Slide 1

NICTA Advanced Course

Theorem Proving
Principles, Techniques, Applications

Slide 2

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
  - Calculational reasoning, mathematics style proofs
  - Hoare logic, proofs about programs

Slide 3

Last Time on HOL

- Defining HOL
- Higher Order Abstract Syntax
- Deriving proof rules
- More automation

Slide 4

The Three Basic Ways of Introducing Theorems

- **Axioms**:

  Expample: **axioms** refl: "$t = t$"

  Do not use. Evil. Can make your logic inconsistent.

- **Definitions:**

  Example: **defs** inj_def: "inj $f \equiv \forall x\ y.\ f\ x = f\ y \longrightarrow x = y$"

- **Proofs:**

  Example: **lemma** "inj $(\lambda x.\ x + 1)$"

  The harder, but safe choice.

Slide 5

The Three Basic Ways of Introducing Types

➜ **typedecl**: by name only

Example: **typedecl** names
Introduces new type *names* without any further assumptions

➜ **types**: by abbreviation

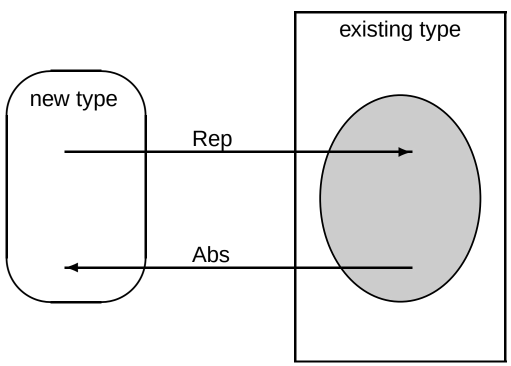
Example: **types** α rel = "$\alpha \Rightarrow \alpha \Rightarrow bool$"
Introduces abbreviation *rel* for existing type $\alpha \Rightarrow \alpha \Rightarrow bool$
Type abbreviations are immediatly expanded internally

➜ **typedef**: by definiton as a set

Example: **typdef** 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 in non-empty.

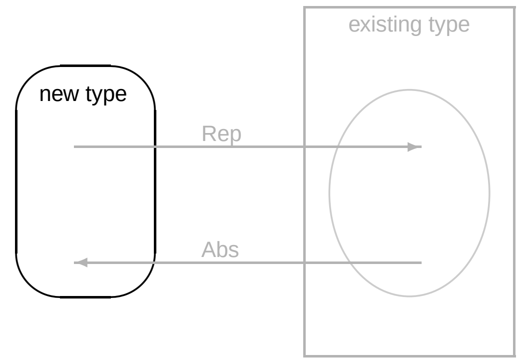
Slide 6

How typedef Works



Slide 7

How typedef Works



Slide 8

Example: Pairs

(α, β) Prod

① Pick existing type: $\alpha \Rightarrow \beta \Rightarrow$ bool

② Identify subset:
(α, β) Prod $= \{f.\ \exists a\ b.\ f = \lambda(x :: \alpha)\ (y :: \beta).\ x = a \wedge y = b\}$

③ We get from Isabelle:

- functions Abs_Prod, Rep_Prod
- both injective
- Abs_Prod (Rep_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

Slide 9

DEMO: INTRODUCTING NEW TYPES

Slide 10

TERM REWRITING

Slide 11

THE PROBLEM

Given a set of equations

$$l_1 = r_1$$
$$l_2 = r_2$$
$$\vdots$$
$$l_n = r_n$$

does equation $l = r$ hold?

Applications in:

➜ **Mathematics** (algebra, group theory, etc)

➜ **Functional Programming** (model of execution)

➜ **Theorem Proving** (dealing with equations, simplifying statements)

Slide 12

TERM REWRITING: THE IDEA

use equations as reduction rules

$$l_1 \longrightarrow r_1$$
$$l_2 \longrightarrow r_2$$
$$\vdots$$
$$l_n \longrightarrow r_n$$

decide $l = r$ by deciding $l \stackrel{*}{\longleftrightarrow} r$

Arrow Cheat Sheet

Slide 13

$\xrightarrow{0}$	$=$	$\{(x,y) \mid x = y\}$	identity
$\xrightarrow{n+1}$	$=$	$\xrightarrow{n} \circ \longrightarrow$	n+1 fold composition
$\xrightarrow{+}$	$=$	$\bigcup_{i>0} \xrightarrow{i}$	transitive closure
$\xrightarrow{*}$	$=$	$\xrightarrow{+} \cup \xrightarrow{0}$	reflexive transitive closure
$\xrightarrow{=}$	$=$	$\longrightarrow \cup \xrightarrow{0}$	reflexive closure
$\xrightarrow{-1}$	$=$	$\{(y,x) \mid x \longrightarrow y\}$	inverse
$\longleftarrow$	$=$	$\xrightarrow{-1}$	inverse
$\longleftrightarrow$	$=$	$\longleftarrow \cup \longrightarrow$	symmetric closure
$\xleftrightarrow{+}$	$=$	$\bigcup_{i>0} \xleftrightarrow{i}$	transitive symmetric closure
$\xleftrightarrow{*}$	$=$	$\xleftrightarrow{+} \cup \xleftrightarrow{0}$	reflexive transitive symmetric closure

How to Decide $l \xleftrightarrow{*} r$

Slide 14

Same idea as for β: look for n such that $l \xrightarrow{*} n$ and $r \xrightarrow{*} n$

Does this always work?

If $l \xrightarrow{*} n$ and $r \xrightarrow{*} n$ then $l \xleftrightarrow{*} r$. Ok.
If $l \xleftrightarrow{*} r$, will there always be a suitable n? **No!**

Example:

Rules: $f\ x \longrightarrow a, \quad g\ x \longrightarrow b, \quad f\ (g\ x) \longrightarrow b$
$f\ x \xleftrightarrow{*} g\ x$ because $f\ x \longrightarrow a \longleftarrow f\ (g\ x) \longrightarrow b \longleftarrow g\ x$
But: $f\ x \longrightarrow a$ and $g\ x \longrightarrow b$ and a, b in normal form

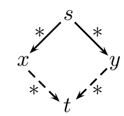
Works only for systems with **Church-Rosser** property:

$$l \xleftrightarrow{*} r \Longrightarrow \exists n.\ l \xrightarrow{*} n \wedge r \xrightarrow{*} n$$

Fact: $\longrightarrow$ is Church-Rosser iff it is confluent.

Confluence

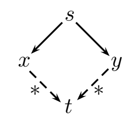
Slide 15



Problem:
is a given set of reduction rules confluent?

undecidable

Local Confluence



Fact: local confluence and termination $\Longrightarrow$ confluence

Termination

Slide 16

$\longrightarrow$ is **terminating** if there are no infinite reduction chains

$\longrightarrow$ is **normalizing** if each element has a normal form

$\longrightarrow$ is **convergent** if it is terminating and confluent

Example:

$\longrightarrow_\beta$ in λ is not terminating, but confluent
$\longrightarrow_\beta$ in $\lambda^\rightarrow$ is terminating and confluent, i.e. convergent

Problem: is a given set of reduction rules terminating?

undecidable

WHEN IS $\longrightarrow$ TERMINATING?

Basic Idea: when the r_i are in some way simpler then the l_i

More formally: $\longrightarrow$ is terminating when
there is a well founded order $<$ in which $r_i < l_i$ for all rules.
(well founded = no infinite decreasing chains $a_1 > a_2 > \ldots$)

Slide 17

Example: $f\ (g\ x) \longrightarrow g\ x$, $g\ (f\ x) \longrightarrow f\ x$

This system always terminates. Reduction order:

$s <_r t$ iff $size(s) < size(t)$ with
$size(s)$ = numer of function symbols in s

① $g\ x <_r f\ (g\ x)$ and $f\ x <_r g\ (f\ x)$
② $<_r$ is well founded, because $<$ is well founded on $\mathbb{N}$

TERM REWRITING IN ISABELLE

Term rewriting engine in Isabelle is called **Simplifier**

apply simp

Slide 18

- ➜ uses simplification rules
- ➜ (almost) blindly from left to right
- ➜ until no rule is applicable.

termination: not guaranteed
(may loop)

confluence: not guaranteed
(result may depend on which rule is used first)

CONTROL

Slide 19

- ➜ Equations turned into simplifaction rules with **[simp]** attribute
- ➜ Adding/deleting equations locally:
  apply (simp add: <rules>) and **apply** (simp del: <rules>)
- ➜ Using only the specified set of equations:
  apply (simp only: <rules>)

Slide 20

DEMO

Slide 21

ISAR

A LANGUAGE FOR STRUCTURED PROOFS

Slide 22

ISAR

apply scripts	**What about..**
➜ unreadable	➜ Elegance?
➜ hard to maintain	➜ Explaining deeper insights?
➜ do not scale	➜ Large developments?
No structure.	**Isar!**

Slide 23

A TYPICAL ISAR PROOF

proof
  assume $formula_0$
  have $formula_1$ **by** simp
  $\vdots$
  have $formula_n$ **by** blast
  show $formula_{n+1}$ **by** ...
qed

proves $formula_0 \Longrightarrow formula_{n+1}$

(analogous to **assumes**/**shows** in lemma statements)

Slide 24

ISAR CORE SYNTAX

proof = **proof** [method] statement* **qed**
  | **by** method

method = (simp ...) | (blast ...) | (rule ...) | ...

statement = **fix** variables ($\bigwedge$)
  | **assume** proposition ($\Longrightarrow$)
  | [**from** name$^+$] (**have** | **show**) proposition proof
  | **next** (separates subgoals)

proposition = [name:] formula

Proof and QED

proof [method] statement* **qed**

lemma "$\llbracket A; B \rrbracket \Longrightarrow A \wedge B$"
proof (rule conjI)
  assume A: "A"
  from A **show** "A" **by** assumption
next
  assume B: "B"
  from B **show** "B" **by** assumption
qed

Slide 25

- **proof** (<method>) applies method to the stated goal
- **proof** applies a single rule that fits
- **proof -** does nothing to the goal

How do I know what to Assume and Show?

Look at the proof state!

lemma "$\llbracket A; B \rrbracket \Longrightarrow A \wedge B$"
proof (rule conjI)

Slide 26

- **proof** (rule conjI) changes proof state to
  1. $\llbracket A; B \rrbracket \Longrightarrow A$
  2. $\llbracket A; B \rrbracket \Longrightarrow B$
- so we need 2 shows: **show** "A" and **show** "B"
- We are allowed to **assume** A,
  because A is in the assumptions of the proof state.

The Three Modes of Isar

- **[prove]**:
  goal has been stated, proof needs to follow.
- **[state]**:
  proof block has openend or subgoal has been proved,
  new *from* statement, goal statement or assumptions can follow.
- **[chain]**:
  from statement has been made, goal statement needs to follow.

Slide 27

lemma "$\llbracket A; B \rrbracket \Longrightarrow A \wedge B$" **[prove]**
proof (rule conjI) **[state]**
  assume A: "A" **[state]**
  from A **[chain]** **show** "A" **[prove]** **by** assumption **[state]**
next **[state]** **...**

Have

Can be used to make intermediate steps.

Example:

Slide 28

lemma "$(x :: \mathsf{nat}) + 1 = 1 + x$"
proof -
  have A: "$x + 1 = \mathsf{Suc}\ x$" **by** simp
  have B: "$1 + x = \mathsf{Suc}\ x$" **by** simp
  show "$x + 1 = 1 + x$" **by** (simp only: A B)
qed

Slide 29

DEMO: ISAR PROOFS

WE HAVE LEARNED TODAY ...

Slide 30

- ➜ Introducing new Types
- ➜ Equations and Term Rewriting
- ➜ Confluence and Termination of reduction systems
- ➜ Term Rewriting in Isabelle
- ➜ First structured proofs (Isar)

EXERCISES

Slide 31

- ➜ use **typedef** to define a new type v with exactly one element.
- ➜ define a constant u of type v
- ➜ show that every element of v is equal to u
- ➜ design a set of rules that turns formulae with $\wedge, \vee, \longrightarrow, \neg$ into disjunctive normal form (= disjunction of conjunctions with negation only directly on variables)
- ➜ prove those rules in Isabelle
- ➜ use **simp only** with these rules on $(\neg B \longrightarrow C) \longrightarrow A \longrightarrow B$