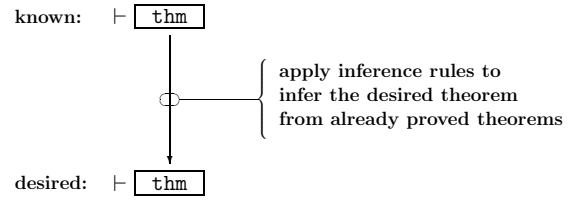


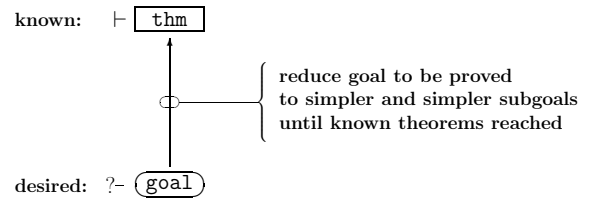
Goal-Directed Proof in HOL

Backward Proof

- Forward proof style:

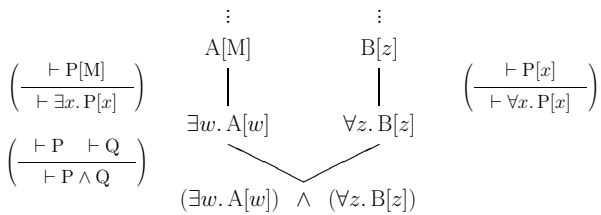


- Goal-directed (or backward) proof style:



Logical Basis of Backward Proof

- The basic idea is:



- Reduction of a goal to subgoals is justified by an inference in the 'opposite direction'.

Backward Proof in HOL

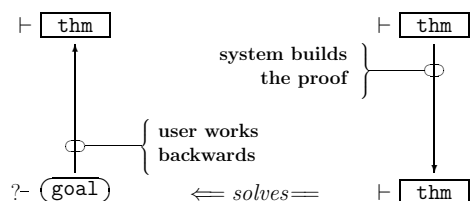
- Goals are represented by values of ML type

$$\text{goal} \stackrel{\text{def}}{=} \underbrace{\text{term list}}_{\text{assumptions}} * \underbrace{\text{term}}_{\text{conclusion}}$$

- Theorems and goals:

$$\underbrace{P_1, \dots, P_n \vdash P}_{:\text{thm}} \text{ solves } \Rightarrow \underbrace{([P_1, \dots, P_n], P)}_{:\text{goal}}$$

- Goal-directed proof in HOL:



Tactics

- A tactic T is a function:

$$T : \text{goal} \rightarrow \underbrace{\text{goal list}}_{\text{subgoals}} \quad \underbrace{(\text{thm list} \rightarrow \text{thm})}_{\text{justification}}$$

- Suppose that for a given goal g

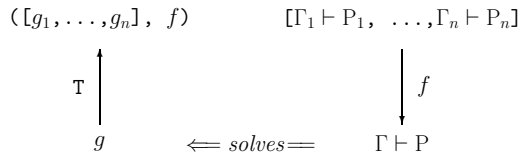
$$T(g) = ([g_1, \dots, g_n], f)$$

If the theorems $\Gamma_1 \vdash P_1, \dots, \Gamma_n \vdash P_n$ solve the goals g_1, \dots, g_n , then

$$f([\Gamma_1 \vdash P_1, \dots, \Gamma_n \vdash P_n])$$

should solve the original goal g .

- In a picture:



Notation for Goals and Tactics

- Notation used in the manual for goals:

$$\Gamma \text{ ?- } P$$

- A tactic is written:

$$\begin{array}{c}
 \langle \text{goal} \rangle \\
 \hline
 \langle \text{subgoal} \rangle \quad \dots \quad \langle \text{subgoal} \rangle
 \end{array}
 \quad \langle \text{name} \rangle \langle \text{args} \rangle$$

- Example:

$$\begin{array}{c}
 A \text{ ?- } t1 \wedge t2 \\
 \hline
 A \text{ ?- } t1 \quad A \text{ ?- } t2
 \end{array}
 \quad \text{CONJ_TAC}$$

This splits a conjunction into two conjuncts.

A Tactic for Conjunctive Goals

- The tactic is:

$$\begin{array}{c}
 A \text{ ?- } t1 \wedge t2 \\
 \hline
 A \text{ ?- } t1 \quad A \text{ ?- } t2
 \end{array}
 \quad \text{CONJ_TAC}$$

- Example:

```

- val (sgs,p) = CONJ_TAC ([], ``T /\ ~F``);
> val sgs = [([],``T``),([],``~F``)] : goal list
> val p = fn : validation

- val th1 = TRUTH;
> val th1 = |- T : thm

- val th2 = NOT_INTRO(SPEC ``F`` FALSITY);
> val th2 = |- ~F : thm

- p [th1,th2];
> val it = |- T /\ ~F : thm
    
```

- Note the ML type abbreviation:

$$\text{validation} \stackrel{\text{def}}{=} \text{thm list} \rightarrow \text{thm}$$

Implementation

- The code for CONJ_TAC is:

```

fun CONJ_TAC (asl,w) =
  let val (l,r) = dest_conj w
  in [(asl,l), (asl,r)],
      fn [th1,th2] => CONJ th1 th2
  end handle _ => failwith "CONJ_TAC";
    
```

- Example:

```

- val (asl,w) = ([], ``T /\ ~F``):goal;
> val asl = [] : term list
> val w = ``T /\ ~F`` : term

- val (l,r) = dest_conj w;
> val l = ``T`` : term
> val r = ``~F`` : term

- [(asl,l),(asl,r)]:goal list;
> val it = [([],``T``),([],``~F``)] : goal list

- fn [th1,th2] => CONJ th1 th2;
> val it = fn : thm list -> thm
    
```

Solving Goals with Theorems

- To use a theorem to solve a goal:

```
A ?- g
===== ACCEPT_TAC (A |- g)
-
```

The theorem must be α -equivalent to the goal.

- Example:

```
- ACCEPT_TAC;
> val it = fn : thm_tactic

- ETA_AX;
> val it = |- !t. (\x. t x) = t : thm

- val (sg,p) =
  ACCEPT_TAC ETA_AX
  ([], 'Q:'a->'b. (\y. Q y) = Q');
> val sg = [] : goal list
> val p = fn : validation

- p [];
> val it = |- !t. (\x. t x) = t : thm
```

- Notice the type abbreviation:

```
thm_tactic def thm -> tactic
```

Summary

- ML type of goals:

```
goal def term list * term
```

- ML type of tactics:

```
tactic def goal -> (goal list * validation)
```

- Tactics and justifications:

- suppose $T(g) = ([g_1, \dots, g_n], f)$
- then if the theorems $\Gamma_1 \vdash P_1, \dots, \Gamma_n \vdash P_n$ solve the subgoals g_1, \dots, g_n , then

$$f [\Gamma_1 \vdash P_1, \dots, \Gamma_n \vdash P_n]$$
 should solve the original goal g .

- Notation and examples:

- notation for goals: $\Gamma ?- P$
- examples: CONJ_TAC and ACCEPT_TAC

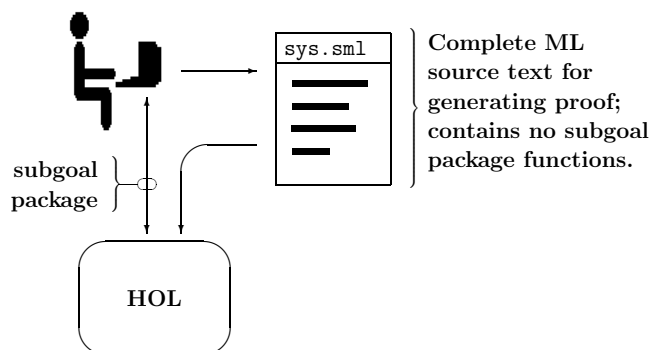
The Subgoal Package

- HOL has a *subgoal package* for finding tactic proofs interactively.

- The subgoal package:

- maintains a stack of subgoals to be proved
- provides functions that operate on these subgoals

- The subgoal package is for *finding* tactic proofs:



Functions in the Subgoal Package

- Put an initial goal (A, g) on the stack:

```
set_goal (A, g);
```

- Expand the current goal using a tactic T :

```
expand T;
```

- Rotate the subgoal stack by n steps:

```
rotate n;
```

- Undo the last change to the goal stack:

```
backup ();
```

- Get the top goal on the stack:

```
top_goal ();
```

Example

- Set the goal:

```
- set_goal ([], 'T /\ ~F');
> val it = New goal stack.
  T /\ ~F : proofs
```

- Expand with CONJ_TAC:

```
- expand CONJ_TAC;
OK..
2 subgoals:
> val it =
  ~F
  T : proofs
```

- Solve the first subgoal:

```
- expand (ACCEPT_TAC TRUTH);
OK..
Goal proved.
|- T

Remaining subgoals:
> val it = ~F : proofs
```

Example Continued

- Solve the second subgoal:

```
- val thm = NOT_INTRO(SPEC 'F' FALSITY);
> val thm = |- ~F : thm

- expand (ACCEPT_TAC thm);
OK..
Goal proved. |- ~F
> val it = Original goal proved.
  |- T /\ ~F : proofs
```

- Get the actual theorem:

```
- val foo = top_thm ();
> val foo = |- T /\ ~F : thm
```

Proof by Contradiction

- The contradiction tactic:

```
      A ?- g
===== CONTR_TAC (B |- F)
      -
```

We should have $B \subseteq A$ and, obviously, $B \neq \{\}$.

- Example:

```
- set_goal ([['n < n'], '1 = 0']);
> val it = New goal stack.
  1 = 0
-----
  n < n : proofs

- show_assums := true;
> val it = () : unit

- val thm = UNDISCH (SPEC 'n:num'
  prim_recTheory.LESS_REFL);
> val thm = [n < n] |- F : thm

- expand (CONTR_TAC thm);
OK..
> val it = Original goal proved.
  [n < n] |- 1 = 0 : proofs
```

Adding Assumptions

- To put a theorem onto the assumption list:

```
      A ?- g
===== ASSUME_TAC (B |- t)
      A u {t} ?- g
```

We require that $B \subseteq A$.

- Example:

```
- set_goal ([['n < m'], 'n < SUC(SUC m)']);
> val it = New goal stack.
  n < SUC (SUC m)
-----
  n < m : proofs

- val thm1 = SPEC 'm:num'
  prim_recTheory.LESS_SUC_REFL;
> val thm1 = |- m < SUC m : thm

- expand (ASSUME_TAC thm1);
OK..
1 subgoal:
> val it =
  n < SUC (SUC m)
-----
  n < m
  m < SUC m : proofs
```

Proving Disjunctions

- To prove a disjunctive goal:

```

A \\/ B
===== DISJ1_TAC
A

```

```

A \\/ B
===== DISJ2_TAC
B

```

- Example:

```

- set_goal ([], 'F \\/ T');
> val it = New goal stack.
F \\/ T
-----
: proofs
- expand DISJ2_TAC;
OK..
1 subgoal:
> val it = T
-----
: proofs
- expand (ACCEPT_TAC TRUTH);
OK..
Goal proved. |- T
> val it = Original goal proved.
|- F \\/ T : proofs

```

Boolean Equality

- The tactic is:

```

A ?- t1 = t2
===== EQ_TAC
A ?- t1 ==> t2    A ?- t2 ==> t1

```

- Example:

```

- set_goal ([], '(?n. n < m) = ~(m = 0)');
> val it = New goal stack.
(?n. n < m) = ~(m = 0)
-----
: proofs
- expand EQ_TAC;
OK..
2 subgoals:
> val it =
~(m = 0) ==> (?n. n < m)
-----
(?n. n < m) ==> ~(m = 0)
-----
: proofs

```

Universal Quantification

- The tactic is:

```

A ?- !x.P
===== GEN_TAC
A ?- P[x'/x]

```

The system chooses the variable x' so that it is not free in $!x.P$ or A .

- For example, set the goal:

```

- set_goal ([['x > 0'], '!x:num. x = x']);
> val it = New goal stack.
!x. x = x
-----
x > 0 : proofs

```

- Expand with GEN_TAC:

```

- expand GEN_TAC;
OK..
1 subgoal:
> val it =
x' = x'
-----
x > 0 : proofs

```

Example Continued

- Current goal:

```

- pr();
> val it =
x' = x'
-----
x > 0 : proofs

```

- We can solve this with:

```

A ?- t = t
===== REFL_TAC
-

```

- Continuing the proof:

```

- expand REFL_TAC;
OK..
Goal proved.
[x > 0] |- x' = x'
> val it = Original goal proved.
[x > 0] |- !x. x = x : proofs

```

Implications

- To assume the antecedent:

```
A ?- P ==> Q
===== DISCH_TAC
A u {P} ?- Q
```

- Example:

```
- set_goal ([], '(?x:'a. P x) ==> ~!x. ~P x');
> val it = New goal stack.
(?x. P x) ==> ~(!x. ~(P x))
-----
: proofs
- expand DISCH_TAC;
OK..
1 subgoal:
> val it =
~(!x. ~(P x))
-----
?x. P x : proofs
```

Stripping a Goal

- STRIP_TAC strips the outermost connective or quantifier using one of the tactics:

```
A ?- P /\ Q      A ?- !x.P      A ?- P ==> Q
=====
A ?- P      A ?- Q      A ?- P[x'/x]      A u {P} ?- Q
```

- It is often (but *not always*) the right thing to do first in typical tactic proofs. For example, set the goal:

```
- set_goal
([], '(!x. (x \/ ~x) ==> !y:num. P y /\ Q y');
> val it = New goal stack.
!x. x \/ ~x ==> (!y. P y /\ Q y)
-----
: proofs
```

- First, strip the universal quantifier:

```
- expand STRIP_TAC;
OK..      1 subgoal:
> val it =
x \/ ~x ==> (!y. P y /\ Q y)
-----
: proofs
```

The Example Continued

- The current goal:

```
- pr();
> val it =
x \/ ~x ==> (!y. P y /\ Q y)
-----
: proofs
```

- Now, assume the antecedent:

```
- expand STRIP_TAC;
OK..
2 subgoals:
> val it =
!y. P y /\ Q y
-----
~x
!y. P y /\ Q y
-----
x : proofs
```

- Note that STRIP_TAC (unlike DISCH_TAC) breaks the disjunctive antecedent into two subgoals.

The Example Concluded

- Continue to break up the goal using STRIP_TAC:

```
- expand STRIP_TAC;
OK..
1 subgoal:
> val it =
P y /\ Q y
-----
x : proofs
- expand STRIP_TAC;
OK..
2 subgoals:
> val it =
Q y
-----
x
P y
-----
x : proofs
```

- Then use other tactics to finish the proof...

Induction

- The induction tactic:

```
A ?- !n:num.P[n]
===== Induct
A ?- P[0] A u {P[n']} ?- P[SUC n']
```

The system chooses n' to be not free in A .

- Example:

```
- set_goal([], '!n. 0 <= n');
> val it = New goal stack.
'!'n. 0 <= n'

-----
: proofs
- expand Induct;
OK.. 2 subgoals:
> val it =
  0 <= SUC n
-----
  0 <= n

0 <= 0
-----
: proofs
```

Rewriting Tactics

- Rewrite the goal using supplied theorems and basic simplifications:

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

- Rewrite the goal using only the supplied theorems:

```
PURE_REWRITE_TAC: thm list -> tactic
```

- Rewrite the using the supplied theorems and the assumptions of the goal:

```
ASM_REWRITE_TAC: thm list -> tactic
PURE_ASM_REWRITE_TAC: thm list -> tactic
```

- There are also 'ONCE_' variants of these:

```
ONCE_REWRITE_TAC: thm list -> tactic
PURE_ONCE_REWRITE_TAC: thm list -> tactic
ONCE_ASM_REWRITE_TAC: thm list -> tactic
PURE_ONCE_ASM_REWRITE_TAC: thm list -> tactic
```

Summary

- The subgoal package:

- `set_goal (A,g);`
- `expand T;`
- `top_thm();`

- Valid and invalid tactics.

- Some basic tactics:

- `CONTR_TAC`, `ASSUME_TAC`, `DISJ1_TAC`, `DISJ2_TAC`,
`EQ_TAC`, `GEN_TAC`, `REFL_TAC`, `EXISTS_TAC`, `SUBST_TAC`,
`DISCH_TAC`, `STRIP_TAC`, `STRIP_ASSUME_TAC`,
`Induct`.

- Rewriting tactics:

```
REWRITE_TAC
ASM_REWRITE_TAC
ONCE_REWRITE_TAC
ONCE_ASM_REWRITE_TAC
PURE_REWRITE_TAC
PURE_ASM_REWRITE_TAC
PURE_ONCE_REWRITE_TAC
PURE_ONCE_ASM_REWRITE_TAC
```