Semantics

J.W. de Bakker

1. History

Programs are written in a programming language, and serve as a means to instruct a computer to perform a given task. As linguistic entities, programs have form and meaning. In the specification of their form, one employs syntactic rules, usually in the form of some grammatical formalism. In semantics, one is concerned with defining meanings of programs in terms of a mathematical model. For $\mathcal{L}$ a programming language (the reader may think of PASCAL, ML or PROLOG as typical examples), one looks for a set $\mathcal{P}$ of mathematical objects and a meaning function $\mathcal{M} : \mathcal{L} \rightarrow \mathcal{P}$ (*) such that, for each $s \in \mathcal{L}$ (each program), one determines its meaning $\mathcal{M}(s) = p$ as object in $\mathcal{P}$. One advantage of having such a meaning function is that we get a notion of equivalence of programs: two programs s_1, s_2 are equivalent if they have the same meaning, i.e. $\mathcal{M}\ (s_1) = \mathcal{M}\ (s_2)$. For example, two procedures may be seen as equivalent if they compute the same function (even though they may be 'programmed' in different ways). In general, the design of semantic models is a rather demanding endeavour. Programming languages are complex entities, and so are the computations specified by the programs of the language. Accordingly, ever since the advent of high-level programming languages (say from 1960 onwards), a rich body of methods and tools has been developed to be used for this purpose.

In the period 1960 to 1970, the emphasis was on the use of general techniques from the theory of computability for the formal definition of (syn-

tax and) semantics of programming languages, e.g., based on (generalized) Markov algorithms, or Van Wijngaarden's two level grammars. Indeed, thanks to the universality of these definitional systems, it was not surprising that complete formal definitions could be given. What was lacking in these definitions was sufficient abstraction from representational (and often arbitrary) detail. Sheer symbol manipulation was often the prevalent approach.

1.1. Denotational semantics

Owing to the pioneering work of D.S. Scott around 1970, the study of semantics returned to the treasured principles of mathematical logic, viz. (i) definitions should be *compositional* (a classic principle due to G. Frege) and (ii) definitions should clearly separate the linguistic realm from the mathematical structure(s) (the domain(s) of interpretation) to which the linguistic constructs are mapped (the $\mathcal{P}$ from definition (*) above). These principles are fundamental for the style of so-called denotational semantics, which has remained one of the major methodologies in semantics till the present day. An even more seminal contribution of Scott was the design of semantic models for the lambda calculus—and many more related languages—couched in the framework of a general theory of (lattice-theoretic) domain equations. Jointly with his coworker, the late C. Strachey, Scott laid the foundations for the semantic analysis of a host of—mostly sequential—programming languages.

1.2. Structural operational semantics

Subsequently, extensions of the general theory were proposed by G.D. Plotkin, especially to cover as well the notions of nondeterminacy and parallelism. Around 1980, two further innovative developments took place. Firstly, the notion of so-called *structural operational semantics* (SOS) was introduced by Plotkin. Its origin can be traced back to automata theory: a transition system $(S, A, \rightarrow)$ consists of a set of *states* S, a set of *actions* A, and a transition relation $\rightarrow \subseteq S \times A \times S$. In automata theory, one would write $\delta(s, a) = s'$, in the SOS-oriented semantics the same fact is written as $s \xrightarrow{a} s'$ (†). Plotkin's idea was to instantiate the abstract set of states S to a concrete set, viz. the set $\mathcal{L}$ of statements in a programming language, and to read (†) as: statement s performs an a-step and then turns into the statement s'—which may, in turn, make a b-step, etc. The formalism of transitions such as (†) turned out to be especially fruitful in the study of *concurrency*, initiated around 1980 in the work of R. Milner on CCS (*A Calculus for Communicating Systems*) and C.A.R. Hoare on CSP (*Communicating Sequential Processes*).

1.3. Algebraic semantics
We next discuss two related areas which have been of prime importance in the history of semantics. Firstly, so-called *algebraic semantics* has gained a central status—as third methodology—alongside the methods of denotational and operational semantics. Here, the theory is built on the foundations of universal algebra (such as the notions of initial and final algebra) and equational logic. Algebraic semantics has turned out to be quite valuable, in particular for a logical underpinning of abstract data types (abstract versions of the data structures of programming). In addition, there are deep connections with the theory of rewriting, the theory of concurrency, and with (the vast variety of) specification formalisms.

1.4. Program logics
A second area of research neighbouring on that of semantics is the theory of program correctness, verification and transformation. Historically, this work dates back to Floyd's inductive assertion method (1967), Dijkstra's structured programming and weakest preconditions (early seventies), and Hoare's axiomatic method for simple sequential languages (1969). Though partly more of a logical/syntactic flavour, this area exploits semantic modelling in the investigation of the soundness of formal systems to prove program correctness or to deduce program transformations. Also, the steps prescribed in refining a program from an abstract specification to an executable—and hopefully efficient—implementation require semantic justification. So much for the history of semantics. Some evidence for the world-wide recognition of the developments sketched above may be inferred from the fact that five of the pioneers named above (E.W. Dijkstra, Hoare, R.W. Floyd, Scott, Milner) are recipients of the Turing award of the American Association for Computing Machinery (the Turing award being the Nobel prize of the computer science profession).

2. CURRENT DEVELOPMENTS

All four areas listed above—denotational semantics, operational semantics, algebraic semantics, program logics—are topics of vigorous current activity. During the last two decades a mathematical discipline called 'category theory' has become increasingly important in semantical investigations in computer science. (This also holds for other areas, like specification or type theory.) Category theory provides an elementary foundational language in which the basic concepts of mathematics can be expressed, not in terms of membership like in set theory, but in terms of 'arrows' (or 'morphisms') between 'objects'. The basic idea is to describe mathematical entities not as what they are made of, but as how they behave. For example, set-theoretically a product consists of a set of pairs, whereas category-theoretically a product is an object with two projection arrows which be-

have in a certain 'universal' way. In category theory one does not 'open' the things under investigation, but one describes them 'from the outside'. More strongly: in category theory one gives specifications instead of implementations. This categorical perspective is fruitful in computer science, where many black boxes occur, of which it is not known what precisely is inside.

Interesting applications of semantics are being developed in the design of semantics driven implementations. *Abstract interpretation* is used as a technique to investigate those properties of programs which may be derived from their 'execution' in restricted—mostly finite—models, e.g. to ascertain termination properties (so-called strictness analysis). SOS-style semantic specifications are at present investigated in a language independent fashion, e.g. by analyzing the feasibility of 'automatically' deriving a denotational semantics or a system of (equational) axioms from a given SOS definition.

Semantics is partly driven by its intrinsic foundational questions, and partly by external developments such as technological advances and associated software innovations. The scene of programming language design has expanded considerably in the decade of the 1980s. The group of the traditional imperative languages (ALGOL, PASCAL), together with an occasional functional language (LISP), formed the starting point of a rapidly growing variety of programming paradigms. Languages for concurrency played a central role in the 1980s. Next, the field of functional languages gained in impact, with the language ML as, possibly, the most influential contemporary representative. *Logic programming* (LP) is an area of much current interest, not in the least thanks to the influential Japanese fifth generation project. One relatively fresh protagonist on this scene is the paradigm of object-oriented (OO) languages, a belated offspring of the 1960s language SIMULA. Smalltalk and C^{++} are more contemporary instances of OO languages. One of the difficult issues at present is how to give a complete semantical account of concurrent object-oriented programming. This involves a combination of two levels: There are objects, which are collections of 'small' programs acting on a local state, specified by a class, and there is on a global level a 'pool of objects', in which one can have (concurrent) interaction via sending of messages.

All these language families pose their own problems and often require 'special-purpose' mathematical tools. For example, functional languages rely heavily on the (theory of the) typed lambda calculus, and LP is a programming variant of Horn clause logic, itself a version of resolution logic (which is, in turn, a way of viewing first order predicate logic).

Finally, we here draw attention to the growing interest for the interface between the semantics of programming languages and that of natural languages as studied in computational linguistics. Semantics has grown in the 1980s, both in depth and in width, facing ever new challenges to assimilate the continuous stream of foundational insights and technological advances.

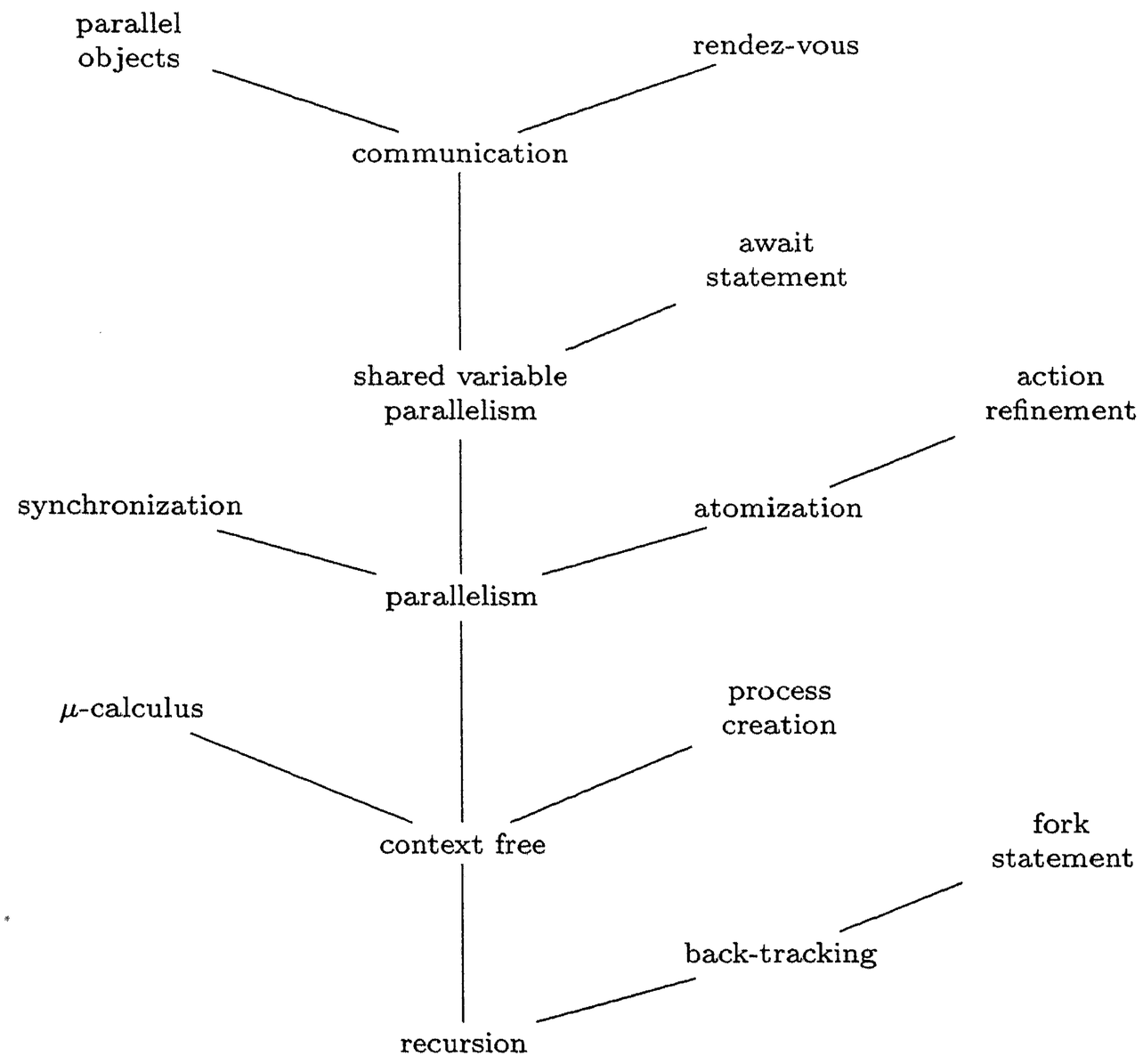

Figure 1. A selection of control flow notions as studied in ref. [4].

3. Contributions of CWI

3.1. Research topics

Parallelism or *concurrency* has been a major focus of semantic research at the CWI in the past decade. An overview is contained in the collection of reprinted papers [2]: [4] is an advanced text/monograph presenting a comprehensive survey of our work since the early eighties (see also figure 1). Characteristic for a good deal of our approach is, on the one hand, the reliance on topological structures in the semantic modelling, and on the other, the large variety of forms of parallelism considered. Not only the more traditional concurrency in an imperative setting, but also parallel versions of LP and OO have been studied in depth. In the period under review a total of nine Ph.D. theses have been completed on the theory of parallel processes in relation to the design and semantics of parallel languages

according to the styles of imperative, dataflow, LP and OO programming, with one further thesis on the proof theory for parallel OO. At present, the main topics in our research are (i) algebraic and coalgebraic approaches to transition systems; (ii) category-theoretic investigations in comparative domain theory; (iii) generalized finiteness conditions in topological models; (iv) predicate—versus state—transformations as theoretical underpinning for a study of refinement; (v) semantics of higher-order and object-oriented processes. In line with the general development mentioned above, category-theoretic tools play an important role in much of this research. For example, it has turned out to be useful to describe transition systems in terms of *coalgebras*, which formally are defined as the dual of algebras. Also observational equivalences, such as the widely used notion of *bisimulation*, can be described in coalgebraic terms. In this manner, a theory of coalgebras is being developed along the lines of (but dual to) universal algebra. This theory seems to have promising applications, for instance, in the semantic description of object-oriented languages.

In comparative domain theory, one of the main issues has been to reconcile the use of metric spaces and partial orders (and their corresponding Hausdorff and non-Hausdorff topologies). Lawvere's view of metric spaces as so-called *enriched categories*, already developed in the early 1970's, offers the right context for this problem. It has led not only to a unification of both theories, but also to new insights concerning, for instance, powerdomains and topology (see also figure 2).

3.2. International and national cooperation

A substantial part of the CWI research in this field over the years has been embedded in international or national collaborative projects. In the first category, we participated in the ESPRIT sponsored project Parallel Architectures and Languages (1984-1989, see [1] for a selection of its results on semantics) and the ESPRIT Basic Research Action Integration—integrating the foundations of functional, logic and object-oriented programming (1989-1992). Currently, our foundational work is supported by the SCIENCE-MASK project—Mathematical Structures in Semantics for Concurrency, and a (national) SION project entitled 'Non-well-founded sets in the semantics of programming languages'. Nationally, we have collaborated for many years with the groups led by G. Rozenberg (Leiden University) and W.P. de Roever (Eindhoven University of Technology), first in the SION-sponsored National Concurrency Project (LPC, 1984-1988), and next in the NFI-project REX—Research and Education in Concurrent Systems, 1988-1993. REX has funded a series of international schools/workshops; [3] contains the proceedings of the 1992 meeting on semantics. Presently, we are involved in a SION funded collaborative project entitled HOOP—Higher Order and Object-Oriented Processes—with as partners the CWI group in

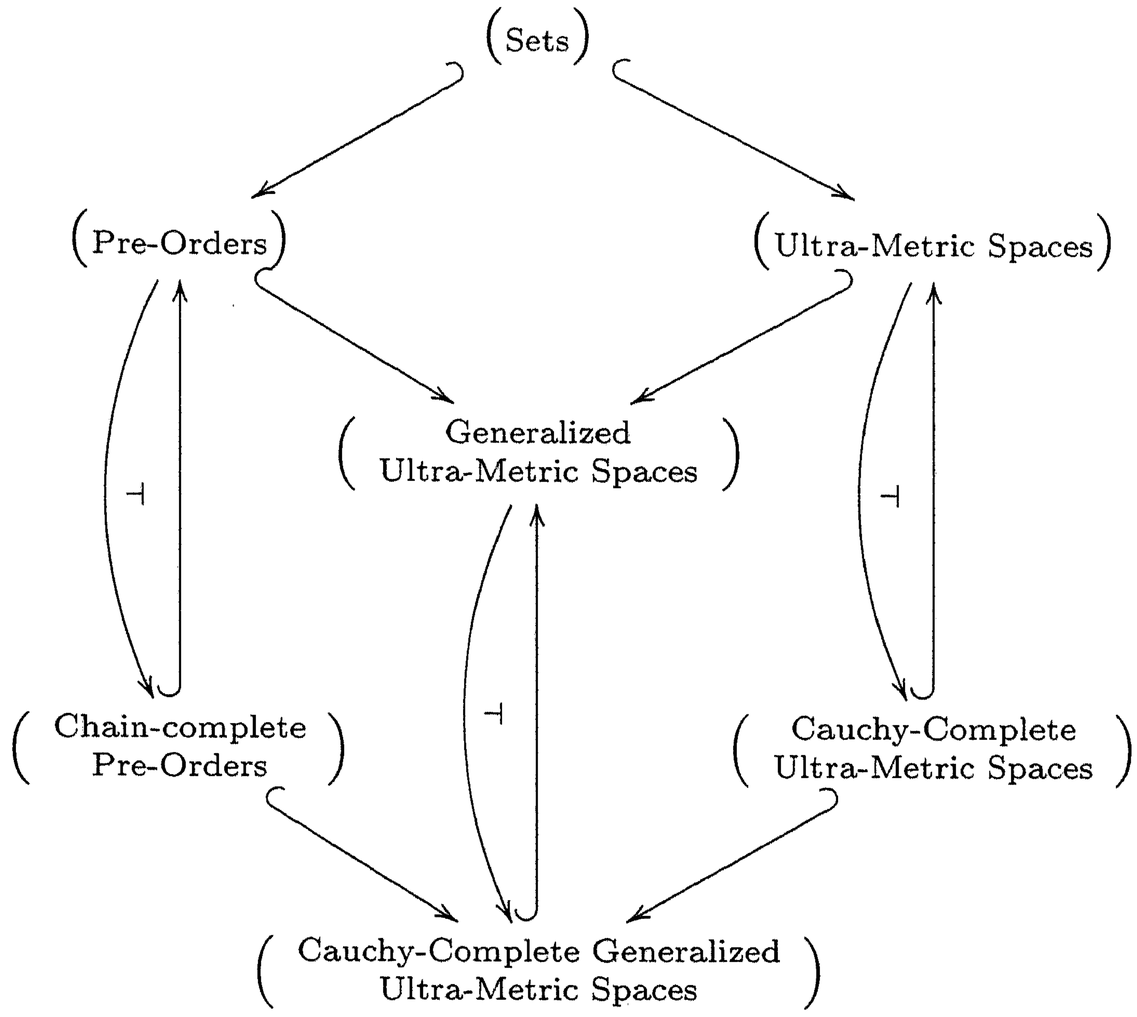

Figure 2. Categories of basic mathematical structures as used in denotational semantics, with some (adjoint) functors between them.

semantics, Leiden University (J. Engelfriet, G. Rozenberg) and Eindhoven University of Technology (J.C.M. Baeten).

REFERENCES

1. J.W. DE BAKKER (ed.). (1989). *Languages for Parallel Architectures—Design, Semantics, Implementation Models*, Wiley.
2. J.W. DE BAKKER, J.J.M.M. RUTTEN (eds.). (1992). *Ten Years of Concurrency Semantics*, selected papers of the Amsterdam Concurrency Group, World Scientific.
3. J.W. DE BAKKER, W.P. DE ROEVER, G. ROZENBERG (eds.). (1993). *Semantics: Foundations and Applications*, Springer Lecture Notes in Computer Science Vol. 666.
4. J.W. DE BAKKER, E.P. DE VINK (1995). *Control Flow Semantics*, MIT Press.