Preface

Present-day programming languages exhibit an overwhelming variety of control flow notions. Our book is devoted to a comprehensive treatment of these, altogether developing operational and denotational semantic models for control flow in 27 languages. More specifically:

- We treat both traditional (e.g., assignment, iteration and recursion, locality) and modern concepts (such as process creation, action refinement, parallel objects, and the rendezvous construct).
- We deal with both *sequential* and *concurrent* languages, on the whole paying about equal attention to each of them.
- We investigate control flow concepts in imperative programming, but also—albeit to a lesser extent—in logic, object-oriented, and functional programming.
- Both schematic ("uniform," in our terminology) and state-based (interpreted or "nonuniform") languages are studied. In the former, the primitive actions are uninterpreted symbols, in the latter the primitives affect the state (e.g., through assignment statements).
- Both deterministic and (possibly unbounded) nondeterministic notions are considered. As to forms of parallelism, both uniform and nonuniform languages, and static and dynamic configurations will be treated. Not only interleaving but also noninterleaving models are presented, though in rather less detail.
- Throughout, finite and infinite behaviour will be specified in full detail.

For each of the 27 languages we develop its operational semantics based on the design of an appropriate labeled transition system specification (in the style of Plotkin's Structural Operational Semantics). In the denotational modeling, we apply the usual compositionality principle in mapping language constructs to elements of a mathematical domain, allowing the application of fixed point techniques to handle recursion. Sometimes, these domains have a linear structure, in that they consist of (functions to) (sets of) (finite or infinite) sequences. Other, more difficult ones are specified as solutions of *domain equations.* In all cases, we provide precise statements on how the operational ($\mathcal{O}$) and denotational ($\mathcal{D}$) models are related.

A large body of semantic definitional techniques will be used in designing these models. We list a few of them:

- We use both uniform (labeled) transition systems, with transitions of the form $s \xrightarrow{a} s'$, and systems with nonuniform transitions, written as $[s,\sigma] \rightarrow [s',\sigma']$. Here

s, s' are statements from the language at hand, a is an atomic action, and σ, σ' are semantic states (i.e., mappings from variables to values).

- We apply both finitely branching and image-finite transition systems, the latter to handle unbounded nondeterminacy.
- In the denotational semantics, we often use the powerful tool of *continuations.* In the design of the transition system specifications, we exploit their—not so standard—syntactic counterparts which we have baptized *resumptions.* The interplay between resumptions and continuations will be of the essence in most of the $\mathcal{O} = \mathcal{D}$ (or variations thereof) proofs.

All models to be described are based on mathematical entities (such as sequences, tree-like objects, or an occasional pomset) which provide a *full history* of the computation at hand. Our main motivation for this kind of modeling—to be contrasted in particular with approaches based on deriving the input/output function associated with a program—is

- the wish to deal with sequential and concurrent systems in one unified framework, together with
- the increased importance of *reactive,* in particular *embedded,* systems in contemporary programming practice, explaining
- the desire to treat finite and infinite behaviour as equally valid types of computation.

In accordance with the above, we have based our semantics on the use of topological, in particular *metric* structures. To be precise, we use *complete* metric spaces: all relevant limits have to be present to cater for infinite computations. We have several reasons for this choice:

- In a history-based model, there is a natural definition of a *distance* between two computations, viz. the following form of the classical Baire metric: two computations have a distance 2^{-n} when the first difference in their behaviour appears after n steps.
- The Baire metric is the starting point for the fruitful exploitation of a number of topological tools. In particular, we refer here to Banach's theorem, stating that each contractive function on a complete metric space has a unique fixed point.
- The use of Banach's theorem is pervasive throughout our work. It enables us to define—by uniqueness of the fixed point of an appropriate (higher-order) operator—both $\mathcal{O}$ and $\mathcal{D}$. Moreover, all semantic operators, ranging from, e.g., simple sequential composition through shared-variable parallelism to the rendezvous operator, are defined as fixed point of some higher-order mapping.

- Semantic domains $\mathbb{P}$ solving domain equations (isometries, to be precise) over complete metric spaces of the form $\mathbb{P} \simeq \mathcal{F}(\mathbb{P})$, for $\mathcal{F}$ a monotonic ("covariant") functor, may be solved as well by an appeal to Banach's theorem. The crucial step here is to set up a framework in which a distance between two complete metric spaces can be defined under certain conditions. Only for $\mathcal{F}$ nonmonotonic or "contravariant" will it be necessary to use tools from category theory. In an Appendix (written by J.J.M.M. Rutten), the solution method developed by America/Rutten (1989) will be described.

The above considerations should not be construed as claiming that metric spaces are the preferred mathematical structures in all semantic modeling. A crucial condition for their successful use is a natural intuition of *computational step* in the language paradigm at hand, such that performing the step contributes an observational element to the computational history. Once this condition is fulfilled, we have no need for the distinction between partial and total elements, a basic ingredient of all order-based semantics. For certain settings—the (untyped) lambda calculus being the prime example—our methodology is not suitable.[1]

We conclude this overview of the design principles of our work with one final remark: the number of languages treated may seem inordinately high and the question may arise whether, after dealing with a good part of them, treatment of further ones might not turn out to become a routine exercise. It has been our experience in studying the selection of 27 languages—still only a small sample of the richness and diversity in the world of programming languages—that this is not at all the case. Though the overall methodology in our semantic modeling stays the same throughout the book, in each individual case considerable variations turned out to be necessary, e.g., in the specification of the transition systems, in the definition of the underlying mathematical structures, or in the argument linking the operational and denotational semantics.

All references pertaining to the book are collected in the bibliographical notes at the end of each chapter. We have aimed at completeness in acknowledging the direct sources of our work, and at providing representative samples of related research. The field of semantics has become too vast to strive for completeness for the latter.

In the Appendices, we have put together some miscellaneous information. Appen-

[1]The domain equation $\mathbb{P} \simeq \mathbb{P} \xrightarrow{1} \mathbb{P}$ (the right-hand side denoting the function space of all nonexpansive mappings from $\mathbb{P}$ to $\mathbb{P}$) has only the one-point space as (ultra) metric solution. On the other hand, for nontrivial constants A, equations such as $\mathbb{P} \simeq A + (\mathbb{P} \xrightarrow{1} \mathbb{P})$ or $\mathbb{P} \simeq A \times (\mathbb{P} \xrightarrow{1} \mathbb{P})$ do have proper solutions.

dices A and B supply proofs of topological facts which we considered somewhat too advanced to incorporate them in the main text. Appendix C (by J.J.M.M. Rutten) describes a category-theory based method to solve domain equations $\mathbb{P} \simeq \mathcal{F}(\mathbb{P})$, where $\mathcal{F}$ is a functor which is possibly not covariant. (Chapter 18 illustrates the need for such an $\mathcal{F}$.) Appendix D provides some hints on further reading.

Our book has been written as an advanced text for senior undergraduate or graduate students and as a reference work for researchers interested in programming languages and their semantics, in concurrency theory, and in application of topology in computer science. To serve the latter audience, considerably more material has been included than can be covered in a one semester course on semantics. For such a course, the instructor might cover Chapters 1, 2, 4, a selection of (the first sections of) Chapters 5 to 8, and, possibly, Chapter 11. In the development of the metric framework (Section 1.1.2, Section 2.1), we have aimed at a complete—though elementary—treatment of the material, including full proofs of the results. In the subsequent semantic applications, it is usually sufficient to work solely with the theorems; familiarity with the techniques used in their proofs is not needed anymore. This holds as well for Chapter 11: Though it builds on the theory of domain equations of Chapter 10, it should be possible to provide only a quick introduction to the main results of this theory, and then to proceed with their application in the use of branching domains to model concurrency.

In order to enhance its use as a textbook, all (but one) of the chapters are provided with exercises, of a varying level of difficulty.

As a text for undergraduates, a substantial part of the material just indicated has been classroom tested for several years at the Vrije Universiteit Amsterdam. Graduate courses based on selected parts of our work have been given at various universities and institutes in Europe (Amsterdam, Rome, Warsaw, Turku, Turin). It seemed especially appropriate that the course in Warsaw was hosted by the Stefan Banach Centre.

Acknowledgments
We first learned about the use of metric spaces in semantics from Maurice Nivat through a memorable lecture series he gave in Amsterdam in 1978. Of seminal importance for our work was the collaboration with Jeff Zucker in the early eighties. We mention in particular his essential share in the development of the (ultra)metric perspective on solving domain equations, and their use in the modeling of imperative concurrency. In the early stages of the research reported here, John-Jules Meyer and Ernst-Rüdiger Olderog made a number of important contributions to

it. Pierre America, Joost Kok and Jan Rutten have—in various combinations of joint papers, the majority of which were to become part of their Ph.D. theses—contributed decisively to the further development of the methodology set forth in our book. We are especially grateful to Jan Rutten who continued to play a leading role in the shaping of the more advanced parts of the theory. In addition, he read most of the drafts of our work—resulting in many amendments—and wrote Appendix C on the categorical approach to the solving of domain equations.

We owe our greatest debt to Franck van Breugel. He carefully scrutinized all successive drafts, providing detailed comments and suggestions for corrections and improvements on countless occasions. If the persons mentioned above were instrumental in the starting of the project, Franck was essential for its completion. Besides all this, a number of sections in the book are based on joint papers with Franck, and several significant ideas from his thesis have found their way into it.

We are furthermore grateful to many people for contributing to the sources on which our investigations are based, for critically reading (parts of) the many drafts of the material, and, more in general, for providing interest and encouragement during the demanding years of preparing the book. In the first category—restricting ourselves to sources from the Amsterdam group and its associates—we are indebted to Jan Bergstra, Frank de Boer, Arie de Bruin, Anton Eliëns, Eiichi Horita, Jan Willem Klop, Catuscia Palamidessi, Marcel de Rooy, Frank Stolker (for several exercises), and Jeroen Warmerdam. In the second group we owe much to the constructive comments by Marcello Bonsangue—who carefully went through the whole text and also prepared the indices—to Bart Jacobs, both for critical remarks and for incisive discussions on the principles of our enterprise, to Jean-Marie Jacquet (mostly pertaining to Chapter 15), and to Daniele Turi. For general support we are thankful to Ralph Back, Mariangiola Dezani, Tony Hoare, Furio Honsell, Marta Kwiatkowska, Mila Majster, Prakash Panangaden, Simona Ronchi della Rocca, and John Tucker.

Finally, we wish to express our thanks to Robert Prior at The MIT Press, as well as Albert Meyer and Michael Garey, for their willingness to publish the book in their Foundations of Computing series.

Jaco de Bakker
CWI and Vrije Universiteit
Amsterdam

Erik de Vink
Vrije Universiteit
Amsterdam