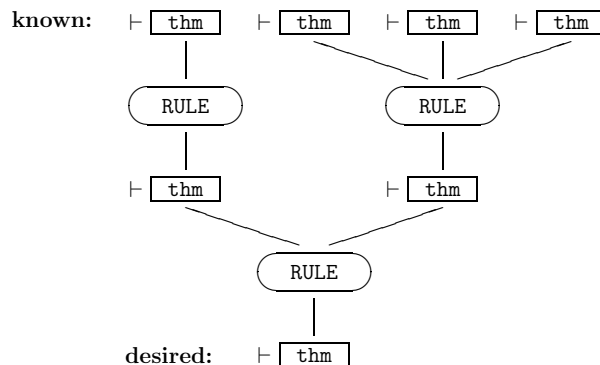


Forward Proof in HOL

Forward Proof in HOL

- The so-called *forward proof style*:

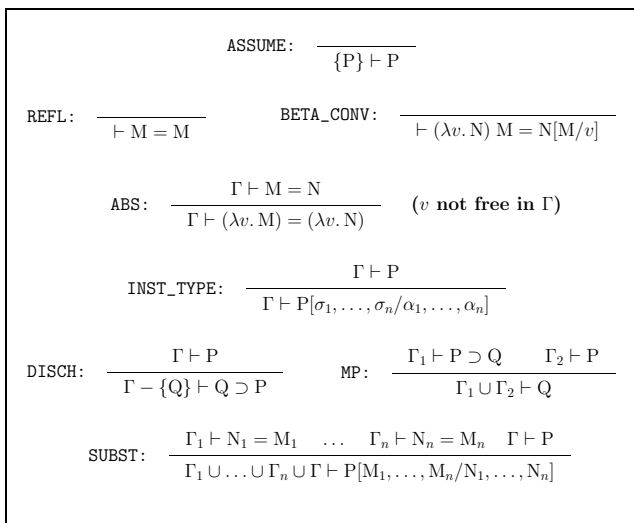


- Forward proofs:

- can be *millions* of (primitive) inferences long
- usually not natural for ‘one-off’ proofs
- but essential for tool building

Forward Proof and Primitive Rules

- All theorems in HOL are ultimately proved using only the primitive inference rules:



- We shall now look at the ML representations of these rules.

Introducing an Assumption

- In logic, the rule is:

$$\text{ASSUME: } \frac{}{\{P\} \vdash P}$$

- This rule allows us to deduce $\{P\} \vdash P$ for any boolean term P , so the HOL rule

$$\text{ASSUME : term} \rightarrow \text{thm}$$

takes this term as an argument.

Examples

- Examples using ASSUME:

```
- ASSUME;  
> val it = fn : term -> thm  
  
- ASSUME ‘‘Q ∧ R’’;  
> val it = . |- Q ∧ R : thm  
  
- show_assums := true;  
> val it = () : unit  
  
- ASSUME ‘‘Q ∧ R’’;  
> val it = [Q ∧ R] |- Q ∧ R : thm  
  
- ASSUME ‘‘n:num’’ handle e => Raise e;  
Exception raised at Thm.ASSUME:  
not a proposition
```

Reflexivity of Equality

- The rule is:

$$\text{REFL: } \frac{}{\vdash M = M}$$

- The HOL function takes a term argument:

```
- REFL ‘‘T’’;  
> val it = |- T = T : thm  
  
- REFL ‘‘x:num’’;  
> val it = |- x = x : thm
```

Discharging an Assumption

- The rule is:

$$\text{DISCH: } \frac{\Gamma \vdash P}{\Gamma - \{Q\} \vdash Q \supset P}$$

- Example:

```
- val th = ASSUME ‘‘F’’;  
> val th = . |- F : thm  
  
- DISCH;  
> val it = fn : term -> thm -> thm  
  
- DISCH ‘‘F’’ th;  
> val it = |- F ==> F : thm  
  
- DISCH ‘‘T’’ th;  
> val it = . |- T ==> F : thm  
  
- DISCH ‘‘x=3’’ (REFL ‘‘x=9’’);  
> val it = |- (x = 3) ==> ((x = 9) = x = 9) : thm
```

Modus Ponens

- The rule is:

$$\text{MP: } \frac{\Gamma_1 \vdash P \supset Q \quad \Gamma_2 \vdash P}{\Gamma_1 \cup \Gamma_2 \vdash Q}$$

- Implementation in HOL:

```
- MP;  
> val it = fn : thm -> thm -> thm  
  
- val th1 = DISCH ‘‘T=T’’ (REFL ‘‘F’’);  
> val th1 = |- (T = T) ==> (F = F) : thm  
  
- MP th1 (REFL ‘‘T’’);  
> val it = |- F = F : thm
```

- Modus ponens works up to α -equivalence:

```
- val th1 =  
  ASSUME ‘‘(!x:num. Q x) ==> ?x. Q x’’;  
> val th1 = . |- (!x. Q x) ==> (?x. Q x) : thm  
  
- val th2 = ASSUME ‘‘!y:num. Q y’’;  
> val th2 = . |- !y. Q y : thm  
  
- MP th1 th2;  
> val it = .. |- ?x. Q x : thm
```

The Abstraction Rule

- The rule is:

$$\text{ABS: } \frac{\Gamma \vdash M = N}{\Gamma \vdash (\lambda v. M) = (\lambda v. N)}$$

where the variable v is not free in Γ .

- An example:

```
- ABS;
> val it = fn : term -> thm -> thm

- val th = REFL 'x:num';
> val it = |- x = x : thm

- ABS 'x:num' th;
> val it = |- (\x. x) = (\x. x) : thm

- ABS 'y:a' it;
> val it = |- (\y x. x) = (\y x. x) : thm
```

Abstraction: Failure Conditions

- The term argument must be a variable:

```
- ABS 'x+y' (ASSUME '(x:a) = x')
  handle e => Raise e;
Exception raised at Thm.ABS:
first argument is not a variable
! Uncaught exception ....
```

- The theorem must be an equation:

```
- ABS 'x:a' (ASSUME 'T')
  handle e => Raise e;
Exception raised at Thm.ABS:
not an equality
! Uncaught exception ....
```

- The variable must not be free in the assumptions of the theorem:

```
- val th = ASSUME '(x:a) = y';
> val th = . |- x = y : thm

- ABS 'x:a' th
  handle e => Raise e;
Exception raised at Thm.ABS:
The variable is free in the assumptions
! Uncaught exception ....
```

Beta Conversion

- The β -conversion rule is:

$$\text{BETA_CONV: } \frac{}{\vdash (\lambda v. N) M = N[M/v]}$$

- In HOL, the function BETA_CONV takes the term $(\lambda v. N) M$ as an argument.

- Example:

```
- BETA_CONV;
> val it = fn : term -> thm

- BETA_CONV '(\x. x=3) 4';
> val it = |- (\x. x = 3) 4 = 4 = 3 : thm

- BETA_CONV '(\x:a. \y. x=y) y';
> val it = |- (\x y. x = y) y = (\y'. y = y') : thm

- BETA_CONV 'x=0' handle e => Raise e;
Exception raised at Thm.BETA_CONV:
not a beta-redex
```

Substitution

- The rule for substitution is SUBST:

$$\frac{\Gamma_1 \vdash N_1 = M_1 \quad \dots \quad \Gamma_n \vdash N_n = M_n \quad \Gamma \vdash P}{\Gamma_1 \cup \dots \cup \Gamma_n \cup \Gamma \vdash P[M_1, \dots, M_n/N_1, \dots, N_n]}$$

where

$$P[M_1, \dots, M_n/N_1, \dots, N_n]$$

means P with *designated* free occurrences of N_i replaced by M_i .

- The 'designating' of occurrences to replace turns out to be a bit complicated:

```
- SUBST;
> val it = fn : {redex: term, residue: thm} list
-> term -> thm -> thm
```

Substitution (continued)

- SUBST takes three arguments:

$$\frac{\text{(thm * term) list} \rightarrow \text{term} \rightarrow \text{thm} \rightarrow \text{thm}}{\Gamma_2 \vdash P}$$

a template with
marked subterms

a list of pairs of substitution equations and marker variables of the form $(\vdash N_1 = M_i, v)$

- A simple example:

```
- val th1 = ASSUME 'x=7';
> val th1 = . |- x = 7 : thm

- SUBST [{redex = 'm:num', residue = th1}
         'm+x = x+m' (REFL 'x+x');
> val it = . |- 7 + x = x + 7 : thm

- SUBST [{redex = 'm:num', residue = th1}
         'x+x = m+x' (REFL 'x+x');
> val it = . |- x + x = 7 + x : thm
```

More Substitution Examples

- Another example:

```
- val th1 = BETA_CONV '(\x. x) 1';
> val th1 = |- (\x. x) 1 = 1 : thm

- val th2 = REFL '(\x.x) 1';
> val th2 = |- (\x. x) 1 = (\x. x) 1 : thm

- val th3 = SUBST [{redex 'm:num', residue=th1}
                  'm = (\x.x) 1' th2];
> val th3 = |- 1 = (\x. x) 1 : thm

- SUBST [redex 'm:num', residue=th3]
- '1 = (\x.x)m' th3;
> val it = |- 1 = (\x. x) ((\x. x) 1) : thm
```

- SUBST allows simultaneous substitutions:

```
- val th1 = ASSUME 'x=7'
  and th2 = ASSUME 'y=x+10';
> val th1 = . |- x = 7 : thm
> val th2 = . |- y = x + 10 : thm

- val th3 = REFL 'x+y';
> val th3 = |- x + y = x + y : thm

- SUBST [{redex 'm1:num', residue= th1},
         {redex 'm2:num', residue= th2}]
         'm1+y = x+m2' th3;
> val it = .. |- 7 + y = x + x + 10 : thm
```

Type Instantiation

- The type instantiation rule is:

$$\text{INST_TYPE: } \frac{\Gamma \vdash P}{\Gamma \vdash P[\sigma_1, \dots, \sigma_n / \alpha_1, \dots, \alpha_n]}$$

- The first argument to INST_TYPE is a list of the type instantiations to be made:

```
- INST_TYPE;
> val it = fn:(hol_type*hol_type)list -> thm -> thm

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

- val th = REFL '(\x:'a. f(x) = (y:'b))';
> val th =
  |- (\(x:'a). (f:'a -> 'b) x = (y:'b)) =
    (\(x:'a). (f:'a -> 'b) x = (y:'b)) :thm

- INST_TYPE [(('num', 'a'),
              ('bool', 'b'))] th;
> val it =
  |- (\(x:num). (f:num -> bool) x = (y:bool)) =
    (\(x:num). (f:num -> bool) x = (y:bool)) :thm
```

Example of Forward Proof

- The proof of $\vdash \neg F$ we did before:

- 1) $\vdash (\lambda b. b \supset F) F = (F \supset F)$ BETA_CONV
- 2) $\vdash (\lambda b. b \supset F) F = (\lambda b. b \supset F) F$ REFL
- 3) $\vdash (F \supset F) = (\lambda b. b \supset F) F$ SUBST 1,2
- 4) $\{F\} \vdash F$ ASSUME
- 5) $\vdash F \supset F$ DISCH 4
- 6) $\vdash (\lambda b. b \supset F) F$ SUBST 3,5
- 7) $\vdash \neg = (\lambda b. b \supset F)$ DEFN of \neg
- 8) $\vdash \neg = \neg$ REFL
- 9) $\vdash (\lambda b. b \supset F) = \neg$ SUBST 7,8
- 10) $\vdash \neg F$ SUBST 6,9

- This proof uses only primitive inference rules.

The Proof in HOL

- Apply beta-conversion:

```
- val t1 = BETA_CONV ``(\b. b ==> F) F``;
> val t1 = |- (\b. b ==> F) F = F ==> F : thm
```

- Reverse the equation just derived:

```
- val t2 = REFL ``(\b. b ==> F) F``;
> val t2 = |- (\b. b ==> F) F = (\b. b ==> F) F : thm

- val t3 = SUBST [{redex ``m:bool``,residue=t1}]
              ``m = (\b. b ==> F) F`` t2;
> val t3 = |- F ==> F = (\b. b ==> F) F : thm
```

- Prove $\vdash F \implies F$ using ASSUME and DISCH:

```
- val t4 = ASSUME ``F``;
> val t4 = . |- F : thm

- val t5 = DISCH ``F`` t4;
> val t5 = |- F ==> F : thm
```

- Conclude as follows:

```
- val t6 = SUBST [{redex ``m:bool``,residue=t3}]
              ``m:bool`` t5;
> val t6 = |- (\b. b ==> F) F : thm
```

The Proof Concluded

- The theorem just derived:

```
- t6;
> val it = |- (\b. b ==> F) F : thm
```

- Reverse the defining equation for negation:

```
- val t7 = REFL ``$~``;
> val t7 = |- ~ = ~ : thm

- val t8 = SUBST [{redex = ``m:bool->bool``,
                  residue = NOT_DEF}]
              ``m = $~`` t7;
> val t8 = |- (\t. t ==> F) = ~ : thm
```

- Substitute with this equation:

```
- val th = SUBST
           [{redex ``m:bool->bool``,residue=t8}]
           ``m F:bool`` t6;
> val th = |- ~F : thm
```

Derived Inference Rules

- Derived inference rules are just ML programs.
- The rule we looked at before:

$$\text{SYM: } \frac{\Gamma \vdash M = N}{\Gamma \vdash N = M}$$

- 1) $\Gamma \vdash M = N$ HYPOTHESIS
- 2) $\vdash M = M$ REFL
- 3) $\Gamma \vdash N = M$ SUBST 1,2

And the ML code that implements it:

```
fun SYM th =
  let val (left,right) = dest_eq(concl th)
      val v = genvar(type_of left)
  in
    SUBST [{redex=v, residue=th}]
          (mk_eq(v,left)) (REFL left)
  end handle _ => failwith "SYM: not an equation";
```

- Let's look at how one develops this code...

Deriving the Rule in HOL

- Start by assuming a typical hypothesis:

```
- val th = ASSUME ``(M:'a) = N``;
> val th = . |- M = N : thm
```

- We need to know what 'M' and 'N' are:

```
- val (left,right) = dest_eq(concl th);
> val left = ``M`` : term
> val right = ``N`` : term
```

- We need a marker variable:

```
- val v = genvar(type_of left);
> val v = ``%genvar%360`` : term
```

- The result is then given by:

```
- SUBST [{redex=v, residue=th}]
> ``~v = ~left`` (REFL left);
> val it = . |- N = M : thm
```

The Derived Rule

- Since M and N were arbitrary, this derivation suggests the following code:

```
- fun SYM th =
  let val (left,right) = dest_eq(concl th)
      val v = genvar(type_of left)
  in
    SUBST [{redex=v, residue=th}]
      (mk_eq(v,left)) (REFL left)
  end handle _ => failwith "SYM: not an equation";
> val SYM = fn : thm -> thm
```

- Test the rule:

```
- F_DEF;
> val it = |- F = (!t. t) : thm

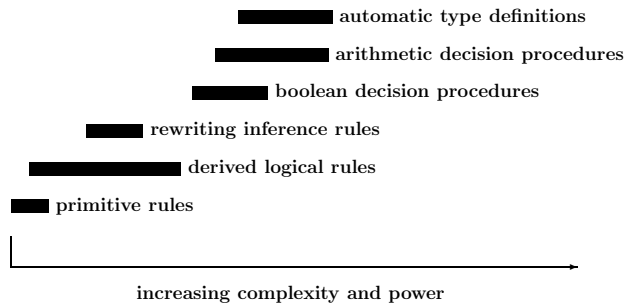
- SYM (F_DEF);
> val it = |- (!t. t) = F : thm

- val thm = ASSUME ``!b. b = T``;
> val thm = . |- !b. b = T : thm

- SYM thm handle e => Raise e;
Exception raised at ??.failwith:
SYM: not an equation
! Uncaught exception: ....
```

Built-in Derived Rules

- There is a wide range of derived inference rules built into the system:



- To become an expert HOL user, you should aim always to be learning new rules and proof techniques.

Example of Forward Proof

- We will use forward proof to derive

$$((\exists x. R x) \supset Q) = (\forall x. R x \supset Q)$$

by proving implication in both directions.

- Show hypotheses during the proof:

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

- To prove the left-to-right implication, first assume the antecedent:

```
- val th1 = ASSUME ``(?x:'a. R(x))=> Q``;
> val th1 = [(?x. R x) ==> Q] |- (?x. R x) ==> Q : thm
```

- Now, assume R x:

```
- val th2 = ASSUME ``R(x:'a):bool``;
> val th2 = R x |- R x
```

Example Continued

- We can now use the rule

$$\frac{\Gamma \vdash P[u/x]}{\Gamma \vdash \exists x. P}$$

to deduce $\exists x. R x$:

```
- th2;
> val it = [R x] |- R x : thm

- val th3 = EXISTS ``?x:'a. R(x)`` ,
                ``x:'a`` th2;
> val th3 = [R x] |- ?x. R x : thm
```

- Now, infer Q:

```
- th1;
> val it = [(?x. R x) ==> Q] |- (?x. R x) ==> Q : thm

- val th4 = MP th1 th3;
> val th4 = [(?x. R x) ==> Q, R x] |- Q : thm
```

- Discharge the assumption R x:

```
- val th5 = DISCH ``R(x:'a):bool`` th4;
> val th5 = [(?x. R x) ==> Q] |- R x ==> Q : thm
```

Example Continued

- Generalize x and discharge the assumption:

```
- val th6 = DISCH_ALL (GEN 'x:'a' th5);
> val th6 = |- ((?x. R x) ==> Q) ==>
  (!x. R x ==> Q) : thm
```

- To prove the reverse implication, again start by assuming the antecedent:

```
- val th7 = ASSUME '!(x:'a).R(x) ==> Q';
> val th7 = [!x. R x ==> Q] |- !x. R x ==> Q : thm
```

- Specialize to x :

```
- th7;
> val it = [!x. R x ==> Q] |- !x. R x ==> Q : thm

- val th8 = SPEC 'x:'a' th7;
> val th8 = [!x. R x ==> Q] |- R x ==> Q : thm
```

- Move $R\ x$ into the assumptions:

```
- val th9 = UNDISCH th8;
> val th9 = [!x. R x ==> Q, R x] |- Q : thm
```

Example Continued

- Now, assume $?x. R\ x$:

```
- val th10 = ASSUME '?x:'a. R(x)';
> val th10 = [?x. R x] |- ?x. R x : thm
```

- We can now use the rule

$$\frac{\Gamma_1 \vdash \exists x. R \quad \Gamma_2 \cup \{R[v/x]\} \vdash Q}{\Gamma_1 \cup \Gamma_2 \vdash Q} \text{ CHOOSE } ("v", (\Gamma_1 \vdash \exists x.R))$$

to conclude Q :

```
- th9;
> val it = [!x. R x ==> Q, R x] |- Q : thm

- val th11 = CHOOSE ('x:'a', th10) th9;
> val th11 = [?x. R x, !x. R x ==> Q] |- Q : thm
```

- Discharge the assumption $?x. R\ x$:

```
- val th12 = DISCH '?x:'a. R(x)' th11;
> val th12 = [!x. R x ==> Q] |- (?x. R x) ==> Q : thm
```

The Example Concluded

- The results so far:

```
- th6;
> val it = |- ((?x. R x) ==> Q) ==>
  (!x. R x ==> Q) : thm

- th12;
> val it = [!x. R x ==> Q] |- (?x. R x) ==> Q : thm
```

- Discharge the remaining assumption:

```
- val th13 = DISCH_ALL th12;
> val th13 = |- (!x. R x ==> Q) ==>
  (?x. R x) ==> Q : thm
```

- Put the two implications together:

```
- val thm = IMP_ANTISYM_RULE th6 th13;
> val thm = |- (?x. R x) ==> Q = (!x. R x ==> Q) : thm
```

- Q.E.D.

Summary — Forward Proof

- Primitive Inference Rules:

ASSUME, REFL, DISCH, MP, ABS,
BETA_CONV, SUBST, INST_TYPE.

- Derived Inference Rules:

SYM, EXISTS, DISCH_ALL, SPEC, GEN,
UNDISCH, CHOOSE, IMP_ANTISYM_RULE.

There are many more built-in derived rules.

Other Rewriting Rules

- Rewriting using the assumptions:

ASM_REWRITE_RULE
ASM_PURE_REWRITE_RULE

These add the assumptions of the theorem to be rewritten to the list of rewrites.

- Rewriting only once:

ONCE_REWRITE_RULE
ONCE_ASM_REWRITE_RULE
PURE_ONCE_REWRITE_RULE
PURE_ONCE_ASM_REWRITE_RULE

These apply at most one rewrite rule to any subterm.

Summary of Rewriting

- Rewrite using standard simplifications:

- REWRITE_RULE
- ASM_REWRITE_RULE
- ONCE_REWRITE_RULE
- ONCE_ASM_REWRITE_RULE

- Rewrite without standard simplifications:

- PURE_REWRITE_RULE
- PURE_ASM_REWRITE_RULE
- PURE_ONCE_REWRITE_RULE
- PURE_ONCE_ASM_REWRITE_RULE

- Rewrite using assumptions:

- ASM_REWRITE_RULE
- PURE_ASM_REWRITE_RULE
- ONCE_ASM_REWRITE_RULE
- PURE_ONCE_ASM_REWRITE_RULE