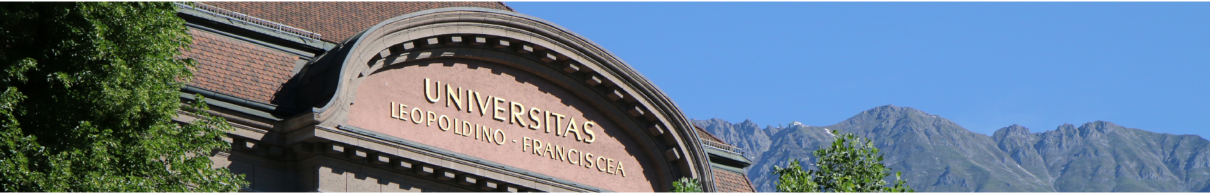
universität innsbruck

Summer Term 2023



Interactive Theorem Proving using Isabelle/HOL

Session 2

René Thiemann

Department of Computer Science

Outline

- The Pure Framework
- Structured Proofs

The Pure Framework

The Minimal Logic Isabelle/Pure

Pure = Generic Natural Deduction Framework

Pure Terms

- inference rules
- logical propositions

Deduction

higher-order resolution (that is, resolution using higher-order unification)

The type `prop`

- Isabelle/Pure contains a type of propositions: `prop`
- let φ `::` `prop` and ψ `::` `prop`, then
  - $\varphi \Longrightarrow \psi$ `::` `prop` (meta-)implication
  - $\bigwedge$`x`. φ `::` `prop` (meta-)quantification
- in Isabelle/HOL, every HOL-formula (t `::` `bool`) is also of type `prop`

Isabelle Symbols

symbol	internal	auto completion	abbreviation
$\Longrightarrow$	`\<Longrightarrow>`	`=` `=` `>`	`.` `>`
$\bigwedge$	`\<And>`	`!` `!`	

Remarks

- $\Longrightarrow$ is right-associative
- propositions with multiple assumptions are encoded by currying

Natural Deduction via Pure Connectives

- every Pure proposition can be read as natural deduction rule
- proposition $P_1 \Longrightarrow \ldots \Longrightarrow P_n \Longrightarrow C$ corresponds to rule

$$\frac{P_1 \quad \ldots \quad P_n}{C}$$

  with premises $P_1, \ldots, P_n$ and conclusion C
- scope of variables (like eigenvariable condition) enforced by $\bigwedge$ `Demo02.thy`
- there is no distinction between inference rules and theorems!

Examples

- `A ⟹ B ⟹ A ∧ B` (conjunction introduction)
- `(A ⟹ B) ⟹ A ⟶ B` (implication introduction)
  in order to prove `A ⟶ B` it suffices to prove `B` under the assumption `A`
- `(⋀ y. P y) ⟹ ∀ x. P x` (all introduction)
  in order to prove `∀ x. P x`, fix some variable `y` and prove `P y`

Schematic Variables

- besides free and bound variables, there are schematic variables (dark blue; these have leading ?)
- schematic variables can be instantiated arbitrarily
- proven inference rules such as $A \Longrightarrow B \Longrightarrow A \wedge B$ in Isabelle are written via schematic variables:

$$?A \Longrightarrow ?B \Longrightarrow ?A \wedge ?B \qquad \text{(thm conjI)}$$

- whenever a proof of a statement is finished, all free variables and outermost $\bigwedge$-variables in that statement are turned into schematic ones;
  example: each of the following two lines result in $?A \Longrightarrow ?B \Longrightarrow ?A \wedge ?B$
  - lemma "$A \Longrightarrow B \Longrightarrow A \wedge B$" ⟨*proof*⟩
  - lemma "$\bigwedge A\ B.\ A \Longrightarrow B \Longrightarrow A \wedge B$" ⟨*proof*⟩
- schematic variables may occur in proof goals, then the user can choose how to instantiate

Apply Single Inference Rule – The rule Method

- remember: each theorem can be seen as inference rule
- assume we have to prove goal with conclusion G
- assume *thm* has shape $P_1 \Longrightarrow \ldots \Longrightarrow P_n \Longrightarrow C$
- `proof (rule` *thm*`)` tries to unify C with G via unifier σ and replaces G by new subgoals coming from instantiated premises $P_1\sigma, \ldots, P_n\sigma$

Example

- consider goal $x < 5 \Longrightarrow x < 3 \wedge x < 2$
- the command `proof (rule conjI)` (conjI: $?A \Longrightarrow ?B \Longrightarrow ?A \wedge ?B$)
  - successfully unifies conclusion $x < 3 \wedge x < 2$ with $?A \wedge ?B$
    - only schematic variables can be instantiated in unification, i.e., here ?A and ?B, but not x
    - unifier: replace ?A by $x < 3$ and ?B by $x < 2$
  - and replaces the previous goal by two new subgoals
    - $x < 5 \Longrightarrow x < 3$
    - $x < 5 \Longrightarrow x < 2$

Another Example

- consider goal `∃ y. 5 < y`
- the command `proof (rule exI)` (`exI: ?P ?x ⟹ ∃ x. ?P x`)
  delivers one new subgoal: `5 < ?y` `Demo02.thy`
- details
  - try higher-order unification of `∃ x. ?P x` and `∃ y. 5 < y`
  - solution: replace `?P` by `λ z. 5 < z`
  - reason: after instantiation we get two terms
    - `∃ x. (λ z. 5 < z) x`
    - `∃ y. 5 < y`
  - these two terms are equivalent modulo $\alpha\beta\eta$
  - the unused schematic variable `?x` is renamed to `?y` since the goal used the name `y` in the existential quantor
  - the new subgoal is `(λ z. 5 < z) ?y` which is equal to `5 < ?y` modulo $\alpha\beta\eta$
- higher-order unification of terms s and t: find σ such that $s\sigma$ and $t\sigma$ are equivalent modulo $\alpha\beta\eta$

Equality in Isabelle

- all terms are normalized w.r.t. $\alpha\beta\eta$
- α-conversion: the names of bound variables are ignored:

  example: `∃ x. P x` is the same as `∃ y. P y`
- β-reduction

  $(\lambda\ x.\ t)\ u$ is the same as $t[x/u]$

  (here, $t[x/u]$ denotes the term t where x gets replaced by u)
- η-expansion

  $t :: ty \Rightarrow ty'$ is the same as $\lambda\ x.\ t\ x$
- `Demo02.thy`

Structured Proofs

Proofs – Outer Syntax

proof	::=	`sorry`	fake proof
	|	`by` *method method*$^?$	atomic proof
	|	`proof` *method*$^?$ *statement** `qed` *method*$^?$	structured proof
statement	::=	`fix` *variables* (`::` *type*)$^?$	arbitrary but fixed values
	|	`assume` *proposition*$^+$	local assumptions
	|	(`from` *fact*$^+$)$^?$ (`have` | `show`) *proposition proof*	(intermediate) result
	|	{ *statement** }	raw proof block
proposition	::=	(*label* :)$^?$ *term*	
fact	::=	*label*	
	|	‹*term*›	literal fact
method	::=	`auto` | `fact` | `rule` *fact* | `-` | ...	
command	::=	`lemma` *proposition proof* | ...	

Remarks

- *symbol*$^?$ denotes optional *symbol*; *symbol** denotes arbitrarily many occurrences of *symbol*

Demo – Drinker's Paradox

- statement: there is a person p, that if p drinks then everyone drinks
- formal proof is contained in `Demo02.thy` and it will illustrate various elements and variations of a proof w.r.t. the previous slide
- the upcoming slides mainly serve as a written down explanation, if something was not mentioned in the theory file or during the live demonstration

Remarks (cont'd)

- without *method* argument `proof` applies method `standard`
- idiom for starting structured proof without initial method "`proof -`"
- special label `this` refers to latest fact
- `show` used for statement that shows conclusion of surrounding `proof ... qed`

Some Proof Methods

- `rule` *fact* – apply single inference rule, namely *fact*
- `standard` – perform a single standard (with respect to current context) inference step
- `-` – do nothing
- `auto` – combines classical reasoning with simplification

Isabelle Symbols – Cartouches

symbol	internal	auto completion	abbreviations
⟨	`\<open>`		` and < <
⟩	`\<close>`		` and > >

Proving Propositions

- prove "$\bigwedge$x. P x" by

```
fix x
have "P x" ⟨proof⟩
```

- prove "A $\Longrightarrow$ B" by

```
assume "A"
have "B" ⟨proof⟩
```

Raw Proof Blocks

- the block

```
{
    fix x y
    assume "P x y" "Q x"
    have "R y" ⟨proof⟩             (* intermediate statement *)
    have "S x" ⟨proof⟩             (* last statement *)
}
```

- is exported as P ?x ?y $\Longrightarrow$ Q ?x $\Longrightarrow$ S ?x

Further Remarks and Statements

- introduce arbitrary but fixed value `x` by `fix x`
- introduce assumption by `assume "..."`
- indicate proposition to be proved by `have "..."` ⟨*proof*⟩
- local definition of `c` by `define c where "c = term"`
  (definition becomes available as theorem `c_def`)
- local abbreviation of `?c` by `let ?c = term`
- abbreviation `?thesis` refers to proposition before current `proof`-`qed`-block
- obtain witness satisfying `P` by `obtain x where "P x"` ⟨*proof*⟩

The rule Method using Current Facts

- on slide 8 it was explained what the rule method does without current facts
  - example Isabelle statement: `have P proof (rule` *thm*`)`
  - *thm* should have form of an introduction rule
  - conclusion in *thm* introduces some specific connective, e.g. $\ldots \Longrightarrow ?A \wedge ?B$
- if there are current facts, the behavior is different and it is tried to apply an elimination rule
  - example Isabelle statement: `from Q have P proof (rule` *thm*`)`
  - *thm* should have form of an elimination rule
  - major premise in *thm* contains specific connective, e.g., $?A \wedge ?B \Longrightarrow \ldots$, which is then unified with Q
  - in detail: given theorem $P_1 \Longrightarrow \ldots \Longrightarrow P_n \Longrightarrow C$, unify major premise P_1 of rule with first of current facts; unify remaining current facts with remaining premises; add rest of premises correspondingly instantiated as new subgoals

Example

```
...
have "x > 5 ∨ x = 2" ⟨proof⟩
from this have "A x"
proof (rule disjE)
  — ‹disjE: ?P ∨ ?Q ⟹ (?P ⟹ ?R) ⟹ (?Q ⟹ ?R) ⟹ ?R›
  show "x > 5 ⟹ A x" ⟨proof⟩
  show "x = 2 ⟹ A x" ⟨proof⟩
qed
```

The Difference Between have and show

- have is used to state arbitrary intermediate propositions
- show is used to discharge a current proof obligation
- show might reject a statement if it does not match a proof obligation
  - if assumptions have been used that are not present in proof obligation
  - if the types of variables are too specific or differ

Examples

- ```
  lemma "P x"
  proof -
    assume "Q x"
    from this show "P x" (* rejected, because of assumption Q x *)
  ```
- ```
  lemma "∃ x. x < 5"
  proof (rule exI)
    show "(3 :: nat) < 5" (* rejected, since type is too specific *)
  ```

The Difference Between HOL- and Meta-Implication/Quantification

- there are meta-connectives $\bigwedge$ and $\Longrightarrow$
- there are HOL-connectives $\forall$ and $\longrightarrow$
- usually the meta-connectives are preferable; example:
  - in `A ⟹ B ⟹ C ⟹ D` we can just assume `B`
  - in `A ⟶ B ⟶ C ⟶ D` we first have to apply implication introduction to access `B`
- the meta-connectives can only be used on the outside, so certain statements require HOL-connectives; example:
  - `∃ x. x > 5 ⟶ (∀ y. P x y)`
    (implication and universal quantor appear below existential quantor)
- consequence: most theorems in Isabelle are written using meta-connectives
  - `lemma "P x ⟹ Q ⟹ R x"` is preferred over
    `lemma "∀ x. P x ⟶ Q ⟶ R x"`

Proofs – Outer Syntax, Extended Grammar

proof	::=	sorry	fake proof
	|	by *method method*?	atomic proof
	|	proof *method*? *statement** qed *method*?	structured proof
statement	::=	fix *variables* (:: *type*)?	arbitrary but fixed values
	|	assume *proposition*+	local assumptions
	|	(from *fact*+)? (have | show) *proposition proof*	(intermediate) result
	|	{ *statement** }	raw proof block
	|	let ?x = *term*	local abbreviation
	|	(from *fact*+)? obtain *vars* where *prop. proof*	get witness
proposition	::=	(*label*:)? *term*	
fact	::=	*label*	
	|	this	previous proposition
	|	‹*term*›	literal fact
method	::=	auto | fact | rule *fact* | - | ...	
command	::=	lemma *proposition proof* | ...	