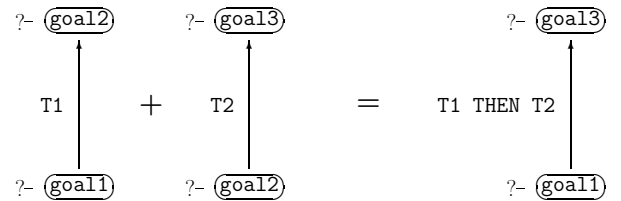


Goal-Directed Proof

Tacticals

Tacticals

- Tacticals are ML functions used to:
 - build more powerful tacticals from simpler ones,
 - modify the behaviour of tacticals,
 - construct complete goal-directed proofs.
- For example, sequencing of tacticals:



- We will now look at the basic HOL tacticals...

Choice of Alternatives

- The tactical ORELSE is an infix ML function:

```
ORELSE : tactic -> tactic -> tactic
```

- If T_1 and T_2 are tacticals, then

T_1 ORELSE T_2

is a tactic which

- first tries the tactic T_1 , and
- applies T_1 if it succeeds, or else
- tries T_2 if T_1 fails

- The ML definition of ORELSE is simple:

```
- infix ORELSE;
infix ORELSE

- fun (f1 ORELSE f2) g = f1 g handle _ => f2 g;
> val ORELSE = fn:('a -> 'b) * ('a -> 'b) -> 'a -> 'b
```

Example of Choice

- Example, try CONJ_TAC or else GEN_TAC

```
- val tac:tactic = CONJ_TAC ORELSE GEN_TAC;
> val tac = fn : tactic

- set_goal ([], ``!x:'a. R x``);
> val it = New goal stack.
!x. R x

-----
: proofs

- expand CONJ_TAC;
OK..
uncaught exception HOL_ERR

- expand tac;
OK..
1 subgoal:
> val it =
R x

-----
: proofs
```

Sequencing

- The tactical THEN is an ML infix:

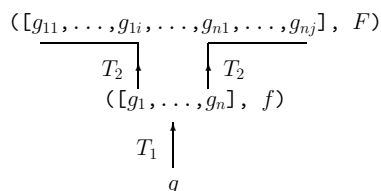
```
THEN : tactic -> tactic -> tactic
```

- If T_1 and T_2 are tactics, then

T_1 THEN T_2

is a tactic which

- first applies the tactic T_1 , and
 - then applies T_2 to *all* the resulting subgoals
- In a picture, T_1 THEN T_2 :



Example of Sequencing

- Example:

```

- set_goal([], '(!x:num. R x) /\ (!y:num. Q y)');
> val it = New goal stack.
      (!x. R x) /\ (!y. Q y)
-----
                                     : proofs
- expand CONJ_TAC;
OK..      2 subgoals:
> val it =
      !y. Q y
-----
      !x. R x
-----
                                     : proofs
- backup();
> val it = New goal stack.
      (!x. R x) /\ (!y. Q y)
-----
                                     : proofs
- expand (CONJ_TAC THEN GEN_TAC);
OK..      2 subgoals:
> val it =
      Q y
-----
      R x
-----
                                     : proofs

```

Alternative Sequencing

- THEN applies its second tactic to *all* the subgoals generated by the first tactic.
- If you want to do something different to each resulting subgoals, use THENL:

```
THENL : tactic -> tactic list -> tactic
```

- If T, T_1, \dots, T_n are tactics, then

T THENL $[T_1, \dots, T_n]$

is a tactic which

- first applies the tactic T , and
 - then applies T_i to the i th resulting subgoal
- If T generates n subgoals, then there must be exactly n tactics in the supplied list.

Repetition

- Tactical that repeats a tactic until failure:

```
REPEAT : tactic -> tactic
```

- If T is a tactic, then the tactic REPEAT T applies T repeatedly until it fails.
- The ML definition of REPEAT:

```

- val ALL_TAC : tactic = fn g => ([g],hd);
> val ALL_TAC = fn : tactic

- fun REPEAT tac g =
  ((tac THEN REPEAT tac) ORELSE ALL_TAC) g ;
> val REPEAT = fn : tactic -> tactic

```

Example of Repetition

- A very common first step:

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

- expand (REPEAT STRIP_TAC);
OK.. 2 subgoals:
> val it =
Q
-----
Q
R

Q
-----
~R
R
: proofs
```

Tacticals and the Subgoal Package

- There are two distinctly *different* things in goal-directed proof:

- the activity of finding a proof
- the completed proof itself

- The subgoal package, and completed proofs:

```
expand [ ] ;
expand [ ] ;
expand [ ] ;
:
```

HOL session

```
THEN [ ]
THEN [ ]
:
```

ML source

- HOL provides some auxiliary functions for packaging-up completed tactic proofs...

Proving and Saving a Theorem

- To prove a theorem and save it:

```
store_thm : (string * term * tactic) -> thm
```

This function:

- takes as arguments name ' $\langle name \rangle$ ', a goal (A, g) and a tactic T ,
- tries to prove the goal using the tactic,
- if the tactic succeeds in proving the goal, returns the resulting theorem, and
- saves the theorem under the name ' $\langle name \rangle$ ' in the current theory.

Example of Packaged Proofs

- Part of the ML sources for the built-in HOL theory of arithmetic:

```
:
val ADD_SYM =
  store_thm
  ("ADD_SYM",
   ``!m n. m + n = n + m``,
   Induct THEN
   GEN_TAC THEN
   ASM_REWRITE_TAC[ADD_0, ADD, ADD_SUC]);
val num_CASES =
  store_thm
  ("num_CASES",
   ``!m. (m = 0) \ / ?n. m = SUC n``,
   Induct THEN
   REWRITE_TAC[NOT_SUC] THEN
   EXISTS_TAC ('m:num') THEN
   REWRITE_TAC[]);
:
```

Summary of Tacticals

- Choice of alternatives:

```
ORELSE : tactic -> tactic -> tactic
```

- Sequencing:

```
THEN : tactic -> tactic -> tactic
```

- Alternative sequencing:

```
THENL : tactic -> tactic list -> tactic
```

- Repetition:

```
REPEAT : tactic -> tactic
```

- Auxiliary functions:

```
TAC_PROOF : (goal * tactic) -> thm  
store_thm : (string * goal * tactic) -> thm
```

List Tacticals

- Two simple *list tacticals* are:

```
EVERY : tactic list -> tactic
```

```
FIRST : tactic list -> tactic
```

- They behave as follows:

- EVERY $[T_1, \dots, T_n]$ applies the tactics T_1, \dots, T_n in sequence to the goal.

- FIRST $[T_1, \dots, T_n]$ applies the first of the tactics T_1, \dots, T_n that succeeds when applied to the goal.

- Thus we have the equivalences:

```
EVERY  $[T_1, \dots, T_n] = T_1$  THEN ... THEN  $T_n$ 
```

```
FIRST  $[T_1, \dots, T_n] = T_1$  ORELSE ... ORELSE  $T_n$ 
```

Mapping Tacticals

- The tacticals

```
MAP_EVERY : ('a -> tactic) -> 'a list -> tactic
```

```
MAP_FIRST : ('a -> tactic) -> 'a list -> tactic
```

take an ML function f of type $'a \rightarrow \text{tactic}$ and a list l of type $'a$ list and

- apply f all the elements in the list l to get a list of tacticals, and
 - apply every tactic in the list (resp. the first tactic that succeeds) to the goal.
- For example

```
MAP_EVERY ASSUME_TAC [|- P1, ..., |-Pn]
```

is equivalent to

```
EVERY [ASSUME_TAC |- P1, ..., ASSUME_TAC |- Pn]
```

Using the Assumptions

- There are three list tacticals for using the assumption of a goal:

```
EVERY_ASSUM : (thm -> tactic) -> tactic
```

```
FIRST_ASSUM : (thm -> tactic) -> tactic
```

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

- These use the assumptions of a goal *as theorems*.
- For example, use the assumptions as inputs to tacticals like ACCEPT_TAC or CONTR_TAC.

Using Every Assumption

- The tactical

```
EVERY_ASSUM : (thm -> tactic) -> tactic
```

takes an ML function f of type $\text{thm} \rightarrow \text{tactic}$ and a goal (A, g) and

- maps ASSUME down the list of assumptions A ,
- maps f down the resulting list of theorems,
- applies every tactic in the resulting list of tactics to the goal.

- You can read

```
EVERY_ASSUM ttac
```

as *'do ttac, using every assumption'*.

Example

- Consider the goal:

```
- set_goal(['n = 1', 'm = 2'], ['n < m']);
> val it = New goal stack.
  n < m
-----
      m = 2
      n = 1
      : proofs
```

- The tactic SUBST1_TAC:

```
      A ?- P
===== SUBST1_TAC (B |- t = u)
      A ?- P[u/t]
```

```
- SUBST1_TAC;
> val it = fn : thm -> tactic
```

Example - concluded

- Substitute into the conclusion:

```
- expand (EVERY_ASSUM SUBST1_TAC);
OK..
1 subgoal:
> val it =
  1 < 2
-----
      m = 2
      n = 1
      : proofs
```

Using the First Useful Assumption

- The tactical

```
FIRST_ASSUM : (thm -> tactic) -> tactic
```

takes an ML function f of type $\text{thm} \rightarrow \text{tactic}$ and a goal (A, g) , and

- maps ASSUME down the list of assumptions A ,
- maps f down the resulting list of theorems,
- applies the first tactic in the resulting list of tactics that succeeds when applied to the goal.

- For example, consider the goal:

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

Example Continued

- After stripping:

```
- expand (REPEAT STRIP_TAC);
OK..      2 subgoals:
> val it =
R
-----
  R
  Q
Q
-----
  R
  Q      : proofs
```

- The conclusion is an assumption, so:

```
- expand (FIRST_ASSUM ACCEPT_TAC);
OK..
Goal proved.
. |- Q

Remaining subgoals:
> val it =
R
-----
  R
  Q      : proofs
```

Using the Entire Assumption List

- The tacticals `FIRST_ASSUM` and `EVERY_ASSUM` are special cases of

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

which takes an ML function f of type

```
thm list -> tactic
```

and a goal (A, g) , and

- maps `ASSUM` down the list of assumptions A ,
- applies f to the resulting list of theorems,
- applies the resulting tactic to the goal.

- For example, `EVERY_ASSUM` is defined by:

```
- val EVERY_ASSUM:thm_tactic->tactic =
  ASSUM_LIST o MAP_EVERY;
> val EVERY_ASSUM = fn : thm_tactic -> tactic
```

Summary — List Tacticals

- For lists of tactics:
 - `EVERY` : tactic list -> tactic
 - `FIRST` : tactic list -> tactic
- Mapping tacticals:
 - `MAP_EVERY` : ('a -> tactic) -> 'a list -> tactic
 - `MAP_FIRST` : ('a -> tactic) -> 'a list -> tactic
- Using assumptions:
 - `EVERY_ASSUM` : thm_tactic -> tactic
 - `FIRST_ASSUM` : thm_tactic -> tactic
 - `ASSUM_LIST` : (thm list -> tactic) -> tactic

'Resolution' Tactics

- In HOL, 'resolution' involves deducing from a list of facts

```
[|- P1, |- P2, ..., |- Pn]
```

and an implication

```
|- Q1 ==> (Q2 ==> ... ==> (Qm ==> Q))
```

some instance of $|- Q$.

- This is done by matching antecedents Q_j with facts P_i and using modus ponens.

Matching Modus Ponens

- Suppose we have two theorems

```
thm1 = |- !x:'a. !y:'a. !z:'a. R[x,y,z] ==> Q[x,y,z]
```

```
thm2 = |- R[a:num, b:num, c:num]
```

and wish to deduce $\text{|- } Q[a, b, c]$

- This could be done as follows:

```
- val ith = INST_TYPE [(('a:num', (':'a')))] thm1;
> val ith = |- !x y z. R[x,y,z] ==> Q[x,y,z]

- val sth = SPECL [(('a:num', ('b:num', ('c:num')))] th;
> val sth = |- R[a,b,c] ==> Q[a,b,c]

- val thm3 = MP sth thm2;
> val it = |- Q[a,b,c]
```

- But MATCH_MP does this automatically:

```
- val thm3 = MATCH_MP thm1 thm2;
|- Q[a,b,c]
```

Resolution

- The main resolution tactic is

```
IMP_RES_TAC : thm -> tactic
```

This takes an implication

```
|- R1 ==> (R2 ==> ... ==> (Rn ==> Q))
```

and does the following:

- match the antecedents R_1, \dots, R_n to the assumptions of the goal
- derive conclusions by modus ponens and add the conclusions to the assumption list.

- The tactic:

```
RES_TAC : tactic
```

does resolution among the assumptions of a goal by calling IMP_RES_TAC for all the implications in the assumption list.

Example of Resolution

- Example:

```
- set_goal ([('a < b', ('b < c'), ('~(c < a)')]);
> val it = New goal stack.
~(c < a)

-----
b < c
a < b      : proofs

- val [_, thm] =
  RES_CANON arithmeticTheory.LESS_TRANS;
> val thm = |- !p n. n < p ==> (!m. m < n ==> m < p):thm

- expand (IMP_RES_TAC thm);
OK..      1 subgoal:
> val it =
~(c < a)

-----
b < c
a < b
!m. m < a ==> m < b
a < c      : proofs
```

- If the matching can be done several ways, all possible results are added to the assumptions.

Example Continued

- IMP_RES_TAC can solve goals:

```
- set_goal ([('a < b', ('b < c'), ('(a < c)')]);
> val it = New goal stack.
a < c

-----
b < c
a < b      : proofs

- arithmeticTheory.LESS_TRANS;
> val it = |- !m n p. m < n /\ n < p ==> m < p : thm

- expand (IMP_RES_TAC it);
OK..
> val it =
Original goal proved.
.. |- a < c : proofs
```

Solving Goals with Resolution

- A goal is solved if resolution generates:
 - the theorem $\Gamma \vdash F$, or
 - a theorem α -equivalent to the goal's conclusion
- Example proof:

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

- expand (REPEAT STRIP_TAC);
OK..      1 subgoal:
> val it =
  Q a
  -----
  !x. R x ==> Q x
    R a          : proofs

- expand RES_TAC;
OK..      Goal proved.
.. |- Q a
> val it =
  Original goal proved.
|- (!(x. R x ==> Q x) /\ R a ==> Q a : proofs
```

Summary — Resolution

- The matching modus ponens rule:
 - MATCH_MP : thm -> thm -> thm
- The resolution tactics:
 - IMP_RES_TAC : thm -> tactic
 - RES_TAC : tactic
- Some related tools:
 - canonical form: RES_CANON
 - one-step resolution: IMP_RES_THEN
 - one-step assumption resolution: RES_THEN

See the manual for details.

An Example Tactic Proof

- The goal to prove is:

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

- Expand with the definition of \leq and strip:

```
- expand(PURE_ONCE_REWRITE_TAC [arithmeticTheory.LESS_OR_E
OK..      1 subgoal:
> val it =
  !m n. n < m \/\ (n = m) ==> (?p. m = n + p)
  -----
                               : proofs

- expand(REPEAT STRIP_TAC);
OK..
2 subgoals:
> val it =
  ?p. m = n + p
  -----
    n < m

  ?p. m = n + p
  -----
    n < m
  : proofs
```

Example Continued

- We can use the fact LESS_ADD_1:

```
- arithmeticTheory.LESS_ADD_1;
> val it = |- !m n. n<m ==> (?p. m = n + p + 1) : thm

- expand(IMP_RES_TAC arithmeticTheory.LESS_ADD_1);
OK..
1 subgoal:
> val it =
  ?p'. m = n + p
  -----
    n < m
    m = n + p + 1
  : proofs
```

- We now substitute for m:

```
- expand (FIRST_ASSUM SUBST1_TAC);
OK..
1 subgoal:
> val it =
  ?p'. n + p + 1 = n + p'
  -----
    n < m
    m = n + p + 1
  : proofs
```

Example Continued

- The proof of this subgoal is now easy:

```
- expand (EXISTS_TAC ('p + 1'));
OK..
1 subgoal:
> val it =
  n + p + 1 = n + p + 1
-----
  n < m
  m = n + p + 1
  : proofs

- expand REFL_TAC;
OK..
Goal proved.
.. |- n + p + 1 = n + p + 1
Goal proved.
.. |- ?p'. n + p + 1 = n + p'
Goal proved.
.. |- ?p. m = n + p
Goal proved.
. |- ?p. m = n + p

Remaining subgoals:
> val it =
  ?p. m = n + p
-----
  n = m
  : proofs
```

Example Continued

- The correct value is 0:

```
- expand (EXISTS_TAC ('0'));
OK..
1 subgoal:
> val it =
  m = n + 0
-----
  n = m      : proofs
```

- An important theorem helps now:

```
- val ADD_THMS = arithmeticTheory.ADD_CLAUSES;
> val ADD_THMS =
  |- (0 + m = m) /\
     (m + 0 = m) /\
     (SUC m + n = SUC (m + n)) /\
     (m + SUC n = SUC (m + n)) : thm

- expand (ASM_REWRITE_TAC [ADD_THMS]);
OK..
Goal proved. . |- m = n + 0
Goal proved. . |- ?p. m = n + p
Goal proved.  |- !m n. n < m \/\ (n = m) ==>
                (?p. m = n + p)

> val it =
  Original goal proved.
  |- !m n. n <= m ==> (?p. m = n + p) : proofs
```

Example Concluded

- The entire proof is:

```
- val thm =
  TAC_PROOF
  (([], ' !m n. (n<=m) ==> ?p. m=n+p',
  PURE_ONCE_REWRITE_TAC [arithmeticTheory.LESS_OR_EQ] T
  REPEAT STRIP_TAC THENL
  [IMP_RES_TAC arithmeticTheory.LESS_ADD_1 THEN
  FIRST_ASSUM SUBST1_TAC THEN
  EXISTS_TAC 'p + 1' THEN
  REFL_TAC,
  EXISTS_TAC '0' THEN
  ASM_REWRITE_TAC [arithmeticTheory.ADD_CLAUSES]]);
> val thm = |- !m n. n <= m ==> (?p. m = n + p) : thm
```