Isabelle Tutorial:

System, HOL and Proofs

Burkhart Wolff
(with contributions by Makarius Wenzel)

Université Paris-Sud

What we will talk about

What we will talk about Isabelle with:

- Elementary Forward Proofs
- Tactic Proofs ("apply style")
- Proof Contexts and Structured Proof

Introduction to tactic backwards "apply style" proofs in Isabelle/HOL

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).

Processing <proof>

- In certain global commands requiring <proof>, the system enters into a "proof mode"
- This means, a proof state is created by

$$\Gamma \vdash_{\Theta} B \Longrightarrow B$$

refined by proof methods, and the required thm is finally extracted from it.

How to Declare Structured Goals

- (Simple) Context Element Declarations are:
  - fixed variables: fix <x> [:: <τ>]
  - assumptions: assume [<thmname>:] „<ϕ>"

    and [<thmname>:] „<ϕ>"

How to Declare Structured Goals

- In contrast (Rich) Context Elements are:

  - fixed variables: fixes <x> [:: <τ>]
  - assumptions: assumes [<thmname>:] „<ϕ>"
  - local definition: defines <x> ≡ <t>
  - reconsidering facts: notes a1=b1 … an=bn
  - intermed. results: have [<thmname>:] „<ϕ>"<proof>

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

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

Simple Proof Commands

- Simple (Backward) Proofs:

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

example:

```
lemma m : "conc (Seq a (Seq b Empty)) (Seq c Empty) =
                Seq a (Seq b (Seq c Empty))"
        by(simp)

schematic_lemma
         m' : "conc (Seq a (Seq b Empty))(Seq c Empty) =
                             ?X"
        by(simp)
```

Backward procedures: “tactic”s

- Concept: tactic is a RELATION on thm's. (mirroring non-determinism in the choice of unifiers or premisses)
- ... implemented in SML:

  thm -> thm Seq

- ... allowing to go
  - backward (via apply(...))
  - alternatives (via back)

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 <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)

Demo IV

- Simple apply-style proofs
  - build demo4 based on demo3
  - prove apply - style:

```
lemma m : "conc (Seq a (Seq b Empty)) (Seq c Empty) =
                Seq a (Seq b (Seq c Empty))"

lemma rev_c : "(reverse (Seq a (Seq b Empty))) =
                (Seq b (Seq a Empty))"


lemma conc_assoc: "conc (conc xs ys) zs = conc xs (conc ys zs)"

lemma reverse_conc: "reverse(conc xs ys)=conc(reverse ys) (reverse xs

lemma reverse_reverse: "reverse (reverse xs) = xs"
```

Resolution

- low-level procedures:
  - rule <thmname>

Resolution

- low-level procedures:
  - rule <thmname>

$$\frac{\alpha_1 \cdots \alpha_m}{\beta}$$

$$\frac{\phi_1 \quad \cdots \quad \phi_i \quad \cdots \quad \phi_n}{\psi}$$

$\phi_1, \ldots, \phi_n$ are current subgoals and ψ is original goal. Isabelle displays

Level ... (n subgoals)
ψ
1. ϕ_1
⋮
n. ϕ_n

$[\![\alpha_1; \ldots; \alpha_m]\!] \Longrightarrow \beta$ is rule.

Resolution

- low-level procedures:
  - rule <thmname>

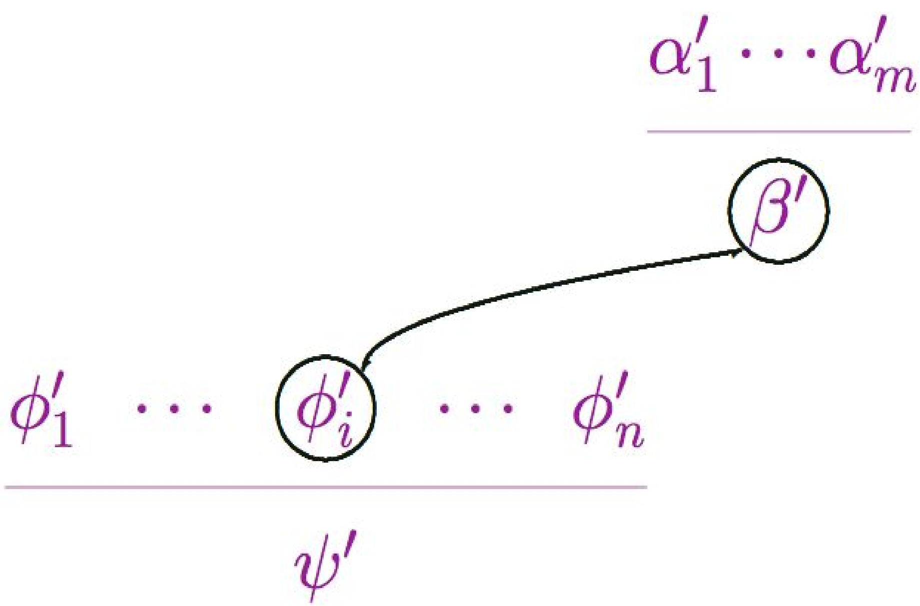
$$\frac{\alpha_1 \cdots \alpha_m}{\beta}$$

$$\frac{\phi_1 \quad \cdots \quad \phi_i \quad \cdots \quad \phi_n}{\psi}$$

Simple scenario where ϕ_i has no premises. Now β must be unifiable with selected subgoal ϕ_i.

Resolution

- low-level procedures:
  - rule <thmname>



Simple scenario where ϕ_i has no premises. Now β must be unifiable with selected subgoal ϕ_i.

We apply the unifier ($'$)

Resolution

- low-level procedures:
  - rule <thmname>

$$\frac{\phi'_1 \cdots \alpha'_1 \cdots \alpha'_m \cdots \phi'_n}{\psi'}$$

Simple scenario where ϕ_i has no premises. Now β must be unifiable with selected subgoal ϕ_i.

We apply the unifier ($'$)

We replace ϕ'_i by the premises of the rule.

Resolution

- low-level procedures:
  - rule <thmname>

$$\frac{\alpha_1 \quad \cdots \alpha_m}{\beta}$$

$$\frac{\phi_1 \quad \cdots \quad \bigwedge x.\phi_i \quad \cdots \quad \phi_n}{\psi}$$

Rule

Resolution

- low-level procedures (lifting over parameters):
  - rule <thmname>

$$\frac{\alpha_1 \quad \cdots \alpha_m}{\beta}$$

$$\frac{\phi_1 \quad \cdots \quad \bigwedge x.\phi_i \quad \cdots \quad \phi_n}{\psi}$$

Rule

Resolution

- low-level procedures(ligting over parameters):
  - rule <thmname>

$$\bigwedge x. \frac{\alpha_1(x) \cdots \alpha_m(x)}{\beta(x)}$$

$$\frac{\phi_1 \quad \cdots \quad \bigwedge x.\phi_i \quad \cdots \quad \phi_n}{\psi}$$

Rule is lifted over x: Apply $[?X \leftarrow ?X(x)]$.

Resolution

- low-level procedures(ligting over parameters):
  - rule <thmname>

$$\bigwedge x. \frac{\alpha_1(x) \cdots \alpha_m(x)}{\beta(x)}$$

$$\frac{\phi_1 \qquad \cdots \qquad \bigwedge x.\phi_i \qquad \cdots \qquad \phi_n}{\psi}$$

Rule is lifted over x: Apply $[?X \leftarrow ?X(x)]$.

As before, β must be unifiable with ϕ_i;

E-Resolution

- low-level procedures:
  - erule <thmname>

E-Resolution

- low-level procedures:
  - erule <thmname>

$$\frac{\alpha_1 \cdots \alpha_m}{\beta}$$

$$[\phi_{i1} \cdots \quad \phi_{il} \quad \cdots \phi_{ik_i}]$$

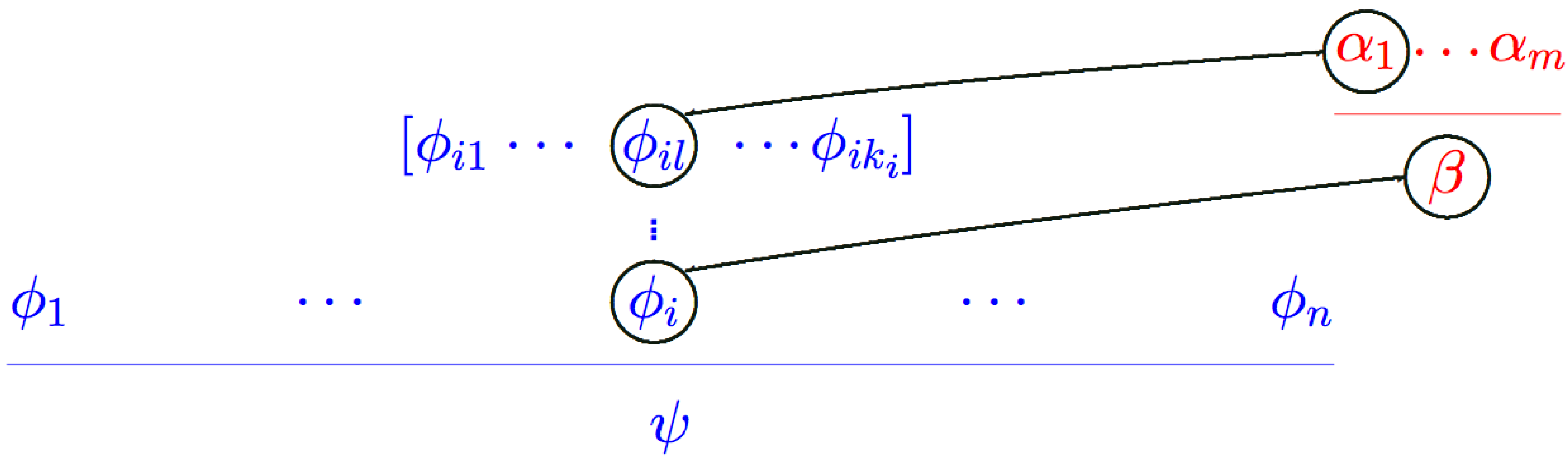
$$\vdots$$

$$\frac{\phi_1 \qquad \cdots \qquad \phi_i \qquad \cdots \qquad \phi_n}{\psi}$$

Same scenario as before

E-Resolution

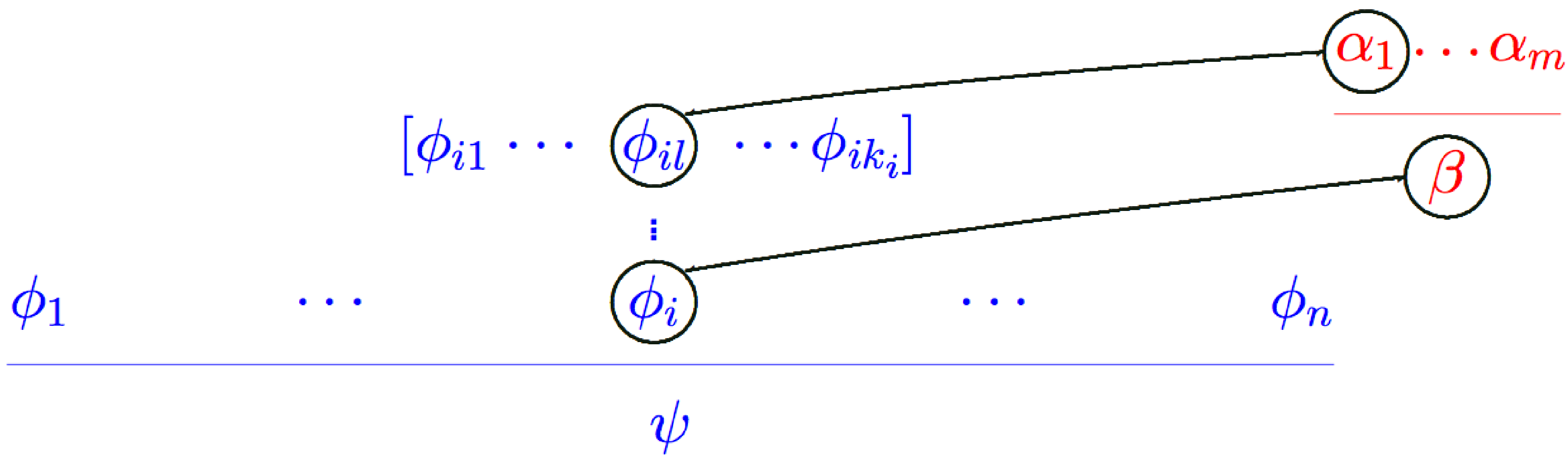
- low-level procedures:
  - erule <thmname>



Same scenario as before, but now β must be unifiable with ϕ_i, and α_1 must be unifiable with ϕ_{il}, for some l.

E-Resolution

- low-level procedures:
  - erule <thmname>



Same scenario as before, but now β must be unifiable with ϕ_i, and α_1 must be unifiable with ϕ_{il}, for some l.

D-Resolution

- low-level procedures:
  - drule <thmname>

D-Resolution

- low-level procedures:
  - drule <thmname>

$$\frac{\begin{array}{ccccc} & & [\phi_{i1} \cdots \phi_{il} \cdots \phi_{ik_i}] & & \\ & & \vdots & & \\ \phi_1 & \cdots & \phi_i & \cdots & \phi_n \end{array}}{\psi} \qquad \frac{\alpha}{\beta}$$

Simple rule

D-Resolution

- low-level procedures:
  - rule <thmname>

$$[\phi_{i1} \cdots \phi_{il} \cdots \phi_{ik_i}] \qquad \frac{\alpha}{\beta}$$

$$\vdots$$

$$\frac{\phi_1 \quad \cdots \quad \phi_i \quad \cdots \quad \phi_n}{\psi}$$

Simple rule, and α must be unifiable with ϕ_{il}, f

D-Resolution

- low-level procedures:
  - drule <thmname>

$$
\dfrac{\begin{array}{ccccc} & [\phi'_{i1} \cdots & \beta' & \cdots \phi'_{ik_i}] & \\ & & \vdots & & \\ \phi'_1 & \cdots & \phi'_i & \cdots & \phi'_n \end{array}}{\psi'}
$$

Simple rule, and α must be unifiable with ϕ_{il}, for some l.

We apply the unifier.

We replace premise ϕ'_{il} with the conclusion of the rule.

Backward Proofs: Example I

- Example:

  lemma "$A \wedge B \longrightarrow B \wedge A$"
  - apply (rule impI)
  - apply (rule conjI)
  - apply (rule conjunct2)   - schematic state
  - apply assumption
  - apply (rule conjunct1)   - schematic state
  - apply assumption
  - done

Backward Proofs: Example II

- Example:

```
lemma "A ∧ B → B ∧ A"
  apply (rule impI)
  apply (rule conjI)
  apply (erule conjunct2)
  apply (erule conjunct1)
  done
```

Backward Proofs: Example III

- Example:

  lemma ex8 1: ”($\forall$ x. p(x)) $\longrightarrow$($\exists$ x. p(x))”
  apply(rule impI)
  apply(rule exI)
  apply(erule spec)
  done

Backward Proofs: Example IV

- Example:

lemma all_istr: "(∀x. A ⟶ B(x)) = (A ⟶ (∀x. B(x)))"

apply(rule iffI)
apply(rule allI)
apply(erule spec)
apply(rule allI)

apply(rule spec) back

apply(drule mp)
apply(assumption)+
done

apply(rule impI)
apply(rule mp)
apply(assumption)
apply(rule impI)

Demo V

- Exos
  - $(A \wedge B) \wedge (C \wedge D) \longrightarrow (B \wedge C) \wedge (D \wedge A)$
  - low and high-level:

    $s\ (s\ (s\ (s\ (zero\ )))) = four \wedge p(zero\ ) \wedge$
    $(\forall x.p(x) \rightarrow p(s(s(x)))) \rightarrow p(four\ )$

  - $(\exists x.\forall y.p(x, y)) \rightarrow (\forall y.\exists x.p(x, y))$
  - $(\exists x.p(f(x))) \rightarrow (\exists x.p(x))$

A Summary of Proof Methods

- advanced procedures:
  - insert <thmname>
    inserts local and global facts into assumptions
  - induct "ϕ"

    searches for appropriate induction scheme using type information and instantiates it

  - cases "ϕ",   case_tac "ϕ"

    searches for appropriate induction scheme using type information and instantiates it

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

What we will talk about Isabelle with:

- Elementary Forward Proofs
- Tactic Proofs (“apply style”)
- Proof Contexts and Structured Proof

Structured Proofs in <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

The Syntactic Category <proof>

- structured proofs:
  - can be nested
  - allow to declare sub-goals
    declaratively
    (eased by pattern-matching and abbreviations)
  - subgoals were matched against the
    proof context
    (order irrelevant, lifting irrelevant)
  - allow for advanced notation
    for matching constructs following
    induction and case distinction
  - extensible (see ITP2014: “Eisbach”)

The Syntactic Category <proof>

- structured proofs:

  proof (<method>)
      <subgoal>
  {next
      <subgoal>}*
  qed

- subgoals:

  <rich context element>* show "ϕ"

Rich Proof Context Elements

- These are
  - fixed variables: fixes <x> [:: <τ>]
  - assumptions: assumes [<thmname>:] „<φ>“
  - local definition: defines <x> ≡ <t>
  - reconsidering facts: notes a1=b1 … an=bn
  - intermed. results: have [<thmname>:] „<φ>“<proof>
  - case - statements: case (<cons> <var>*)