CSCE 330 Fall 2000

Notes on Denotational Semantics

Wednesday 00/9/13

The main reference for these notes is Section 9.4 of: Ghezzi, Carlo and Mehdi Jazayeri. *Programming Language Concepts, 2nd ed.*, New York: John Wiley and Sons, 1987.

We consider a very simple language with only arithmetic and Boolean expressions; moreover, all variables are of integer type, and the control structures are restricted to conditionals and **while** pretest loops.

The *state* of a program P (at a given time) is a triple, $s_P =< mem_P, i_P, o_P >$, where

- $mem_P : Id_P \to Z \cup \{undef\}$
  mem_P is called the *memory function*, and it gives the value of each identifier. $undef$ is a special symbol that is not an integer.

- $i_P \in Z^*$ and $o_P \in Z^*$
  i_P and o_P are called the *input stream* and *output stream*, respectively, and they are strings (or sequences) of integers.

Each language instruction is specified by a state transformation.

- We begin with arithmetic expressions. Let EX be the set of all legal arithmetic expressions.
  $dsem_{EX} : EX \times S \to Z \cup \{error\}$
  $dsem_{EX}(E, s) = error$ if $s =< mem, i, o >$ and $mem(v) = undef$ for some variable v occurring in E; otherwise
  $dsem_{EX}(E, s) = e$ if $s =< mem, i, o >$ and e is the result of evaluating E after replacing each variable v occurring in E with $mem(v)$.

- Let AS be the set of all legal assignment statements.
  $dsem_{AS} : AS \times S \to S \cup \{error\}$
  $dsem_{AS}(x := E, s) = error$ if $dsem_{EX}(E, s) = error$; otherwise
  $dsem_{AS}(x := E, s) = s'$, where $s' =< mem', i', o' >, s =< mem, i, o >$ $, i' = i, o' = o, mem'(y) = mem(y)$ for all $y \neq x, mem'(x) = dsem_{EX}(E, s)$

- Suppose that the input statements in our language are written as $read(x)$, which means that the next input value is assigned to x. Let RD be the set of all legal input statements.
  $dsem_{RD} : RD \times S \to S \cup error$
  $dsem_{RD}(read(x), s) = error$ if $s =< mem, i, o >$ and i is empty; otherwise
  $dsem_{RD}(read(x), s) = s'$, where $s =< mem, i, o >, s' =< mem', i', o' >$ $, o = o', i = Ii'$ for some $I \in Z$ and some $i' \in Z^*, mem(y) = mem'(y)$ for all $y \neq x$, and $mem(x) = I$.

- Let WR be the set of all legal **write** statements.
  $dsem_{WR} : WR \times S \rightarrow S \cup error$
  $dsem_{WR}(write(x), s) = error$ if $s =< mem, i, o >$ and $mem(x) = undef$, otherwise
  $dsem_{WR}(write(x), s) = s'$, where $s =< mem, i, o >, s' =< mem', i', o' >$ $, mem = mem', i = i', o' = oO$, where $O = mem(x)$

- Let SL be the set of all statement lists.
  $dsem_{SL} : SL \times S \rightarrow S \cup error$
  $dsem_{S}L$ is defined recursively as follows:
  $dsem_{SL}(empty\_list, s) = s$
  $dsem_{SL}(H;T, s) = error$ if $dsem(H, s) = error$; otherwise
  $dsem_{SL}(H;T, s) = dsem_{SL}(T, dsem(H, s))$

- Let BOOL be the set of all Boolean (relational) expressions.
  $dsem_{BOOL} : BOOL \times S \rightarrow S \cup undef$
  is defined almost exactly as $dsem_{EX}$.

- Let IF be the set of all $if \ldots then \ldots else \ldots fi$ conditional (or selection) statements. Let B be a Boolean expression. Let $L1$ and $L2$ be statement lists.
  $dsem_{IF}(\text{ifBthenL1elseL2fi}, s) = error$ if $dsem_{BOOL}(B, s) = undef$; otherwise
  $dsem_{IF}(\text{ifBthenL1elseL2fi}, s) = U$, where if $dsem(B, s) = true$, then $U = dsem_{SL}(L1, s)$, else $U = dsem_{SL}(L2, s)$

- Let DO be the set of all syntactically correct $while \ldots do \ldots od$ pretest loop statements.
  $dsem_{DO}(\text{whileBdoLod}, s) = error$ if $dsem_{BOOL}(B, s) = undef$; otherwise
  $dsem_{DO}(\text{whileBdoLod}, s) = s$ if $dsem_{BOOL}(B, s) = false$; otherwise
  $dsem_{DO}(\text{whileBdoLod}, s) = error$ if $dsem_{SL}(L, s) = error$; otherwise
  $dsem_{DO}(\text{whileBdoLod}, s) = dsem_{DO}(\text{whileBdoLod}, dsem_{SL}(L, s))$

- Let $PROG$ be the set of all syntactically correct programs in our language. The *language semantics* is defined by the following function.
  $dsem_{PROG} : PROG \times Z^* \rightarrow Z^* \cup error$
  Let L is the statement list that makes up the program.
  $dsem_{PROG}(L, i) = out(dsem_{SL}(L, init(i)))$, where

  - $init(i) =< mem0, i, o >$, where $mem0(x) = undef$ for all identifiers x and o is the empty string
  - $out(error) = error$
  - $out(< mem, i, o >) = o$