CS571

- Notes 06N
- Operational semantics of commands

Semantics of commands

- Abstract syntax
  \[ C ::= \text{nop} \mid I := E \mid C_1;C_2 \mid \text{if } B \text{ then } C_1 \text{ else } C_2 \text{ end} \mid \text{while } B \text{ do } C \text{ end} \]
  - A command, unlike an expression, changes the state
  - General scheme:
    \[ [C, \sigma] \rightarrow \sigma' \]
  - The relation is a different one from the evaluation relation

Semantics of assignment

- Rules
  - Nop: \[ \text{nop}, \sigma \rightarrow \sigma \]
  - Assignment:
    \[ \begin{align*}
    [E, \sigma] & \rightarrow \nu \\
    [I := E, \sigma] & \rightarrow [I \mapsto \nu] \sigma
    \end{align*} \]
  - The store function \( \sigma \) is changed so that the identifier \( I \) is paired with the value of the expression \( E \)
Semantics of the sequence

- Rule:
  \[ [C_1, \sigma] \rightarrow \sigma' \quad [C_2, \sigma'] \rightarrow \sigma'^* \]
  \[ [C_1; C_2, \sigma] \rightarrow \sigma'^* \]
- The sequencing comes from the store in the “middle” - \( \sigma'^* \)

Semantics of the conditional

- Conditional needs two rules, one for each branch:
  \[ [B, \sigma] \rightarrow \text{true} \quad [C_1, \sigma] \rightarrow \sigma' \]
  \[ [B, \sigma] \rightarrow \text{false} \quad [C_2, \sigma] \rightarrow \sigma'^* \]
  \[ \text{if } B \text{ then } C_1 \text{ else } C_2, \sigma \rightarrow \sigma' \quad \text{if } B \text{ then } C_1 \text{ else } C_2, \sigma \rightarrow \sigma'^* \]
- The result will be a different store depending on the branch taken

Problem with the loop

- Problem: loops are iterative, but we have no language (logic, sets, etc.) with that property – we must use recursion
- The basic idea is that the loop continues by executing the body and then repeating the whole loop again until the exit test is false:
  
  \* while B do C end == C;while B do C end
Semantics of the loop

- Rule is:
  
  \[
  \begin{align*}
  [B, \sigma] & \rightarrow \text{true} \quad [C; \text{while } B \text{ do } C \text{ end}, \sigma] \rightarrow \sigma' \\
  [\text{while } B \text{ do } C \text{ end}, \sigma] & \rightarrow \sigma'
  \end{align*}
  \]

- Simpler is (expanding the sequence)
  
  \[
  \begin{align*}
  [B, \sigma] & \rightarrow \text{true} \quad [C, \sigma] \rightarrow \sigma'' \quad [\text{while } B \text{ do } C \text{ end}, \sigma''] \rightarrow \sigma' \\
  [\text{while } B \text{ do } C \text{ end}, \sigma] & \rightarrow \sigma'
  \end{align*}
  \]

Semantics of the loop - termination

- Rule for termination is
  
  \[
  [B, \sigma] \rightarrow \text{false} \quad [\text{while } B \text{ do } C \text{ end}, \sigma] \rightarrow \sigma
  \]

- The store is unchanged when the test is false

An example program

- Program is multiplication using repeated addition:

  \[
  X := 2; \\
  Y := 3; \\
  M := 0; \\
  \text{while } X \text{ greater } 0 \text{ do} \\
  M := M + Y; \\
  X := X - 1 \\
  \text{end}
  \]
**Working the rules**

- Split the program into two: the assignment sequence and the loop itself
- Initial store is \( \sigma_0 \) (could be empty)
- Do each assignment separately:

  \[
  \begin{align*}
  [2, \sigma] \rightarrow 2 & \quad [3, \sigma] \rightarrow 3 \\
  [X := 2, \sigma] \rightarrow [X := 2] \sigma & \quad [Y := 3, \sigma] \rightarrow [Y := 3] \sigma \\
  [0, \sigma] \rightarrow 0 & \quad [M := 0, \sigma] \rightarrow [M := 0] \sigma
  \end{align*}
  \]

**Sequencing**

- Put these three in a sequence with:

  \[
  \sigma_1 = [X \mapsto 2] \sigma \text{ and } \sigma_2 = [Y \mapsto 3] \sigma
  \]

- Then:

  \[
  \begin{align*}
  [2, \sigma] \rightarrow 2 & \quad [3, \sigma] \rightarrow 3 \\
  [X := 2, \sigma] \rightarrow \sigma & \quad [Y := 3, \sigma] \rightarrow \sigma \\
  [0, \sigma] \rightarrow 0 & \quad [M := 0, \sigma] \rightarrow \sigma
  \end{align*}
  \]

**Handling the loop**

- The basic scheme has as many "unfolding" of the loop as necessary, with termination (when the test is false) at the top

  \[
  \begin{align*}
  [0, \sigma] \rightarrow \text{true} & \quad [C, \sigma] \rightarrow \sigma' \\
  [B, \sigma'] \rightarrow \text{true} & \quad [C, \sigma'] \rightarrow \sigma' \quad [\text{while B do C end} \sigma'] \rightarrow \sigma
  \end{align*}
  \]

- Work bottom-up, not top-down
The loop body

- For our program:

\[
\begin{align*}
&M, \sigma^n 
\rightarrow \sigma^n(M) \\
&Y, \sigma^n \rightarrow \sigma^n(Y) \\
&X, \sigma^n \rightarrow \sigma_{\text{init}}(X) \\
&\lfloor 1/\sigma_{\text{init}} \rfloor + 1 \\
&M := M \text{ plus } Y, \sigma^n \rightarrow \sigma_{\text{init}} \\
&\lfloor X := X \text{ minus } 1, \sigma_{\text{init}} \rfloor \rightarrow \sigma^n.
\end{align*}
\]

- This will do for iteration of the loop – the \( m \)th time, where:

\[
\sigma_{\text{init}} := \lfloor M \mapsto \sigma^n(M) \rfloor, \sigma^n(Y) \] and \( \sigma^n(\lfloor X := X \text{ minus } 1, \sigma_{\text{init}} \rfloor) \rightarrow \sigma^n.\]

Putting it all together

- We can work bottom-up
- The stores are:

\[
\begin{align*}
\sigma^n_0 & = \lfloor M \mapsto 0 \rfloor, \lfloor Y \mapsto 3 \rfloor, \lfloor X \mapsto 2 \rfloor \sigma_0 \\
\sigma^n_1 & = \lfloor M \mapsto 3 \rfloor, \lfloor X \mapsto 1 \rfloor \sigma^n_0 \\
\sigma^n_2 & = \lfloor M \mapsto 0 \rfloor, \lfloor X \mapsto 0 \rfloor \sigma^n_0
\end{align*}
\]

- which is, in fact, \( \sigma_{\text{init}} \), because, in the next iteration, \( X \text{ greater } 0 \) is false

Summary

- Operational semantics can handle assignment, evaluation of expressions, and even loops through recursive unfolding
- It can even be used for making inferences about programs in general