Tactics and Generic Proof Procedures

Christian Urban

Apply vs ML

```
lemma disj_swap:
  shows "P ∨ Q ⟹ Q ∨ P"
apply(erule disjE)
apply(rule disjI2)
apply(assumption)
apply(rule disjI1)
apply(assumption)
done
```

Apply vs ML

```
lemma disj_swap:
  shows "P ∨ Q ⟹ Q ∨ P"
apply(erule disjE)
apply(rule disjI2)
apply(assumption)
apply(rule disjI1)
apply(assumption)
done
```

```
let
  val ctxt  = @{context}
  val goal  = @{prop "P ∨ Q ⟹ Q ∨ P"}
  val facts = []
  val schms = ["P", "Q"]
in
  Goal.prove ctxt schms facts goal
   (fn _ =>
      etac @{thm disjE} 1
      THEN rtac @{thm disjI2} 1
      THEN atac 1
      THEN rtac @{thm disjI1} 1
      THEN atac 1)
end
```

Tactics and tactic

```
val foo_tac =
     (etac @{thm disjE} 1
      THEN rtac @{thm disjI2} 1
      THEN atac 1
      THEN rtac @{thm disjI1} 1
      THEN atac 1)
```

```
 lemma
  shows "P ∨ Q ⟹ Q ∨ P"
apply(tactic {* foo_tac *})
done
```

Tactics and tactic

```
val foo_tac =
     (etac @{thm disjE} 1
      THEN rtac @{thm disjI2} 1
      THEN atac 1
      THEN rtac @{thm disjI1} 1
      THEN atac 1)
```

```
 lemma
  shows "P ∨ Q ⟹ Q ∨ P"
apply(tactic {* foo_tac *})
done
```

THEN just strings tactics together (tactic combinators or tacticals).

Type of Tactics

The type of tactics is

```
thm -> thm Seq.seq
```

The lazy sequences are possible successor states. The simplest tactics are:

```
fun no_tac thm = Seq.empty

fun all_tac thm = Seq.single thm
```

Type of Tactics

The type of tactics is

```
thm -> thm Seq.seq
```

The lazy sequences are possible successor states. The simplest tactics are:

```
fun no_tac thm = Seq.empty

fun all_tac thm = Seq.single thm
```

The possibilities can be explored on the Isabelle level using **back**.

Goal States are Theorems

This might be surprising, since in general there are still subgoals to be proved.

Goal States are Theorems

This might be surprising, since in general there are still subgoals to be proved.

```
fun my_print_tac ctxt thm =
let
  val _ = tracing (Syntax.string_of_term ctxt (prop_of thm))
in
  Seq.single thm
end
```

Goal States are Theorems

This might be surprising, since in general there are still subgoals to be proved.

```
fun my_print_tac ctxt thm =
let
  val _ = tracing (Syntax.string_of_term ctxt (prop_of thm))
in
  Seq.single thm
end
```

In general a goal state is the theorem

$$S_1 \ldots S_n \Longrightarrow \#C$$

Tactics for Manipulating the Goal States

```
lemma shows "P ⟹ P"
apply(tactic {* atac 1 *})

lemma shows "P ∧ Q"
apply(tactic {* resolve_tac [@{thm conjI}] 1 *})

lemma shows "P ∧ Q ⟹ False"
apply(tactic {* eresolve_tac [@{thm conjE}] 1 *})

lemma shows "False ∧ True ⟹ False"
apply(tactic {* dresolve_tac [@{thm conjunct2}] 1 *})
```

Tactics for Manipulating the Goal States

```
lemma
  shows "True = False"
apply(tactic {* cut_facts_tac [@{thm True_def}, @{thm False_def}] 1 *})
```

goal (1 subgoal):
 1. $\llbracket$True $\equiv$ (λx. x) = (λx. x); False $\equiv$ $\forall$P. P$\rrbracket$ $\Longrightarrow$ True = False

Pre-Instatiations

Becuase of schematic variables, theorems need to be often pre-instantiated.

```
lemma
  shows "∀x ∈ A. P x ⟹ Q x"
apply(tactic {* dresolve_tac [@{thm bspec}] 1 *})
```

```
goal (2 subgoals):
  1. ?x ∈ A
  2. P ?x ⟹ Q x
```

Pre-Instatiations

Becuase of schematic variables, theorems need to be often pre-instantiated.

```
lemma
  shows "∀ x ∈ A. P x ⟹ Q x"
apply(tactic {* dresolve_tac [@{thm bspec}] 1 *})
```

```
goal (2 subgoals):
 1. ?x ∈ A
 2. P ?x ⟹ Q x
```

```
@{thm disjI1} RS @{thm conjI}
> ⟦?P1; ?Q⟧ ⟹ (?P1 ∨ ?Q1) ∧ ?Q
```

MRS, RL,. . .

Tacticals

```
val foo_tac' = EVERY' [etac @{thm disjE},
                       rtac @{thm disjI2},
                       atac,
                       rtac @{thm disjI1},
                       atac]
```

Tacticals

```
val foo_tac' = EVERY' [etac @{thm disjE},
                        rtac @{thm disjI2},
                        atac,
                        rtac @{thm disjI1},
                        atac]
```

A tactic to analyse the topmost logical connective:

```
val sel_tac =  FIRST' [rtac @{thm conjI},
                        rtac @{thm impI},
                        rtac @{thm notI},
                        rtac @{thm allI}, K all_tac]
```

Tacticals

```
val sel_tac =  FIRST' [rtac @{thm conjI},
                       rtac @{thm impI},
                       rtac @{thm notI},
                       rtac @{thm allI}, K all_tac]
```

```
val sel_tac' = TRY o FIRST' [rtac @{thm conjI},
                             rtac @{thm impI},
                             rtac @{thm notI},
                             rtac @{thm allI}]
```

A Decision Procedure for PIL

$$\frac{}{A, \Gamma \vdash A} \qquad \frac{}{f, \Gamma \vdash C}$$

$$\frac{A, B, \Gamma \vdash C}{A \wedge B, \Gamma \vdash C} \qquad \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}$$

$$\frac{A, \Gamma \vdash C \quad B, \Gamma \vdash C}{A \vee B, \Gamma \vdash C} \qquad \frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \quad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}$$

$$\frac{A \longrightarrow B, \Gamma \vdash A \quad B, \Gamma \vdash C}{A \longrightarrow B, \Gamma \vdash C} \qquad \frac{A, \Gamma \vdash B}{\Gamma \vdash A \longrightarrow B}$$

A Decision Procedure for PIL

$$\frac{A \longrightarrow B, \Gamma \vdash A \quad B, \Gamma \vdash C}{A \longrightarrow B, \Gamma \vdash C}$$

is replaced by

$$\frac{A \longrightarrow B \longrightarrow C, \Gamma \vdash D}{(A \wedge B) \longrightarrow C, \Gamma \vdash D}$$

$$\frac{A \longrightarrow C, B \longrightarrow C, \Gamma \vdash D}{(A \vee B) \longrightarrow C, \Gamma \vdash D}$$

$$\frac{B \longrightarrow C, \Gamma \vdash A \longrightarrow B \quad B, \Gamma \vdash D}{(A \longrightarrow B) \longrightarrow C, \Gamma \vdash D}$$

$$\frac{B, A, \Gamma \vdash C}{A \longrightarrow B, A, \Gamma \vdash C}$$

Simple Implementation

```
val apply_tac =
let
  val intros = [@{thm conjI}, @{thm disjI1}, @{thm disjI2},
                @{thm impI}, @{thm iffI}]
  val elims  = [@{thm FalseE}, @{thm conjE}, @{thm disjE},
                @{thm iffE}, @{thm impE2}, @{thm impE3},
                @{thm impE4}, @{thm impE5}, @{thm impE1}]
in
  atac
  ORELSE' resolve_tac intros
  ORELSE' eresolve_tac elims
end
```

Simple Implementation

```
val apply_tac =
let
  val intros = [@{thm conjI}, @{thm disjI1}, @{thm disjI2},
                @{thm impI}, @{thm iffI}]
  val elims  = [@{thm FalseE}, @{thm conjE}, @{thm disjE},
                @{thm iffE}, @{thm impE2}, @{thm impE3},
                @{thm impE4}, @{thm impE5}, @{thm impE1}]
in
  atac
  ORELSE' resolve_tac intros
  ORELSE' eresolve_tac elims
end
```

```
lemma
  shows "((((P ⟶ Q) ⟶ P) ⟶ P) ⟶ Q) ⟶ Q"
apply(tactic {* (DEPTH_SOLVE o apply_tac) 1 *})
done
```

SUBPROOF

See example.

Setting up Goals

$$(P\ 2 = P\ 3 \longrightarrow P\ 3 \wedge P\ 2 \wedge P\ 1) \wedge$$
$$(P\ 1 = P\ 2 \longrightarrow P\ 3 \wedge P\ 2 \wedge P\ 1) \wedge$$
$$(P\ 1 = P\ 3 \longrightarrow P\ 3 \wedge P\ 2 \wedge P\ 1) \longrightarrow$$
$$P\ 3 \wedge P\ 2 \wedge P\ 1$$

rhs n = $\bigwedge$i...n. P i

lhs n = $\bigwedge$i...n. P i = P (i + 1 mod n) $\longrightarrow$ rhs n

de_bruijn n = lhs (2*n+1) $\longrightarrow$ rhs (2*n+1)

Setting up Goals

$$
\begin{aligned}
&(P\ 2 = P\ 3 \longrightarrow P\ 3 \wedge P\ 2 \wedge P\ 1) \wedge \\
&(P\ 1 = P\ 2 \longrightarrow P\ 3 \wedge P\ 2 \wedge P\ 1) \wedge \\
&(P\ 1 = P\ 3 \longrightarrow P\ 3 \wedge P\ 2 \wedge P\ 1) \longrightarrow \\
&\qquad P\ 3 \wedge P\ 2 \wedge P\ 1
\end{aligned}
$$

```
fun P n =
  @{term "P::nat ⇒ bool"} $ (mk_number @{typ "nat"} n)

fun rhs 1 = P 1
  | rhs n = mk_conj (P n, rhs (n - 1))

fun lhs 1 n = mk_imp (mk_eq (P 1, P n), rhs n)
  | lhs m n =  mk_conj
                 (mk_imp (mk_eq (P (m - 1), P m), rhs n),
                  lhs (m - 1) n)
```

Setting up Goals

```
fun de_bruijn ctxt n =
let
  val i = 2*n+1
  val goal = mk_Trueprop (mk_imp (lhs i i, rhs i))
in
  Goal.prove ctxt ["P"] [] goal
   (fn _ => (DEPTH_SOLVE o apply_tac) 1)
end

de_bruijn @{context} 1
```

Handling Schematic Variables

$$
\begin{aligned}
&(b = c \longrightarrow a \wedge b \wedge c) \wedge \\
&(a = b \longrightarrow a \wedge b \wedge c) \wedge \\
&(a = c \longrightarrow a \wedge b \wedge c) \longrightarrow \\
&\qquad\qquad\qquad\qquad a \wedge b \wedge c
\end{aligned}
$$

Handling Schematic Variables

$$
\begin{aligned}
&(b = c \longrightarrow a \wedge b \wedge c) \wedge \\
&(a = b \longrightarrow a \wedge b \wedge c) \wedge \\
&(a = c \longrightarrow a \wedge b \wedge c) \longrightarrow \\
&\qquad\qquad a \wedge b \wedge c
\end{aligned}
$$

$$
\begin{aligned}
&(?b = ?c \longrightarrow ?a \wedge ?b \wedge ?c) \wedge \\
&(?a = ?b \longrightarrow ?a \wedge ?b \wedge ?c) \wedge \\
&(?a = ?c \longrightarrow ?a \wedge ?b \wedge ?c) \longrightarrow \\
&\qquad\qquad ?a \wedge ?b \wedge ?c
\end{aligned}
$$

```
fun de_bruijn ctxt n =
let
 val i = 2*n+1
 val bs = replicate (i+1) "b"
 val (nbs, ctxt') = Variable.variant_fixes bs ctxt
 val fbs = map (fn z => Free (z, @{typ "bool"})) nbs
 fun P n = nth fbs n

 fun rhs 1 = P 1
   | rhs n = mk_conj (P n, rhs (n - 1))

 fun lhs 1 n = mk_imp (mk_eq (P 1, P n), rhs n)
   | lhs m n =  mk_conj (mk_imp
               (mk_eq (P (m - 1), P m), rhs n), lhs (m - 1) n)

 val goal = mk_Trueprop (mk_imp (lhs i i, rhs i))
in
 Goal.prove ctxt' [] [] goal
   (fn _ => (DEPTH_SOLVE o apply_tac) 1)
end
```

```
fun de_bruijn ctxt n =
let
  val i = 2*n+1
  val bs = replicate (i+1) "b"
  val (nbs, ctxt') = Variable.variant_fixes bs ctxt
  val fbs = map (fn z => Free (z, @{typ "bool"})) nbs
  fun P n = nth fbs n

  fun rhs 1 = P 1
    | rhs n = mk_conj (P n, rhs (n - 1))

  fun lhs 1 n = mk_imp (mk_eq (P 1, P n), rhs n)
    | lhs m n =  mk_conj (mk_imp
                  (mk_eq (P (m - 1), P m), rhs n), lhs (m - 1) n)

  val goal = mk_Trueprop (mk_imp (lhs i i, rhs i))
in
  Goal.prove ctxt' [] [] goal
    (fn _ => (DEPTH_SOLVE o apply_tac) 1)
end
```

```
fun de_bruijn ctxt n =
let
  val i = 2*n+1
  val bs = replicate (i+1) "b"
  val (nbs, ctxt') = Variable.variant_fixes bs ctxt
  val fbs = map (fn z => Free (z, @{typ "bool"})) nbs
  fun P n = nth fbs n

  fun rhs 1 = P 1
    | rhs n = mk_conj (P n, rhs (n - 1))

  fun lhs 1 n = mk_imp (mk_eq (P 1, P n), rhs n)
    | lhs m n =  mk_conj (mk_imp
                  (mk_eq (P (m - 1), P m), rhs n), lhs (m - 1) n)

  val goal = mk_Trueprop (mk_imp (lhs i i, rhs i))
in
  Goal.prove ctxt' [] [] goal
    (fn _ => (DEPTH_SOLVE o apply_tac) 1)
  |> singleton (ProofContext.export ctxt' ctxt)
end
```