{P} \ldots \{Q\}

---

**Content**

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

---

**Finding Theorems**

Command `find_theorems` (C-c C-f) finds combinations of:

- pattern: "\(+\)+" 
- tns of simp rules: simp: "\(\times\) \(\times\)" 
- intro/elim/dest on current goal 
- lemma name: name: assoc 
- exclusions thereof: -name: "HOL."

Example:

`find_theorems dest -"hd" name: "List."`

finds all theorems in the current context that

- match the goal as dest rule,
- do not contain the constant "hd"
- are in the List theory (name starts with "List")

---

**Last Time**

- Calculations: also/finally 
- (trans)-rules 
- Code generation
Isar: define and defines

Can define local constant in Isar proof context:

\[
\text{proof}
\]

\[
\text{define } f \equiv \text{big term}
\]

\[
\text{have } g = f x \ldots
\]

like definition, not automatically unfolded (f_def)
different to let ?! = "big term"

Also available in lemma statement:

\[
\text{lemma } \ldots
\]

\[
\text{fixes } \ldots
\]

\[
\text{assumes } \ldots
\]

\[
\text{defines } \ldots
\]

\[
\text{shows } \ldots
\]

IMP - a small Imperative Language

Commands:

\[
\text{datatype com} = \text{SKIP} \\
| \text{Assign loc aexp} | \text{Semi com com} | \text{Cond bexp com com} | \text{While bexp com}
\]

\[
| (:=) | (\_ := \_ ) | (\_ \_ ) | (\text{IF } \_ \_ \_ \text{THEN } \_ \_ \_ \text{ELSE } \_ \_ ) | (\text{WHILE } \_ \_ \_ \text{DO } \_ \_ \_ \text{OD})
\]

\[
\text{types loc} = \text{string} \\
\text{types state} = \text{loc } \Rightarrow \text{nat}
\]

\[
\text{types aexp} = \text{state } \Rightarrow \text{nat} \\
\text{types bexp} = \text{state } \Rightarrow \text{bool}
\]

Example Program

Usual syntax:

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

Expressions are functions from state to bool or nat:

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

- Syntax of commands and expressions
- State of programs (function from variables to values)

Now we need: the meaning (semantics) of programs

How to define execution of a program?

→ A wide field of its own
→ 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**

\[
\begin{align*}
\langle \text{SKIP}, \sigma \rangle & \rightarrow \sigma \\
\langle x := e, \sigma \rangle & \rightarrow \sigma[x \mapsto v] \\
\langle c_1, \sigma' \rangle & \rightarrow \sigma'' \\
\langle c_1, c_2, \sigma \rangle & \rightarrow \sigma'' \\
\langle \text{IF } b \text{ } \text{THEN } c_1 \text{ } \text{ELSE } c_2, \sigma \rangle & \rightarrow \sigma' \\
\langle \text{WHILE } b \text{ } \text{DO } c, \sigma \rangle & \rightarrow \sigma''
\end{align*}
\]

**DEMO: THE DEFINITIONS IN ISABELLE**
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 8 implements the factorial.

**Lemma** \((\text{factorial}, \sigma) \rightarrow \sigma' \rightarrow \sigma' \mathord{B} = \text{fac} (\sigma \mathord{A})\)

(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:
- \{\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

---

**DEMO: EXAMPLE PROOF**
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:
\[|\{P\} c \{Q\}\| = (\forall \sigma. P \sigma \land \langle c, \sigma\rangle \rightarrow \sigma' \implies Q \sigma')\]

Total Correctness:
\[|\{P\} c \{Q\}\| = (\forall \sigma. P \sigma \implies \exists \sigma'. \langle c, \sigma\rangle \rightarrow \sigma' \land Q \sigma')\]

This lecture: partial correctness only (easier)

Hoare Rules

\[\{P\} \text{SKIP} \quad \{P; e\} \quad x := e \quad \{P\}\]

\[\{P\} c_1 \{R\} \quad \{R; c_2\} \quad \{P\} c_1; c_2 \quad \{Q\}\]

\[\{P \land b\} c_1 \{Q\} \quad \{P \land \neg b\} c_2 \quad \{Q\}\]

\[\{P\} \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2 \quad \{P\} \quad \{Q\}\]

\[\{P\} \text{WHILE } b \text{ DO } c \text{ OD} \quad \{P\} \quad \{Q\}\]

Are the Rules Correct?

Soundness: \[|\{P\} c \{Q\}\| \implies |\{P\} c \{Q\}\|\]

Proof: by rule induction on \[|\{P\} c \{Q\}\|

Demo: Hoare Logic in Isabelle