Preuves Interactives et Applications

Burkhart Wolff

https://www.lri.fr/~wolff/teach-material/2017-18/M2-CSMR/index.html

Université Paris-Saclay

Advanced Proof Techniques inb Isabelle/HOL

Revisions

- Elementary apply-style (backward) proofs
- Elementary attributed (forward) proofs
- Advanced apply-style proof techniques

Introduction to more Advanced Proof Techniques

- induction and case-splitting
- Rewriting
- Tableaux-provers (fast, blast, auto …)
- A magic device: sledgehammer

Simple Proof Commands

- Simple (Backward) Proofs:

```
lemma  <thmname> :
  [ <contextelem>+ shows ]"<ϕ>"
  <proof>
```

  - where <contextelem> declare elements of a proof context Γ (to be discussed further)
  - where <proof> is just a call of a high-level proof method by(simp), by(auto), by(metis), by(arith) or the discharger sorry (for the moment).

The Syntactic Category <proof>

- Notations for proofs so far:
  - ellipses:

    sorry, oops
  - "one-liners" simp and auto:

    by(<method>)   (abbrev: apply(...) done)
  - "apply-style proofs", backward-proofs:

    apply(<method>) … apply(<method>)

    done <method>
  - structured proofs:

    proof (<method>) … qed

A Summary of Proof Methods

- low-level procedures and
  versions with explicit substitution:
  - assumption
  - rule_tac <subst> in <thmname>
  - erule_tac <subst> in <thmname>
  - drule_tac <subst> in <thmname>

- … where <subst> is of the form:
  x_1="ϕ_1" and x_n="ϕ_n

A Summary of Proof Methods

- low-level procedures:
  - assumption (unifies conclusion vs. a premise)
  - subst [(asm)] <thmname>

    does one rewrite-step
    (by instantiating the HOL subst-rule)
  - rule <thmname>

    PROLOG - like resolution step using HO-Unification
  - erule <thmname>

    elimination resolution (for ND elimination rules)
  - drule <thmname>

    destruction resolution (for ND destriction rules)

A Summary of Proof Methods

- forward proof constructions by attributes
  - <thm>[THEN <thm>] (unifies conclusion vs. premise)
  - <thm>[OF <thm>] (unifies premise vs. conclusion)
  - <thm>[symmetric] (flips an equation)
  - <thm>[of (<term> | _)*] (instantiates variables)
  - <thm>[simp] (simplifies a thm)
  - <thm>[simp only: <thm>] (simplifies a thm)

Introduction to more Advanced Proof Techniques

- induction and case-splitting
- rewriting (= simplification)
- tableaux-provers (fast, blast, auto …)
- a magic device: sledgehammer

A Summary of Proof Methods

- advanced procedures:
  - insert <thmname>, insert <thmname>[„[„ of <subst>“]“]

    inserts local and global facts into assumptions
  - induct_tac “ϕ”, induct “ϕ” [arbitrary : „<variable>“]

    searches for appropriate induction scheme using type information and instantiates it
  - case_tac “ϕ”, cases “ϕ”,

    searches for appropriate case splitting scheme using type information and instantiates it

The Simplifier

Supports Rewriting, in particular:

- Rewriting of HO-Patterns,
- Ordered Rewriting
- Conditional Rewriting
- Context - Rewriting
- Automatic Case-Splitting

INSTRUMENTATION NECESSARY, so it is necessary to tell which rule should be used HOW. Simplification is quite predictable, using[[simp_trace]] shuts on tracing of the rewriter

The Simplifier

What is a higher-Order Pattern ?

It is a λ-term of form that is:

- constant head, i.e. of the form $c\ t_1 \ldots t_n$
- linear in free variables
- All HO Variables occur only in the form:

  $F(x_1 \ldots x_n)$ for distinct x_i

Seems very limited ? Well, you can have λ ...

Consider the rule:

$$\forall(\lambda\ x.\ P(x) \wedge Q(x)) = \forall(\lambda\ x.\ P(x)) \wedge (\forall(\lambda\ x.Q(x))$$

The Simplifier

Supports Rewriting, in particular:

- Rewriting of HO-Patterns, i.e. rules of the form:

  <lhs> = <rhs>

  where lhs is a HO-Pattern, where
  lhs is linear in the free variables and
  free variables in rhs occur also in lhs

```
apply(simp add: <rule>)
```

The Simplifier

Supports Rewriting, in particular:

- Ordered Rewriting:

  There is an implicit wf-ordering on terms.

  Rewriting is only done if the re-written term is smaller.

  Commutativity: $a+b = b+a$

  With a little trickery, one can have ACI rewriting:

disj_comms(2):	(P v Q v R) = (Q v P v R)
disj_comms(1):	(P v Q) = (Q v P)
disj_ac(3):	((P v Q) v R) = (P v Q v R)
disj_ac(2):	(P v Q v R) = (Q v P v R)
disj_ac(1):	(P v Q) = (Q v P)
disj_absorb:	(A v A) = A
disj_left_absorb:	(A v A v B) = (A v B)

The Simplifier

Supports Rewriting, in particular:

- Conditional Rewriting

if_P: $P \Longrightarrow (\text{if } P \text{ then } x \text{ else } y) = x$

if_not_P: $\neg P \Longrightarrow (\text{if } P \text{ then } x \text{ else } y) = y$

```
apply(simp add: if_P if_not_P)
```

(Not necessary, somewhere in the library it is stated:

```
declare if_P [simp] if_not_P [simp] )
```

...)

The Simplifier

Supports Rewriting, in particular:

- Context - Rewriting

HOL.if_cong:

$$b = c \Longrightarrow$$
$$(c \Longrightarrow x = u) \Longrightarrow$$
$$(\neg\, c \Longrightarrow y = v) \Longrightarrow$$
$$(\text{if } b \text{ then } x \text{ else } y) = (\text{if } c \text{ then } u \text{ else } v)$$

HOL.conj_cong:

$$P = P' \Longrightarrow (P' \Longrightarrow Q = Q') \Longrightarrow (P \land Q) = (P' \land Q')$$

```
apply(simp  cong: if_cong)
```

The Simplifier

Supports Rewriting, in particular:

- Automatic Case-Splitting
  (by a new type of rule which is NOT constant head)

split_if_asm: P (if Q then x else y) $= (\neg (Q \wedge \neg P\ x \vee \neg Q \wedge \neg P\ y))$

split_if: P (if Q then x else y) $= ((Q \longrightarrow P\ x) \wedge (\neg Q \longrightarrow P\ y))$

For any data type (example: Option):

Option.option.split_asm:

P (case x of None $\Rightarrow$ f1 | Some x $\Rightarrow$ f2 x) =

$(\neg (x = \text{None} \wedge \neg P\ f1 \vee (\exists a.\ x = \text{Some}\ a \wedge \neg P\ (f2\ a))))$

Option.option.split:

P (case x of None $\Rightarrow$ f1 | Some x $\Rightarrow$ f2 x) =

$((x = \text{None} \longrightarrow P\ f1) \wedge (\forall a.\ x = \text{Some}\ a \longrightarrow P\ (f2\ a)))$

```
apply(simp  split: split_if_asm split_if)
```

fast, blast and auto

Tableaux Provers

- For Logic terms and Set terms
- Uses all rules classified as
  - introduction rule   (keyword: intro)
    - works on conclusion of a goal
  - elimination rule   (keyword: elim)
    - works on assumptions of a goal
  - destruction drule (keyword:: dest)
    - works on assumptions of a goal
      applies modus ponens destructively
    - frule works on assumptions of a goal,
      applies modus ponens destructively

fast, blast and auto

fast

- will apply safe intro/elim/drule's blindly
  (these are rules like conjI, conjE, disjE, ...
  allI, exE, ... Rules that will transform a
  subgoal into an equivalent one, without
  loosing "logical content")
- with backtrack on unsafe rules
  (refines a subgoal into a logically stronger one,
  can lead into a dead end).

fast works for HO-Terms, but is fairly slow slow

blast

- dito, but resticted to first-order reasoning

fast, blast and auto

fast

- will apply safe intro/elim/drule's blindly (these are rules like conjI, conjE, disjE, ... allI, exE, ... Rules that will transform a subgoal into an equivalent one, without loosing “logical content”)
- will do backtrack-search on unsafe rules (refines a subgoal into a logically stronger one, can lead into a dead end. Ex: exI, allE).

fast works for HO-Terms, but is fairly slow

blast

- dito, but resticted to first-order reasoning

fast, blast and auto

blast

- works similarly like fast,
  but is resticted to first-order reasoning

  Substantially faster than fast,
  can treat transitivity rules.

auto

- intertwines simp, blast, and fast

A Summary of Proof Methods

- advanced automated procedures:
  - simp **[**add: <thmname>+**]** **[**del: <thmname>+**]**
    [split: <thmname>+**]** **[**cong: <thmname>+**]**
  - auto **[**simp: <thmname>+**]**
    [intro: <thmname>+**]** **[**intro [!]: <thmname>+**]**
    [dest: <thmname>+**]** **[**dest [!]: <thmname>+**]**
    [elim: <thmname>+**]** **[**elim[!]: <thmname>+**]**
  - metis <thmname>+
  - arith

Magic Device:

- sledgehammer - command.
  - asks well-known automatic first-order theorem provers such as
    - Vampire
    - E
    - CVC4
    - Z3

    ... if they can construct a proof based on all Isabelle theorems existing at this point, reconstructs an Isabelle proof.
  - does not work for proofs involving HO or induction.

Conclusion

- Isabelle focusses on interactive proofs (enabling presentation of intermediate steps, and structuring of proofs and prover instrumentations)
- … but this does not mean that there are no automatic proof techniques available and that classical ATP's are "better" in any sense …