1 Conversions

Conversions are functions that construct equality theorems from terms. Given a term t, a conversion will try to construct a theorem of form $\vdash t = u$.

- `BETA_CONV`

  If t has form $(\lambda x\ A)u$, then the result has form $t = A[x := u]$.

Conversionals are functions that modify conversions.

- `ONCE_DEPTH_CONV c`

  If t has a subterm u on which c constructs a theorem $\vdash u = v$, then (ONCE_DEPTH_CONV c) constructs the theorem $\vdash t[u] = t[v]$. In case, more u are possible, top-down order decides which one is chosen.

- ```
  DEPTH_CONV c           // Use c for a one time rewriting sweep.
  TOP_DEPTH_CONV c       // Rewrite as long as possible using c.
  REDEPTH_CONV c         // Rewrite as long as possible using c.
  ```

2 Rules

Rules are functions that construct theorems. In HOL light, constructing a theorem is the same as proving it.

- `ASSUME t`

  Constructs the theorem $t \vdash t$.

- `CONJ thm1 thm2`

  If **thm1** has form $\Gamma_1 \vdash A$, and **thm2** has form $\Gamma_2 \vdash B$, then the result has form $\Gamma_1, \Gamma_2 \vdash A \wedge B$.

- `CONJUNCT1 thm`

  If **thm** has form $\Gamma \vdash A \wedge B$, then the result is the theorem $\Gamma \vdash A$.

- `CONJUNCT2 thm`

  If **thm** has form $\Gamma \vdash A \wedge B$, then the result is the theorem $\Gamma \vdash B$.
  ```

- `MP thm1 thm2`

  If **thm1** has form $\Gamma_1 \vdash A \to B$, and **thm2** has form $\Gamma_2 \vdash A$, then the result equals $\Gamma_1, \Gamma_2 \vdash B$.

- `CONTR t thm`

  Prove $\Gamma \vdash t$ from $\Gamma \vdash F$. (F is the false constant)

- `DISJ1 thm term`

  If **thm** has form $\Gamma \vdash A$ and **term** has form B, then the result is the theorem $\Gamma \vdash A \vee B$.

- `DISJ2 thm term`

  If **thm** has form $\Gamma \vdash B$ and **term** has form A, then the result is the theorem $\Gamma \vdash A \vee B$.

- `DISJ_CASES thm1 thm2 thm2`

  If **thm1** has form $\Gamma_1 \vdash A \vee B$, **thm2** has form $\Gamma_2, A \vdash C$ and **thm3** has form $\Gamma_3, B \vdash C$, then the result has form $\Gamma_1, \Gamma_2, \Gamma_3 \vdash C$.

- `EXISTS t1 t2 thm`

  Introduces an existential quantifier from a witness. $\mathbf{t_1}$ must have form $\exists x\ P(x)$, and **thm** must have form $\Gamma \vdash P[x := \mathbf{t_2}]$.

- `CHOOSE x,thm1 thm2`

  Should be applied on a pair consisting of a term and a theorem, and a second theorem. **thm1** must have form $\Gamma_1 \vdash \exists x\ P(x)$, x must be an eigenvariable, and **thm2** must have form $\Gamma_2, P(x) \vdash A$. The result will be the theorem

  $$\Gamma_1, \Gamma_2 \vdash A.$$

- `GEN x thm`

  If **thm** has form $\Gamma \vdash A$, and x is a variable that is not free in Γ, then the result equals $\Gamma \vdash \forall x\ A$.

- `DISCH term t`

  If **thm** has form $\Gamma \vdash A$, then the result has form $\Gamma \backslash \{t\} \vdash t \to A$.

- `INST_TYPE [ v1,t1; ..., vn,tn ] thm`

  Instantiate type variables in **thm** by the substitution $\{v_1 := t_1, \ldots, v_n := t_n\}$. Type variables are in principle not visible, but you can get them by decomposing the sequent, and use type_of.

- `INST [ v1,t1, . . ., vn, tn ] thm`

  Instantiate the variables in **thm** by the substitution $\{v_1 := t_1, \ldots, v_n := t_n\}$.

- `ISPEC t thm`

  If **thm** has form $\Gamma \vdash \forall x\ A$, then the result has form $\Gamma \vdash A[x := t]$. If necessary, ISPEC also instantiates type variables.

- `ITAUT f`

  Try to prove f automatically in propositional logic and construct the theorem $\vdash f$ it is succeeds.

- `NOT_ELIM thm`

  Replace $\neg A$ by $A \rightarrow \bot$ in conclusion of theorem.

- `NOT_INTRO thm`

  Replace $A \rightarrow \bot$ by $\neg A$ in conclusion of theorem.

- `CONV_RULE c thm`

  Remember that a conversion is a function that makes an equality theorem $\vdash t = u$ from a term t. CONV_RULE c applies c on the conclusion A of **thm** and if it succeeds, it uses the equality to replace A by A'. The result is a new theorem $\Gamma \vdash A'$.

- `GSYM thm`

  If **thm** has form $\Gamma \vdash \forall \overline{x}\ t_1 = t_2$, then the result has form $\Gamma \vdash \forall \overline{x}\ t_2 = t_1$.

3 The Goal Editor

- The proof editor is entered by typing

  ```
  g ‘formula‘ ;;
  ```

Do not forget to use back quotes.

- `b( );;`

  Backtrack.

- `e( T );;`

  Expand the goal, using tactic T. A list of useful tactics is given below.

- `p( );;`

  Prints the main subgoal.

- `r(I);;`

  Rotates the subgoals by I. This rule proves nothing, but it is useful if you want to see which subgoals there are, or you want to work on another goal. I can be positive or negative.

- If you managed to prove all subgoals, then the original goal has become a theorem. It is called

  ```
  top_thm( )
  ```

  You can save it in a variable by typing

  ```
  let v = top_thm( );;
  ```

4 Tactics

Tactics are the operations that you can type in the E-command of the proof editor. A tactic constists of two parts: A function f_1 that transforms a goal $\Gamma \vdash A$ into a new set of goals

$$\Gamma_1 \vdash A_1, \ldots, \Gamma_n \vdash A_n,$$

and a function f_2 of type $\mathbf{thm}^i \rightarrow \text{thm}$ that constructs $\Gamma \vdash A$ from $\Gamma_1 \vdash A_1, \ldots, \Gamma \vdash A_n$. When you apply the tactic, the editor uses f_1 to construct a list of new subgoals. If you manage to prove all the subgoals, the editor will use f_2 to construct the original goal.

- `STRIP_TAC`

  Simplifies the conclusion and the assumptions of the goal in standard (but somewhat unpredictable) way, looking at its main operator. The rule also splits (for example, when a premisse has $\vee$ as main operator) If you want to simplify a goal completely, use `(REPEAT STRIP_TAC)`. This will repeat `STRIP_TAC` until it fails.

- `REWRITE_TAC [ th1 ; th2 ... ; thn ]`

  Rewrite goal, using the theorems th1 $\cdots$ thn, which must have the forms of rewrite rules. As a general rule, you should direct all equalities in such a way that they can be used as rewrite rules, because rewriting is very useful.

- `ASM_REWRITE_TAC [ th1 ; th2; ...; thn ]`

  Same as REWRITE_TAC but now also assumptions that look like rewrite rules can be used. Note that an assumption a that does not look like a rewrite rule, is replaced by $a \Rightarrow \top$, so that it can still simply the goal. I think that negative assumptions $\neg a$ are replaced by the rule $a \Rightarrow \bot$.

- `EQ_TAC`

  Replaces a $\Leftrightarrow$ in the goal by two implications.

- `DISCH_TAC`

  If the goal has form $A \rightarrow B$, then A is added to the assumptions and B becomes the new goal. If the goal has form $\neg A$ then A is added to the assumptions, and $\bot$ becomes the new goal.

- `ASM_CASES_TAC term`

  Do a case split on term. The result consists of two subgoals. In the first, it is assumed that term = true. In the second, it is assumed that term = false.

- `ASSUME_TAC thm`

  If the goal has form $\Gamma \vdash A$ and **thm** has form $\Gamma' \vdash B$ with $\Gamma' \subseteq \Gamma$, then the new goal will be $\Gamma, B \vdash A$.

- `MP_TAC thm`

  If the goal has form $\Gamma \vdash A$ and and **thm** has form $\Gamma' \vdash B$ with $\Gamma' \subseteq \Gamma$, then the new goal will be $\Gamma \vdash B \rightarrow A$.

- `DISJ_CASES_TAC thm`

  If the goal has form $\Gamma \vdash C$ and **thm** has form $\Gamma' \vdash A \lor B$, with $\Gamma' \subseteq \Gamma$, then the goal will be split into two goals

  $$\Gamma, A \vdash C \text{ and } \Gamma, B \vdash C.$$

- `EXISTS_TAC t`

  If the goal has form $\Gamma \vdash \exists x\ P(x)$, then it becomes $\Gamma \vdash P(t)$. If you see mysterious failures, then the reason could be that you have to provide type information in term t.

- `CHOOSE_TAC thm`

  If the goal has form $\Gamma \vdash A$ and **thm** has form $\Gamma' \vdash \exists x\ P(x)$, with $\Gamma' \subseteq \Gamma$, then the new goal will be $\Gamma, P(x) \vdash A$, where x is an eigenvariable.

- `GEN_TAC`
  `X_GEN_TAC`

  GEN_TAC allows to prove a formula of form $\forall x\ P(x)$ by proving $P(y)$ for an eigenvariable y. X_GEN_TAC does the same, but it allows you to specify the eigenvariable.

- `DISJ1_TAC`

  If the goal has form $\Gamma \vdash A \lor B$, then the new goal will have form $\Gamma \vdash A$.

- `DISJ2_TAC`

  If the goal has form $\Gamma \vdash A \lor B$, then the new goal will have form $\Gamma \vdash B$.

- `CONJ_TAC`

  If the goal has form $\Gamma \vdash A \land B$, then create two new goals

  $$\Gamma \vdash A \text{ and } \Gamma \vdash B.$$

- `CONTR_TAC`

  Replace the goal $\Gamma \vdash A$ by $\Gamma \vdash F$.

- `ANTS_TAC`

  Replaces a goal of form $\Gamma \vdash (A \to B) \to C$ by two goals

  $$\Gamma \vdash A \text{ and } \Gamma \vdash B \to C.$$

- `CONV_TAC conv`

  conv is a conversion, i.e. a function, s.t. **conv**t returns a theorem $\vdash t = t'$. When applied on a goal of form $\Gamma \vdash t$, the new goal will be $\Gamma \vdash t'$.

  See the section on conversions for a list of conversions.

- `ITAUT_TAC`

  Try to prove goal using ITAUT.

- `MATCH_MP_TAC thm`

  If the goal has form $\Gamma \vdash B$, and **thm** has form

  $$\Gamma' \vdash \forall x_1 \cdots x_n \; A'[x_1, \ldots, x_n] \rightarrow B'[x_1, \ldots, x_n],$$

  s.t. $\Gamma' \subseteq \Gamma$ and $B'[x_1, \ldots, x_n]$ can be matched into B with substitution $\{x_1 := t_1, \ldots, x_n := t_n\}$, then the new goal will be $\Gamma \vdash A'[t_1, \ldots, t_n]$.

- `ASM_MESON_TAC [ thm1, ..., thmn ]`

  Try to prove the goal automatically using a built-in theorem prover MESON, using the theorems $\mathbf{thm_1}, \ldots, \mathbf{thmn}$ as premisses. If you type `MESON_TAC` without `ASM`, the assumptions of the goal will not be used.

- `ASM_REWRITE_TAC [ thm1, ..., thmn ]`

  Rewrite conclusion of goal using equalities in the assumptions, the equalities **thm1**, …, **thmn**, and some additional set of equalities called **basic_rewrites**.

- `SPEC_TAC t,v`

  Replaces in the conclusion of the goal, every occurrence of t by variable v and universally quantifies by $\forall v$.

- `( REPEAT t )`

  Repeats tactic t until it fails.

- `t1 THEN t2`

  First apply t_1, and after that t_2 on all subgoals generated by t_2.

5 Other Useful Functions

- `type_of t`

  Returns the type of t.

- `concl thm`

  Returns the conclusion of a theorem as term.

Looking into the structure of a term:

It is sometimes necessary to look at the type variables, and it can make your scripts more robust if you can reuse parts of the formulas.

- `lhs t`

  If t is an equality $t_1 = t_2$, then the result is t_1.

- `rhs t`

  If t is an equality $t_1 = t_2$, then the result is t_2.

- `rator t`

  If t is an application term $a \cdot b$, then the result equals a.

- `rand t`

  If t is an application term $a \cdot b$, then the result equals b.

- `bndvar t`

  If t is an abstraction term $\lambda x\ u$, then the result equals x.

- `body t`

  If t is an abstraction term $\lambda x\ u$, then the result equals u.

- `top_goal( ).`

  Returns a pair containing of the list of premisses and the conclusion of the current goal.

Building Terms

- ```
  mk_comb t1,t2     // Note that this is a pair, not two arguments!
  mk_icomb t1, t2
  ```

  Builds the application $(t_1\ t_2)$. mk_icomb instantiates type variables in t_1, so it is a bit stronger.

- ```
  mk_abs v,t       // This is a pair.
  ```

  Builds $\lambda v\ t$.

- ```
  mk_binder s v,t
  ```

  Builds the term $(s\ (\lambda v\ t))$. s must be a string (with double quotes). You probably think that the same term can also be built using mk_abs and mk_comb, but problems with type variables seem to make it impossible to do this.

- ```
  mk_pair t1,t2
  ```

  Makes a pair from a pair. This may seem useless, but the argument is an OCAML pair, and the result is a HOL pair.

- ```
  mk_eq t1,t2
  ```

  Makes equality $t_1 = t_2$. If you try to use mk_comb, you will run into type problems, but you can use mk_icomb.

Definitions:
  ```