CS571

• Notes 06N
• Operational semantics of commands

Semantics of commands

• Abstract syntax
  * C ::= nop | I := E | C1; C2 | if B then C1 else C2 end |
  while B do C 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:
    \[ [E, \sigma] \rightarrow u \]
    \[ [I := E, \sigma] \rightarrow [I \mapsto u] \sigma \]

• 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:
  \[ [C_1, \sigma] \rightarrow \sigma' \]
  \[ [C_2, \sigma'] \rightarrow \sigma'' \]

  \[ [C_1; C_2, \sigma] \rightarrow \sigma'' \]

• The sequencing comes from the store in the “middle” - \( \sigma' \)
Semantics of the conditional

- Conditional needs two rules, one for each branch:
  \[ [B, \sigma] \rightarrow \text{true} \quad [C_1, \sigma] \rightarrow \sigma' \]
  \[ [B, \sigma] \rightarrow \text{false} \quad [C_2, \sigma] \rightarrow \sigma^* \]

- 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:
  \[ [B, \sigma] \rightarrow \text{true} \quad [C ; \text{while B do C end}, \sigma] \rightarrow \sigma' \]
  \[ [\text{while B do C end}, \sigma] \rightarrow \sigma' \]

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

Semantics of the loop - termination

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

- The store is unchanged when the test is false
An example program

- Program is multiplication using repeated addition:

\[
\begin{align*}
X &:= 2; \\
Y &:= 3; \\
M &:= 0; \\
\text{while } X \text{ greater 0 do} & \\
\quad M &:= M \text{ plus } Y; \\
\quad X &:= X \text{ minus 1} \\
\text{end}
\end{align*}
\]

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:

\[
\begin{align*}
[2, \sigma_0] &\rightarrow 2 \\
[X:=2, \sigma_0] &\rightarrow [X \rightarrow 2] \sigma_0 \\
[3, \sigma_1] &\rightarrow 3 \\
[Y:=3, \sigma_1] &\rightarrow [Y \rightarrow 3] \sigma_1 \\
[0, \sigma_2] &\rightarrow 0 \\
[M:=0, \sigma_2] &\rightarrow [M \rightarrow 0] \sigma_2
\end{align*}
\]

Sequencing

- Put these three in a sequence with:

\[
\begin{align*}
\sigma_1 &= [X \mapsto 2] \sigma_0 \text{ and } \sigma_2 &= [Y \mapsto 3] \sigma_1 \\
\text{Then:}
\end{align*}
\]

\[
\begin{align*}
[2, \sigma_0] &\rightarrow 2 \\
[X:=2, \sigma_0] &\rightarrow \sigma_1 \\
[3, \sigma_1] &\rightarrow 3 \\
[Y:=3, \sigma_1] &\rightarrow \sigma_2 \\
[0, \sigma_2] &\rightarrow 0 \\
[M:=0, \sigma_2] &\rightarrow \sigma_m
\end{align*}
\]

- Where \( \sigma_m = [M \mapsto 0] \sigma_1 = [M \mapsto 0] [Y \mapsto 3] [X \mapsto 2] \sigma_0 \)

Handling the loop

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

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

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

- For our program:

\[
\begin{align*}
[M, \sigma^m] &\rightarrow [\sigma^m(M)] [Y, \sigma^m] \rightarrow [\sigma^m(Y)] \\
[M := M + Y, \sigma^m] &\rightarrow [\sigma^m(X)] [X, \sigma^m] \rightarrow [\sigma^m(X)] [1, \sigma^m] \rightarrow [1] \\
[M := M + Y, \sigma^m] &\rightarrow [\sigma^m(X)] [X := X - 1, \sigma^m] \rightarrow [\sigma^m(X)] \\
[M := M + Y, X := X - 1, \sigma^m] &\rightarrow [\sigma^m(X)]
\end{align*}
\]

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

\[
\sigma_{\text{loop}} = [M \mapsto \sigma^m(M) + \sigma^m(Y)] \sigma^m \text{ and } \sigma_{\text{loop}}^{-1} = [X \mapsto \sigma_{\text{loop}}(X) + 1] \sigma_{\text{loop}}
\]

Putting it all together

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

\[
\begin{align*}
\sigma_0^i &= [M \mapsto 0] [Y \mapsto 3] [X \mapsto 2] \sigma_0 \\
\sigma_1^i &= [M \mapsto 3] [X \mapsto 1] \sigma_1 \\
\sigma_2^i &= [M \mapsto 6] [X \mapsto 0] \sigma_2
\end{align*}
\]

- which is, in fact, \( \sigma_{\text{out}} \), because, in the next iteration, \( X > 0 \) 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