Principles of Programming Languages

http://www.di.unipi.it/~andrea/Didattica/PLP-15/

Prof. Andrea Corradini
Department of Computer Science, Pisa

Lesson 14

- Introduction to Denotational semantics

Describing a Programming Language

- Syntax, **semantics** and pragmatics
- Semantics defines the meaning of programs
- Various kinds of semantics
  - Operational
  - Algebraic / Axiomatic
  - **Denotational**
  - Game Theoretical
  - …
- Used for:
  - Unambiguous specification of meaning of programs
    - Correctness of implementations
  - Proving properties or equivalence of programs
  - Evaluating alternative constructs in design phase

Denotational Semantics

- Developed by Dana Scott and Christopher Strachey (~1970)
- Topic of course MOD (Models of Computations) [summer semester]
  - Mathematical foundations (Domain theory)
  - Complete semantics of simple programming languages: IMP (imperative) and HOFL (functional)
- Our presentation is orthogonal
  - Foundations: almost none and informally
  - "Descriptive" use of semantics:
    - to understand programming constructs
    - to compare them across different programming languages
  - We follow
    "*R.D. Tennent: The denotational semantics of programming languages, Communications of the ACM, Volume 19 Issue 8, Aug. 1976*"

Basics and Syntax of LOOP

- The **denotational semantics** of a programming language map programs to mathematical objects (**denotations**) representing the *meaning* of the programs
- This is done **compositionally** on the syntax of the program
- The abstract syntax of a language defines
  - a collection of *syntactic domains*, corresponding to non-terminal symbols
    - Example: **Prog, Exp, Com, Var, …**
  - a collection of *operations* on syntactic domains corresponding to productions

Abstract syntax of the LOOP language [Tennent76]

Exp ::= 0 | **succ** Exp | Var

Com ::= Var := Exp | Com ; Com | **to** Exp **do** Com

Prog ::= **read** Var ; Com ; **write** Exp

- A LOOP program computes a function on natural numbers

Productions as operations

0: → Exp

succ: Exp → Exp

in: Var → Exp

seq : Com x Com → Com

assign: Var x Exp → Com

rep: Exp x Com → Com

prog : Var x Com x Exp → Prog

Denotational Semantics of LOOP

For each *syntactic domain* a corresponding *semantic domain* is defined, and the meaning is given by a *semantic interpretation function*

- ***P*** **: Prog → N → N** (→ associates right, read "Prog → (N → N)")
  - *The meaning of a program is a function from N to N*
- Since Prog ::= **read** Var ; Com ; **write** Exp, to define *P* compositionally we need the semantics of Var, Com and Exp
- For evaluating variables, we introduce the domain of *states*:
  - S = Var → N thus a state *s* ∈ S is a function from Var to N
  - for a state s ∈ S, s{*v*} is the content of variable *v*
  - Def: s[*n*/*v*] is a state s.t. s[*n*/*v*]{*x*} = *n* if *v* = *x*, else s[*n*/*v*]{*x*} = s{*x*}

Note: we use {_} instead of [[_]], the classical notation

Denotational Semantics of LOOP: Expressions and Commands

We define *E* and *C* by induction:

- E: Exp → S → N
  - $E\{0\}\ s = 0$
  - $E\{\textbf{succ}\ e\}\ s = E\{e\}\ s + 1$
  - $E\{v\}\ s = s(v)$

Abstract syntax of the LOOP language [Tennent76]

Exp ::= 0 | **succ** Exp | Var

Com ::= Var := Exp | Com ; Com | **to** Exp **do** Com

Prog ::= **read** Var ; Com ; **write** Exp

Commands change the state:

- C: Com → S → S
  - $C\{v := e\}\ s = s[n/v]$ *where* $n = E\{e\}\ s$
  - $C\{c_1; c_2\}\ s = (\ C\{c_2\} \circ C\{c_1\}\ )s\ \ [= C\{c_2\}\ (C\{c_1\}\ s)\ ]$
  - $C\{\textbf{to}\ e\ \textbf{do}\ c\}\ s = ((C\{c\})^n)\ s$ *where* $n = E\{e\}\ s$
- Note: $f^0(x) = x$, $f^{n+1}(x) = f(f^n(x))$

Denotational Semantics of LOOP: Programs

- *P* : Prog → N → N
  - *The meaning of a program is a function from N to N*
- *P*{**read** *v* ; *c* ; **write** *e*}n = *E*{*e*}s
  where s = *C*{*c*}$s_0[n/v]$
  where $s_0\{w\} = 0$ for each variable *w*

Just a simple example to stress compositionality

- Language LOOP is total
- There is no conditionally controlled iteration
- More complex domains are needed in general