universität innsbruck

Summer Term 2019

Interactive Theorem Proving using Isabelle/HOL

Session 9

Christian Sternagel

Department of Computer Science

Topics

calculational reasoning, case analysis, code generation, computation induction, data type invariants, document preparation, finding theorems, first steps, functional programming in HOL, higher-order logic, history and motivation, induction, inductive definitions, Isabelle basics, Isabelle/Isar, Isabelle/ML, IsaFoR/CeTA, locales, manual termination proofs, multisets, natural deduction, notation, proof methods, PSL: a high-level proof strategy language, rule induction, rule inversion, session management, sets, simplification, sledgehammer, structural induction, structured proof, The Archive of Formal Proofs, the certification approach, total recursive functions, type classes, type definitions, well-foundedness

Overview

- Proof Methods
- Well-Foundedness
- Manual Termination Proofs
- Exercises

Some Useful Attributes

- `of` – instantiation of schematic variables (by position from left to right)
  `‹?x = ?x›[of y] ⇝ ‹y = y›`
- `OF` – discharge assumptions using existing facts (by position)
  `‹?A ⟹ ?A›[OF TrueI] ⇝ ‹True›`
- `symmetric` – get symmetric version of equation
  `‹?a = ?b›[symmetric] ⇝ ‹?b = ?a›`
- `rule_format` – replace HOL connectives by Pure connectives
  `‹∀x. ?P x ⟶ ?Q›[rule_format] ⇝ ‹?P ?x ⟹ ?Q›`
- `THEN` – composition of facts
  `‹?A ∈ Pow ?B ⟹ ?A ⊆ ?B›[THEN ‹?A ⊆ ?B ⟹ ?c ∈ ?A ⟹ ?c ∈ ?B›] ⇝`
  `‹?A ∈ Pow ?B ⟹ ?c ∈ ?A ⟹ ?c ∈ ?B›`
- `simp`, `intro`, `elim`, `dest` – declare fact simplification/introduction/elimination/destruction rule

Kinds of Rules

- simplification rules – (conditional) equations used from left to right
- introduction rules – if conclusion of rule matches conclusion of subgoal, replace it by premises of rule (generating one new subgoal per premise)
- destruction rules – replace first premise of subgoal matching major premise of rule by conclusion (together with remaining premises) of rule
- elimination rules – like destruction rules, but rule is supposed to not loose (destruct) information (compare `conjunct1` with `conjE`)

Examples

- `have "∀x. P x" apply (rule allI)` ⇝ `⋀x. P x`
- `have "A ∧ B ⟹ C" apply (drule conjunct2)` ⇝ `B ⟹ C`
- `have "A ∨ B ⟹ C" apply (erule disjE)` ⇝ 1. `A ⟹ C` 2. `B ⟹ C`

Equational Proof Methods

- `unfold` *fact*$^+$ – exhaustively apply equational facts (replacing left-hand sides by right-hand sides); usually as initial method
- `simp`/`simp_all` – exhaustively apply simp rules to first/all subgoal(s)

Proof Methods for Classical Reasoning

- (`intro` | `elim`) *fact*$^+$ – exhaustively apply intro/elim rules; usually as initial method
- `blast` (`best`, `fast`) – solve first subgoal by exhaustive proof search (up to certain bound) using all known intro/dest/elim rules (using best-first search, depth-first search)

Combined Proof Methods

- `force` (`fastforce`, `bestsimp`) – solve first subgoal by combination of equational and classical reasoning
- `auto` – apply combination of equational and classical reasoning to all subgoals and leave result as new subgoals

Modifiers for Classical Methods

classical methods (like `blast` and `auto`) take following modifiers:

- `intro`: *fact*$^+$ – add additional intro rules
- `dest`: *fact*$^+$ – add additional dest rules
- `elim`: *fact*$^+$ – add additional elim rules
- `del`: *fact*$^+$ – delete classical rules

Note

when used with combined methods (like `force` and `auto`), modifiers for simplifier use prefix `simp` (like `simp add:`, `simp del:`, ...)

Your favorite definition of well-foundedness?

Definition

relation $<$ is well-founded iff it meets one of following (equivalent) conditions:

- no infinite chains of shape $x_1 > x_2 > x_3 > \dots$ (C)
- $<$ admits induction, that is, for all P: $(\forall x.(\forall y < x.P(y)) \to P(x)) \to \forall x.P(x)$ (I)
- all elements are accessible, where x is accessible iff all $y < x$ are accessible (A)
- all nonempty sets have minimal element (M)
- no nonterminating set, where N nonterminating iff $N \neq \varnothing \wedge \forall x \in N.\exists y \in N.y < x$ (N)

`Demo09.thy` – Equivalence Proof

prove that (I) implies (A) and (N) implies (C)

Trivially Well-Founded Relations

- the empty relation is well-founded
- every finite acyclic relation is well-founded

(R acyclic iff no x such that $(x, x) \in R^+$)

Measure Functions

- every measure function $f : A \to \mathbb{N}$ induces relation $M_f = \{(x, y).\, f(x) < f(y)\}$
- M_f well-founded for any f by construction
- Isabelle notation for M_f is '`measure f`'

Lexicographic Product of Relations

- lexicographic product of relations R and S, written R `<*lex*>` S, give by $((x_1, x_2), (y_1, y_2)) \in R$ `<*lex*>` S iff $(x_1, y_1) \in R$ or both, $x_1 = y_1$ and $(x_2, y_2) \in S$
- lexicographic product of well-founded relations is well-founded

Demo09.thy – Alternative Implementation for Grouping Repeated List Elements

```
fun split_same :: "'a ⇒ 'a list ⇒ ('a list × 'a list)"
  where
    "split_same x [] = ([], [])"
  | "split_same x (y # ys) =
    (if x = y then let (us, vs) = split_same x ys in (y # us, vs)
    else ([], y # ys))"

fun group :: "'a list ⇒ 'a list list"
where
  "group [] = []"
| "group (x#xs) = (let (ys,zs) = split_same x xs in (x#ys) # group zs)"
```

Additional Simp Rules for Termination

- sometimes sufficient to add simp rules (only for termination proofs)
- use attribute `termination_simp`

Demo09.thy – An Odd Even/Odd Function

```
fun evenodd :: "nat ⇒ bool ⇒ bool"
  where
    "evenodd 0 True ⟷ True"
  | "evenodd x False ⟷ ¬ (evenodd x True)"
  | "evenodd (Suc x) True ⟷ evenodd x False"
```

Explicit Termination Proofs

- replace `fun f :: "T" ...` by `function (sequential) f :: "T" ... by (pat_completeness) auto`
- and add `termination by (relation "R") auto` for some well-founded relation `R` that covers recursive calls
- note: by default `fun` uses method `lexicographic_order` for termination proofs

Demo09.thy – Higher-Order Recursion

```
datatype tree = Tree nat "tree list"

fun map :: "('a ⇒ 'b) ⇒ 'a list ⇒ 'b list"
  where
    "map f [] = []"
  | "map f (x#xs) = f x # map f xs"

fun mirror :: "tree ⇒ tree"
  where
    "mirror (Tree n ts) = Tree n (rev (map mirror ts))"
```

Congruence Rules

- express on which values higher-order arguments have to agree to yield same result
- `fundef_cong` – declares congruence rule for function definitions

Exercises (start from `Exercises09.thy`)

URL
http://cl-informatik.uibk.ac.at/teaching/ss19/itp/thys/Exercises09.thy

Further Reading

- Alexander Krauss.
  Defining Recursive Functions in Isabelle/HOL.
  Isabelle documentation, 2018.
- Alexander Krauss.
  Automating Recursive Definitions and Termination Proofs in Higher-Order Logic.
  PhD thesis, Institut für Informatik, Technische Universität München, 2009.

Important Concepts

- `best` *(method)*
- `bestsimp` *(method)*
- `blast` *(method)*
- `del` *(modifiers)*
- `dest` *(attribute/modifier)*
- `elim` *(attribute/method/modifier)*
- `fast` *(method)*
- `fastforce` *(method)*
- `force` *(method)*
- `fundef_cong` *(attribute)*
- `intro` *(attribute/method/modifier)*
- `lexicographic_order` *(method)*
- `of` *(attribute)*
- `OF` *(attribute)*
- `rule_format` *(attribute)*
- `simp` *(attribute)*
- `symmetric` *(attribute)*
- `THEN` *(attribute)*
- `termination_simp` *(attribute)*
- `unfold` *(method)*