Content

→ Intro & motivation, getting started

→ Foundations & Principles
  • Lambda Calculus, natural deduction  [1,2]
  • Higher Order Logic  [3^a]
  • Term rewriting  [4]

→ Proof & Specification Techniques
  • Inductively defined sets, rule induction  [5]
  • Datatypes, recursion, induction  [6, 7]
  • Hoare logic, proofs about programs, C verification  [8^b,9]
  • (mid-semester break)
  • Writing Automated Proof Methods  [10]
  • Isar, codegen, typeclasses, locales  [11^c,12]

^a a1 due; ^b a2 due; ^c a3 due
Last Time

- Conditional term rewriting
- Case Splitting with the simplifier
- Congruence rules
- AC Rules
- Knuth-Bendix Completion (Waldmeister)
- Orthogonal Rewrite Systems
Specification Techniques

Sets
Sets in Isabelle

Type ’a set: sets over type ’a

→ \{\}, \{e_1, \ldots, e_n\}, \{x. \ P \ x\}
→ e \in A, \ A \subseteq B
→ A \cup B, \ A \cap B, \ A - B, \ -A
→ \bigcup x \in A. B x, \ \bigcap x \in A. B x, \ \bigcap A, \ \bigcup A
→ \{i..j\}
→ \text{insert :: } \alpha \Rightarrow \alpha \ \text{set} \Rightarrow \alpha \ \text{set}
→ f' A \equiv \{y. \exists x \in A. \ y = f \ x\}
→ \ldots
Proofs about Sets

Natural deduction proofs:

- equalityI: \([A \subseteq B; B \subseteq A] \implies A = B\)
- subsetI: \((\forall x. x \in A \implies x \in B) \implies A \subseteq B\)
- . . . (see Tutorial)
Bounded Quantifiers

\[ \forall x \in A. \ P x \equiv \forall x. \ x \in A \rightarrow P x \]
\[ \exists x \in A. \ P x \equiv \exists x. \ x \in A \land P x \]
\[ \text{balll: } (\bigwedge x. \ x \in A \implies P x) \implies \forall x \in A. \ P x \]
\[ \text{bspec: } [\forall x \in A. \ P x; x \in A] \implies P x \]
\[ \text{bexl: } [P x; x \in A] \implies \exists x \in A. \ P x \]
\[ \text{bexe: } [\exists x \in A. \ P x; \bigwedge x. [x \in A; P x] \implies Q] \implies Q \]
Demo

Sets
The Three Basic Ways of Introducing Theorems

→ Axioms:
  Example: \textit{axiomatization where refl: } "t = t"
  Do not use. Evil. Can make your logic inconsistent.

→ Definitions:
  Example: \textit{definition inj where } "inj
  \( f \equiv \forall x \ y. \ f \ x = f \ y \implies x = y" 
  Introduces a new lemma called inj_def.

→ Proofs:
  Example: \textit{lemma } "inj \ (\lambda x. \ x + 1)"
  The harder, but safe choice.
The Three Basic Ways of Introducing Types

- **typedef**: by name only
  
  Example: **typedef** names
  Introduces new type *names* without any further assumptions

- **type_synonym**: by abbreviation
  
  Example: **type_synonym** \( \alpha \) rel = "\( \alpha \Rightarrow \alpha \Rightarrow bool \)"
  Introduces abbreviation *rel* for existing type \( \alpha \Rightarrow \alpha \Rightarrow bool \)
  Type abbreviations are immediately expanded internally

- **typedef**: by definition as a set
  
  Example: **typedef** new_type = "\{some set\}" <proof>
  Introduces a new type as a subset of an existing type.
  The proof shows that the set on the rhs is non-empty.
How typedef works

new type

existing type

Rep

Abs
How typedef works

new type

Rep

Abs

existing type
Example: Pairs

\[(\alpha, \beta) \text{ Prod}\]

1. Pick existing type: \(\alpha \Rightarrow \beta \Rightarrow \text{bool}\)
2. Identify subset:
\[(\alpha, \beta) \text{ Prod} = \{ f . \exists a b. f = \lambda(x :: \alpha)(y :: \beta). x = a \land y = b\}\]
3. We get from Isabelle:
   - functions Abs_Prod, Rep_Prod
   - both injective
   - \(\text{Abs}_{\text{Prod}}(\text{Rep}_{\text{Prod}} x) = x\)
4. We now can:
   - define constants Pair, fst, snd in terms of Abs_Prod and Rep_Prod
   - derive all characteristic theorems
   - forget about Rep/Abs, use characteristic theorems instead
Demo

Introducing new Types
Inductive Definitions
Example

\[
\begin{align*}
\langle \text{skip}, \sigma \rangle & \rightarrow \sigma \\
\langle x := e, \sigma \rangle & \rightarrow \sigma[x \mapsto v]
\end{align*}
\]

\[
\begin{align*}
\langle c_1, \sigma \rangle & \rightarrow \sigma' \\
\langle c_2, \sigma' \rangle & \rightarrow \sigma'' \\
\langle c_1; c_2, \sigma \rangle & \rightarrow \sigma''
\end{align*}
\]

\[
\begin{align*}
\llbracket b \rrbracket \sigma = \text{False} \\
\langle \text{while } b \text{ do } c, \sigma \rangle & \rightarrow \sigma
\end{align*}
\]

\[
\begin{align*}
\llbracket b \rrbracket \sigma = \text{True} \\
\langle c, \sigma \rangle & \rightarrow \sigma' \\
\langle \text{while } b \text{ do } c, \sigma' \rangle & \rightarrow \sigma'' \\
\langle \text{while } b \text{ do } c, \sigma \rangle & \rightarrow \sigma''
\end{align*}
\]
What does this mean?

→ \langle c, \sigma \rangle \longrightarrow \sigma'  \quad \text{fancy syntax for a relation} \quad (c, \sigma, \sigma') \in E
→ \text{relations are sets: } E :: (\text{com } \times \text{ state } \times \text{ state}) \ \text{set}
→ \text{the rules define a set inductively}

But which set?
Simpler Example

0 \in \mathbb{N} \quad n \in \mathbb{N} \quad n + 1 \in \mathbb{N}

\rightarrow N \text{ is the set of natural numbers } \mathbb{N}
\rightarrow \text{But why not the set of real numbers? } 0 \in \mathbb{R}, \ n \in \mathbb{R} \Rightarrow n + 1 \in \mathbb{R}
\rightarrow \mathbb{N} \text{ is the smallest set that is consistent with the rules.}

Why the smallest set?

\rightarrow \text{Objective: no junk. Only what must be in } X \text{ shall be in } X.
\rightarrow \text{Gives rise to a nice proof principle (rule induction)}
\rightarrow \text{Alternative (greatest set) occasionally also useful: coinduction}
Rule Induction

\[ 0 \in \mathbb{N} \quad \frac{n \in \mathbb{N}}{n + 1 \in \mathbb{N}} \]

induces induction principle

\[ [P \ 0; \land n. \ P \ n \rightarrow P \ (n + 1)] \rightarrow \forall x \in \mathbb{N}. \ P \ x \]
Demo

Inductive Definitions
We have learned today ...

- Sets
- Type Definitions
- Inductive Definitions