_P are called invariant*,* non-terminating condition *and* postcondition*, respectively.*

Proof. If $\mathrm{I}_P \leq \mathrm{T}_P \vee \Pi_P$, the theorem 4.1.6 says that for all $M \in \mathbb{M}(D)$, $M \models \mathrm{I}_P$ implies $M \models \mathrm{T}_P \vee \Pi_P$. By definition of satisfaction, this happens if and only if either $M \models \mathrm{T}_P$ or $M \models \Pi_P$. We have by hypothesis that $M_n \models \mathrm{I}_P$ and then $M \models \mathrm{T}_P \vee \Pi_P$ but $M_n \not\models \mathrm{T}_P$, so $M_n \models \Pi_P$ must be the case. ■

At last, we are able to return to the basic problem of proving total correctness of a program P. We will follow these steps:

1. We will state a precondition Φ and a postcondition Ψ, where Φ, $\Psi \in L(\mathcal{M}(D))$.

2. Using theorem 4.3.2, we will prove that $P \models \blacklozenge(M_1, M_n)$ for certain M_1, $M_n \in \mathbb{M}(D)$.

3. Applying theorem 4.3.3 and the inference rules of $\mathcal{L}(\mathcal{M}(D))$, we will show that if $M_1 \models \Phi$ then $M_n \models \Psi$.

4. As P can be a complex program, steps 1–3 can be applied to each of its components and then the rules 4.2.1 will be used to reassemble the components.

4.4 Examples

Now it is time to put to work all the system developed in this chapter. Our first example will be a proof of correctness of program 4.1.8.

1. Let the precondition be

$$\Box\{\!|(0,0,0)|\!\}$$

and the postcondition

$$\Box\{\!|(x_1, y_1, k), \ldots, (x_{3^k}, y_{3^k}, k)|\!\}$$

where (x_i, y_i) is one of the points in the k-th iteration of the Sierpinsky function (henceforth called a k-Sierpinsky point). Clearly there is only one multiset meeting the precondition and the same holds for the postcondition. In other words, we want to show

$$P \models \blacklozenge(\{\!|(0,0,0)|\!\}, \{\!|(x_1, y_1, k), \ldots, (x_{3^k}, y_{3^k}, k)|\!\}).$$

For convenience, let us abbreviate $\{\!|(x/2 - 2/5, y/2 - 2/5, z + 1), (x/2 - 2/5, y/2 + 2/5, z + 1), (x/2 + 2/5, y/2 + 2/5, z + 1)|\!\}$ by $A(x, y, z)$.

2. Let $(M_1, M_2) \ldots \in T(Sier, Skip)$ and let the relation $\succ \subseteq (\mathbb{R} \times \mathbb{R} \times \mathbb{N})^2$ be defined by

$$(a, b, c) \succ (a', b', c') \qquad \text{if and only if} \qquad c < c' \text{ and } c' \leq k.$$

Clearly $\succ$ is a well-founded order. Therefore $\succ_{DM}$ is also well-founded.

Now, if $z < k$ and $N = (M - \{\!|(x, y, z)|\!\}) \uplus A(x, y, z)$ then $M \succ_{DM} N$. Then the traces in $T(Sier, Skip)$ have the shape $(M_1, M_2) \ldots (M_{n-1}, M_n)$ and

$$Sier \models \blacklozenge(M_1, M_n).$$

3. Define

$$\begin{aligned} \mathrm{I}_{Sier} &= \bigvee_{i=1}^{n} (\Box(M_{i-1} - \{\!|(x, y, z)|\!\}) \uplus A(x, y, z)) \\ \mathrm{T}_{Sier} &= \Diamond\{\!|z < k|\!\} \\ \Pi_{Sier} &= \Box\{\!|(x_1, y_1, k), \ldots, (x_{3^k}, y_{3^k}, k)|\!\}. \end{aligned}$$

If $M \models \mathrm{I}_{Sier}$ then $M \models \mathrm{T}_{Sier} \vee \Pi_{Sier}$ and, by 4.1.6, we get $\mathrm{I}_{Sier} \leq \mathrm{T}_{Sier} \vee \Pi_{Sier}$. Clearly, for every i, $M_i \models \mathrm{I}_{Sier}$, but $M_n \not\models \mathrm{T}_{Sier}$ (otherwise M_n would not be the final multiset). Then

$$M_n \models \Box\{\!|(x_1, y_1, k), \ldots, (x_{3^k}, y_{3^k}, k)|\!\}.$$

Step 4 is omitted as *Sier* is an atomic rule. ■

The second case is example 4.1.9. We will split the program and apply steps 1–3 to each part. Let us begin with *Max*.

Suppose our initial multiset is $\{\!|m_1, \ldots, m_n|\!\}$. The precondition is now $\Diamond\{\!|t, t, t|\!\}$ and the postcondition is

$$\Box\{\!|\bigwedge_{i=1}^{n} m_i \leq z, \bigwedge_{i=1}^{n-1} m_i \leq w|\!\}.$$

The multiset order $\succ$ is the simplest:

$$M \succ N \qquad \text{if and only if} \qquad |M| > |N|.$$

Assume $(M_1, M_2) \ldots \in T(Max, Skip)$. It is clear that $M_i \succ M_{i+1}$ and that $\succ$ is well-founded. Then all traces in $T(Max, Skip)$ are finite of length $n/2$ (with $n \in \mathbb{N}$) and then

$$Max \models \blacklozenge(M_1, M_n).$$

Now, make

$$\begin{aligned}
\mathrm{I}_{Max} &= \Diamond\{\!\!\{ \bigwedge_{i=1}^{n} m_i \leq z, \bigwedge_{i=1}^{n-1} m_i \leq w \}\!\!\} \\
\mathrm{T}_{Max} &= \Diamond\{\!\!\{ t, t, t \}\!\!\} \\
\Pi_{Max} &= \Box\{\!\!\{ \bigwedge_{i=1}^{n} m_i \leq z, \bigwedge_{i=1}^{n-1} m_i \leq w \}\!\!\}.
\end{aligned}$$

Again, $\mathrm{I}_{Max} \leq \mathrm{T}_{Max} \vee \Pi_{Max}$ and for every i, $M_i \models \mathrm{I}_{Max}$. Additionally, $M_n \not\models \mathrm{T}_{Max}$ (by axiom A_{11} in 4.1.4) and therefore

$$M_n \models \Box\{\!\!\{ \bigwedge_{i=1}^{n} m_i \leq z, \bigwedge_{i=1}^{n-1} m_i \leq w \}\!\!\}.$$

Regarding *Prod*, our multiset order is the same as before and then

$$Prod \models \blacklozenge(N_1, N_m).$$

If $N_1 = \{\!\!\{n_1, \ldots, n_p\}\!\!\}$ and

$$\begin{aligned}
\mathrm{I}_{Prod} &= \bigvee_{i \in \mathbb{N}} \Box\{\!\!\{ \prod_{j=1}^{i} x_j = \prod_{k=1}^{p} n_k \}\!\!\} \\
\mathrm{T}_{Prod} &= \Diamond\{\!\!\{ t, t \}\!\!\} \\
\Pi_{Prod} &= \Box\{\!\!\{ \prod_{k=1}^{p} n_k \}\!\!\},
\end{aligned}$$

is clear that $N_i \models \mathrm{I}_{Prod}$ as $N_i \models \Box\{\!\!\{ \prod_{j=1}^{r} = \prod_{k=1}^{p} n_k \}\!\!\}$ with $r = |N_i|$.

On the other hand, $N_m \not\models \mathrm{T}_{Prod}$ and then $N_M \models \Box\{\!\!\{ \prod_{k=1}^{p} n_k \}\!\!\}$. If we take $N_1 = \{\!\!\{z, w\}\!\!\}$ then

$$Prod \models \blacklozenge(\{\!\!\{z, w\}\!\!\}, \{\!\!\{zw\}\!\!\}).$$

Applying now the sequential composition and absorption rules:

$$Prod \circ Max \models \blacklozenge(\{\!\!\{m_1, \ldots, m_n\}\!\!\}, \{\!\!\{zw\}\!\!\}),$$

where $\{\!\!\{z, w\}\!\!\} \models \Box\{\!\!\{ \bigwedge_{i=1}^{n} m_i \leq z, \bigwedge_{i=1}^{n-1} m_i \leq w \}\!\!\}$. ■

Finally we shall prove that $Fib \models \blacklozenge(\{\!\!\{n\}\!\!\}, \{\!\!\{ \mathrm{Fib}\, n\}\!\!\})$, where $\mathrm{Fib}\, n$ stands for the n-Fibonacci number. That is, the precondition is $\Box\{\!\!\{n\}\!\!\}$ and the postcondition is $\Box\{\!\!\{ \mathrm{Fib}\, n\}\!\!\}$.

Consider the traces $(M_1, M_2)\ldots \in T(Sum, Skip)$. Again, the multiset order

$$M \succ_{Sum} N \qquad \text{if and only if} \qquad |M| > |N|$$

is well-founded and $M_i \succ_{Sum} M_{i+1}$ and then the traces are finite and

$$Sum \models \blacklozenge(M_1, M_m).$$

Now let

$$\begin{aligned} \mathrm{I}_{Sum} &= \bigvee_{i\in\mathbb{N}} \Box\{\!\!\{ \sum_{j=1}^{i} x_j = \mathrm{Fib}\, n\}\!\!\} \\ \mathrm{T}_{Sum} &= \Diamond\{\!\!\{t, t\}\!\!\} \\ \Pi_{Sum} &= \Box\{\!\!\{ \mathrm{Fib}\, n\}\!\!\}. \end{aligned}$$

It is clear that

$$\Pi_{Sum} = \bigvee_{i\in\mathbb{N}} \Box\{\!\!\{ \sum_{i=1}^{i} x_j = \mathrm{Fib}\, n\}\!\!\} \wedge \Box\{\!\!\{t\}\!\!\}$$

and then $\mathrm{I}_{Sum} \leq \mathrm{T}_{Sum} \vee \Phi_{Sum}$.

Consider now the traces in $T(Sum, Skip)$ such that $M_1 = \{\!\!\{1\}\!\!\}^{\mathrm{Fib}\, n}$. We have that for all $i \leq n$, $M_i \models \mathrm{I}_{Sum}$. On the other hand, $M_m \not\models \mathrm{T}_{Sum}$ (otherwise M_m would not be the final element) and then $M_m \models \Box\{\!\!\{ \mathrm{Fib}\, n\}\!\!\}$. Therefore

$$Sum \models \blacklozenge(\{\!\!\{1\}\!\!\}^{\mathrm{Fib}\, n}, \{\!\!\{ \mathrm{Fib}\, n\}\!\!\}).$$

It is time to examine $Pred + One$. Let $(N_1, N_2)\ldots \in T(Pred + One, Skip)$ and let

$$M \succ_{Pred+One} N \quad \text{iff} \quad \begin{cases} N = (M - \{\!\!\{0\}\!\!\}) \uplus \{\!\!\{1\}\!\!\} & \text{or} \\ N = (M - \{\!\!\{x\}\!\!\}) \uplus \{\!\!\{x-1, x-2\}\!\!\} & \text{if } x > 1. \end{cases}$$

Again, $\succ_{Pred+One}$ is well-founded and then the traces in $T(Pred + One, Skip)$ have the form $(N_1, N_2)\ldots(N_{k-1}, N_k)$. Therefore

$$Pred + One \models \blacklozenge(N_1, N_k).$$

Let us restrict our attention to the traces starting with the multiset $\{\!\!\{n\}\!\!\}$. Define now

$$\begin{aligned} \mathrm{I}_{Pred+One} &= \bigvee_{i\in\mathbb{N}} \Box\{\!\!\{ \sum_{j=1}^{i} \mathrm{Fib}\, x_j = \mathrm{Fib}\, n\}\!\!\} \\ \mathrm{T}_{Pred+One} &= \Diamond\{\!\!\{x > 1\}\!\!\} \vee \Diamond\{\!\!\{0\}\!\!\} \\ \Pi_{Pred+One} &= \Box\{\!\!\{1\}\!\!\}^{\mathrm{Fib}\, n}. \end{aligned}$$

As $M \models \Pi_{Pred+One}$ if and only if $M \models \mathrm{I}_{Pred+One}$ but $M \not\models \mathrm{T}_{Pred+One}$, we conclude that $\mathrm{I}_{Pred+One} \leq \mathrm{T}_{Pred+One} \vee \Pi_{Pred+One}$. Obviously, $\{\!|n|\!\} \models \Box\{\!|\operatorname{Fib} n = \operatorname{Fib} n|\!\}$ and then $\{\!|n|\!\} \models \mathrm{I}_{Pred+One}$. If we shift our attention to the successors of $\{\!|n|\!\}$ we can notice that if $M_i \models \mathrm{I}_{Pred+One}$ then

$$M_{i+1} = \begin{cases} N = (M - \{\!|0|\!\}) \uplus \{\!|1|\!\} & \text{or} \\ N = (M - \{\!|x|\!\}) \uplus \{\!|x-1, x-2|\!\} & \text{if } x > 1 \end{cases}$$

satisfies $\mathrm{I}_{Pred+One}$, as $\operatorname{Fib} 0 = \operatorname{Fib} 1$ and $\operatorname{Fib} x = \operatorname{Fib}(x-1) + \operatorname{Fib}(x-2)$. Lastly,

$$M_m \not\models \Diamond\{\!|x > 1|\!\} \vee \Diamond\{\!|0|\!\}$$

(again, it would not be the last multiset in the trace if it did) and then $M_m \models \Box\{\!|1|\!\}^{\operatorname{Fib} n}$. Hence

$$Pred + One \models \blacklozenge(\{\!|n|\!\}, \{\!|1|\!\}^{\operatorname{Fib} n})$$

and we conclude with an application of the sequential and absorption rules:

$$Sum \circ (Pred + One) \models \blacklozenge(\{\!|n|\!\}, \{\!|\operatorname{Fib} n|\!\}). \qquad \blacksquare$$

And with this last example the presentation of the Gamma proof system concludes and we can recapitulate our results:

A sound and (conditionally) complete multiset logic was presented. This multiset logic complemented a transition trace logic, which was also proved sound and (conditionally) complete. The transition trace logic is the basis for a correctness proof method for general Gamma programs (including proofs of termination of programs). Some examples of the application of the method were also offered.

Chapter 5

Locales, bags and pipelining

In this chapter we shall discuss three different issues which can be useful to extend the results of chapter 3 and 4 both in a practical and in a theoretical direction: a localic presentation of the multiset logic, the relation between the multiset logic and some other views about multisets and the application of multiset logic to the analysis of program transformations.

There are other ways of presenting the multiset logic of chapter 4. In the chosen presentation, the emphasis is on the satisfaction relation between propositions and multisets. Alternatively, we could have begun with the logic itself and then tried to "recover" the multisets as the entities satisfying the logic. *Geometric logics* were proposed by Vickers (1989) as proof systems in which the satisfaction of a property can be proven with a finite number of observations. A geometric logic is a frame where least upper bounds and greatest lower bounds are regarded as disjunctions and conjunctions, respectively. Then the objects satisfying the propositions in the logic —called *points*— are explained in terms of a *locale*. The satisfaction relation imposes an order on the points, together with some other nice properties. The theoretical foundations for this view lay, again, on the work of Stone (1936) and Johnstone (1982). However, we will leave some open questions about the equivalence of this view and the multiset logic of chapter 4.

Multisets are also an important topic in database theory. Databases are collections of data, and these collections can be seen either as sets or

multisets. The latter are particularly useful when dealing with partial or incomplete information in databases (which is normally the rule). Libkin (1994) worked on the semantics of incomplete information in databases and, among other results, produced a multiset language with a rich set of operations, most of them of polynomial complexity.

In a previous work, Libkin & Wong (1993) also defined two orders for multisets. One of them is equivalent to the multiset order arising from our logic. This implies that the verification of the satisfaction of a formula by a multiset can be done in polynomial time (provided the satisfaction relation for elements is also of polynomial complexity).

But some negative results arise from the equivalence between the two orders too. Heckmann (1995) proposed the *lower bag-domain* and proved that its order is equivalent to the one in Libkin & Wong (1993). He also showed that Scott domains are not preserved by the bag-domain construction, and this limitation can be extended to the order generated by our multiset logic.

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. Hankin, Le Métayer & Sands (1998) and Weichert (1999) studied in depth this problem in the context of Gamma programs. The latter strengthened a series of requirements for a pipelining transformation to be safe. Those requirements can be re-stated in terms of the multiset logic and then the whole formal apparatus of chapter 4 can be used to verify them.

A section is devoted to the discussion of each topic. We shall start with the localic version of the multiset logic. The bag language and orders of Libkin & Wong (1995) will follow afterwards. The final subject will be the pipeline transformation, accompanied by an example.

Other people have also explored the relationship between multisets and databases. Gunter's (1992) mixed powerdomain and Vickers's (1992) predicate geometric logic address the representation of partial information in a set of data. A research program to connect their views to our own multiset logic will be presented in the concluding chapter.

5.1 A locale for the multiset logic

A few concepts are necessary before introducing locales. To begin with, we have the two point lattice $\mathbf{2} = \{false, true\}$ with $false \leq true$ (again, the Sierpinski space regarded as a lattice).

Let us remember that a frame is a distributive lattice which is also cocomplete, although it is not necessarily complete (see 2.4.2). But a frame is also a special kind of logic, where the operator $\sqcap$ is the conjunction $\wedge$ and $\sqcup$ is the disjunction $\vee$. The fact that only finite conjunctions but arbitrary disjunctions need to exist reflects the property that propositions can be proven with a finite number of observations: to verify a conjunction we need to verify each of its components (then the conjunction cannot be infinite); to verify a disjunction we just need to find a component which is true and there is no need to verify the others. This type of logic is called *geometric*.

To make the relationship between frames and geometric logics even stronger we introduce some mappings from frames to lattices:

Definition 5.1.1 *Let F be a frame and L a lattice. A* frame homomorphism *is a function $f : F \to L$ such that*

$$f(x \wedge y) = f(x) \wedge f(y) \qquad and \qquad f(x \vee y) = f(x) \vee f(y).$$

If F is a frame, the set

$$\mathsf{pt}\, F = \{f : F \to \mathbf{2} \mid f \text{ is a frame homomorphism}\}$$

is called the points *of F. The set $f^{-1}(true)$ is the* true kernel *of f.*

Because f is a frame homomorphism, we have that $x \sqsubseteq y$ implies $f(x) \sqsubseteq f(y)$ and, as a consequence:

a) $f^{-1}(true)$ is upper closed;

b) $x, y \in f^{-1}(true)$ implies $x \wedge y \in f^{-1}(true)$;

c) if $\bigvee_{i \in I} x_i \in f^{-1}(true)$ then there is a $i \in I$ such that $x_i \in f^{-1}(true)$.

In other words, $f^{-1}(true)$ is a complete prime filter (see definition 2.4.3). Now we have all the notation required for locales:

Definition 5.1.2 *Let F be a frame. The pair $(F, \mathsf{pt}F)$ is the* locale *of F. A locale is associated with a satisfaction relation $\models \subseteq \mathsf{pt}F \times F$:*

$$f \models \phi \qquad \text{if and only if} \qquad \phi \in f^{-1}(true).$$

For all $\phi \in F$ define the set

$$U_\phi = \{f \in \mathsf{pt}F \mid f \models \phi\}.$$

These open sets form a topology of $\mathsf{pt}F$:

$$\emptyset = U_\bot \qquad \mathsf{pt}F = U_\top \qquad U_\phi \cap U_\phi = U_{\phi\wedge\psi} \qquad \bigcup_{i\in I} U_{\phi_i} = U_{\bigvee_{i\in I}\phi_i}$$

as F has a $\bot$ and a $\top$, and finite meets and arbitrary joins.

The structure of the set $\mathsf{pt}F$ looks a little mysterious. But the relation $\models$ introduces some order in the picture:

Definition 5.1.3 *Given a F, the specialization order $\sqsubseteq_S$ on $\mathsf{pt}F$ is the relation defined by*

$$x \sqsubseteq_S y \qquad \text{if and only if} \qquad x \models \phi \text{ implies } y \models \phi, \text{ for all } \phi \in F.$$

If F and G are frames, a function $h : F \to G$ is a *homeomorphism* if $x \sqsubseteq y$ implies $h(x) \sqsubseteq h(y)$ and there is another function $h' : G \to F$ such that $h \circ h' = Id_F$ and $h' \circ h = Id_G$. If there is a homeomorphism between F and G, they are called homeomorphic. In symbols: $F \simeq G$.

The set of compact elements in a topological space was defined shortly after definition 2.4.5. As there is a parallelism between intersection and conjunction, on the one hand, and union and disjunction, on the other, compactness can easily be extended to frames. As was the case with topological spaces, the set of compact elements in a frame F is denoted by $K(F)$. Because of this parallel, any element definable only through infinite disjunctions is excluded from $K(F)$: disjunctions are the equivalent to unions, and if c is a compact element, C is a set of opens and $c \sqsubseteq \bigvee C$, then there is a finite $C' \subseteq C$ such that $c \sqsubseteq \bigvee C'$.

Definition 5.1.4 *The frame F is* coherent *if and only if*

1. *It is algebraic (see definition 2.4.4).*
2. *The set $K(F)$ is a sublattice (ie, it is closed under finite meets).*

The previous definitions are put to some use in the following series of results:

Theorem 5.1.5 *Let F be a coherent frame. The next statements are true:*

- $\mathsf{pt}\,F$ *is* directed cocomplete*: if $D \subseteq \mathsf{pt}\,F$ is directed, then $\bigsqcup D$ exist.*
- *F is* complete*: for all $\phi, \psi \in F$, if $f \models \phi$ implies $f \models \psi$ then $\phi \leq \psi$.*
- *F is* sound*: the inverse implication of completeness holds as part of the definition of frame homomorphisms.*

This theorem summarizes propositions 7.2.3, 5.3.5 and 9.2.4 by Vickers (1989), where the reader can find a proof.

Now it is time to come back to the multiset logic. To begin with, $\mathcal{L}(\mathbb{M}(D))$ is a frame and then all of the previous definitions apply. Consider the locale $(\mathcal{L}(\mathbb{M}(D)), \mathsf{pt}\,\mathcal{L}(\mathbb{M}(D)))$. In an ideal world, the points in the locale should be finite multisets and the satisfaction relation $\models$ should coincide with the same relation as defined in chapter 4. The desirability of this coincidence does not arise from moral issues, but from the fact that $\mathcal{L}(\mathbb{M}(D))$ is complete with respect to $\mathsf{pt}\,\mathcal{L}(\mathbb{M}(D))$ and, moreover, $\mathsf{pt}\,\mathcal{L}(\mathbb{M}(D))$ is a domain whenever D is so. This is the subject of the following theorems. To begin with, we need to prove coherence of $\mathcal{L}(\mathbb{M}(D))$ and the next two lemmas will lead us there.

Lemma 5.1.6 $\mathrm{A} \in K(\mathcal{L}(\mathbb{M}(D)))$ *if and only if* $\mathrm{A} = \bigwedge_{i=1}^{m} \Box\{\!|a_1^i, \ldots, a_n^i|\!\}$ *and each of the a_j^i's is compact in $\mathcal{L}(D)$.*

Proof. If $\mathrm{A} \in K(\mathcal{L}(\mathbb{M}(D)))$ then A must be a finite conjunction of compacts (by definition of compactness). Take first the simplest case: $\mathrm{A} = \Box\{\!|a_1, \ldots, a_n|\!\}$. Now suppose there is a non-compact a_i, that is there exists a directed set $B \subseteq L(D)$ such that $a_i \leq \bigvee B$ and $a_i \leq b$ for no $b \in B$. Consider

now the set $B' = \{\Box\{\!|a_1, \ldots, a_{i-1}, b, a_{i+1}, \ldots, a_n|\!\} \mid b \in B\}$. By A_{13} in 4.1.4, B' is directed like B. According to A_{14}

$$\begin{aligned}\Box\{\!|a_1, \ldots, a_n|\!\} &\leq \Box\{\!|a_1, \ldots, a_{i-1}, \bigvee B, a_{i+1}, \ldots, a_n|\!\} \\ &\leq \bigvee_{b \in B} \Box\{\!|a_1, \ldots, a_{i-1}, b, a_{i+1}, \ldots, a_n|\!\} \\ &= \bigvee B'.\end{aligned}$$

However, there is no $b' \in B'$ such that $\Box\{\!|a_1, \ldots, a_n|\!\} \leq b'$, which contradicts our assumptions that $\Box\{\!|a_1, \ldots, a_n|\!\}$ was compact. Therefore all a_i's are compact.

If $\mathrm{A} = \bigwedge_{i=1}^{m} \Box\{\!|a_1^i, \ldots, a_n^i|\!\}$ with $m \geq 2$, then the argument previously explained can be repeated for each $\Box\{\!|a_1^i, \ldots, a_n^i|\!\}$ and we will reach the same conclusion.

The other direction of the lemma is trivial. ■

Lemma 5.1.7 *If $\mathfrak{L}(D)$ is an algebraic frame, then $\mathfrak{L}(\mathbb{M}(D))$ is algebraic too.*

Proof. For every $\alpha \in \mathfrak{L}(D)$ and $\mathrm{A} \in \mathfrak{L}(\mathbb{M}(D))$, let

$$K_\alpha = \{\psi \in K(\mathfrak{L}(D)) \mid \psi \leq \alpha\} \quad \text{and} \quad K_{\mathrm{A}} = \{\Psi \in K(\mathfrak{L}(\mathbb{M}(D))) \mid \Psi \leq \mathrm{A}\}.$$

We need to prove that K_{A} is directed and $\mathrm{A} = K_{\mathrm{A}}$.

Case $\mathrm{A} = \Box\{\!|\alpha_1, \ldots, \alpha_n|\!\}$. Take the formula $\Box\{\!|\bigvee K_{\alpha_1}, \ldots, \bigvee K_{\alpha_n}|\!\}$. These statements hold:

(a) $\Box\{\!|\bigvee K_{\alpha_1}, \ldots, \bigvee K_{\alpha_n}|\!\} = \Box\{\!|\alpha_1, \ldots, \alpha_n|\!\}$.

(b) $\Box\{\!|\bigvee K_{\alpha_1}, \ldots, \bigvee K_{\alpha_n}|\!\} = \bigvee K_{\mathrm{A}}$.

For (a), we just need to consider that $\mathfrak{L}(D)$ is algebraic and then $\alpha_i = \bigvee K_{\alpha_i}$. For (b), take a formula $\Psi \in K_{\mathrm{A}}$, with

$$\Psi = \bigwedge_{i=1}^{m} \Box\{\!|\psi_1^i, \ldots, \psi_n^i|\!\}.$$

As $\Psi \leq \mathrm{A}$, there are permutations σ_1^Ψ, …, σ_m^Ψ such that $\psi^i_{\sigma_i^\Psi(j)} \leq \alpha_j$. Now, remember that for all i and j, ψ_j^i is compact (lemma 5.1.6). On the other

hand, if $c \in K_{\alpha_i}$, there is a $\Gamma \in K_{\mathrm{A}}$ such that $\Gamma = \Box\{\!|\gamma_i, \ldots, c, \ldots, \gamma_n|\!\}$ (otherwise, we could easily build a compact Γ which would not belong to K_{A}). Then

$$\{\psi^i_{\sigma_i^\Psi(j)}\}_{\Psi \in K_{\mathrm{A}}} = K_{\alpha_j}$$

and then (b) follows.

Finally, K_{A} is directed. If Φ and $\Psi \in K_{\mathrm{A}}$ and $\Phi = \bigwedge_{i=1}^m \{\!|\phi^i_1, \ldots, \phi^i_n|\!\}$ and $\Psi = \bigwedge_{i=1}^p \{\!|\psi^i_1, \ldots, \psi^i_n|\!\}$, there are again permutations σ_1^Φ, $\ldots$, σ_m^Φ and σ_1^Ψ, $\ldots$, σ_p^Ψ such that

$$\phi^i_{\sigma_i^\Phi(j)} \leq \alpha_j \qquad \text{and} \qquad \psi^i_{\sigma_i^\Psi(j)} \leq \alpha_j,$$

and given the facts that (i) $\mathcal{L}(D)$ is algebraic; (ii) $\phi^i_{\sigma_i^\Phi(j)} \wedge \psi^i_{\sigma_i^\Psi(j)}$ is compact, and (iii) it belongs to K_{α_j}, we have that $\Phi \wedge \Psi$ can be built using those meets as elements. Therefore K_{A} is directed.

Case $\mathrm{A} = \bigwedge_{i=1}^m \Box\{\!|\alpha^i_1, \ldots, \alpha^i_n|\!\}$. The argument of the previous case can be repeated for each $\Box\{\!|\alpha^i_1, \ldots, \alpha^i_n|\!\}$ (as they are finite).

Case $\mathrm{A} = \Diamond\{\!|\alpha_1, \ldots, \alpha_n|\!\}$. As $\Diamond\{\!|\alpha_1, \ldots, \alpha_n|\!\} = \bigvee_{m \in \mathbb{N}} \Box(\{\!|\alpha_1, \ldots, \alpha_n|\!\} \uplus \{\!|t|\!\}^m)$ and $\phi \leq t$ for every $\phi \in \mathcal{L}(D)$, this case is reduced to the first one. ■

Theorem 5.1.8 *The multiset logic $\mathcal{L}(\mathbb{M}(D))$ is coherent, provided $\mathcal{L}(D)$ is so.*

Proof. We already know that $\mathcal{L}(\mathbb{M}(D))$ is algebraic. We only have to prove that $K(\mathcal{L}(\mathbb{M}(d)))$ is closed under finite meets. Assume that Φ and $\Psi \in K(\mathcal{L}(\mathbb{M}(D)))$, where

$$\Phi = \bigwedge_{i=1}^m \{\!|\phi^i_1, \ldots, \phi^i_n|\!\} \qquad \text{and} \qquad \Psi = \bigwedge_{i=1}^p \{\!|\psi^i_1, \ldots, \psi^i_n|\!\}.$$

By A_{12} in 4.1.4 we obtain:

$$\bigwedge_{i=1}^m \{\!|\phi^i_1, \ldots, \phi^i_n|\!\} = \bigvee_{\substack{\sigma_1 \in \Sigma(n) \\ \ldots \\ \sigma m \in \Sigma_n}} \Box\{\!|\phi^1_{\sigma_1(1)} \wedge \ldots \wedge \phi^m_{\sigma_m(1)}, \ldots, \phi^1_{\sigma_1(n)} \wedge \ldots \wedge \phi^m_{\sigma_m(n)}|\!\}$$

$$\bigwedge_{i=1}^p \{\!|\psi^i_1, \ldots, \psi^i_n|\!\} = \bigvee_{\substack{\tau_1 \in \Sigma(n) \\ \ldots \\ \tau_p \in \Sigma_n}} \Box\{\!|\psi^1_{\tau_1(1)} \wedge \ldots \wedge \psi^p_{\tau_p(1)}, \ldots, \psi^1_{\tau_1(n)} \wedge \ldots \wedge \psi^p_{\tau_p(n)}|\!\}$$

.

By distributivity of conjunction over disjunction:

$$\Phi \wedge \Psi = \bigvee_{\substack{\sigma_1 \in \Sigma(n) \\ \vdots \\ \sigma_m \in \Sigma_n \\ \tau_1 \in \Sigma(n) \\ \vdots \\ \tau_p \in \Sigma_n}} \Box\{\!|(\phi^1_{\sigma_1(1)} \wedge \ldots \wedge \phi^m_{\sigma_m(1)}) \wedge (\psi^1_{\tau_1(1)} \wedge \ldots \wedge \psi^p_{\tau_p(1)}), \ldots, (\phi^1_{\sigma_1(n)} \wedge \ldots \wedge \phi^m_{\sigma_m(n)}) \wedge (\psi^1_{\tau_1(n)} \wedge \ldots \wedge \psi^p_{\tau_p(n)})|\!\}$$

which is compact, as all disjunctions and conjunctions appearing inside are finite and $K(\mathcal{L}(D))$ is closed under them. ■

Corollary 5.1.9 *Completeness of* $\mathcal{L}(\mathbb{M}(D))$. *If for all* $\mathbf{M} \in \mathsf{pt}\,\mathcal{L}(\mathbb{M}(D))$, $\mathbf{M} \models \phi$ *implies* $\mathbf{M} \models \psi$ *then* $\phi \leq \psi$.

Proof. As $\mathcal{L}(\mathbb{M}(D))$ is coherent, theorem 5.1.5 applies. ■

As a nice additional result we have that $\langle \mathsf{pt}\,\mathbb{M}(D), \sqsubseteq_S \rangle$ is directed cocomplete, again thanks to 5.1.5.

We come back now to the comparison between $\mathbb{M}(D)$ and $\mathsf{pt}\,\mathcal{L}(\mathbb{M}(D))$. First of all we can also have a specialization order for $\mathbb{M}(D)$:

$$M \sqsubseteq_S N \qquad \text{if and only if} \qquad M \models \phi \text{ implies } N \models \phi.$$

Now we would like to have a function $h : \mathbb{M}(D) \to \mathsf{pt}\,\mathcal{L}(\mathbb{M}(D))$ such that if $M \in \mathbb{M}(D)$ and $h(M) = \mathbf{M}$

$$M \models \phi \qquad \text{if and only if} \qquad \mathbf{M} \models \phi.$$

The function h would definitely be one-to-one and would also preserve the specialization order $\sqsubseteq_S$. The point is: is h properly defined, namely, does $h(M)$ always exist? Additionally, is h a homeomorphism, and then $\mathbb{M}(D)$ and $\mathsf{pt}\,\mathcal{L}(\mathbb{M}(D))$ are essentially the same? The answer to the first question is still open. About the second question, the final result of the next section entails that the answer is *no*. It also entails that $\langle \mathbb{M}(D), \sqsubseteq_S \rangle$ is *not* a domain. While this may sound discouraging, a proper characterization of the relation between $\mathbb{M}(D)$ and $\mathsf{pt}\,\mathcal{L}(\mathbb{M}(D))$ might illuminate where the problem with $\mathsf{pt}\,\mathcal{L}(\mathbb{M}(D))$ lies and what can be done about it.

5.2 Multisets logic and bag languages

Libkin & Wong (1993) defined a language for expressing operations on multisets (or bags): the *nested bag language*. The language includes operators for adding new elements to a bag, for constructing a bag singleton, for the bag union and for applying functions to elements in a bag. They also define the following operations for bags:

1 $count : \mathbb{M}(D) \to \mathbb{N}$. We have that $count(d, M)$ is $M(d)$.

2 $monus : \mathbb{M}(D) \times \mathbb{M}(D) \to \mathbb{M}(D)$. $monus(M, N) = M - N$.

3 $max : \mathbb{M}(D) \times \mathbb{M}(D) \to \mathbb{M}(D)$. $max(M, N) = P$, where

$$P(d) = \max\{M(d), N(d)\}.$$

4 $min : \mathbb{M}(D) \times \mathbb{M}(D) \to \mathbb{M}(D)$. $min(M, N) = P$, where

$$P(d) = \min\{M(d), N(d)\}.$$

5 $eq : D \times D \to \mathbb{B}$. Equality test.

6 $member : D \times \mathbb{M}(D)$. Membership test.

7 $subbag : \mathbb{M}(D) \times \mathbb{M}(D) \to \mathbb{B}$. It is the predicate $M \subseteq N$.

8 $unique : \mathbb{M}(D) \to \mathbb{M}(D)$. If $unique(M) = N$, then $N \subseteq M$ and $M(d) \geq 1$ if and only if $N(d) = 1$.

As they proved in their paper, operation 2 can express the operations 3–7. The operation *unique* is independent of the others and should be included as a primitive. They also introduced two different orders for multisets:

Definition 5.2.1 *Let $\langle D, \leq \rangle$ be a poset. Define the following orders on $\mathbb{M}(D)$.*

$$M \leq_{cwa} N \quad \text{if and only if} \quad N = monus(M, \{\!|a|\!\}) \uplus \{\!|b|\!\}, \quad \text{where } a \leq b$$
$$M \leq_{owa} N \quad \text{if and only if} \quad M \leq_{cwa} N \quad \text{or} \quad N = M \uplus \{\!|b|\!\}.$$

The orders $\leq_{cwa}$ and $\leq_{owa}$ are called closed world assumption order *and* open world assumption order*, respectively. The symbols $\trianglelefteq_{cwa}$ and $\trianglelefteq_{owa}$ denote their corresponding transitive closures.*

The open world and closed world assumption orders have two very interesting properties, viz:

Theorem 5.2.2 *Let D be a poset and let M and $N \in \mathbb{M}(D)$. Then the statements 1 and 2 hold:*

1. *The orders $\trianglelefteq_{cwa}$ and $\trianglelefteq_{cwb}$ on $\mathbb{M}(D)$ are partial orders.*
2. *There is an algorithm of $O(n^{5/2})$ time complexity which can verify whether $M \trianglelefteq_{cwa} N$ or $M \trianglelefteq_{owa} N$ (where n stands for the cardinality of the smallest of M and N).*

They appear as propositions 4.7 and 4.8 of Libkin (1994). Suppose that $M = \{\!|m_1, \ldots, m_k|\!\}$ and $N = \{\!|n_1, \ldots, n_p|\!\}$. The proof requires finding a permutation σ such that $m_i \leq n_{\sigma(i)}$ for every $i \leq k$. This is almost the definition of $\models$ in $\mathcal{L}(\mathbb{M}(D))$. Let us not allow the opportunity to escape:

Theorem 5.2.3 *Let D be a domain and let $\leq_S$ be the specialization order on D (see definition 5.1.3). If M and $N \in \mathbb{M}(D)$, define $\leq_{cwa}$ as*

$$M \leq_{cwa} N \qquad \text{if and only if} \qquad N = (M - \{\!|m|\!\}) \uplus \{\!|n|\!\}, \quad \text{with } m \leq_S n.$$

As before, $\trianglelefteq_{cwa}$ is the transitive closure of $\leq_{cwa}$. We introduce now another poset: $\langle D \cup \mathcal{L}(D), \leq_{\models} \rangle$ where

$$x \leq_{\models} \phi \qquad \text{if and only if} \qquad x \in D, \quad \phi \in \mathcal{L}(D) \quad \text{and} \quad x \models \phi,$$

and define $\trianglelefteq_{owa}$ on $\mathbb{M}(D) \cup \mathcal{L}(\mathbb{M}(D))$ using $\leq_{\models}$ as the basis. Then these statements are valid:

1. *If $\sqsubseteq_S$ is the specialization order on $\mathbb{M}(D)$, then $M \sqsubseteq_S N$ if and only if $M \trianglelefteq_{cwa} N$.*
2. *$M \models \Phi$ if and only if $M \trianglelefteq_{owa} \Phi$.*
3. *The statements $M \sqsubseteq_S N$ and $M \models \Phi$ can be verified in polynomial time (with respect to the size of M).*

Proof. 1. Suppose $M = \{\!|m_1, \ldots, m_k|\!\}$, $N = \{\!|n_1, \ldots, n_k|\!\}$ and $M \sqsubseteq_S N$. We want to prove that there is a permutation σ such that $m_i \leq_S n_{\sigma(i)}$. Take the formula

$$\Phi = \Box\{\!|x = m_1, \ldots, x = m_k|\!\}.$$

Obviously, $M \models \Phi$ and by hypothesis $N \models \Phi$, that is, there is a σ such that

$$n_{\sigma(1)} \models x = m_1, \ldots, n_{\sigma(k)} \models x = m_k.$$

But then $m_i \leq_S n_{\sigma(i)}$ for all $i \leq k$. Therefore $M \trianglelefteq_{cwa} N$.

For the inverse implication, we assume $M \trianglelefteq_{cwa} N$, ie there is a σ such that $m_i \leq_S n_{\sigma(i)}$. Now suppose that $M \models \Box\{\!|\phi_1, \ldots, \phi_k|\!\}$. Then there is a permutation τ with the virtue that

$$m_1 \models \phi_{\tau(1)}, \ldots, m_k \models \phi_{\tau(k)}.$$

Hence, by definition of $\leq_S$, we obtain:

$$n_{\sigma(1)} \models \phi_{\tau(1)}, \ldots, n_{\sigma(k)} \models \phi_{\tau(k)},$$

and we conclude that $N \models \Phi$.

2. Suppose $M \models \Phi$. If $\Phi = \Box\{\!|\phi_1, \ldots, \phi_k|\!\}$ then there is a permutation σ which gives

$$m_1 \models \phi_{\sigma(1)}, \ldots, m_k \models \phi_{\sigma(k)},$$

ie, $m_i \leq_{\models} \phi_{\sigma(i)}$ for all $i \leq k$ and then $M \trianglelefteq_{cwa} \Phi$.

This time let $\Phi = \Diamond\{\!|\phi_1, \ldots, \phi_p|\!\}$ (with $p \leq k$ by definition of $\models$ in $\mathcal{L}(\mathbb{M}(D))$). Remember that $\Diamond\{\!|\phi_1, \ldots, \phi_p|\!\} = \bigvee_{m \in \mathbb{N}} \Box(\{\!|\phi_1, \ldots, \phi_p|\!\} \uplus \{\!|t|\!\}^m)$. Then there is a permutation τ such that

$$m_{\tau(1)} \models \phi_1, \ldots, m_{\tau(p)} \models \phi_p.$$

Regarding the other $m_{\tau(j)}$'s, it is clear that

$$m_{\tau(p+1)}, \ldots, m_{\tau(k)} \models t,$$

and then $M \trianglelefteq_{owa} \Phi$.

Now, if $M \trianglelefteq_{owa} \Phi$, $M \models \Phi$ comes from the definition of $\models$ in $\mathcal{L}(\mathbb{M}(D))$.

3 follows trivially from theorem 5.2.2. ■

This result has a series of very important consequences:

- $\langle \mathbb{M}(D), \sqsubseteq_S \rangle$ is not a domain. The specialization order coincides with $\preceq_{cwa}$ and this order, in turn, is equivalent to Heckmann's lower bag-domain, *which does not preserve Scott domains*.
- It is not the case that $\langle \mathbb{M}(D), \sqsubseteq_S \rangle \simeq \langle \mathsf{pt}\,\mathcal{L}(\mathbb{M}(D)), \sqsubseteq_S \rangle$. This follows from the fact that $\langle \mathsf{pt}\,\mathcal{L}(\mathbb{M}(D)), \sqsubseteq_S \rangle$ is a domain (theorems 5.1.5 and 5.1.8).
- If the relation $\models$ in $\mathcal{L}(D)$ can be verified in polynomial time, then $\models$ in $\mathcal{L}(\mathbb{M}(D))$ can be checked in polynomial time, too. This suggests that, in principle, it is possible to implement the multiset logic in an efficient way. In the same way, reaction conditions can be tested in polynomial time. In chapter 6 we will propose some possible ways of taking advantage of this result.

And now it is time to move on to the final topic of the chapter.

5.3 Program transformations

Hankin et al. (1998) analyzed the problem of the pipelining transformation in the context of Gamma programming. 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 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. Again, Hankin et al. (1998) proposed conditions that would guarantee the correctness of the transformation. Weichert (1999) proved that those conditions needed to be strengthened for the transformation to preserve some refinement properties. The aim of this section is to set this work in the language of the multiset logic and show

how the latter can help understand and prove the conditions. Let us start with Weichert's (1999) presentation of the problem.

In addition to the program orders $\sqsubseteq_{IO}$ and $\sqsubseteq_C$ an order based in state-program pairs can be considered:

Definition 5.3.1 *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 $\langle P_1, M\rangle \to \langle P_1', M'\rangle$ then there are P_2', N' such that $\langle P_2, N\rangle \to^* \langle P_2', N'\rangle$ and $(\langle P_1', M'\rangle, \langle P_2', N'\rangle) \in R$;*
3. *if $\langle P_1, M\rangle \to M$ then there are P_2', N' such that $\langle P_2, N\rangle \to^* \langle P_2', N'\rangle$, $(\langle P_1, M\rangle, \langle P_2', N'\rangle) \in R$ and $\langle P_2', N'\rangle \to N'$.*

Finally $P_1 \sqsubseteq_{SB} P_2$ 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 $\equiv_{SB}$.

Consider the following Gamma programs:

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

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 program order 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, Weichert (1999) 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 reenable 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, Weichert (1999) proved that ($**$) implies:

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

On the other hand, when $\sqsubseteq_{IO}$ is considered 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? Weichert (1999) assumed implicitly there is a way of establishing both conditions, without going any further in that direction. Fortunately, the multiset logic of chapter 4 gives us such a method.

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 $M \models (\Diamond\{\!|\phi_1, \ldots, \phi_n|\!\} \wedge \Box\{\!|z \in g_1(D^p), \ldots, z \in g_k(D^p)|\!\}) \vee (\Box\{\!|\phi_1, \ldots, \phi_n|\!\} \wedge \Diamond\{\!|z \in g_1(D^p), \ldots, z \in g_k(D^p)|\!\})$.

Both conditions can be proved (or disproved) using the rules of chapter 4. As a matter of fact, and assuming that the logic of the elements in D is complete, the test of $(*')$ and $(**')$ becomes 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

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

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

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

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

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

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

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}).

It is time for applications now that the notation has been explained. Ciancarini et al. (1996) offered this example, also analyzed by Weichert (1999):

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

Weichert proved 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.

Let us summarize the results of the chapter: we presented a more theoretical setting for the multiset logic, introducing the set $\mathsf{pt}\,\mathcal{L}(\mathbb{M}(D))$, with some positive results (for instance, coherence of $\mathcal{L}(\mathbb{M}(D))$). But also some negative results were found. We explored the relationship between the multiset logic and Libkin and Wong's multiset orders. We had positive and negative results too. Among the former, the possibility of implementation of the

multiset logic in polynomial time. Among the latter, the non-preservation of Scott domains when the specialization order for multisets is considered. And finally, we developed an application of the multiset logic in the analysis of the pipelining transformation.

Chapter 6

Conclusions and further work

It is time to summarize the results achieved in this thesis. This will allow us to present an account of possible extensions of the work and to delineate a brief research programme.

6.1 What has been done

Let us recapitulate the aims of this thesis before starting with the conclusions. The two most important ones were to find a fully abstract semantics for Gamma and to develop a proof system for the language. Some other issues appeared on the way: the connections between the multiset logic (a key component of the proof system) and previous work on multiset languages (specially that of Libkin & Wong (1993)), as well as another theoretical question: what is the relationship between multisets and their localic counterpart.

A proof system has no point if it is not applied to prove properties of programs. Apart from the correctness proofs in chapter 4, another example of application seemed necessary. The chosen subject was the pipelining transformation. We will review now the results in each of this areas:

Gamma semantics. Although there are alternative operational semantics for Gamma (Hankin et al. (1993), Chaudron (1998), Ciancarini et al. (1996)), all of them have been well studied and a clear picture has emerged. This

was definitely not the case for the denotational semantics. The resumption semantics of Gay & Hankin (1996*b*) proposed an interesting model for the language, but it failed to be fully abstract. Sands (1993*a*) put forward the transition trace semantics, based on Brookes's (1993) methods. But again, full abstraction was missing.

Our proposal used the transition trace semantics as a starting point, restricting the unbounded behaviour of the so-called environment. In so doing, the new denotational model became fully abstract. Additionally, this allowed us to lay the foundations for a new attempt at applying DTLF to Gamma (with the precedents of Gay & Hankin (1996*b*) and Gay & Hankin (1996*a*)).

Gamma logic. The work on proof systems for Gamma is extensive, as has already been pointed out at the start of chapter 4. Beginning with the method advanced by Banâtre & Le Métayer (1990) and then going to the transition trace logic of Gay & Hankin (1996*a*), passing through Errington et al. (1993), we have various logical systems or techniques for Gamma. We can add to the stock the work by Chaudron (1998) and Reynolds (1996). Nevertheless, none of these examples constitutes a full proof system addressing *at the same time* all the following issues: a) a multiset logic; b) proofs of correctness of programs; c) termination.

The situation is now different: not only has our Gamma logic dealt with these three points, but it has also done so with a fully integrated proof system, equipped with suitable axioms and inference rules. A proof of its soundness and conditional completeness was also given. Moreover, this Gamma logic has its theoretical foundations on the denotational semantics previously introduced and it thus creates a rounded view of the language.

Multisets in databases. Databases have been a very well studied field in computer science. Some of the techniques and results developed in that area are of potential utility to Gamma logic and programming, as multisets play a central role in both.

We have established some connections with Libkin & Wong's (1993) bag orders. This has some nice practical implications regarding efficiency

of both Gamma implementations and proof system. For example, it would be possible to test reaction conditions in polynomial time.

Some negative results have also emerged—namely, the non-preservation of Scott domains when we go from basic types to multisets of these types. This is a consequence of Heckmann's (1995) theorem.

The localic version of the multiset logic. DTLF opened up a rich series of theoretical perspectives. Our multiset logic arose from practical considerations, but its potential theoretical scope is an equally interesting subject. In this work we have started to explore the relationship between the "ground" view and the localic view of the multiset logic. Coherence of the multiset logic was proved. But the assessment of the exact relationship between $\mathbb{M}(D)$ and $\mathsf{pt}\,\mathcal{L}(\mathbb{M}(D))$ is not complete yet, although we know they are not the same: if D is a Scott domain, $\mathsf{pt}\,\mathcal{L}(\mathbb{M}(D))$ is so too, but $\mathbb{M}(D)$ does not preserve Scott domains, as we said above.

Program transformation. Hankin et al. (1998) and Weichert (1999) have analyzed the requirements for a successful application of pipelining transformation, one of the most common instances of program transformation. Particularly, the latter stated and proved a series of propositions which cover a full range of possibilities: atomic rules and non-simple programs, on the one hand, refinements based on input-output preserving order or on state-based simulation, on the other. But in his paper, he assumed we already have a way of proving some essential properties of reaction conditions and their combination in parallel or sequential programs. We believe that the final section of chapter 5 fills this implicit gap with a set of explicit rules.

After this summary of results, it is the turn of describing what has been left for the future.

6.2 What is to be done

The most straightforward way of continuing the work of this thesis is to round off some of the results—let us call this task *completion*. The second (and more challenging) way is to *extend* and apply the ideas to different subjects.

• On the semantics side, some completions could be done. The interaction between Gamma semantics (based on sets of multiset-pair traces) and multisets semantics (Heckmann's (1995), Gunter's (1992) mixed domain and Vickers's (1992) bag domain) can be explored.

The new transition trace model solved the full abstraction problem at a cost: incurring what Winskel (1993) called *encoding the operational semantics into the denotational description*. Although there is nothing wrong with this approach in itself, a more mathematically abstract description could facilitate the extension of our method to other languages. It would also provide a deeper insight into the denotational semantics of parallel languages in general.

• Anyway, as it stands, our semantical model can be extended to cover some developments of the Gamma language. First of all, some alternative operators have been suggested by Sands (1993*a*): the *vanilla parallel composition* and the *nondeterministic choice*

$$P \parallel Q \qquad \text{and} \qquad P \vee Q, \qquad \text{respectively.}$$

In contrast to the operator $+$, the vanilla parallel composition does not require synchronous termination of its operands, as shown by the following SOS rules:

$$\frac{\langle Q, M\rangle \to M}{\langle P \parallel Q, M\rangle \to \langle P, M\rangle} \qquad \frac{\langle P, M\rangle \to M}{\langle P \parallel Q, M\rangle \to \langle Q, M\rangle}.$$

The nondeterministic choice takes one of its operands and executes it, discarding the other. The decision of which of them to take is made before *any* evaluation has happened:

$$\langle P \vee Q, M\rangle \to \langle P, M\rangle \qquad \langle P \vee Q, M\rangle \to \langle Q, M\rangle.$$

The semantic function $T : \mathbb{G} \to \mathcal{P}_l(\mathbb{T}(\mathbb{M}(D)))$ can accommodate these new operators straightforwardly—and so they can be integrated into the transition trace model as a whole, without altering full abstraction in all likelihood.

• Gamma is a first order programming language: programs cannot be composed into higher order programs. Le Métayer (1994) addressed this limitation, producing higher-order Gamma. Let us make a brief introduction of the higher-order language.

The distinction between programs and multisets disappears in the extended language. The basic component of higher-order Gamma is the configuration:

$$[P, V_1 = M_1, \ldots, V_n = M_n]$$

where P is a program (including the "empty" program $\emptyset$) and each of the M_i's is a multiset expression. The latter have the following syntax

$$M := \emptyset \mid M \uplus M \mid M - M \mid \{\!|A|\!\}.$$

In this last definition A is a variable which stands for an action, whose syntax is

$$A := C \mid C.V_i \mid x_i \mid M \mid \langle \text{other} \rangle$$

where C is a configuration, x_i is a single variable, M is a multiset expression and $\langle$other$\rangle$ can be any function as defined in first order Gamma actions. The operation $C.V_i$ extracts the stable multiset expression M_i from the configuration C. A configuration is called *passive* if $P = \emptyset$, and *active* otherwise.

In non-formal terms, the program part of a configuration reacts with the multisets M_i (which can themselves contain configurations) until no further reaction is possible. Then the configuration becomes passive. We have two new SOS rules for the operational semantics:

$$\frac{M \to M'}{\{\!|M|\!\} \uplus N \to \{\!|M'|\!\} \uplus N}$$

$$\frac{M_i \to M_i'}{[P, V_1 = M_1, \ldots, V_i = M_i, \ldots] \to [P, V_1 = M_1, \ldots, V_i = M_i', \ldots]},$$

that is, the computation of an active configuration can happen inside a multiset and a multiset containing an active configuration can change inside a higher-order configuration.

As far as I am aware, there is no work on the denotational semantics of higher-order Gamma, making it an obvious topic to apply the ideas of this thesis.

• I already mentioned the possibility of using the same methods to produce fully abstract semantics for other languages with parallelism. This is also a clear way of extending our present work.

• On the logical side, we can consider some extensions too:

  ◇ Apart from the application of the multiset logic to the pipelining transformation, some other refinement and program development techniques are susceptible of a similar treatment.

  ◇ Given the functional setting of the denotational semantics and the existence of a bag language of polynomial complexity, automation or semi-automation of proofs in Gamma logic may be not so difficult to implement.

  ◇ The multiset logic is general enough to be applicable to other contexts apart from the Gamma proof system. Again, database theory springs to mind.

• At last, we have the completion of some of the work in chapter 5:

  ◇ To find what is the exact relationship between $\mathbb{M}(D)$ and $\mathsf{pt}\,\mathcal{L}(\mathbb{M}(D))$ in the locale.

  ◇ To relate our multiset logic with the domain-like logic which can be built using Heckmann's (1995) bag-domain.

Bibliography

Abramsky, S. (1987), Domain Theory and the Logic of Observable Properties, PhD thesis, University of London.

Abramsky, S. (1991), 'Domain theory in logical form', *Annals of Pure and Applied Logic* **51**, 1–77.

Abramsky, S. & Jung, A. (1990), Domain theory, *in* Abramsky, Gabbay & Maibaum (1990).

Abramsky, S., Gabbay, D. & Maibaum, T., eds (1990), *Handbook of Logic in Computer Science*, Oxford University Press.

AMS (1994), *Proceedings of the* DIMACS *workshop on specifications of parallel algorithms*, Dimacs series in Discrete Mathematics, American Mathematical Society.

Andreoli, J.-M., Hankin, C. & Le Métayer, D., eds (1996), *Coordination Programming: Mechanisms, Models and Semantics*, Imperial College Press.

Banerjee, U., Gelernter, D., Nicolau, A. & Padua, D., eds (1993), *Proc. 5th Workshop on Languages and Compilers for Parallel Computing, Berlin 1992*, Lecture notes in Computer Science 757, Springer.

Banâtre, J. & Le Métayer, D. (1990), 'The GAMMA model and its discipline of programming', *Science of Computer Programming* **15**, 55–77.

Banâtre, J.-P. & Le Métayer, D. (1993), 'Programming by multiset transformation', *Communications of the ACM* **36 (1)**, 98–111.

Barnsley, M. (1993), *Fractals Everywhere*, Academic Press.

Berry, G. & Boudol, G. (1992), 'The chemical abstract machine', *Theoretical Computer Science* **96**, 217–248.

Bjørner, D., Broy, M. & Pottosin, I. V., eds (1993), *Formal Methods in Programming and their Applications, International Conference, June/July 1993*, Lecture Notes in Computer Science 735, Springer.

Bourgois, M. (1997), Advantage of formal specifications: a case study of replication in Lotus notes, *in* Najm & Stefani (1997).

Brookes, S. (1985), An axiomatic treatment of a parallel programming language, *in* Parikh (1985).

Brookes, S. (1992), An axiomatic treatment of partial correctness and deadlock in a shared variable parallel language, Technical Report CMU-CS-92-154, School of Computer Science, Carnegie Mellon University.

Brookes, S. (1993), Full abstraction for a shared variable parallel language, *in* IEE (1993), pp. 98–109.

Broy, M. & Schmidt, G., eds (1982), *International Colloquium on Automata, Languages and Programs*, Lecture notes in Computer Science 140, Springer.

Burn, G., Gay, S. & Ryan, M., eds (1993), *Theory and Formal Methods 1993*, Workshops in Computing, Springer.

Chandy, K. M. & Misra, J. (1988), *Parallel Program Design: a Foundation*, Addison-Wesley.

Chaudron, M. R. (1998), Separating Computation and Coordination in the Design of Parallel and Distributed Programs, PhD thesis, Riijksuniversiteit Leiden.

Ciancarini, P. & Wolf, A. L., eds (1999), *Coordination Languages and Models, 3rd International Conference Coordination '99, Amsterdam, The Netherlands, April 1999*, Lecture Notes in Computer Science 1594, Springer.

Ciancarini, P., Gorrieri, R. & Zavattaro, G. (1996), An alternative semantics for the parallel operator of the calculus of Gamma programs, *in* Andreoli, Hankin & Le Métayer (1996).

Creveuil, C. & Moguérou, G. (1991), 'Développment systématique d'un algorithme de segmentation d'images à l'aide de Gamma', *Techniques et Science Informatiques* **10**(2), 125–137.

Davey, B. & Priestley, H. (1990), *Introduction to Lattices and Order*, Cambridge Mathematical Textbooks, Cambridge University Press.

Dershowitz, N. & Manna, Z. (1979), 'Proving termination with multiset ordering', *Communications of the ACM* **22**, 465–476.

Dijkstra, E. (1976), *A Discipline of Programming*, Prentice-Hall.

Edalat, A., Jourdan, S. & McCusker, G., eds (1996), *Advances in Theory and Formal Methods, Theory and Formal Methods Workshop 1996*, Imperial College.

Errington, L., Hankin, C. & Jensen, T. (1993), Reasoning about Gamma programs, *in* Burn, Gay & Ryan (1993).

Fourman, M., Johnstone, P. & Pitts, A., eds (1992), *Applications of Categories in Computer Science, Proceedings of the LMS Symposium Durham, 1991*, London Mathematical Society Lecture Note Series 177, Cambridge University Press.

Fradet, P. & Le Métayer, D. (1996), Structured gamma, Technical Report PI-989, Irisa.

Gay, S. & Hankin, C. (1996*a*), Gamma and the logic of transition traces, *in* Edalat, Jourdan & McCusker (1996).

Gay, S. & Hankin, C. (1996*b*), A program logic for Gamma, *in* Andreoli et al. (1996).

Gödel, K. (1930), 'Vollständigkeit der axiome des logischen funktionenkalküls', *Monatshefte für Mathematik und Physik* **37**, 349–360.

Gödel, K. (1931), 'Über formal unentscheidbare sätze der principia mathematica und verwandter systeme i', *Monatshefte für Mathematik und Physik* **38**, 173–198.

Gunter, C. (1992), 'The mixed powerdomain', *Theoretical Computer Science* **103**, 311–334.

Hankin, C., Le Métayer, D. & Sands, D. (1993), A calculus of Gamma programs, *in* Banerjee, Gelernter, Nicolau & Padua (1993).

Hankin, C., Le Métayer, D. & Sands, D. (1998), 'Refining multiset transformers', *Theoretical Computer Science* **192**, 233–258.

Heckmann, R. (1995), 'Lower bag domains', *Fundamenta Informaticae* **24**(3), 259–281.

Hennessy, M. & Milner, R. (1985), 'Algebraic laws for non-determinism and concurrency', *Journal of the ACM* **85**(32), 137–161.

Hernández Quiroz, F. (1998), A multiset logic for gamma, Theory and Formal Methods Workshop 1998, Bath.

Hernández Quiroz, F. (1999), An extension of a gamma proof system, Coordina Workshop. Amsterdam.

Hoare, C. (1969), 'An axiomatic basis for computer programming', *Communications of the ACM* **12**, 576–580.

IEE (1993), *Eighth Annual IEEE Symposium on Logic in Computer Science*, IEEE Computer Society Press.

Jensen, T. (1992), Abstract Interpretation in Logical Form, PhD thesis, Department of Computing, Imperial College of Science, Technology and Medicine.

Johnstone, P. (1982), *Stone Spaces*, Cambridge Studies in Advanced Mathematics 3, Cambridge University Press.

Lamport, L. (1994), 'The temporal logic of actions', *ACM Transactions on Programming Languages and Systems* **16**(3), 872–923.

Le Métayer, D. (1994), Higher-order multiset programming, *in* AMS (1994).

Libkin, L. (1994), Aspects of Partial Information in Databases, PhD thesis, Department of Computing Science, University of Pennsylvania.

Libkin, L. & Wong, L. (1993), Query languages for bags, Technical Report MS-CIS-93-36, Department of Computer Science, University of Pennsylvania.

Libkin, L. & Wong, L. (1995), 'On representation and querying incomplete information in databases with multisets', *Information Processing Letters* **56**, 209–214.

McEvoy, H. (1996), Gamma, chromatic typing and vegetation, *in* Andreoli et al. (1996).

Najm, E. & Stefani, J.-B., eds (1997), *Formal Methods for Open Object-based Distributed Systems. International Workshop 1996, Paris*, Chapman & Hall.

Parikh, R., ed. (1985), *Logics of Programs*, Lecture Notes in Computer Science 193, Springer.

Plotkin, G. (1981), *Domains (aka Post-graduate Lectures in Advanced Domain Theory)*, Dept. of Computer Science, University of Edinburgh.

Plotkin, G. (1983), A metalanguage for predomains, *in* Programming Methodology Group (1983).

Programming Methodology Group, ed. (1983), *Workshop on the Semantics of Programming Languages*, Chalmers University of Technology.

Reynolds (1996), Temporal semantics for Gamma, *in* Andreoli et al. (1996).

Sands, D. (1993*a*), A compositional semantics of combining forms for Gamma programs, *in* Bjørner, Broy & Pottosin (1993).

Sands, D. (1993*b*), Laws of parallel synchronised termination, *in* Burn et al. (1993).

Sands, D. (1996), Composed reduction systems, *in* Andreoli et al. (1996).

Scott, D. (1982), Domains for denotational semantics, *in* Broy & Schmidt (1982).

Smyth, M. (1983), 'The largest cartesian closed category of domains', *Theoretical Computer Science* **27**, 109–119.

Stone, M. (1936), 'The theory of representations for boolean algebras', *Transactions of the American Mathematical Society* pp. 37–111.

Vickers, S. (1989), *Topology via Logic*, Cambridge Tracts in Theoretical Computer Science 5, Cambridge University Press.

Vickers, S. (1992), Geometric theories and databases, *in* Fourman, Johnstone & Pitts (1992).

Weichert, M. (1999), Pipelining the molecule soup: a plumber's approach to Gamma, *in* Ciancarini & Wolf (1999).

Winskel, G. (1993), *The Formal Semantics of Programming Languages*, The MIT Press.

Zhang, G. (1991), *Logic of Domains*, Progress in Theoretical Computer Science, Birkhäuser, Boston.