CSCE 330 Fall 2001
NOTES ON DENOTATIONAL SEMANTICS
Monday 01/09/10


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, \( s_P = \langle \text{mem}_P, i_P, o_P \rangle \), where

- \( \text{mem}_P : \text{Id}_P \to Z \cup \{\text{undef}\} \)
  \( \text{mem}_P \) is called the memory function, and it gives the value of each identifier. \( \text{undef} \) 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.
  \( \text{sem}_{EX} : EX \times S \to Z \cup \{\text{error}\} \)
  \( \text{sem}_{EX}(E, s) = \text{error} \) if \( s = \langle \text{mem}, i, o \rangle \) and \( \text{mem}(v) = \text{undef} \) for some variable \( v \) occurring in \( E \); otherwise
  \( \text{sem}_{EX}(E, s) = e \) if \( s = \langle \text{mem}, i, o \rangle \) and \( e \) is the result of evaluating \( E \) after replacing each variable \( v \) occurring in \( E \) with \( \text{mem}(v) \).

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

- Suppose that the input statements in our language are written as \( \text{read}(x) \), which means that the next input value is assigned to \( x \). Let \( RD \) be the set of all legal input statements.
  \( \text{sem}_{RD} : RD \times S \to S \cup \{\text{error}\} \)
  \( \text{sem}_{RD}(\text{read}(x), s) = \text{error} \) if \( s = \langle \text{mem}, i, o \rangle \) and \( i \) is empty; otherwise
  \( \text{sem}_{RD}(\text{read}(x), s) = s', \) where \( s = \langle \text{mem}, i, o \rangle, s' = \langle \text{mem}', i', o' \rangle, o = o, i = I^t \) for some \( I \in Z \) and some \( i' \in Z^* \), \( \text{mem}(y) = \text{mem}'(y) \) for all \( y \neq x \), and \( \text{mem}(x) = I \).
• Let \( WR \) be the set of all legal write statements.
  \[
dsem_{WR} : WR \times S \rightarrow S \cup \{ \text{error} \}
\]
  \[
dsem_{WR}(\text{write}(x), s) = \text{error} \text{ if } s = \langle \text{mem}, i, o \rangle \text{ and mem}(x) = \text{undef},
\]
  otherwise
  \[
dsem_{WR}(\text{write}(x), s) = s', \text{ where } s = \langle \text{mem}, i, o \rangle, s' = \langle \text{mem}', i', o' \rangle,
\]
  \[
  \text{mem} = \text{mem}', i = i', o = o', \text{ where } O = \text{mem}(x)
\]

• Let \( SL \) be the set of all statement lists.
  \[
dsem_{SL} : SL \times S \rightarrow S \cup \{ \text{error} \}
\]
  \[
dsem_{SL}(\text{empty list}, s) = s
\]
  \[
dsem_{SL}(H; T, s) = \text{error} \text{ if } dsem(H, s) = \text{error}; \text{ otherwise}
\]
  \[
dsem_{SL}(H; T, s) = dsem_{SL}(T, dsem(H, s))
\]

• Let \( BOOL \) be the set of all Boolean (relational) expressions.
  \[
dsem_{BOOL} : BOOL \times S \rightarrow \{ \text{true}, \text{false} \} \cup \{ \text{undef} \}
\]
  is defined almost exactly as \( dsem_{EX} \).

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

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

• Let \( PROG \) be the set of all syntactically correct programs in our language.
  The language semantics is defined by the following function.
  \[
dsem_{PROG} : PROG \times Z^* \rightarrow Z^* \cup \{ \text{error} \}
\]
Let \( L \) is the statement list that makes up the program.
  \[
dsem_{PROG}(L, i) = \text{out}(dsem_{SL}(L, \text{init}(i)), \text{ where}
\]
  \[
  - \text{init}(i) = \langle \text{mem0}, i, o \rangle, \text{ where mem0}(x) = \text{undef} \text{ for all identifiers}
\]
  \[
  x \text{ and } o \text{ is the empty string}
\]
  \[
  - \text{out(\text{error}) = error}
\]
  \[
  - \text{out(\langle \text{mem}, i, o \rangle) = o}
\]