Last Time

- Calculations: also/finally
- \{\text{trans}\}-rules
- Code generation

Content

- Intro & motivation, getting started
  - Rough timeline
    - Intro & motivation, getting started
  - Foundations & Principles
    - Lambda Calculus, natural deduction
      - Higher Order Logic
      - Term rewriting
    - Higher Order Logic
    - Higher Order Logic
  - Proof & Specification Techniques
    - Isar
      - Inductively defined sets, rule induction
    - Datatypes, recursion, induction
    - Calculational reasoning, mathematics style proofs
    - Calculational reasoning, mathematics style proofs
    - Calculational reasoning, mathematics style proofs
    - Hoare logic, proofs about programs
    - Hoare logic, proofs about programs

"a1 out; a1 due; a2 out; a2 due; session break; a3 out; a3 due"
IMP - a small Imperative Language

Commands:

<table>
<thead>
<tr>
<th>datatype com</th>
<th>SKIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assign loc aexp</td>
<td>( x := e )</td>
</tr>
<tr>
<td>Semi com com</td>
<td>( c_1 ; c_2 )</td>
</tr>
<tr>
<td>Cond bexp com com</td>
<td>( \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2 )</td>
</tr>
<tr>
<td>While bexp com</td>
<td>( \text{WHILE } b \text{ DO } c )</td>
</tr>
</tbody>
</table>

types loc = string

types state = loc \( \rightarrow \) nat

Types:

types aexp = state \( \rightarrow \) nat

Types:

types bexp = state \( \rightarrow \) bool

Example Program

Usual syntax:

\[
\begin{align*}
B & := 1; \\
\text{WHILE } A \neq 0 \text{ DO} \\
B & := B \times A; \\
A & := A - 1 \\
\text{OD}
\end{align*}
\]

Expressions are functions from state to bool or nat:

\[
\begin{align*}
B & := (\lambda \sigma. 1); \\
\text{WHILE } (\lambda \sigma. A \neq 0) \text{ DO} \\
B & := (\lambda \sigma. B \times \sigma A); \\
A & := (\lambda \sigma. A - 1) \\
\text{OD}
\end{align*}
\]

What does it do?

So far we have defined:

\( \to \) Syntax of commands and expressions

\( \to \) State of programs (function from variables to values)

Now we need: the meaning (semantics) of programs

How to define execution of a program?

\( \to \) A wide field of its own

\( \to \) Some choices:

- Operational (inductive relations, big step, small step)
- Denotational (programs as functions on states, state transformers)
- Axiomatic (pre-post conditions, Hoare logic)

Structural Operational Semantics

\[
\begin{align*}
\langle \text{SKIP} , \sigma \rangle & \rightarrow \sigma \\
\langle x := e , \sigma \rangle & \rightarrow \sigma[x := e] \\
\langle c_1 ; c_2 , \sigma \rangle & \rightarrow \sigma' \\
\langle \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2 , \sigma \rangle & \rightarrow \sigma' \\
\langle \lambda \sigma. e \sigma , \sigma \rangle & \rightarrow e' \\
\langle \text{WHILE } b \text{ DO } c \rangle & \rightarrow \sigma'
\end{align*}
\]
Structural Operational Semantics

\[ \delta \sigma = \text{False} \]

\[
\begin{align*}
\langle \text{WHILE } b \text{ DO } c \text{ END}, \sigma \rangle & \rightarrow \sigma \\
\langle c, \sigma \rangle & \rightarrow \sigma'
\end{align*}
\]

\[ \delta \sigma = \text{True} \]

\[
\begin{align*}
\langle \text{WHILE } b \text{ DO } c \text{ END}, \sigma' \rangle & \rightarrow \sigma'' \\
\langle \text{WHILE } b \text{ DO } c \text{ END}, \sigma \rangle & \rightarrow \sigma''
\end{align*}
\]

Proofs about Programs

Now we know:

\[ \rightarrow \text{ What programs are: Syntax} \]
\[ \rightarrow \text{ On what they work: State} \]
\[ \rightarrow \text{ How they work: Semantics} \]

So we can prove properties about programs

Example:

Show that example program from slide 6 implements the factorial.

\[
\text{lemma } \langle \text{factorial}, \sigma \rangle \rightarrow \sigma' \equiv \sigma' B = \text{fac } (\sigma A)
\]

(\text{where } \text{fac } 0 = 0, \text{ fac } (\text{Suc } n) = (\text{Suc } n) \ast \text{ fac } n)
Too tedious

Induction needed for each loop

Is there something easier?

Floyd/Hoare

Idea: describe meaning of program by pre/post conditions

Examples:

\[
\begin{align*}
\{ \text{True} \} & \quad x := 2 \quad \{ x = 2 \} \\
\{ y = 2 \} & \quad x := 21 \times y \quad \{ x = 42 \} \\
\{ x = n \} & \quad \text{IF } y < 0 \text{ THEN } x := x + y \text{ ELSE } x := x - y \quad \{ x = n - |y| \} \\
\{ A = n \} & \quad \text{factorial} \quad \{ B = \text{fac } n \}
\end{align*}
\]

Proofs: have rules that directly work on such triples

Slide 13

Meaning of a Hoare-Triple

\[ \{ P \} \ c \ { Q \} \]

What are the assertions \( P \) and \( Q \)?

- Here: again functions from state to bool (shallow embedding of assertions)
- Other choice: syntax and semantics for assertions (deep embedding)

What does \( \{ P \} \ c \ { Q \} \) mean?

Partial Correctness:

\[ \models \{ P \} \ c \ { Q \} \quad \equiv \quad (\forall \sigma \sigma'. \ P \sigma \land \langle c, \sigma \rangle \rightarrow \sigma' \implies Q \sigma') \]

Total Correctness:

\[ \models \{ P \} \ c \ { Q \} \quad \equiv \quad (\forall \sigma. \ P \sigma \implies \exists \sigma'. \langle c, \sigma \rangle \rightarrow \sigma' \land Q \sigma') \]

This lecture: partial correctness only (easier)

Slide 15

Hoare Rules

\[
\begin{align*}
\{ P \} & \quad \text{SKIP} \quad \{ P \} \\
\{ P[x := e] \} & \quad x := e \quad \{ P \} \\
\{ P \} & \quad c_1 \quad \{ R \} \\
\{ P \} & \quad c_2 \quad \{ Q \} \\
\{ P \} & \quad \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2 \quad \{ Q \} \\
\{ P \} & \quad \text{WHILE } b \text{ DO } c \text{ OD} \quad \{ Q \}
\end{align*}
\]

Slide 16
Hoare Rules

\[ \vdash \{ P \} \text{SKIP} \{ P \} \]

\[ \vdash \{ \lambda \sigma. P \sigma(x := e \sigma) \} x := e \{ P \} \]

\[ \vdash \{ P \} c_1 \{ R \} \vdash \{ R \} c_2 \{ Q \} \]

\[ \vdash \{ \lambda \sigma. P \sigma \land b \sigma \} c_1 \{ R \} \vdash \{ \lambda \sigma. P \sigma \land \neg b \sigma \} c_2 \{ Q \} \]

\[ \vdash \{ \lambda \sigma. P \sigma \rightarrow Q \sigma \} \implies \{ P \ \text{WHILE} \ b \ \text{DO} \ Q \} \]

\[ \vdash \{ \lambda \sigma. P \sigma \rightarrow P' \sigma \} \implies \{ P' \} c \{ Q' \} \]

\[ \vdash \{ P' \} c \{ Q' \} \implies \{ Q \} \]

Are the Rules Correct?

Soundness: \( \vdash \{ P \} c \{ Q \} \implies \{ P \} c \{ Q \} \)

Proof: by rule induction on \( \vdash \{ P \} c \{ Q \} \)

Demo: Hoare Logic in Isabelle