Semantics and Domain Theory – Notes on lecture 1

Herman Geuvers

1 Content of the course

• semantics: "assigning meaning to programs" (more generally: to phrases in a formal language)
• domain theory: the mathematical theory of the sets-with-structure necessary to achieve this

One contrasts
• operational semantics: "evaluation"
• axiomatic semantics: "logic" (assertions about programs)
• denotational semantics: "model theory"

1.1 Operational semantics

• inductive definitions
• grammars
• systems of inference rules defining derivations of judgments
• definition by structural recursion (on syntax, on rules)
• proof by induction
• specification of behaviour by execution/evaluation
• exp ⇒ val, evaluation judgments, for appropriate notions of "exp" (expressions) and "val" (values).
• config ⇒ st, execution judgments, for an abstract machine with configurations "config" and final states "st"
• styles of operational semantics
  – big-step: evaluation/execution all in one go;
  – small-step: consider intermediate configurations of an abstract machine; take transitive closure to reach a final state
• operational semantics gives an intensional theory of behaviour: how, not what.
1.2 Denotational semantics

- abstract mathematical structures
- behaviour described in terms of mathematical functions operating on such abstract values
- **extensional** theory of behaviour: what, not how
- "equal values map to equal results": need notions of equality of abstract programs
- typically recursive (rather than inductive) definitions
- "a general theory of recursive definitions"

Another view on denotational semantics/domain theory is: "a general theory of partiality/partial recursive definitions"

1.3 Programming languages

- The simple imperative language (IMP or WHILE):
  - assignment to variables, conditional execution, unbounded iteration sequential composition (need to put smaller program together to make larger ones)
  - origins: Turing/von Neumann model (1930s), ForTran (Backus, 1957)
  - meanings given by transformations on states (partial functions from variables to values), so: ... higher-order functions ...

- Functional languages, essentially varieties of lambda calculus (PCF):
  - function application and abstraction, possibly built on top of some primitives.
  - origins: Church (untyped 1930s, typed 1940s), LISP (McCarthy, 1957).
  - meanings given by... already in terms of ... higher-order functions (again!).
  - defined (usually) by (possibly many) syntactic categories (phrase types) $T, E, C, ...$ specified by a (context-free) grammar.
  - each phrase type $C$ gives rise to a set of the well-formed phrases of that type. We identify $C$ with this set.

1.4 Domains

A domain is ...

- an appropriate target $\llbracket C \rrbracket$ for interpreting (giving meaning to) elements of $C$;
- it will turn out to be analysed in terms of suitable set-with-structure (an order relation reflecting partial states of knowledge).

1.5 Denotational Semantics

A denotational semantics for the phrase type $C$ is simply a mapping

$$ [-] : C \to \llbracket C \rrbracket.$$ 

So we overload $[-]$ in the usual way; these double braces are usually called *Scott brackets* after Dana Scott, who basically founded the subject; he was originally a student of Tarski) $[-]$ should respect/reflect the appropriate structure on $C$ and $\llbracket C \rrbracket$. 
1.6 Summarising

The principal aim of the subject:

to develop the interplay between these two points-of-view on the meaning/behaviour of programs, that is, to relate

• execution of programs,
  e.g. prog, config ⇒ config

• mathematical description in terms of functions,
  e.g. \([\text{prog}] : [\text{config}] \rightarrow [\text{config}]\)

taking into account

• partiality

• recursion

• appropriate notions of equality, and simulation, between programs

For next time: DENS (Winskel), Ch. 1 For those who haven’t followed Semantiek en Correctheid: read pages 7-14 of Nielsen&Nielsen