Lecture 29

Topics

1. PS4 is out, I will add one more problem on IMP design – adding functions with state. We might also consider environments that give names to constants. There is an extra credit problem to explain Blum’s Size Theorem. There will be a PS5 as well.

2. We will study the simple imperative programming language IMP from Winskel’s book *The Formal Semantics of Programming Languages*, MIT Press, 1993. Chapter 2. PS4 includes material from Chapter 10.

3. Functional vs. Imperative languages, “imperative” is for commands, “do this to the state”. We need *assignable* identifiers also called reference variables in the ML family.

4. Winskel gives a syntax, the easy bit, and a *large step* operational semantics (à la Khan) and a *small step* operational semantics (à la Plotkin).

5. We will add *functional procedures* to IMP (to get “FIMP”?). That will be part of PS4. This will be a collective “design” problem.

---

**Defining IMP** (with some CS6110 flourishes)

Abstract syntax

Arithmetic expressions (p.12)

\[ aexp ::= n \mid x \mid \text{add}(a_1; a_2) \mid \text{sub}(a_1; a_2) \mid \text{mult}(a_1; a_2) \]

Syntactic conventions

\[ n, m \text{ range over } \mathbb{N} \]

\[ x, y \text{ range over } \text{Loc} \] (locations, variables)

\[ a \text{ range over } Aexp \] (arithmetic expressions)

\[ b \text{ range over } Bexp \] (Boolean expressions)

\[ c \text{ range over } Cmd \] (Commands – Com)

Some “Winskel ways” requiring parenthesis to make the expressions unique – a bit strange to me. Here are his syntactic classes: \( Aexp, Bexp, Cmd \) (he uses Com for Commands).

I show Algol68 syntax:

\[ \text{if } bexp \text{ then } c_0 \text{ else } c_1 \text{ fi, while } bexp \text{ do } c \text{ od, and until } bexp \text{ do } c \text{ od, etc.} \]
Structured operational semantics (big step, à la Khan, also called natural semantics).

Evaluation requires states. Winskel writes $\sigma$, we write $s$. $s : \text{Loc} \rightarrow \mathbb{N}$ (recall Loc are locations in the memory or state).

The rules have the format

\[
< \text{exp}, s > \rightarrow \text{value} \\
< \text{cmd}, s > \rightarrow \text{state}
\]

On page 16 he suggests a refinement or top down style of reading the rules. Wirth suggested the same and called them refinement style rules. In lecture we will write them this way.

Evaluation of sums p.14

Rule instances p.15 Note: $s_0(\text{Init}) = 0$

Derivation tree p.15

\[
\vdash < \text{Init} + 5 + (7 + 9), s_0 > \rightarrow 5 + 16
\]

\[
\vdash < \text{Init} + 5, s_0 > \rightarrow 5 + 5
\]

\[
\vdash < \text{Init}, s_0 > \rightarrow s_0(\text{Init}) = 0
\]

\[
\vdash < 5, s_0 > \rightarrow 5
\]

\[
\vdash < 7, s_0 > \rightarrow 7
\]

\[
\vdash < 9, s_0 > \rightarrow 9
\]

Evaluation of Commands 2.4, p.19

Using $< C, s > \rightarrow s'$

For example: $< X := 5, s > \rightarrow s[5/X]$

List of Commands

- skip $< \text{skip}, s > \rightarrow s$
- sequencing $< C_0; C_1, s > \rightarrow$
- conditionals $< \text{if } b \text{ then } C_0 \text{ else } C_1, s > \rightarrow$
- while loops (page 20)

\[
\vdash < \text{while } b \text{ do } C \text{ od }, s > \rightarrow s'
\]

\[
\vdash < b, s > \rightarrow \text{true}
\]

\[
\vdash < C, s > \rightarrow s''
\]

\[
\vdash < \text{while } b \text{ do } C, s'' > \rightarrow s'
\]

\[
\vdash < \text{while } b \text{ do } C \text{ od }, s > \rightarrow s
\]

\[
\vdash < b, s > \rightarrow \text{false}
\]