Slide 1

NICTA Advanced Course
Theorem Proving
Principles, Techniques, Applications

\{P\} \ldots \{Q\}

Slide 2

CONTENT

\rightarrow Intro & motivation, getting started with Isabelle
\rightarrow Foundations & Principles
  \bullet Lambda Calculus
  \bullet Higher Order Logic, natural deduction
  \bullet Term rewriting
\rightarrow Proof & Specification Techniques
  \bullet Inductively defined sets, rule induction
  \bullet Datatypes, recursion, induction
  \bullet More recursion, Calculational reasoning
  \bullet Hoare logic, proofs about programs
  \bullet Locales, Presentation

Slide 3

LAST TIME

\rightarrow Recdef
\rightarrow More induction
\rightarrow Well founded orders
\rightarrow Well founded recursion
\rightarrow Calculations: also/finally
\rightarrow \{trans\}-rules

Slide 4

A CRASH COURSE IN SEMANTICS

LAST TIME

IMP - A SMALL IMPERATIVE LANGUAGE
IMP - a small imperative language

Commands:

\[ \text{datatype} \ \text{com} = \text{SKIP} \]
\[ \text{Assign loc aexp} (\_ := \_) \]
\[ \text{Semi com com} (\_ 
\]
\[ \text{Cond bexp com com} (\text{IF } \_ \text{ THEN } \_ \text{ ELSE } \_) \]
\[ \text{While bexp com} (\text{WHILE } \_ \text{ DO } \_ \text{ OD}) \]

Slide 5

| types loc | = string |
| types state | = loc \Rightarrow \text{nat} |
| types aexp | = \text{state} \Rightarrow \text{nat} |
| types bexp | = \text{state} \Rightarrow \text{bool} |

Example Program

Usual syntax:

\[ B := 1; \]
\[ \text{WHILE } A \neq 0 \text{ DO} \]
\[ B := B \ast A; \]
\[ A := A - 1 \]
\[ \text{OD} \]

Slide 6

Expressions are functions from state to bool or nat:

\[ B := (\lambda \sigma. \ 1); \]
\[ \text{WHILE } (\lambda \sigma. \ A \neq 0) \text{ DO} \]
\[ B := (\lambda \sigma. \ B \ast A); \]
\[ A := (\lambda \sigma. \ A - 1) \]
\[ \text{OD} \]

What does it do?

So far we have defined:

\[ \rightarrow \text{ Syntax of commands and expressions} \]
\[ \rightarrow \text{ State of programs (function from variables to values)} \]

Now we need: the meaning (semantics) of programs

How to define execution of a program?

\[ \rightarrow \text{ A wide field of its own (visit a semantics course!)} \]
\[ \rightarrow \text{ Some choices:} \]
- Operational (inductive relations, big step, small step)
- Denotational (programs as functions on states, state transformers)
- Axiomatic (pre-/post conditions, Hoare logic)

Structural Operational Semantics

\[ (\text{SKIP}, \sigma) \longrightarrow \sigma \]

\[ e \sigma = v \]

\[ (x := e, \sigma) \longrightarrow \sigma[x \leftarrow v] \]

\[ (c_1, \sigma) \longrightarrow \sigma' \]
\[ (c_2, \sigma') \longrightarrow \sigma'' \]
\[ (c_1; c_2, \sigma) \longrightarrow \sigma'' \]

\[ b \sigma = \text{True} \]
\[ (\text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2, \sigma) \longrightarrow \sigma' \]

\[ b \sigma = \text{False} \]
\[ (\text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2, \sigma) \longrightarrow \sigma' \]
**Slide 9**

**Structural Operational Semantics**

\[
\begin{align*}
& b \sigma = \text{False} \\
& (\text{WHILE } b \text{ DO } c \text{ OD}, \sigma) \rightarrow \sigma
\end{align*}
\]

**Slide 11**

**Proofs about Programs**

Now we know:

- What programs are: Syntax
- On what they work: State
- How they work: Semantics

So we can prove properties about programs

**Example:**

Show that example program from slide 6 implements the factorial.

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

(where \( \text{fac } 0 = 0 \), \( \text{fac } (\text{Suc } n) = (\text{Suc } n) \times \text{fac } n \))

---

**Slide 10**

**Demo: The Definitions in Isabelle**

---

**Slide 12**

**Demo: Example Proof**

---
**Slide 13**

Too tedious

Induction needed for each loop

Is there something easier?

**Slide 14**

Floyd/Hoare

Idea: describe meaning of program by pre/post conditions

Examples:

- \{ \text{True} \} \quad x := 2 \quad \{ x = 2 \}
- \{ y = 2 \} \quad x := 21 \ast 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 \}

Proofs: have rules that directly work on such triples

**Slide 15**

Meaning of a Hoare-Triple

\{ P \} \quad c \quad \{ 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:

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

Total Correctness:

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

This lecture: partial correctness only (easier)

**Slide 16**

Hoare Rules

\[
\begin{array}{c}
\{ P \} \quad \text{SKIP} \quad \{ P \} \\
\{ P \} \quad x := e \quad \{ P \} \\
\{ P \} \quad c_1 \quad \{ Q \}
\end{array}
\]  

\[
\begin{array}{c}
\{ P \} \quad c_1 \quad \{ R \} \quad c_2 \quad \{ Q \} \\
\{ P \} \quad c_1 ; c_2 \quad \{ Q \}
\end{array}
\]  

\[
\begin{array}{c}
\{ P \} \quad \text{WHILE} \ \text{b} \ \text{DO} \ \text{c} \ \text{OD} \quad \{ Q \} \\
\{ P \} \quad \text{IF} \ \text{b} \ \text{THEN} \ c_1 \ \text{ELSE} \ c_2 \quad \{ Q \}
\end{array}
\]  

\[
\begin{array}{c}
\{ P \} \quad \{ P' \} \quad c \quad \{ Q' \} \quad \{ Q \}
\end{array}
\]
Slide 17

**Hoare Rules**

\[ \vdash \{ P \} \text{SKIP} \{ P \} \]
\[ \vdash \{ \lambda \sigma. P (\sigma (x := e \sigma)) \} \quad x := e \quad \{ P \} \]
\[ \vdash \{ P \} \quad c_1 \{ R \} \quad \vdash \{ R \} \quad c_2 \{ Q \} \]
\[ \vdash \{ P \} \quad c_1 ; c_2 \{ Q \} \]

\[ \vdash \{ \lambda \sigma. P \sigma \wedge b \sigma \} \quad c_1 \{ R \} \quad \vdash \{ \lambda \sigma. P \sigma \wedge \neg b \sigma \} \quad c_2 \{ Q \} \]
\[ \vdash \{ P \} \quad \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2 \{ Q \} \]

\[ \vdash \{ \lambda \sigma. P \sigma \wedge b \sigma \} \quad c \quad \wedge \sigma. P \sigma \wedge \neg b \sigma \Rightarrow Q \sigma \]
\[ \vdash \{ P \} \quad \text{WHILE } b \text{ DO } c \text{ OD } \{ Q \} \]

\[ \wedge \sigma. P \sigma \Rightarrow P' \sigma \quad \vdash \{ P' \} \quad c \{ Q' \} \quad \wedge \sigma. Q' \sigma \Rightarrow Q \sigma \]
\[ \vdash \{ P \} \quad c \quad \{ Q \} \]

Slide 18

**ARE THE RULES CORRECT?**

**Soundness:** \[ \vdash \{ P \} \quad c \{ Q \} \quad \Rightarrow \vdash \{ P \} \quad c \{ Q \} \]

**Proof:** by rule induction on \[ \vdash \{ P \} \quad c \{ Q \} \]

Slide 19

**Nicer, but still kind of tedious**

Hoare rule application seems boring & mechanical.

**Automation?**

**Problem:** While – need creativity to find right (invariant) \( P \)

**Solution:**

\[ \Rightarrow \text{ annotate program with invariants} \]
\[ \Rightarrow \text{ then, Hoare rules can be applied automatically} \]

**Example:**

\[ \{ M = 0 \wedge N = 0 \} \]
\[ \text{WHILE } M \neq a \text{ INV } \{ N = M \ast b \} \text{ DO } N := N + b ; M := M + 1 \text{ OD} \]
\[ \{ N = a \ast b \} \]

Slide 19

**Solution:**

\[ \Rightarrow \text{ annotate program with invariants} \]
\[ \Rightarrow \text{ then, Hoare rules can be applied automatically} \]

**Example:**

\[ \{ M = 0 \wedge N = 0 \} \]
\[ \text{WHILE } M \neq a \text{ INV } \{ N = M \ast b \} \text{ DO } N := N + b ; M := M + 1 \text{ OD} \]
\[ \{ N = a \ast b \} \]

Slide 20

**Weakest Preconditions**

\[ \text{pre } c \mid Q = \text{ weakest } P \text{ such that } \{ P \} = \{ Q \} \]

With annotated invariants, easy to get:

pre \( \text{SKIP } Q \)
\[ = Q \]

pre \( (x := a) \mid Q \)
\[ = \lambda \sigma. Q(\sigma(x := a \sigma)) \]

pre \( (c_1 ; c_2) \mid Q \)
\[ = \text{pre } c_1 (\text{pre } c_2 Q) \]

pre \( \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2 \mid Q \)
\[ = \lambda \sigma. (b \rightarrow \text{pre } c_1 Q \sigma) \wedge \neg b \rightarrow \text{pre } c_2 Q \sigma \]

pre \( \text{WHILE } b \text{ INV } I \text{ DO } c \text{ OD} \mid Q \)
\[ = I \]
VERIFICATION CONDITIONS

{\text{pre} c \ Q} \ c \ \{Q\} \text{ only true under certain conditions}

These are called verification conditions \( \text{vc} c \ Q \):

- \( \text{vc SKIP} \ Q = \text{True} \)
- \( \text{vc} (x := a) \ Q = \text{True} \)
- \( \text{vc} (c_1 ; c_2) \ Q = \text{vc} c_2 \ Q \land (\text{vc} c_1 \ (\text{pre} c_2 \ Q)) \)
- \( \text{vc} (\text{IF} b \ \text{THEN} c_1 \ \text{ELSE} c_2) \ Q = \text{vc} c_1 \ Q \land \text{vc} c_2 \ Q \)
- \( \text{vc} (\text{WHILE} b \ \text{INV} I \ \text{DO} c \ \text{OD}) \ Q = \text{vc} c \ I \land \text{vc} \ I \land \{\text{pre} c \ I \} \land (\forall \sigma. I \sigma \land b \sigma \rightarrow \text{pre} c \ I \sigma) \land \text{vc} C I \land \{P\} \ c \ \{Q\} \)

SYNTAX TRICKS

- \( x := \lambda \sigma. 1 \) instead of \( x := 1 \) sucks
- \( \{\lambda \sigma. \sigma \ x = n\} \) instead of \( \{x = n\} \) sucks as well

Problem: program variables are functions, not values
Solution: distinguish program variables syntactically

Choices:
- declare program variables with each Hoare triple
  - nice, usual syntax
  - works well if you state full program and only use vcg
- separate program variables from Hoare triple (use extensible records), indicate usage as function syntactically
  - more syntactic overhead
  - program pieces compose nicely

RECORDS IN ISABELLE

Records are tuples with named components

Example:

\begin{verbatim}
record A =  a :: nat \
b :: int
\end{verbatim}

- Selectors: \( a :: A \Rightarrow \text{nat}, \ b :: A \Rightarrow \text{int}, \ a \ r = \text{Suc} 0 \)
- Constructors: \( \{a = \text{Suc} 0, \ b = -1\} \)
- Update: \( r(\{a := \text{Suc} 0\}) \)

Records are extensible:

\begin{verbatim}
record B = A + \
c :: nat list
\end{verbatim}

\( \{a = \text{Suc} 0, \ b = -1, \ c = [0,0]\} \)
More

Available now in Isabelle:
- procedures
- with blocks and local variables
- and (mutual) recursion
- exceptions
- arrays
- pointers

We're working at:
- nondeterminism
- probability
- object orientation

Exercises

Write a program in IMP that calculates quotient and reminder of $x \in \mathbb{N}$ and $y \in \mathbb{N}$
- Find the right invariant for its while loop.
- Show its correctness in Isabelle:

\[
\{ \text{True} \} \quad \text{program} \quad \{ Q \wedge y + R = x \wedge R < y \}
\]

Write an IMP program that sorts arrays (lists) by insertion sort.
- Formulate and show its correctness in Isabelle.

We have seen today ...

- Syntax and semantics of IMP
- Hoare logic rules
- Soundness of Hoare logic
- Verification conditions
- Example program proofs