Interactive Theorem Proving in HOL4

Course 06: Basic Tactics

Dr Chun TIAN
chun.tian@anu.edu.au

5 September 2024

Australian National University

Acknowledgement of Country

We acknowledge and celebrate the First Australians on whose traditional lands we meet, and pay our respect to the elders past and present.

More information about Acknowledgement of Country can be found here and here

Goal-Directed Proofs and Tactics

Goal-Directed Proofs

- User starts by setting a proof goal (as statements of the targeting theorem);
- *Tactics* (like `CONJ_TAC`) to reduce the current goal to zero or more subgoals;
- *Tacticals* (like `THEN` and `THEN1`) to organize tactics in tree-like structure.
- When no subgoal is left, the forward proof is automatically constructed from bottom up, to generate the final theorem.

Category of Built-in Tactics

- Goal deconstruction;
- Quantifiers;
- Assumption management;
- Subgoal management;
- Rewriting;
- Case analysis (case splits);
- Induction;
- Renaming and abbreviation;
- Automatic Provers.

Goal Deconstruction: Conjunctive Goals (1)

Prove two conjunctives separately

```
CONJ_TAC : tactic
```

Before

```
    Initial goal:

    P /\ Q: proof
```

After

```
    Q

    P (* first subgoal *)

2 subgoals
    : proof
```

Goal Deconstruction: Conjunctive Goals (2)

Prove the first conjunctive, then use the first to prove the second

```
CONJ_ASM1_TAC : tactic
```

Before

```
    Initial goal:

    P /\ Q: proof
```

After

```
        Q
    ------------------------------------
     0.  P

    P

2 subgoals
```

Why it works?

$\vdash P \wedge Q \iff P \wedge (P \Rightarrow Q).$

Goal Deconstruction: Conjunctive Goals (3)

Use the 2nd conjunctive to prove the 1st one (then prove the 2nd)

```
CONJ_ASM2_TAC : tactic
```

Before

```
Initial goal:

P /\ Q: proof
```

After

```
Q


     P
------------------------------------
 0.  Q


2 subgoals
```

Alternative Approach

Use `ONCE_REWRITE_TAC [CONJ_SYM]` to rewrite the goal to `Q /\ P`.

Goal Deconstruction: Conjunctive Goals (4)

```
hurdUtils.STRONG_CONJ_TAC : tactic
```

Before

```
    Initial goal:

    P /\ Q: proof
```

After

```
val it =

   P ==> Q

   P

2 subgoals
```

For the other direction

Use `ONCE_REWRITE_TAC [CONJ_SYM]` and then `STRONG_CONJ_TAC` to get $Q \Rightarrow P$.

Goal Deconstruction: Disjunctive Goals (1)

Prove the first disjunctive subgoal only

`DISJ1_TAC : tactic`

Before

```
    Initial goal:

    P \/ Q: proof
```

After

```
1 subgoal:
val it =

   P

   : proof
```

See also

`DISJ2_TAC` for proving the second disjunctive only.

Goal Deconstruction: Disjunctive Goals (2)

Use the first (negated) disjunctive to prove the second

```
hurdUtils.STRONG_DISJ_TAC : tactic
```

Before

```
    Initial goal:

    ~P \/ Q: proof
```

After

```
1 subgoal:
> val it =

        Q
   ------------------------------------
    0.  P

   : proof
```

Quantifiers: Eliminate Universal Quantifier

```
qx_gen_tac  : term quotation -> tactic
qx_genl_tac : term quotation list -> tactic
```

Before

```
    Initial goal:

    !x y. P x y: proof
```

Tactic

```
qx_genl_tac [‘a‘, ‘b‘]
```

After

```
1 subgoal:
val it =

   P a b

   : proof
```

Alternatives

```
X_GEN_TAC ‘‘x‘‘ >> X_GEN_TAC ‘‘y‘‘
```

“rpt GEN_TAC” or “NTAC 2 STRIP_TAC”.

Quantifiers: Introduce Universal Quantifier

```
qspec_tac     : term quotation * term quotation -> tactic
qid_spec_tac : term quotation list -> tactic
```

Before

```
   Initial goal:

   P a b: proof
```

Tactic

```
qspec_tac (‘b‘, ‘y‘)
```

After

```
1 subgoal:
val it =


   !y. P a y

   : proof
```

Alternatives

`qid_spec_tac ‘a‘` is equivalent to `qspec_tac (‘a‘,‘a‘)`.

Quantifiers: Eliminate Existential Quantifier

```
qexists_tac : term quotation -> tactic
```

Before

```
    Initial goal:

    ?n. SUC n = 1: proof
```

Tactic

```
qexists_tac ‘0‘
```

After

```
1 subgoal:
val it =

   SUC 0 = 1

   : proof
```

Quantifiers: Choose for Existential Quantifier

Choose a variable for existential quantifier and push to assumptions

```
Q.X_CHOOSE_TAC : term quotation -> thm_tactic
STRIP_TAC : tactic
```

Before

```
   Initial goal:

   (?x. P x) ==> Q: proof
```

Tactic

```
DISCH_THEN (Q.X_CHOOSE_TAC ‘y‘)
```

After

```
1 subgoal:
val it =

        Q
   ------------------------------------
    0.  P y

   : proof
```

Assumption Management (1)

Adding a theorem as a new assumption

```
ASSUME_TAC : thm_tactic (= thm -> tactic)
```

Before

```
    Initial goal:

    ~P ==> ~Q: proof
```

Tactic

```
ASSUME_TAC (Q.SPECL [‘P‘, ‘Q‘]
                     CONTRAPOS_THM)
```

After

```
1 subgoal:
val it =

         ~P ==> ~Q
   ------------------------------------
    0.   ~P ==> ~Q <=> Q ==> P

   : proof
```

Assumption Management (2)

Move proposition from goal to assumption

```
DISCH_TAC : tactic
STRIP_TAC : tactic
```

Before

```
   Initial goal:

   P ==> Q
```

Tactic

```
DISCH_TAC
```

After

```
val it =

       Q
   ------------------------------------
    0.  P

   : proof
```

Assumption Management (3)

Moving last assumption to the goal

```
POP_ASSUM : thm_tactic -> tactic
MP_TAC    : thm_tactic
```

Before

```
       R
   ------------------------------------
    0.  P
    1.  Q

   : proof
```

Tactic

```
POP_ASSUM MP_TAC
```

After

```
val it =

       Q ==> R
   ------------------------------------
    0.  P

   : proof
```

Assumption Management (4)

Moving matched assumption to the goal

```
Q.PAT_X_ASSUM : term quotation -> thm_tactic -> tactic
```

Before

```
      R
    ------------------------------------
     0.  P
     1.  Q

    : proof
```

Tactic

```
Q.PAT_X_ASSUM ‘P‘ MP_TAC
```

After

```
val it =

      P ==> R
    ------------------------------------
     0.  Q

    : proof
```

Subgoal Management (1)

Using subgoals

- Goal-directed proofs do not always revert the informal proof;
- Good formal proofs are in forwarding direction, aligned with the informal proofs;
- Subgoals are stage work of the proof stored into assumptions;
- Subgoals make proofs read easier;
- Subgoals can be either in forward or backward styles.

Subgoal Management (2)

Prove and place a theorem on the assumptions of the goal

```
op by : term quotation * tactic -> tactic
```

Before

```
    Initial goal:

    !x. P x: proof
```

Tactic

```
‘Q‘ by cheat
```

After

```
val it =

        !x. P x
   ------------------------------------
    0.  Q

   : proof
```

The subgoal inherits all assumptions of the current goal.

Subgoal Management (3)

Prove and place a theorem into the goal

```
hurdUtils.Know : term quotation -> tactic
```

Before

```
    Initial goal:

    !x. P x: proof
```

Tactic

```
Know `Q :bool`
```

After

```
2 subgoals:
val it =

   Q ==> !x. P x

   Q

2 subgoals
```

Subgoal Management (4)

Replace the goal's conclusion with a sufficient alternative.

```
hurdUtils.Suff : term quotation -> tactic
```

Before

```
   Initial goal:

   !x. P x: proof
```

Tactic

```
Suff `Q :bool`
```

After

```
   Q

   Q ==> !x. P x

2 subgoals
```

Rewriting Tactics (1)

Rewriting goal using theorems

```
REWRITE_TAC : (thm list -> tactic)
```

Before

```
val it =
   Proof manager status: 1 proof.
   1. Incomplete goalstack:
        Initial goal:
        SUC n = n + 1
   : proofs
> ADD1;
val it = |- !m. SUC m = m + 1: thm
```

Tactic

```
REWRITE_TAC [ADD1]
```

After

```
OK..
val it =
   Initial goal proved.
   |- SUC n = n + 1: proof
```

Rewriting Tactics (2)

Rewriting goal using theorems and assumptions

```
ASM_REWRITE_TAC : (thm list -> tactic)
```

Before

```
       n + 1 = 2
  ------------------------------------
   0.  n = 1

  : proof
```

Tactic

```
ASM_REWRITE_TAC []
```

After

```
1 subgoal:
val it =

       1 + 1 = 2
  ------------------------------------
   0.  n = 1

  : proof
```

Rewriting Tactics (3)

Rewriting goal and assumptions using simpset, theorems and *newer* assumptions

```
FULL_SIMP_TAC : simpset -> thm list -> tactic
```

Before

```
     m + 1 = 2
  ------------------------------------
   0.  m = n
   1.  n = 1

  : proof
```

Tactic

```
FULL_SIMP_TAC std_ss []
```

After

```
1 subgoal:
val it =

     1 + 1 = 2
  ------------------------------------
   0.  m = 1
   1.  n = 1

  : proof
```

Rewriting Tactics (4)

Rewriting goal and assumptions using simpset, theorems and *older* assumptions

```
REV_FULL_SIMP_TAC : simpset -> thm list -> tactic
```

Before

```
     m + 1 = 2
   ------------------------------------
    0.  n = 1
    1.  m = n

   : proof
```

Tactic

```
FULL_SIMP_TAC std_ss []
```

After

```
1 subgoal:
val it =

       1 + 1 = 2
   ------------------------------------
    0.  n = 1
    1.  m = 1

   : proof
```

Induction (1)

How induction works

1. There must be an induction theorem to apply, e.g.:

   [numTheory.INDUCTION]
   $\vdash P\ 0 \wedge (\forall n.\ P\ n \Rightarrow P\ (\mathtt{SUC}\ n)) \Rightarrow \forall n.\ P\ n$

2. The tactic HO_MATCH_MP_TAC is used for applying the induction theorem.

Before

```
   Initial goal:


   !n. n + n = 2 * n: proof
```

Tactic

```
HO_MATCH_MP_TAC numTheory.INDUCTION
```

After

```
val it =


   0 + 0 = 2 * 0 /\
   !n. n + n = 2 * n ==>
       SUC n + SUC n = 2 * SUC n

   : proof
```

Induction (2)

Induction-related tactics

```
Induct                 : tactic
Induct_on              : term quotation -> tactic
```

Before

```
    Initial goal:

    !n. n + n = 2 * n: proof
```

Tactics

`Induct`, or `Induct_on ‘n‘`

After

```
         SUC n + SUC n = 2 * SUC n
    ------------------------------------
     0.  n + n = 2 * n

    0 + 0 = 2 * 0

2 subgoals
    : proof
```