Automatic Deduction — LVA 703522
Introduction to Isabelle

Clemens Ballarin

Universität Innsbruck

HOL

Contents

- Intro & motivation, getting started with Isabelle
- Foundations & Principles
  - Lambda Calculus
  - Types & Classes
  - Natural Deduction
  - Term rewriting
- Proof & Specification Techniques
  - Isar: mathematics style proofs
  - Inductively defined sets, rule induction
  - Datatypes, structural induction
  - Recursive functions & code generation

Types

Types in Isabelle

$$\tau \quad ::= \quad B \mid {'\nu} \mid {'?\nu} \mid (\tau, \ldots, \tau)\ K \mid \tau :: C$$

B	base types
ν	type variables
K	type constructors
C	sorts

- **Base types:** `bool`, `int`, ...
- **Type variables:** `'a`, `'a1`, `'name`, `'?a`, ...
- **Type constructors:** `int list`, `'a list`, `'a` $\Rightarrow$ `'b`, ...
- **Sorts:** `'a :: order`, `'a :: {plus, order}`, ...
  Restrict a type to one or more classes.

Terms in Isabelle

$$t \quad ::= \quad v \;\mid\; ?v \;\mid\; c \;\mid\; (t\ t) \;\mid\; (\lambda x.\, t) \;\mid\; (t :: \tau)$$

v, x	variable names
c	constants

- **Variables & constants:** `a`, `a1`, `name`, ...
- **Type constraints:** `f :: 'a ⇒ 'b`
  Restrict a term to a type.
- **Schematic variables:** variables that can be instantiated.

Type Classes

Similar to Haskell's type classes, but with semantic properties

> **class** *order* =
> **fixes** *less_eq* (**infix** "$\leq$" *50*)
> **and** *less* (**infix** "$<$" *50*)
> **assumes** *order_refl:* "$x \leq x$"
> **and** *order_trans:* "$[\![x \leq y; y \leq z]\!] \Longrightarrow x \leq z$"
> **and** ...

Theorems can be proved in the abstract

> **lemma** (**in** *order*) *order_less_trans:*
> "$\bigwedge x.\ [\![x < y; y < z]\!] \Longrightarrow x < z$"

Here x, y and z have type $'a :: order$.

Type Classes

Can be used for subtyping

> **class** *linorder* $=$ *order* $+$
>   **assumes** *linorder_linear:* "$x \leq y \vee y \leq x$"

Can be instantiated

> **instance** *nat* :: "$\{$*order, linorder*$\}$" **by** $\ldots$

Schematic Variables

Two operational roles of variables.

- In lemmas they must be **instantiated** when applied.

$$[\![X; Y]\!] \Longrightarrow X \land Y$$

- During proofs they must not be instantiated.

  lemma "$x + 0 = 0 + x$"

  Convention: lemma must be true for all x.

Isabelle has **free** (x), **bound** (x), and **schematic** (?x) variables.

Only schematic variables can be instantiated.

Free converted into schematic after proof is finished.

Higher-Order Unification

Unification:
Find substitution σ on variables for terms s, t such that
$\sigma(s) = \sigma(t)$

In Isabelle:
Find substitution σ on schematic variables such that
$\sigma(s) =_{\alpha\beta\eta} \sigma(t)$

Examples:

$$
\begin{array}{llll}
?X \wedge ?Y & =_{\alpha\beta\eta} & x \wedge x & [?X \mapsto x, ?Y \mapsto x] \\
?P\ x & =_{\alpha\beta\eta} & x \wedge x & [?P \mapsto \lambda x.\ x \wedge x] \\
P\ (?f\ x) & =_{\alpha\beta\eta} & ?Y\ x & [?f \mapsto \lambda x.\ x, ?Y \mapsto P]
\end{array}
$$

Higher-Order: schematic variables can be functions.

Higher-Order Unification

- Unification modulo $\alpha\beta$ is semi-decidable
- Unification modulo $\alpha\beta\eta$ is undecidable
- Higher-Order Unification has possibly infinitely many most general solutions

But:

- Most cases are well-behaved
- Important fragments (like Higher-Order Patterns) are decidable

Higher-Order Patterns

Higher-Order Pattern:

- is a term in β-normal form where
- each occurrence of a schematic variable is of the from $?f\ t_1\ \ldots\ t_n$
- and the $t_1\ \ldots\ t_n$ are η-convertible into n distinct bound variables

Preview: Proofs in Isabelle

Proofs in Isabelle

General schema

lemma name: "⟨*goal*⟩"
  apply ⟨*method*⟩
  apply ⟨*method*⟩
  ...
  done

- Sequential application of methods until all subgoals are solved.

The Proof State

1. $\bigwedge x_1 \ldots x_p . [\![A_1; \ldots ; A_n]\!] \Longrightarrow B$

2. $\bigwedge y_1 \ldots y_q . [\![C_1; \ldots ; C_m]\!] \Longrightarrow D$

$x_1 \ldots x_p$	Parameters
$A_1 \ldots A_n$	Local assumptions
B	Current (sub)goal

Isabelle Theories

Syntax

theory $\langle$*name*$\rangle$
imports $\langle$*import*$_1\rangle \ldots \langle$*import*$_n\rangle$
begin

(declarations, definitions, theorems, proofs, ...)*

end

- $\langle$*name*$\rangle$: name of theory. Must live in file $\langle$*name*$\rangle$`.thy`
- $\langle$*import*$_i\rangle$: name of imported theory. Import transitive.

Unless you need something special:

theory $\langle$*name*$\rangle$
imports Main
begin

Natural Deduction

Natural Deduction Rules

$$\frac{A \qquad B}{A \wedge B}\ \text{conjI} \qquad \frac{A \wedge B \qquad [\![A; B]\!] \Longrightarrow C}{C}\ \text{conjE}$$

$$\frac{A}{A \vee B} \quad \frac{B}{A \vee B}\ \text{disjI1/2} \qquad \frac{A \vee B \qquad A \Longrightarrow C \qquad B \Longrightarrow C}{C}\ \text{disjE}$$

$$\frac{A \Longrightarrow B}{A \longrightarrow B}\ \text{disjE} \qquad \frac{A \longrightarrow B \qquad A \qquad B \Longrightarrow C}{C}\ \text{impE}$$

For each connective ($\wedge$, $\vee$, etc):
introduction and elemination rules

Proof by Assumption

apply assumption

proves

1. $[\![B_1; \ldots; B_m]\!] \Longrightarrow C$

by unifying C with one of the B_i

There may be more than one matching B_i
and multiple unifiers.

Backtracking!

Explicit backtracking command: **back**

Intro Rules

Intro rules decompose formulae to the right of $\Longrightarrow$.

apply (rule ⟨*intro-rule*⟩)

Intro rule $[\![A_1; \ldots; A_n]\!] \Longrightarrow A$ means

- To prove A it suffices to show $A_1 \ldots A_n$

Applying rule $[\![A_1; \ldots; A_n]\!] \Longrightarrow A$ to subgoal C:

- unify A and C
- replace C with n new subgoals $A_1 \ldots A_n$

Elim Rules

Elim rules decompose formulae on the left of $\Longrightarrow$.

apply (erule <elim-rule>)

Elim rule $[\![A_1; A_2; \ldots; A_n]\!] \Longrightarrow A$ means

- If I know A_1 and want to prove A it suffices to show $A_2 \ldots A_n$

Applying rule $[\![A_1; \ldots; A_n]\!] \Longrightarrow A$ to subgoal C:
Like rule but also

- unifies first premise of rule with an assumption
- eliminates that assumption

Demo: Propositional Reasoning

Iff, Negation, True and False

$$\frac{A \Longrightarrow B \qquad B \Longrightarrow A}{A = B}\ \textsf{iffI} \qquad \frac{A = B \qquad [\![A \longrightarrow B; B \longrightarrow A]\!] \Longrightarrow C}{C}\ \textsf{iffE}$$

$$\frac{A = B}{A \Longrightarrow B}\ \textsf{iffD1} \qquad \frac{A = B}{B \Longrightarrow A}\ \textsf{iffD2}$$

$$\frac{A \Longrightarrow \mathit{False}}{\neg A}\ \textsf{notI} \qquad \frac{\neg A \qquad A}{P}\ \textsf{notE}$$

$$\frac{}{\mathit{True}}\ \textsf{TrueI} \qquad \frac{\mathit{False}}{P}\ \textsf{FalseE}$$

Equality

$$\frac{}{t = t}\ \text{refl} \qquad \frac{s = t}{t = s}\ \text{sym} \qquad \frac{r = s \qquad s = t}{r = t}\ \text{trans}$$

$$\frac{s = t \qquad P\ s}{P\ t}\ \text{subst}$$

Rarely needed explicitly — used implicitly by term rewriting.

Classical

$$\frac{}{P = \mathit{True} \ \lor \ P = \mathit{False}} \ \text{True\_or\_False}$$

$$\frac{}{P \lor \neg P} \ \text{excluded\_middle}$$

$$\frac{\neg A \Longrightarrow \mathit{False}}{A} \ \text{ccontr} \qquad \frac{\neg A \Longrightarrow A}{A} \ \text{classical}$$

- excluded_middle, ccontr and classical not derivable from the other rules.
- If we include True_or_False, they are derivable.

They make the logic classical, non-constructive.

Cases

$$\frac{}{P \vee \neg P}\ \text{excluded\_middle}$$

is a case distinction on type $bool$.

Isabelle can do case distinctions on arbitrary terms:

apply (case_tac ⟨*term*⟩)

Safe and Not so Safe

Safe rules preserve provability:

conjI, impI, notI, iffI, refl, ccontr, classical, conjE, disjE

$$\frac{A \qquad B}{A \wedge B} \text{ conjI}$$

Unsafe rules can turn a provable goal into an unprovable one:

disjI1, disjI2, impE, iffD1, iffD2, notE

$$\frac{A}{A \vee B} \text{ disjI1}$$

Apply safe rules before unsafe ones.

Demo: More Rules

Quantifiers

Scope

- Scope of parameters: whole subgoal
- Scope of $\forall, \exists, \ldots$: ends with meta-level connective: $\Longrightarrow$, $\equiv$ or ;.

Example:

$$\bigwedge x\ y.\ [\![\ \forall y.\ P\ y \longrightarrow Q\ z\ y;\ \ Q\ x\ y\ ]\!] \implies \exists x.\ Q\ x\ y$$

means

$$\bigwedge x\ y.\ [\![\ (\forall y_1.\ P\ y_1 \longrightarrow Q\ z\ y_1);\ \ Q\ x\ y\ ]\!] \implies (\exists x_1.\ Q\ x_1\ y)$$

Natural Deduction for Quantifiers

$$\frac{\bigwedge x.\ P\ x}{\forall x.\ P\ x}\ \textsf{allI} \qquad \frac{\forall x.\ P\ x \qquad P\ ?x \Longrightarrow R}{R}\ \textsf{allE}$$

$$\frac{P\ ?x}{\exists x.\ P\ x}\ \textsf{exI} \qquad \frac{\exists x.\ P\ x \qquad \bigwedge x.\ P\ x \Longrightarrow R}{R}\ \textsf{exE}$$

- allI and exE introduce new parameters ($\bigwedge x$).
- allE and exI introduce new unknowns ($?x$).

Instantiating Rules

apply (rule_tac x = "⟨*term*⟩" in ⟨*rule*⟩)

Like rule, but ?x in ⟨*rule*⟩ is instantiated by ⟨*term*⟩ before application.

Similar: erule_tac

- x is in ⟨*rule*⟩, not in goal.
- ⟨*term*⟩ may contain parameters from the goal and those introduced in Isar texts (later).

Two Successful Proofs

1. $\forall x.\ \exists y.\ x = y$

apply (rule allI)

1. $\bigwedge x.\ \exists y.\ x = y$

Best practice

apply (rule_tac x = "x" in exI)

1. $\bigwedge x.\ x = x$

apply (rule refl)

simpler & clearer

Exploration

apply (rule exI)

1. $\bigwedge x.\ x = ?y\ x$

apply (rule refl)

$?y \mapsto \lambda u.u$

shorter & trickier

Two Unsuccessful Proofs

1. $\exists y.\ \forall x.\ x = y$

apply (rule_tac x = ??? in exI)

apply (rule exI)
1. $\forall x.\ x = ?y$

apply (rule allI)
1. $\bigwedge x.\ x = ?y$

apply (rule refl)
$?y \mapsto x$ yields $\bigwedge x'.x' = x$

Principle

$?f\ x_1 \ldots x_n$ can only be replaced by term t
if params$(t) \subseteq x_1, \ldots, x_n$.

Safe and Unsafe Rules

Safe	allI, exE
Unsafe	allE, exI

Create parameters first, unknowns later

Demo: Quantifier Proofs

Parameter Names

Parameter names are chosen by Isabelle

1. $\forall\ x.\ \exists y.\ x = y$

apply (rule allI)
1. $\bigwedge x.\ \exists y.\ x = y$

apply (rule_tac x = "x" in exI)

Brittle!

Renaming Parameters

1. $\forall x.\ \exists y.\ x = y$

apply (rule allI)
1. $\bigwedge x.\ \exists y.\ x = y$

apply (rename_tac N)
1. $\bigwedge N.\ \exists y.\ N = y$

apply (rule_tac x = "N" in exI)

In general

(rename_tac $x_1 \dots x_n$) renames the rightmost (inner) n parameters to $x_1 \dots x_n$.

Forward Proof: frule and drule

apply (frule ⟨*rule*⟩)

Rule:	$[\![A_1; \ldots; A_m]\!] \Longrightarrow A$
Subgoal:	1. $[\![B_1; \ldots; B_n]\!] \Longrightarrow C$
Substitution:	$\sigma(B_i) \equiv \sigma(A_1)$
	Unifiable assumption B_i is chosen.
New subgoals:	1. $\sigma([\![B_1; \ldots; B_n]\!] \Longrightarrow A_2)$
	⋮
	m-1. $\sigma([\![B_1; \ldots; B_n]\!] \Longrightarrow A_m)$
	m. $\sigma([\![B_1; \ldots; B_n; A]\!] \Longrightarrow C)$

Like frule but also deletes B_i: **apply** (drule ⟨*rule*⟩)

Examples for Forward Rules

$$\frac{P \wedge Q}{P}\ \textsf{conjunct1} \qquad \frac{P \wedge Q}{Q}\ \textsf{conjunct2}$$

$$\frac{P \longrightarrow Q \qquad P}{Q}\ \textsf{mp}$$

$$\frac{\forall x.\ P\ x}{P\ ?x}\ \textsf{spec}$$

Forward Proof: OF

$$r \text{ [OF } r_1 \ldots r_n]$$

Prove assumption 1 of theorem r with theorem r_1, and assumption 2 with theorem r_2, etc ...

Rule r	$[\![A_1; \ldots; A_m]\!] \Longrightarrow A$
Rule r_1	$[\![B_1; \ldots; B_n]\!] \Longrightarrow B$
Substitution	$\sigma(B) \equiv \sigma(A_1)$
r [OF r_1]	$\sigma([\![B_1; \ldots; B_n; A_2; \ldots; A_m]\!] \Longrightarrow A)$

May use underscore to omit an argument:
r [OF _ r_2] proves assumption 2 with theorem r_2.

Forward proofs: THEN

r_1 [THEN r_2]   means r_2 [OF r_1]

Demo: Forward Proofs

Hilbert's Epsilon Operator

(David Hilbert, 1862-1943)

$\varepsilon\ x.\ P\ x$ is a value that satisfies P (if such a value exists)

ε also known as description operator.
In Isabelle the ε-operator is written SOME $x.\ P\ x$

$$\frac{P\ ?x}{P\ (\text{SOME}\ x.\ P\ x)}\ \text{someI}$$

More Epsilon

ε implies Axiom of Choice:

$$\forall x.\ \exists y.\ Q\ x\ y \Longrightarrow \exists f.\ \forall x.\ Q\ x\ (f\ x)$$

Existential and universal quantification can be defined with ε.

Isabelle also knows the definite description operator ι:

$$\frac{}{(\mathsf{THE}\ x.\ x = a) = a}\ \mathsf{the\_eq\_trivial}$$

More Proof Methods

apply (intro ⟨*intro-rules*⟩)	repeatedly applies intro rules
apply (elim ⟨*elim-rules*⟩)	repeatedly applies elim rules
apply clarify	applies all safe rules that do not split the goal
apply safe	applies all safe rules
apply fast **apply** best	sequent based automatic search tactics
apply blast	an automatic tableaux prover (works well on predicate logic)
apply metis	resolution prover for first-order logic with equality

Epsilon and Automation Demo

Attributes

Review:
Safe and unsafe rule; heuristics: use safe before unsafe

This can be automated

Automated methods (fast, blast, clarify etc) are not hardwired.
Use attributes to declare safe and unsafe intro and elim rules.

Syntax:

$[\langle kind \rangle !]$	for safe rules ($\langle kind \rangle$ one of intro, elim, dest)
$[\langle kind \rangle]$	for unsafe rules

More on Automation

Application (roughly):
do safe rules first, search/backtrack on unsafe rules only

Example:

declare attribute globally	**declare** conjI [intro!] allE [elim]
remove attribute gloabllay	**declare** allE [rule del]
use locally	**apply** (blast intro: someI)
delete locally	**apply** (blast del: conjI)

Demo: Attributes

We Have Learned so far...

- Proof rules propositional logic
- Proof rules for predicate calculus
- Safe and unsafe rules
- Forward proof
- The Epsilon Operator
- Some automation (classical reasoner)