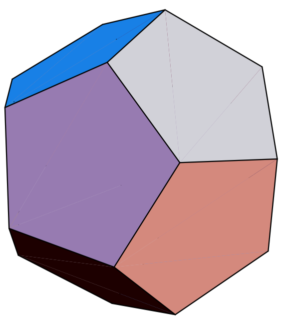
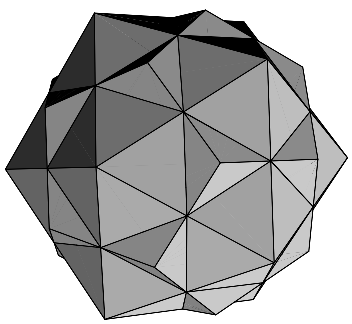
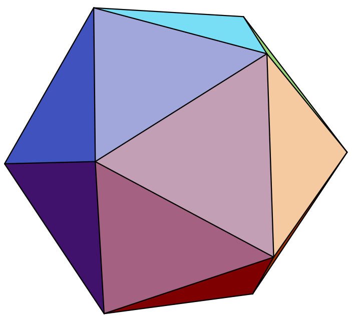
Summary of HOL System Version 2

Wai Wong

Contents

The motivation of compiling this summary was my incapability of memorizing many rules, tactics and theorems in the system. I found myself very often flip through the thick *reference manual* just to find a suitable tactic or theorem while doing proofs. It is very inconvenient since the manual is very bulky. So I decided to make a brief summary or even just a list of all useful ML functions or names. The result is this summary.

This summary is divided into five sections: inference rules, conversions, tactics and tacticals, built-in theorems, and libraries.

In the first three sections, the names of the ML functions implementing the rules, conversions and tactics are listed along with their ML types and brief syntax. These are followed by the inference rules implemented by the rule functions, the theorems returned by the conversion functions and the tactics. Brief notes are given where appropriate. The material for these three sections was collected from the help files. This is converted into LaTeX format with the aid of emacs and other standard system utilities.

The built-in theorems section lists all theorems available when the system is started up. The last section lists several libraries in the format similar to the output of `print_theory`. The generation of the LaTeX source for these two sections is automatic. I created a library `latex-hol` which contains functions for converting HOL terms, theorems and theories into LaTeX source. It uses the new pretty printer in the library `prettyp`. The convertion rules are based on the pretty printing rules for HOL developed by Richard Boulton who also developed the `prettyp` library. They were modifid to output LaTeX commands. Only slight editing is required to correct bad line breaks in the resulting LaTeX files. A new pretty printer for the `sets` library was written by myself. It outputs the sets in the same format as in conventional mathematic.

I hope this summary will be very useful to all of you, I welcome any comments and suggestions.

1st June 1991

W.Wong
University of Warwick
`ww@uk.ac.warwick.eng`

Upgraded to HOL version 2 on 17 Sept. 1991

1 HOL Inference Rules

- `ABS "x" :(term -> thm -> thm)`

$$\frac{A \vdash t_1 = t_2}{A \vdash (\lambda x.t_1) = (\lambda x.t_2)}$$

where x is not free in A

- `ADD_ASSUM "t" :(term -> thm -> thm)`

$$\frac{A \vdash t'}{A, t \vdash t'}$$

- `ALPHA "t1" "t2" :(term -> term -> thm)`

$$\frac{}{\vdash t_1 = t_2}$$

t_1 and t_2 are α-convertable

- `AP_TERM "t" :(term -> thm -> thm)`

$$\frac{u = v}{t\,u = t\,v}$$

- `AP_THM (|- u = v) "t" :(thm -> term -> thm)`

$$\frac{u = v}{u\,t = v\,t}$$

- `ASM_REWRITE_RULE [th1;...thn] :(thm list -> thm -> thm)`

$$\frac{A \vdash t}{A \vdash t'}$$

where t' is obtained by rewriting t with its assumptions, the theoroms $th_1; \ldots; th_n$ and the theorems bound to `basic_rewrites`

- `BETA_RULE :(thm -> thm)`

$$\frac{A \vdash (\lambda x.t[x])y}{A \vdash t[y/x]}$$

- `BODY_CONJUNCTS :(thm -> thm list)`

$$\frac{A \vdash \forall x.(t_1 \wedge \cdots) \wedge \cdots (\forall y. \cdots \wedge t_n)}{A \vdash t_1 \quad \cdots \quad A \vdash t_n}$$

flattens out conjuncts even in bodies of forall

- `CCONTR "t" :(term -> thm -> thm)`

$$\frac{A, \neg t \vdash F}{A \vdash t}$$

classical contradiction rule

- `CHOOSE ("v", |- ?x.t[x]) (|- t')`
  `:((term # thm) -> thm -> thm)`

$$\frac{A_1 \vdash \exists x.t[x] \quad A_2, t[v] \vdash t'}{A_1 \bigcup A_2 \vdash t'}$$

where the variable `v` is not free anywhere except in the assumption $t[v]$ of t'

- `CONJ :(thm -> thm -> thm)`

$$\frac{A_1 \vdash t_1 \quad A_2 \vdash t_2}{A_1 \bigcup A_2 \vdash t_1 \wedge t_2}$$

- `CONJUNCT1 :(thm -> thm)`

$$\frac{A \vdash t_1 \wedge t_2}{A \vdash t_1}$$

- `CONJUNCT2 :(thm -> thm)`

$$\frac{A \vdash t_1 \wedge t_2}{A \vdash t_2}$$

- `CONJUNCTS :(thm -> thm list)`

$$\frac{A \vdash t_1 \wedge \cdots \wedge t_n}{A \vdash t_1 \quad \cdots \quad A \vdash t_n}$$

where $n > 0$

- `CONJ_DISCH "t" :(term -> thm -> thm)`

$$\frac{A, t \vdash t_1 = t_2}{A \vdash (t \wedge t_1) = (t \wedge t_2)}$$

- `CONJ_DISCHL ["t1";..."tn"] :(term list -> thm -> thm)`

$$\frac{A, t_1, \ldots, t_n \vdash t = u}{A \vdash (t_1 \wedge \cdots \wedge t_n \wedge t) = (t_1 \wedge \cdots \wedge t_n \wedge u)}$$

- `CONJ_LIST n :(int -> thm -> thm list)`

$$\frac{A \vdash t_1 \wedge \ldots \wedge t_n}{A \vdash t_1 \quad \cdots \quad A \vdash t_n}$$

where $n > 0$

- `CONJ_PAIR :(thm -> (thm # thm))`

$$\frac{A \vdash t_1 \wedge t_2}{A| - t_1 \quad A| - t_2}$$

- `CONTR "t" :(term -> thm -> thm)`

$$\frac{A \vdash F}{A \vdash t}$$

intuitionistic contradiction rule

- `CONTRAPOS :(thm -> thm)`

$$\frac{A \vdash t_1 \supset t_2}{A \vdash \neg t_2 \supset \neg t_1}$$

- `DISCH "t1" :(term -> thm -> thm)`

$$\frac{A, t_1 \vdash t_2}{A \vdash t_1 \supset t_2}$$

- `DISCH_ALL :(thm -> thm)`

$$\frac{A, t_1, \ldots, t_n \vdash t}{A \vdash t_1 \supset \cdots \supset t_n \supset t}$$

- `DISJ1 (A |- t1) "t2" :(thm -> conv)`

$$\frac{A \vdash t_1}{A \vdash t_1 \vee t_2}$$

- `DISJ2 "t1" (A |- t2) :(term -> thm -> thm)`

$$\frac{A \vdash t_2}{A \vdash t_1 \vee t_2}$$

- `DISJ_CASES :(thm -> thm -> thm -> thm)`

$$\frac{A \vdash t_1 \vee t_2 \quad A_1, t_1 \vdash t \quad A_2, t_2 \vdash t}{A \bigcup A_1 \bigcup A_2 \vdash t}$$

- `DISJ_CASES_UNION :(thm -> thm -> thm -> thm)`

$$\frac{A \vdash t_1 \vee t_2 \quad A_1, t_1 \vdash t_3 \quad A_2, t_2 \vdash t_4}{A \bigcup A_1 \bigcup A_2 \vdash t_3 \vee t_4}$$

- `DISJ_IMP :(thm -> thm)`

$$\frac{A \vdash t_1 \vee t_2}{A \vdash \neg t_1 \supset t_2}$$

- `EQF_ELIM :(thm -> thm)`

$$\frac{A \vdash t = F}{A \vdash \neg t}$$

- `EQF_INTRO :(thm -> thm)`

$$\frac{A \vdash \neg t}{A \vdash t = F}$$

- `EQT_ELIM :(thm -> thm)`

$$\frac{A \vdash t = T}{A \vdash t}$$

- `EQT_INTRO :(thm -> thm)`

$$\frac{A \vdash t}{A \vdash t = T}$$

- `EQ_IMP_RULE :(thm -> (thm # thm))`

$$\frac{A \vdash t_1 = t_2}{A \vdash t_1 \supset t_2 \quad A \vdash t_2 \supset t_1}$$

- `EQ_MP :(thm -> thm -> thm)`

$$\frac{A_1 \vdash t_1 = t_2 \quad A_2 \vdash t_1}{A_1 \bigcup A_2 \vdash t_2}$$

- `EXISTENCE :(thm -> thm)`

$$\frac{A \vdash \exists\forall x.P[x]}{A \vdash \exists x.P[x]}$$

- `EXISTS ("?x.t[x]", "t'") :((term # term) -> thm -> thm)`

$$\frac{A \vdash t[t']}{A \vdash \exists x.t[x]}$$

- `EXISTS_EQ "x" :(term -> thm -> thm)`

$$\frac{A \vdash t_1 = t_2}{A \vdash (\exists x.t_1) = (\exists x.t_2)}$$

- `EXISTS_IMP "x" :(term -> thm -> thm)`

$$\frac{A \vdash P \supset Q}{A \vdash (\exists x.P) \supset (\exists x.Q)}$$

- `EXT :(thm -> thm)`

$$\frac{A \vdash \forall x.t_1 x = t_2 x}{A \vdash t_1 = t_2}$$

  where x is not free in A, t_1 or t_2

- `FORALL_EQ "x" :(term -> thm -> thm)`

$$\frac{A \vdash t_1 = t_2}{A \vdash (\forall x.t_1) = (\forall x.t_2)}$$

- `GEN "x" :(term -> thm -> thm)`

$$\frac{A \vdash t[x]}{A \vdash \forall x.t[x]}$$

  where x is not free in A

- `GENL ["x1";..."xn"] :(term list -> thm -> thm)`

$$\frac{A \vdash t}{A \vdash \forall x_1 \ldots x_n.t}$$

  where x_i for $i = 1, \ldots, n$ are not free in A

- `GEN_ALL :(thm -> thm)`

$$\frac{A \vdash t[x_1, \ldots, x_n]}{A \vdash \forall x_1 \ \ldots \ x_n.t[x_1, \ldots, x_n]}$$

- `GEN_REWRITE_RULE`
  `:((conv -> conv) -> thm list -> thm list -> thm -> thm)`

  This is the main rewriting rule which is called by other rewriteing rules

- `GSPEC  :(thm -> thm)`

$$\frac{\vdash \forall x_1 \ldots x_n . t}{\vdash t[g_1/x_1]\cdots[g_n/x_n]}$$

  where g_i are variables generated automatically by the system

- `HALF_MK_ABS  :(thm -> thm)`

$$\frac{A \vdash \forall x . u\, x = t}{A \vdash u = \lambda x . t}$$

- `IMP_ANTISYM_RULE  :(thm -> thm -> thm)`

$$\frac{A_1 \vdash t_1 \supset t_2 \quad A_2 \vdash t_2 \supset t_1}{A_1 \bigcup A_2 \vdash t_1 = t_2}$$

- `IMP_CANON  :(thm -> thm list)`

$$\frac{t_1 \wedge t_2}{t_1, t_2}$$

$$\frac{(t_1 \wedge t_2) \supset t}{t_1 \supset (t_2 \supset t)}$$

$$\frac{(t_1 \vee t_2) \supset t}{t_1 \supset t,\; t_2 \supset t}$$

$$\frac{(\exists x . t_1) \supset t_2}{t_1[x'/x] \supset t_2}$$

$$\frac{\forall x . t_1}{t_1[x'/x]}$$

  the above rules are used to put theorem into canonical form (horn clauses)

- `IMP_CONJ  :(thm -> thm -> thm)`

$$\frac{A_1 \vdash P \supset Q \quad A_2 \vdash R \supset S}{A_1 \bigcup A_2 \vdash P \wedge R \supset Q \wedge S}$$

- `IMP_ELIM  :(thm -> thm)`

$$\frac{A \vdash t_1 \supset t_2}{A \vdash \neg t_1 \vee t_2}$$

- `IMP_TRANS  :(thm -> thm -> thm)`

$$\frac{A_1 \vdash t_1 \supset t_2 \quad A_2 \vdash t_2 \supset t3}{A_1 \bigcup A_2 \vdash t_1 \supset t3}$$

- `INDUCT  :((thm # thm) -> thm)`

$$\frac{A_1 \vdash t[0] \quad A_2 \vdash \forall n . t[n] \supset t[\mathsf{SUC}\, n]}{A_1 \bigcup A_2 \vdash \forall n . t[n]}$$

- `INST  [("ti","xi")] :((term # term) list -> thm -> thm)`

$$\frac{A \vdash t[x_i]}{A \vdash t[t_i]}$$

- `INST_TYPE  [(":tyi",":vtyi")]`
  `:((type # type) list -> thm -> thm)`

$$\frac{A \vdash t}{A \vdash t[ty_1, \ldots, ty_n / vty_1, \ldots, vty_n]}$$

  where type variables vty_i not in A

- `INST_TY_TERM ([("ti","xi")],[(":tyi",":vtyi")])`
  `:(((term # term) list # (type # type) list) -> thm -> thm)`

$$\dfrac{\dfrac{A \vdash t[x_1,\ldots,x_m]}{A \vdash t[ty_1,\ldots,ty_n/vty_1,\ldots,vty_n]}}{A \vdash t[t_1,\ldots,t_m]}$$

  combination of `INST` and `INST_TYPE`

- `ISPEC "t':ty'" :(term -> thm -> thm)`

$$\frac{A \vdash \forall x{:}ty.\,t}{A \vdash t[t'/x]}$$

- `ISPECL ["ti:tyi"] :(term list -> thm -> thm)`

$$\frac{A \vdash \forall x_1 \ldots x_n.t}{A \vdash t[t_1/x_1,\ldots,t_n/x_n]}$$

  where t_i is free for x_i in t

- `LIST_CONJ :(thm list) -> thm`

$$\frac{A_1 \vdash t_1 \quad A_2 \vdash t_2 \quad \cdots \quad A_n \vdash t_n}{A_1 \bigcup A_2 \bigcup \cdots \bigcup A_n \vdash t_1 \wedge t_2 \wedge \cdots \wedge t_n}$$

- `LIST_INDUCT :((thm # thm) -> thm)`

$$\frac{A_1 \vdash t[[\,]] \quad A_2 \vdash \forall l.t[l] \supset \forall h.t[\mathsf{CONS}\,h\,l]}{A_1 \bigcup A_2 \vdash \forall l.t[l]}$$

- `LIST_MK_EXISTS ["xi"] :(term list -> thm -> thm)`

$$\frac{A \vdash t_1 = t_2}{A \vdash (\exists x_1 \ldots x_n.t_1) = (\exists x_1 \ldots x_n.t_2)}$$

  where x_i are not free in A

- `LIST_MP :(thm list -> thm -> thm)`

$$\frac{\vdash t_1 \quad \cdots \quad \vdash t_n \quad \vdash t_1 \supset \cdots \supset t_n \supset t}{\vdash t}$$

- `MATCH_MP (|- !x1 ... xn. t ==> u) (|-t')`
  `:(thm -> thm -> thm)`

$$\frac{\vdash \forall x_1 \ldots x_n.t \supset u \quad \vdash t'}{u'}$$

  mathces t' with t

- `MK_ABS :(thm -> thm)`

$$\frac{A \vdash \forall x.(t_1 = t_2)}{A \vdash (\lambda x.t_1) = (\lambda x.t_2)}$$

- `MK_COMB :((thm # thm) -> thm)`

$$\frac{A_1 \vdash f = g \quad A_2 \vdash x = y}{A_1 \bigcup A_2 \vdash fx = gy}$$

- `MK_EXISTS :(thm -> thm)`

$$\frac{A \vdash (\forall x.t_1 = t_2)}{A \vdash (\exists x.t_1) = (\exists x.t_2)}$$

- `MP :(thm -> thm -> thm)`

$$\frac{A_1 \vdash t_1 \supset t_2 \quad A_2 \vdash t_1}{A_1 \bigcup A_2 \vdash t_2}$$

- `NEG_DISCH "t" :(term -> thm -> thm)`

$$\frac{A, t \vdash F}{A \vdash \neg t}$$

- `NOT_ELIM :(thm -> thm)`

$$\frac{A \vdash \neg t}{A \vdash t \supset F}$$

- `NOT_EQ_SYM :(thm -> thm)`

$$\frac{A \vdash \neg(t_1 = t_2)}{A \vdash \neg(t_2 = t_1)}$$

- `NOT_INTRO :(thm -> thm)`

$$\frac{A \vdash t \supset F}{A \vdash \neg t}$$

- `ONCE_ASM_REWRITE_RULE [th1;...thn]`
  `:(thm list -> thm -> thm)`

$$\frac{A \vdash t}{A \vdash t'}$$

  where t' is obtained from t by rewriting *once* with the assumptions in A, $th_1, \ldots, th_n$ and the theorems bound to `basic_rewrites`

- `ONCE_REWRITE_RULE [th1,...thn] :(thm list -> thm -> thm)`

$$\frac{A \vdash t}{A \vdash t'}$$

  where t' is obtained from t by rewriting *once* with $th_1, \ldots, th_n$ and the theorems bound to `basic_rewrites`

- `PART_MATCH f (|-!x1 ... xn. t==>u)`
  `:((term -> term) -> thm -> conv)`

  called by `MATCH_MP`. `f` takes the part of the theorem to be matched.

- `PROVE_HYP (|- t1) (|- t2) :(thm -> thm -> thm)`

$$\frac{A \vdash t_1 \quad B, t_1 \vdash t_2}{A \bigcup B \vdash t_2}$$

- `PURE_ASM_REWRITE_RULE :(thm list -> thm -> thm)`

$$\frac{A \vdash t}{A \vdash t'}$$

  where t' is obtained from t by rewriting with $th_1, \ldots, th_n$ and the assumptions in A

- `PURE_ONCE_ASM_REWRITE_RULE [th1;...thn]`
  `:(thm list -> thm -> thm)`

$$\frac{A \vdash t}{A \vdash t'}$$

  where t' is obtained from t by rewriting *once* with only $th_1, \ldots, th_n$ and the assumption A

- `PURE_ONCE_REWRITE_RULE [th1,...thn]`
  `:(thm list -> thm -> thm)`

$$\frac{A \vdash t}{A \vdash t'}$$

  where t' is obtained from t by rewriting *once* with only $th_1, \ldots, th_n$

- `PURE_REWRITE_RULE [th1,...thn] :(thm list -> thm -> thm)`

$$\frac{A \vdash t}{A \vdash t'}$$

  where t' is obtained from t by rewriting with only $th_1, \ldots, th_n$

- `RES_CANON :(thm -> thm list)`

$$\frac{t_1 \wedge t_2}{t_1, t_2}$$

$$\frac{(t_1 \wedge t_2) \supset t}{t_1 \supset (t_2 \supset t)}$$

$$\frac{(t_1 \vee t_2) \supset t}{t_1 \supset t,\ t_2 \supset t}$$

$$\frac{(\exists x.t_1) \supset t_2}{t_1[x'/x] \supset t_2}$$

$$\frac{\forall x.t_1}{t_1[x'/x]}$$

$$\frac{\forall x.t_1 \supset t_2}{t_1 \supset \forall x.t_2}$$

the above rules are used to put theorem into canonical form (horn clauses) for resolustion

- `REWRITE_RULE [th1,...thn] :(thm list -> thm -> thm)`

$$\frac{A \vdash t}{A \vdash t'}$$

where t' is obtained from t by rewriting with $th_1, \ldots, th_n$ and the theorems bound to `basic_rewrites`

- `RIGHT_BETA :(thm -> thm)`

$$\frac{A \vdash t_1 = (\lambda x.t_2)t_3}{A \vdash t_1 = t_2[t_3/x]}$$

- `RIGHT_LIST_BETA :(thm -> thm)`

$$\frac{A \vdash t_1 = (\lambda x_1 \ldots x_n.t_2)u_1 \ldots u_n}{A \vdash t_1 = t_2[u_1 \ldots u_n/x_1 \ldots x_n]}$$

- `SELECT_ELIM :(thm -> (term # thm) -> thm)`

$$\frac{A_1 \vdash P(\$\exists P) \quad A_2, \texttt{"P v"} \vdash t}{A_1 \bigcup A_2 \vdash t}$$

v occurs nowhere except in $P\,v$

- `SELECT_EQ "x" :(term -> thm -> thm)`

$$\frac{A \vdash t_1 = t_2}{A \vdash (@x.t_1) = (@x.t_2)}$$

- `SELECT_INTRO :(thm -> thm)`

$$\frac{A \vdash P\,t}{A \vdash P(\$@P)}$$

- `SELECT_RULE :(thm -> thm)`

$$\frac{A \vdash \exists x.t[x]}{A \vdash t[@x.t[x]]}$$

- `SPEC "t" :(term -> thm -> thm)`

$$\frac{A \vdash \forall x.u}{A \vdash u[t/x]}$$

- `SPECL ["t1";...;"tn"] :(term list -> thm -> thm)`

$$\frac{A \vdash \forall x_1 \ldots x_n.t[x_1, \ldots, x_n]}{A \vdash t[t_1, \ldots, t_n]}$$

- `SPEC_ALL :(thm -> thm)`

$$\frac{A \vdash \forall x_1 \ldots x_n.t[x_i]}{A \vdash t[x_i'/x_i]}$$

- `SPEC_VAR  :(thm -> (term # thm))`

$$\vdash \forall x.t \leadsto \texttt{"x'"}, \vdash t[x'/x]$$

- `SUBS  [|- ti=ui] :(thm list -> thm -> thm)`

$$\frac{\vdash t_1 = u_1 \quad \cdots \quad \vdash t_n = u_n \quad \vdash t[t_1,\ldots,t_n]}{\vdash t[u_1,\ldots,u_n]}$$

- `SUBST  [((A1 |- t1=u1), x1); ...  ;((An |- tn=un), xn)]`
  `"t[x1,...,xn]" (A |- t[t1,...,tn])`
  `:((thm # term) list -> term -> thm -> thm)`

$$\frac{A_1 \vdash t_1 = u_1 \quad \cdots \quad A_n \vdash t_n = u_n \quad A \vdash t[t_1,\ldots,t_n]}{A \bigcup A_1 \bigcup \cdots \bigcup A_2 \vdash t[u_i]}$$

- `SUBST_MATCH  (|- u=v) :(thm -> thm -> thm)`

$$\frac{\vdash u = v \quad \vdash t[u]}{A \vdash t[v/u]}$$

- `SUBS_OCCS  [[ni],|- ti=ui]`
  `:((int list # thm) list -> thm -> thm)`

$$\frac{\vdash t_1 = u_1 \quad \cdots \quad \vdash t_n = u_n \quad t[t_1,\ldots,t_n]}{t[u_1,\ldots,u_n]}$$

  replaces each t_i by u_i in t, at occurrence numbers specified in the list `ni`

- `SYM  :(thm -> thm)`

$$\frac{A \vdash t_1 = t_2}{A \vdash t_2 = t_1}$$

- `TRANS  :(thm -> thm -> thm)`

$$\frac{A_1 \vdash t_1 = t_2 \quad A_2 \vdash t_2 = t_3}{A_1 \bigcup A_2 \vdash t_1 = t_3}$$

- `UNDISCH  :(thm -> thm)`

$$\frac{A \vdash t_1 \supset t_2}{A, t_1 \vdash t_2}$$

- `UNDISCH_ALL  :(thm -> thm)`

$$\frac{A \vdash t_1 \supset \cdots \supset t_n \supset t}{A, t_1, \ldots, t_n \vdash t}$$

2 HOL Conversions

- `ABS_CONV conv "\x.t[x]" :(conv -> conv)`

  applies `conv` to `t[x]`

- `AC_CONV (assoc_thm, comm_thm) "lhs = rhs" :((thm # thm) -> conv)`

  A derived rule for proving simple AC-equivalence. `assoc_thm` and `comm_thm` are the theorems asserting the associativity and commutativity of the operator in `lhs` and `rhs`. If both sides are AC-equivalent, a theorem $\vdash (lhs = rhs) = \mathsf{T}$ is returned. It fails otherwise.

- `ADD_CONV "n + m" :conv`

$$\vdash n + m = s$$

- `ALL_CONV "t" :conv`

$$\vdash t = t$$

- `ALPHA_CONV "x" "(\y. t)" :(term -> conv)`

$$\vdash \lambda y.t = \lambda x.t[x/y]$$

- `AND_EXISTS_CONV "(?x.P) /\ (?x.Q)" :conv`

$$\vdash (\exists x.P) \wedge (\exists x.Q) = (\exists x.P \wedge Q)$$

- `AND_FORALL_CONV "(!x.P) /\ (!x.Q)" :conv`

$$\vdash (\forall x.P) \wedge (\forall x.Q) = (\forall x.P \wedge Q)$$

- `ANTE_CONJ_CONV "a1 /\ a2 ==> c" :conv`

$$\vdash a_1 \wedge a_2 \supset c = (a_1 \supset (a_2 \supset c))$$

- `ASSUME "A" :conv`

$$A \vdash A$$

- `BETA_CONV "(\x.u)v" :conv`

$$\vdash (\lambda x.u)v = u[v/x]$$

- `CHANGED_CONV conv  :(conv -> conv)`

  fail if `conv` does not change its input

- `COND_CONV "c => u | v" :conv`

$$\begin{array}{ll} \vdash (\mathsf{T} \Rightarrow u \mid v) = u & \text{if } c = \mathsf{T} \\ \vdash (\mathsf{F} \Rightarrow u \mid v) = u & \text{if } c = \mathsf{F} \\ \vdash (c \Rightarrow u \mid v) = u & \text{if } u \text{ and } v \text{ are } \alpha\text{-equivalent} \end{array}$$

- `CONJUNCTS_CONV "t1" "t2" :((term # term) -> thm)`

$$\vdash t_1 = t_2$$

  if t_1 and t_2 are equivalent using idempotence, symetry and associativity of $\wedge$

- `CONJ_SET_CONV "[t1;...;tn]" "[u1;...;um]" :(term list -> term list -> thm)`

$$\vdash (t_1 \wedge \ldots \wedge t_n) = (u_1 \wedge \ldots \wedge u_n)$$

- `CONTRAPOS_CONV "P ==> Q" :conv`

$$\vdash (P \supset Q) = (\neg Q \supset \neg P)$$

- `CONV_RULE conv th  :(conv -> thm -> thm)`

  applies `conv` to the conclusion of `th`

- `DEF_EXISTS_RULE :conv`

  proves that a function defined by a definitional equation exists

- `DEPTH_CONV conv tm :(conv -> conv)`

  applies `conv` recursively to `tm`

- `ETA_CONV "\x. t x" :conv`

$$\vdash (\lambda x.tx) = t$$

- `EVERY_CONV convl tm :(conv list -> conv)`

  performs every conversion in `convl`

- `EXISTS_AND_CONV "?x. P /\ Q" :conv`

$$\begin{array}{rl} \vdash (\exists x.P \wedge Q) = (\exists x.P) \wedge Q & \text{if } x \text{ is free in } P \text{ but not in } Q \\ \vdash (\exists x.P \wedge Q) = P \wedge (\exists x.Q) & \text{if } x \text{ is free in } Q \text{ but not in } P \\ \vdash (\exists x.P \wedge Q) = (\exists x.P) \wedge (\exists x.Q) & \text{if } x \text{ is free in neither } P \text{ nor } Q \end{array}$$

- `EXISTS_GREATEST_CONV`
  `"(?x.P[x]) /\ (?y.!z.z > y ==> ~P[z])" :conv`

$$\vdash (\exists x.P[x]) \wedge (\exists y.\forall z.z > y \supset \neg P[z]) = \exists x.P[x] \wedge \forall z.z > x \supset \neg P[z]$$

- `EXISTS_IMP_CONV "?x. P ==> Q" :conv`

$$\begin{array}{rl} \vdash (\exists x.P \supset Q) = (\forall x.P) \supset Q & \text{if } x \text{ is free in } P \text{ but not in } Q \\ \vdash (\exists x.P \supset Q) = P \supset (\exists x.Q) & \text{if } x \text{ is free in } Q \text{ but not in } P \\ \vdash (\exists x.P \supset Q) = (\forall x.P) \supset (\exists x.Q) & \text{if } x \text{ is free in neither } P \text{ nor } Q \end{array}$$

- `EXISTS_LEAST_CONV "?n:num. P[n]" :conv`

$$\vdash (\exists n.P[n]) = \exists n.P[n] \wedge \forall n'.(n' < n) \supset \neg P[n']$$

- `EXISTS_NOT_CONV "?x. ~P" :conv`

$$\vdash (\exists x.\neg P) = \neg(\forall x.P)$$

- `EXISTS_OR_CONV "?x. P \/ Q" :conv`

$$\vdash (\exists x.P \vee Q) = (\exists x.P) \vee (\exists x.Q)$$

- `EXISTS_UNIQUE_CONV "?!x. P[x]" :conv`

$$\vdash ?!x.P[x] = \exists x.P[x] \wedge \forall xy.P[x] \wedge P[y] \supset (x = y)$$

- `FIRST_CONV convl tm :(conv list -> conv)`

  performs the first successful conversion of those in `convl`

- `FORALL_AND_CONV "!x. P /\ Q" :conv`

$$\vdash (\forall x.P \wedge Q) = (\forall x.P) \wedge (\forall x.Q)$$

- `FORALL_IMP_CONV "!x. P ==> Q" :conv`

$$\begin{array}{rl} \vdash (\forall x.P \supset Q) = (\exists x.P) \supset Q & \text{if } x \text{ is free in } P \text{ but not } Q \\ \vdash (\forall x.P \supset Q) = P \supset (\forall x.Q) & \text{if } x \text{ is free in } Q \text{ but not in } P \\ \vdash (\forall x.P \supset Q) = (\exists x.P) \supset (\forall x.Q) & \text{if } x \text{ is free in neither } P \text{ nor } Q \end{array}$$

- `FORALL_NOT_CONV "!x. ~P" :conv`

$$\vdash (\forall x.\neg P) = \neg(\exists x.P)$$

- `FORALL_OR_CONV "!x. P \/ Q" :conv`

$$\begin{array}{rl} \vdash (\forall x.P \vee Q) = (\forall x.P) \vee Q & \text{if } x \text{ is free in } P \text{ but not in } Q \\ \vdash (\forall x.P \vee Q) = P \vee (\forall x.Q) & \text{if } x \text{ is free in } Q \text{ but not in } P \\ \vdash (\forall x.P \vee Q) = (\forall x.P) \vee (\forall x.Q) & \text{if } x \text{ is free in neither } P \text{ nor } Q \end{array}$$

- `FRONT_CONJ_CONV [t1;...;tn] t :(term list -> conv)`

$$\vdash (t_1 \wedge \cdots \wedge t \wedge \cdots \wedge t_n) = (t \wedge t_1 \wedge \cdots \wedge t_n)$$

- `FUN_EQ_CONV "f = g" :conv`

$$\vdash (f = g) = (\forall x. f x = g x)$$

- `GEN_ALPHA_CONV "x" "(\y.t)" :(term -> conv)`

$$\vdash \lambda y.t = \lambda x.t[x/y]$$
$$\vdash \forall y.t = \forall x.t[x/y]$$
$$\vdash \exists y.t = \exists x.t[x/y]$$
$$\vdash @y.t = @x.t[x/y]$$
$$\vdash \exists\forall y.t = \exists\forall x.t[x/y]$$

- `GEN_BETA_CONV "(\x.u) v" :conv`

$$\vdash (\lambda x.u)v = u[v/x]$$
$$\vdash (\lambda(x,y).u)p = u[\mathsf{FST}\,p, \mathsf{SND}\,p/x,y]$$

- `LEFT_AND_EXISTS_CONV "(?x.P) /\ Q" :conv`

$$\vdash (\exists x.P) \wedge Q = (\exists x'.P[x'/x] \wedge Q)$$

- `LEFT_AND_FORALL_CONV "(!x.P) /\ Q" :conv`

$$\vdash (\forall x.P) \wedge Q = (\forall x'.P[x'/x] \wedge Q)$$

- `LEFT_IMP_EXISTS_CONV "(?x.P) ==> Q" :conv`

$$\vdash (\exists x.P) \supset Q = (\forall x'.P[x'/x] \supset Q)$$

- `LEFT_IMP_FORALL_CONV "(!x.P) ==> Q" :conv`

$$\vdash (\forall x.P) \supset Q = (\exists x'.P[x'/x] \supset Q)$$

- `LEFT_OR_EXISTS_CONV "(?x.P) \/ Q" :conv`

$$\vdash (\exists x.P) \vee Q = (\exists x'.P[x'/x] \vee Q)$$

- `LEFT_OR_FORALL_CONV "(!x.P) \/ Q" :conv`

$$\vdash (\forall x.P) \vee Q = (\forall x'.P[x'/x] \vee Q)$$

- `LENGTH_CONV "LENGTH[x1;...;xn]" :conv`

$$\vdash \mathsf{LENGTH}[x_1; \ldots; x_n] = n$$

- `LIST_BETA_CONV "(\x1 ... xn. t)t1 ... tn" :conv`

$$\vdash (\lambda x_1 \ldots x_n.t)t_1 \ldots t_n = t[t_1/x_1] \ldots [t_n/x_n]$$

- `NOT_EXISTS_CONV "~(?x.P)" :conv`

$$\vdash \neg(\exists x.P) = \forall x.\neg P$$

- `NOT_FORALL_CONV "~(!x.P)" :conv`

$$\vdash \neg(\forall x.P) = \exists x.\neg P$$

- `NO_CONV :conv`

  always fails

- `ONCE_DEPTH_CONV conv tm :(conv -> conv)`

  applies `conv` *once* to the first suitable sub-term(s)

- `OR_EXISTS_CONV "(?x.P) \/ (?x.Q)" :conv`

$$\vdash (\exists x.P) \vee (\exists x.Q) = (\exists x.P \vee Q)$$

- `OR_FORALL_CONV "(!x.P) \/ (!x.Q)" :conv`

$$\vdash (\forall x.P) \vee (\forall x.Q) = (\forall x.P \vee Q)$$

- `conv1 ORELSEC conv2 :(conv -> conv -> conv)` infix
  performs `conv1` if failed then perform `conv2`
- `PAIRED_BETA_CONV "(\(x1,...,xn).t) (t1,...,tn)" :conv`
  $$\vdash (\lambda(x_1,\ldots,x_n).t)(t_1,\ldots,t_n) = t[t_1,\ldots,t_n/x_1,\ldots,x_n]$$
- `PAIRED_ETA_CONV "(\(x1,...,xn).t (t1,...,tn))" :conv`
  $$\vdash (\lambda(x_1,\ldots,x_n).t(t_1,\ldots,t_n)) = t$$
- `RAND_CONV conv "t1 t2" :(conv -> conv)`
  applies `conv` to `t2`
- `RATOR_CONV conv "t1 t2" :(conv -> conv)`
  applies `conv` to `t1`
- `REDEPTH_CONV conv t :(conv -> conv)`
  re-traverses term after each conversion
- `REFL "t" :conv`
  $$\vdash t = t$$
- `REPEATC conv :(conv -> conv)`
  applies a conversion zero or more time
- `REWRITES_CONV net :(conv term_net -> conv)`
  creates a conversion from a net
- `REWRITE_CONV |- !x1...xn.  t = u :(thm -> conv)`
  matches $x_1 \ldots x_n : t' \leadsto \vdash t' = u'$
- `RIGHT_AND_EXISTS_CONV "P /\ (?x.Q)" :conv`
  $$\vdash P \wedge (\exists x.Q) = (\exists x'.P \wedge (Q[x'/x])$$
- `RIGHT_AND_FORALL_CONV "P /\ (!x.Q)" :conv`
  $$\vdash P \wedge (\forall x.Q) = (\forall x'.P \wedge (Q[x'/x])$$
- `RIGHT_CONV_RULE conv (A |- t1 = t2) :(conv -> thm -> thm)`
  $$A \vdash t_1 = t_2'$$
  where t_2' is the result of applying `conv` to t_2
- `RIGHT_IMP_EXISTS_CONV "P ==> (?x.Q)" :conv`
  $$\vdash P \supset (\exists x.Q) = (\exists x'.P \supset (Q[x'/x])$$
- `RIGHT_IMP_FORALL_CONV "P ==> (!x.Q)" :conv`
  $$\vdash P \supset (\forall x.Q) = (\forall x'.P \supset (Q[x'/x])$$
- `RIGHT_OR_EXISTS_CONV "P \/ (?x.Q)" :conv`
  $$\vdash P \vee (\exists x.Q) = (\exists x'.P \vee (Q[x'/x])$$
- `RIGHT_OR_FORALL_CONV "P \/ (!x.Q)" :conv`
  $$\vdash P \vee (\forall x.Q) = (\forall x'.P \vee (Q[x'/x])$$
- `SELECT_CONV "P [@x.P [x]]" :conv`
  $$\vdash P[@x.P[x]] = \exists x.P[x]$$
- `SKOLEM_CONV "!x1...xn. ?y. t" :conv`
  $$\vdash (\forall x_1 \ldots x_n.\exists y.t) = (\exists y'.\forall x_1 \ldots x_n.tm[y' x_1 \ldots x_n/y])$$
  where y' is a primed variant of y not free in the input term

- `SUBST_CONV [(|- ti = ui,"xi");...] "t[x1,...xn]" "t[t1,...tn]" :((thm # term) list -> term -> conv)`

$$\vdash t[t_1,\ldots,t_n] = t[u_1,\ldots,u_n]$$

- `SUB_CONV conv tm :(conv -> conv)`

  applies `conv` to all top-level subterms of `tm`

- `SWAP_EXISTS_CONV "?x y. P" :conv`

$$\vdash (\exists x\, y.P) = (\exists y\, x.P)$$

- `SYM_CONV "t1 = t2" :conv`

$$\vdash (t_1 = t_2) = (t_2 = t_1)$$

- `conv1 THENC conv2 :(conv -> conv -> conv)` infix

  performs `conv1` then `conv2`

- `TOP_DEPTH_CONV conv tm :(conv -> conv)`

  rewrites `tm` trying conversions at top level before descending

- `TRY_CONV conv :(conv -> conv)`

  $\equiv$ `conv ORELSEC ALL_CONV`

- `X_FUN_EQ_CONV "x" "f = g" :(term -> conv)`

$$\vdash (f = g) = \forall x. f x = g x$$

- `X_SKOLEM_CONV "f" "!x1...xn. ?y. t" :(term -> conv)`

$$\vdash (\forall x_1 \ldots x_n.\exists y.t) = (\exists f.\forall x_1 \ldots x_n.t[f x_1 \ldots x_n/y])$$

- `bool_EQ_CONV "b1 = b2" :conv`

$\vdash (b_1 = b_2) = \mathsf{T}$	if b_1 and b_2 are identical boolean terms
$\vdash (b_1 = b_2) = \mathsf{F}$	if b_1 and b_2 are T or F and different
$\vdash (b_1 = b_2) = b_2$	if b_1 is T
$\vdash (b_1 = b_2) = b_1$	if b_2 is T

- `let_CONV "let v1 = x1 and ... and vn = xn in tm[v1,...,vn]" :conv`

$$\vdash \mathsf{let} v_1 = x_1 \mathsf{and} \ldots \mathsf{and} v_n = x_n \mathsf{in}$$
$$tm[v_1,\ldots,v_n] = tm[x_1,\ldots,x_n/v_1,\ldots,v_n]$$

- `list_EQ_CONV conv "[x1;...;xn] = [y1;...;ym]" :(conv -> conv)`

$\vdash ([x_1;\ldots;x_n] = [y_1;\ldots;y_m]) = \mathsf{F}$	if $\neg(n = m)$ or `conv` proves $\vdash (x_i = y_i) = \mathsf{F}$ for any i, where $1 \le i \le n, m$
$\vdash ([x_1;\ldots;x_n] = [y_1;\ldots;y_m]) = \mathsf{T}$	if $(n = m)$ and x_i is syntactically identical to y_i for all i, or if $(n = m)$ and `conv` proves $\vdash (x_i = y_i) = \mathsf{T}$ for all i, where $1 \le i \le n$

- `num_CONV "n" :conv`

$$\vdash n = \mathsf{SUC}\, m$$

- `num_EQ_CONV "n = m" :conv`

$\vdash (n = m) = \mathsf{T}$	if $n = m$
$\vdash (n = m) = \mathsf{F}$	if $\neg(n = m)$

3 HOL Tactics and Tacticals

- `ACCEPT_TAC (|- th) :thm_tactic`

$$\frac{th'}{\overline{\quad -\quad}}$$

solves any goal whose conclusion is α-convertible of `th`

- `ALL_TAC :tactic`

$$\frac{t}{\overline{t}}$$

passes on a goal unchanged

- `ALL_THEN :thm_tactical`

equivalent to `I`

- `ANTE_RES_THEN f (|-th) :thm_tactical`

$$\frac{\{t_1,\ldots,t_n\} \vdash t}{\overline{\{t_1,\ldots,t_n,u_1,\ldots,u_m\}t'}}$$

applies `f` to the result of resolving `th` with t_i which is in implicative form and then applies the resulting tactic to the goal

- `AP_TERM_TAC :tactic`

$$\frac{fx = fy}{\overline{x = y}}$$

- `AP_THM_TAC :tactic`

$$\frac{fx = gx}{\overline{f = g}}$$

- `ASM_CASES_TAC "u" :(term -> tactic)`

$$\frac{t}{\overline{\{u\}t \quad \{\neg u\}t}}$$

- `ASM_REWRITE_TAC [th1;...;thn] :(thm list -> tactic)`

$$\frac{\{t_1,\ldots,t_m\}t}{\overline{\{t_1,\ldots,t_m\}t'}}$$

t' is obtained from t by rewriteing with $t_1,\ldots,t_m$,$th_1,\ldots,th_n$ and the theorems bound to `basic_rewrites`

- `ASSUME_TAC (|- u) :thm_tactic`

$$\frac{t}{\overline{\{u\}\,t}}$$

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

$$\frac{\{t_1,\ldots,t_n\}t}{\overline{\{t_1,\ldots,t_n\}t'}}$$

applies `f` to the list of assumptions which have be converted to `:thm`s by `ASSUME`

- `BETA_TAC :tactic`

$$\frac{(\lambda x.t[x])y}{\overline{t[y/x]}}$$

- `BOOL_CASES_TAC "u" :(term -> tactic)`

$$\frac{t}{\overline{\{u = T\}t \quad \{u = F\}t}}$$

- `CASES_THENL [f1;...;fn] (|-x. u1 \/ ...  \/ un)`
  `:(thm_tactic list -> thm_tactic)`

  where t_i is obtained by applying the tactic $f_i(u_i \vdash u_i)$ to t

- `CHANGED_TAC  tac :(tactic -> tactic)`

  repeatedly apply `tac` to the goal until no new goals are produced

- `CHECK_ASSUME_TAC th :thm_tactic`

$$\frac{\{t_1, \ldots, t_n\}t}{\{t_1, \ldots, t_n, th\}t'}$$

  checks to see whether `th` can sovle the goal by contradiction or α-conversion, then discard it if is useless, otherwise put it in the assumption

- `CHOOSE_TAC (|-?x.u) :thm_tactic`

$$\frac{\{t_1, \ldots, t_n\}t}{\{t_1, \ldots, t_n, \exists x.u\}t'}$$

  $\equiv$ `CHOOSE_THEN ASSUME_TAC`

- `CHOOSE_THEN  f (|-?x.u) :thm_tactical`

$$\frac{\{t_1, \ldots, t_n\}t}{\{t_1, \ldots, t_n\}t'}$$

  applies `f` to `u` then applies the resulting tactic to the goal

- `COND_CASES_TAC :tactic`

$$\frac{t[p \Rightarrow u \mid v]}{\{p\}t[u] \quad \{\neg p\}t[v]}$$

- `CONJUNCTS_THEN  f (|- u /\v) :thm_tactical`

  apply `f` to $(u \vdash u)$ and $(v \vdash v)$ then apply the resulting tactics to the goal in turn

- `CONJUNCTS_THEN2  f1 f2 (|- u /\v)`
  `:(thm_tactic -> thm_tactical)`

  apply `f1` to `u` and `f2` to `v` respectively, then apply the resulting tactics to the goal in that order

- `CONJ_TAC :tactic`

$$\frac{t_1 \wedge t_2}{t_1 \quad t_2}$$

- `CONTR_TAC  (|-F) :thm_tactic`

$$\frac{t}{-}$$

  solve goal given contradictory theorem

- `CONV_TAC  conv :(conv -> tactic)`

$$\frac{t}{t'}$$

  t' is the result of applying `conv` to t

- `DISCARD_TAC  (|-th) :thm_tactic`

  check that a theorem is useless, then ignore it

- `DISCH_TAC :tactic`

$$\frac{u \supset v}{\{u\}\ v}$$

$$\frac{\neg u}{\{u\}F}$$

- `DISCH_THEN  f :(thm_tactic -> tactic)`

$$\frac{\{A\}\ u \supset v}{\{A, u'\}\ v'}$$

apply `f` to the antecedants of implications and then the resulting tactic to the goal

- `DISJ1_TAC :tactic`

$$\frac{t_1 \vee t_2}{t_1}$$

- `DISJ2_TAC :tactic`

$$\frac{t_1 \vee t_2}{t_2}$$

- `DISJ_CASES_TAC  (|- u \/ v) :thm_tactic`

$$\frac{t}{\{u\}\,t \quad \{v\}\,t}$$

- `DISJ_CASES_THEN  f (|-u \/ v) :thm_tactical`

$$\frac{t}{t_1 \quad t_2}$$

where t_1 is obtained by applying the tactic $f(u \vdash u)$ on t and t_2 is obtained by applying the tactic $f(v \vdash v)$ on t

- `DISJ_CASES_THEN2 f1 f2 (|-u \/ v)`
  `:(thm_tactic -> thm_tactical)`

$$\frac{t}{t_1 \quad t_2}$$

where t_1 is obtained by applying the tactic $f_1(u \vdash u)$ on t and t_2 is obtained by applying the tactic $f_2(v \vdash v)$ on t

- `DISJ_CASES_THENL  [f1;...;fn] (|-u1\/...\/un)`
  `:(thm_tactic list -> thm_tactic)`

$$\frac{t}{t_1 \quad \cdots \quad t_n}$$

where t_i is obtained by applying the tactic $f_i(u_i \vdash u_i)$ on t

- `EQ_TAC :tactic`

$$\frac{u = v}{u \supset v \quad v \supset u}$$

- `EVERY [tac1;...;tacn] :(tactic list -> tactic)`

applies the tactics in the list in turn

- `EVERY_ASSUM f :(thm_tactic -> tactic)`

$$\frac{\{u_1, \ldots, u_n\}t}{\{u_1, \ldots, u_n\}t'}$$

where t' is obtained by applying $f\,(u_i \vdash u_i)$ in turn on t

- `EVERY_TCL [f1;...;fn] :(thm_tactical list -> thm_tactical)`

applies the theorem tacticals in turn

- `EXISTS_TAC "u" :(term -> tactic)`

$$\frac{\exists x.t[x]}{t[u/x]}$$

- `FAIL_TAC str :(string -> tactic)`

  it always fails with the given string `str`

- `FILTER_ASM_REWRITE_TAC p [th1;...thn] :((term -> bool) -> thm list -> tactic)`

$$\frac{\{t_1,\ldots,t_m\}t}{\{t_1,\ldots,t_m\}t'}$$

  where t' is obtained by rewriting t with $th_1,\ldots,th_n$, those assumptions satisfied the predicate `p` and the theorems bound to `basic_rewrites`

- `FILTER_DISCH_TAC tm :(term -> tactic)`

$$\frac{\{A\}\ u \supset v}{\{A, u\}\ v}$$

  equivalent to `FILTER_DISCH_THEN STRIP_ASSUME_TAC`

- `FILTER_DISCH_THEN f tm :(thm_tactic -> term -> tactic)`

$$\frac{\{A\}\ u \supset v}{\{A, u'\}\ v}$$

  like `DISCH_THEN` but fails if the antecedent mentions the term

- `FILTER_GEN_TAC tm :(term -> tactic)`

$$\frac{\forall x.t}{t}$$

  fails if `tm` equals x

- `FILTER_ONCE_ASM_REWRITE_TAC p [th1;...thn] :((term -> bool) -> thm list -> tactic)`

$$\frac{\{t_1,\ldots,t_m\}t}{\{t_1,\ldots,t_m\}t'}$$

  where t' is obtained by rewriting t *once* with $th_1,\ldots,th_n$, those assumptions satisfied the predicate `p` and the theorems bound to `basic_rewrites`

- `FILTER_STRIP_TAC "x" :(term -> tactic)`

$$\frac{t}{\{\cdots\}t_1 \quad \cdots \quad \{\cdots\}t_n}$$

  like `STRIP_TAC` but if t contains `x` then do nothing

- `FILTER_STRIP_THEN f "x" :(thm_tactic -> term -> tactic)`

  like `STRIP_TAC` but if t contains `x` then do nothing

- `FIRST [tac1;...;tacn] :(tactic list -> tactic)`

$$\frac{t}{t'}$$

  apply `tac1`, if it fails then apply `tac2`, etc.

- `FIRST_ASSUM f :(thm_tactic -> tactic)`

$$\frac{\{A_1,\ldots,A_n\}t}{\{A_1,\ldots,A_n\}t'}$$

  apply the tactic $f(A_1 \vdash A_1)$ to the goal, if it fails then apply $f(A_2 \vdash A_2),\ldots$

- `FIRST_TCL [f1,...fn] :(thm_tactical list -> thm_tactical)`

  applies the first successful tactical

- `FREEZE_THEN f (|-u) :thm_tactical`

  applies $f(u \vdash u)$ to the goal and freeze $(u \vdash u)$ to prevent instantiation

- `GEN_REWRITE_TAC`
  `:((conv -> conv) -> thm list -> thm list -> tactic)`

  this is the general rewriting tactic which is called by other rewrite tactics

- `GEN_TAC :tactic`

$$\frac{\forall x.t[x]}{\overline{t[x']}}$$

  x' is a variant of x not free in t

- `GSUBST_TAC`
  `:(((term # term) list -> term -> term) -> thm list -> tactic)`

  does substitution in the goal

- `IMP_RES_TAC  th :thm_tactic`

$$\frac{\{t_1,\ldots,t_n\}t}{\overline{\{t_1,\ldots,t_n,u_1,\ldots,u_m\}t}}$$

  where u_i are the results of resolving `th` with the assumptions of the goal

- `IMP_RES_THEN  f th :thm_tactical`

$$\frac{\{t_1,\ldots,t_n\}t}{\overline{\{t_1,\ldots,t_n,u_1,\ldots,u_m\}t'}}$$

  applies the tactic generating function `f` to the results of resolving `th` with the assumptions of the goal and then apply the tactic to the goal

- `INDUCT_TAC :tactic`

$$\frac{\forall n.t[n]}{\overline{t[0] \quad \{t[n]\}\, t[\mathsf{SUC}\, n]}}$$

  mathematical induction over the natural numbers

- `INDUCT_THEN  th f :(thm -> thm_tactic -> tactic)`

$$\frac{\forall x{:}ty.t[x]}{\overline{t[b] \quad \{t[x]\}t'[x]}}$$

  general induction tactic for recursive types. `th` is the induction theorem of the type $:ty$ and `f` is applied to the induction hypothesis to tranform t to t'

- `LIST_INDUCT_TAC :tactic`

$$\frac{\forall l.t[l]}{\overline{t[\mathsf{NIL}] \quad \{t[l]\}\, \forall h.t[\mathsf{CONS}\, h\, l]}}$$

  mathematical induction over finite lists

- `MAP_EVERY f [x1,...xn] :((* -> tactic) -> * list -> tactic)`

$$\frac{t}{\overline{t'}}$$

  where t' is obtained from applying the tactics generated by applying `f` to each element of the list

- `MAP_FIRST f [x1,...xn] :((* -> tactic) -> * list -> tactic)`

$$\frac{t}{\overline{t'}}$$

  where t' is obtained from the first successfully applied tactic generated by mapping `f` to the list

- `MATCH_ACCEPT_TAC th :thm_tactic`

$$\frac{t}{\overline{-}}$$

solves a gaol that matchs the given theorem `th`

- `MATCH_MP_TAC (|-!x1..xn.s ==> !y1...ym.t) :thm_tactic`

$$\frac{\{A\}t'}{\{A\}\exists z_1 \dots z_p.s'}$$

where z's are thoses x's that do not appear free in t. This matches the goal to the conclusion of an implicative theorem and reduce it to some instance of the antecedant

- `MP_TAC (|-u) :thm_tactic`

$$\frac{t}{u \supset t}$$

- `NO_TAC :tactic`

this tactic always fails

- `NO_THEN :thm_tactical`

this always fails

- `ONCE_ASM_REWRITE_TAC [th1;...;thn] :(thm list -> tactic)`

$$\frac{\{t_1, \dots, t_m\}t}{\{t_1, \dots, t_m\}t'}$$

where t' is obtained by rewriting t *once* with $th_1, \dots, th_n$, the assumptions and the theorems bound to `basic_rewrites`

- `ONCE_REWRITE_TAC [th1;...;thn] :(thm list -> tactic)`

$$\frac{\{t_1, \dots, t_m\}t}{\{t_1, \dots, t_m\}t'}$$

where t' is obtained by rewriting t *once* with $th_1, \dots, th_n$, and the theorems bound to `basic_rewrites`

- `ORELSE tac1 tac2 :(tactic -> tactic -> tactic)` infix

apply `tac1`, if it fails then apply `tac2`

- `ORELSE_TCL f1 f2 tac th`
`:(thm_tactical -> thm_tactical -> thm_tactical)`

applies `(f1 tac th)`, if it fails applies `(f2 tac th)`

- `POP_ASSUM f :(thm_tactic -> tactic)`

$$\frac{\{t_1, t_2, \dots, t_n\}t}{\{t_2, \dots, t_n\}t'}$$

where t' is obtained by applying $f\,t_1$ to t

- `POP_ASSUM_LIST f :((thm list -> tactic) -> tactic)`

$$\frac{\{t_1, t_2, \dots, t_n\}t}{t'}$$

where t' is obtained from applying the tactic(s) generated by $f[t_1; \dots; t_n]$ to t

- `PURE_ASM_REWRITE_TAC [th1;...;thn] :(thm list -> tactic)`

$$\frac{\{t_1, \dots, t_m\}t}{\{t_1, \dots, t_m\}t'}$$

where t' is obtained by rewriting t with $th_1, \dots, th_n$ and the assumptions

- `PURE_ONCE_ASM_REWRITE_TAC [th1;...;thn] :(thm list -> tactic)`

$$\frac{\{t_1,\ldots,t_m\}t}{\{t_1,\ldots,t_m\}t'}$$

where t' is obtained by rewriting t *once* with $th_1,\ldots,th_n$ and the assumptions

- `PURE_ONCE_REWRITE_TAC [th1;...;thn] :(thm list -> tactic)`

$$\frac{\{t_1,\ldots,t_m\}t}{\{t_1,\ldots,t_m\}t'}$$

where t' is obtained by rewriting t once with $th_1,\ldots,th_n$

- `PURE_REWRITE_TAC [th1;...;thn] :(thm list -> tactic)`

$$\frac{\{t_1,\ldots,t_m\}t}{\{t_1,\ldots,t_m\}t'}$$

where t' is obtained by rewriting t with $th_1,\ldots,th_n$

- `REFL_TAC :tactic`

$$\frac{t=t}{-}$$

- `REPEAT tac :(tactic -> tactic)`

apply `tac` repeatedly until it fails

- `REPEAT_GTCL :(thm_tactical -> thm_tactical)`

applies `ttcl` repeatedly and trap failures. used to repeat `IMP_RES_THEN`

- `REPEAT_TCL ttcl ttac th :(thm_tactical -> thm_tactical)`

applies `ttcl` repeatedly

- `RES_TAC :tactic`

$$\frac{\{t_1,\ldots,t_n\}t}{\{t_1,\ldots,t_n,u_1,\ldots,u_n\}t}$$

resolve using the assumptions

- `RES_THEN f :(thm_tactic -> tactic)`

$$\frac{\{t_1,\ldots,t_n\}t}{\{t_1,\ldots,t_n,u_1,\ldots,u_n\}t'}$$

apply `f` to the results of resolving the assumptions and then apply the generated tactic(s) to t

- `REWRITE_TAC [th1;...;thn] :(thm list -> tactic)`

$$\frac{\{t_1,\ldots,t_m\}t}{\{t_1,\ldots,t_m\}t'}$$

where t' is obtained by rewriting t with $th_1,\ldots,th_n$ and the theorems bound to `basic_rewrites`

- `RULE_ASSUM_TAC rule :((thm -> thm) -> tactic)`

$$\frac{\{t_1\ldots,t_m\}t}{\{t'_1\ldots,t'_m\}t}$$

apply the inference rule to the assumptions

- `SPEC_TAC ("u","x") :((term # term) -> tactic)`

$$\frac{t[u]}{\forall x.t[x]}$$

- `STRIP_ASSUME_TAC  th :thm_tactic`

$$\frac{t}{\{u_1,\ldots,u_n\}\,t}$$

where $u_1,\ldots,u_n$ are the parts of `th`

- `STRIP_GOAL_THEN  f :(thm_tactic -> tactic)`

strip one outer $\forall$, $\wedge$ or $\supset$ from the goal

- `STRIP_TAC :tactic`

$$\frac{\forall x.t[x]}{t[x]}$$

$$\frac{\{A\}u\wedge v}{\{A\}u \quad \{A\}v}$$

$$\frac{\{A\}(u_1\wedge\ldots\wedge u_n)\supset v}{\{A,u_1,\ldots,u_n\}v}$$

strip one outer connective from the goal

- `STRIP_THM_THEN f th :thm_tactical`

applies `CONJUNCTS_THEN` or else `DISJ_CASES_THEN` or else `CHOOSE_THEN`

- `STRUCT_CASES_TAC |-(?y1.(x = u1[y1])) \/...\/(?yn.(x=un[yn])) :thm_tactic`

$$\frac{t[x]}{t[u_1[y_1]] \quad \cdots \quad t[u_n[y_n]]}$$

split a goal according to the structure theorem

- `SUBGOAL_THEN "u" ttac :(term -> thm_tactic -> tactic)`

$$\frac{t}{\{u\}t \quad u}$$

makes a new subgoal `u` and also provides it as a theorem to `ttac` to apply to the goal t

- `SUBST1_TAC |-u=v :thm_tactic`

$$\frac{t[u]}{t[v]}$$

- `SUBST_ALL_TAC |-u=v :thm_tactic`

$$\frac{\{t_1[u],\ldots,t_n[u]\}t[u]}{\{t_1[v],\ldots,t_n[v]\}t[v]}$$

- `SUBST_OCCS_TAC [([oij],|-ui=vi)] :((int list # thm) list -> tactic)`

$$\frac{t[u_i]}{t[v_i]}$$

substitutes at the jth occurrences of the ith variable in the goal

- `SUBST_TAC [|-ui=vi] :(thm list -> tactic)`

$$\frac{t[u_1,\ldots,u_n]}{t[v_1,\ldots,v_n]}$$

- `TAC_PROOF (t,tac) :((goal # tactic) -> thm)`

$$\frac{t}{-}$$

prove the goal `t` using `tac`

- `THEN tac1 tac2 :(tactic -> tactic -> tactic)` infix

  apply `tac1` then `tac2` to the goal

- `THENL tac [tac1;...;tacn]`
  `:(tactic -> tactic list -> tactic)` infix

  if `tac` produces n subgoals, apply `taci` to the ith subgoal

- `THEN_TCL ttcl1 ttcl2 tac`
  `:(thm_tactical -> thm_tactical -> thm_tactical)`

  applies `ttcl1` and `ttcl2` in turn

- `TRY tac :(tactic -> tactic)`

  apply `tac` to the goal if failed do `ALL_TAC`

- `UNDISCH_TAC  "t1" :(term -> tactic)`

$$\frac{\{t_1,\ldots,t_n\}t}{\{t_2,\ldots,t_n\}t_1 \supset t}$$

- `VALID tac :(tactic -> tactic)`

  makes `tac` valid and fails if it returns an invalid proof

- `X_CASES_THEN :(term list list -> thm_tactical)`

- `X_CASES_THENL [["vij"]] [fi] th`
  `:(term list list -> thm_tactic list -> thm_tactic)`

$$\frac{\{A\}t_1}{\{A\}t'_1}\mathtt{f1} \vdash u_1\left[\frac{v_{1j}}{x_{1j}}\right] \quad \cdots \quad \frac{\{A\}t_n}{\{A\}t'_n}\mathtt{fn} \vdash u_n\left[\frac{v_{nj}}{x_{nj}}\right]$$

  where `th` is a generalized disjunction in the form $\vdash (\exists x_{1j}.u_1[x_{1j}]) \vee \cdots \vee (\exists x_{nj}.u_n[x_{nj}])$ and

- `X_CHOOSE_TAC  "y" (|-?x.u) :(term -> thm_tactic)`

$$\frac{\{A\}t}{\{A, u[y/x]\}t}$$

  $\equiv$ `X_CHOOSE_THEN "x" ASSUME_TAC`

- `X_CHOOSE_THEN  "y" f (|-?x.u) :(term -> thm_tactical)`

  Applies `f` to $u[y/x] \vdash u[y/x]$ and then apply the resulting tactic to the goal

- `X_GEN_TAC  "y" :(term -> tactic)`

$$\frac{\forall x.t[x]}{t[y]}$$

4 Pre-proved Theorems

Theorems listed in this section are built-in in the system. They are loaded automatically when the name is encountered in the input. They are divided into subsections according to the type. Within each subsection, definitions are list first then are the theorems.

4.1 Axioms

These are axioms of the HOL logic

`ARB_THM` $\vdash \$= \,=\, \$=$

`BOOL_CASES_AX` $\vdash \forall t.\,(t = \mathsf{T}) \vee (t = \mathsf{F})$

`ETA_AX` $\vdash \forall t.\,(\lambda x.\, t\, x) = t$

`IMP_ANTISYM_AX` $\vdash \forall t_1\, t_2.\,(t_1 \supset t_2) \supset ((t_2 \supset t_1) \supset (t_1 = t_2))$

`INFINITY_AX` $\vdash \exists f.\, \mathsf{ONE\_ONE}\, f \wedge \neg \mathsf{ONTO}\, f$

`SELECT_AX` $\vdash \forall P\, x.\, P\, x \supset P\,(\$\varepsilon\, P)$

4.2 Definitions of basic logical constants

These are the primitive logical constants. All of them have special syntex.

`AND_DEF` $\vdash \$\wedge = (\lambda t_1\, t_2.\,(\forall t.\,(t_1 \supset (t_2 \supset t)) \supset t))$

`ARB` $\vdash \mathsf{ARB} = (\varepsilon x.\, \mathsf{T})$

`COND_DEF` $\vdash \mathsf{COND} = (\lambda t\, t_1\, t_2.\,(\varepsilon x.\,((t = \mathsf{T}) \supset (x = t_1)) \wedge ((t = \mathsf{F}) \supset (x = t_2))))$

`EXISTS_DEF` $\vdash \$\exists = (\lambda P.\, P\,(\$\varepsilon\, P))$

`EXISTS_UNIQUE_DEF` $\vdash \$\exists\forall = (\lambda P.\, \$\exists\, P \wedge (\forall x\, y.\, P\, x \wedge P\, y \supset (x = y)))$

`FORALL_DEF` $\vdash \$\forall = (\lambda P.\, P = (\lambda x.\, \mathsf{T}))$

`F_DEF` $\vdash \mathsf{F} = (\forall t.\, t)$

`LET_DEF` $\vdash \mathsf{LET} = (\lambda f\, x.\, f\, x)$

`NOT_DEF` $\vdash \neg = (\lambda t.\, t \supset \mathsf{F})$

`ONE_ONE_DEF` $\vdash \forall f.\, \mathsf{ONE\_ONE}\, f = (\forall x_1\, x_2.\,(f\, x_1 = f\, x_2) \supset (x_1 = x_2))$

`ONTO_DEF` $\vdash \forall f.\, \mathsf{ONTO}\, f = (\forall y.\,(\exists x.\, y = f\, x))$

`OR_DEF` $\vdash \$\vee = (\lambda t_1\, t_2.\,(\forall t.\,(t_1 \supset t) \supset ((t_2 \supset t) \supset t)))$

`RES_ABSTRACT` $\vdash \forall P\, B.\, \mathsf{RES\_ABSTRACT}\, P\, B = (\lambda x.\,(P\, x \Rightarrow B\, x \mid \mathsf{ARB}))$

`RES_EXISTS` $\vdash \forall P\, B.\, \mathsf{RES\_EXISTS}\, P\, B = (\exists x.\, P\, x \wedge B\, x)$

`RES_FORALL` $\vdash \forall P\, B.\, \mathsf{RES\_FORALL}\, P\, B = (\forall x.\, P\, x \supset B\, x)$

`RES_SELECT` $\vdash \forall P\, B.\, \mathsf{RES\_SELECT}\, P\, B = (\varepsilon x.\, P\, x \wedge B\, x)$

`T_DEF` $\vdash \mathsf{T} = ((\lambda x.\, x) = (\lambda x.\, x))$

4.3 Logical tautologies

These theorems are not stored in any theory.

`ABS_SIMP` $\vdash \forall t_1\, t_2.\, (\lambda x.\, t_1)\, t_2 = t_1$

`AND1_THM` $\vdash \forall t_1\, t_2.\, t_1 \wedge t_2 \supset t_1$

`AND2_THM` $\vdash \forall t_1\, t_2.\, t_1 \wedge t_2 \supset t_2$

`AND_CLAUSES`
$\vdash \forall t.\, (\mathsf{T} \wedge t = t) \wedge (t \wedge \mathsf{T} = t) \wedge (\mathsf{F} \wedge t = \mathsf{F}) \wedge (t \wedge \mathsf{F} = \mathsf{F}) \wedge (t \wedge t = t)$

`AND_INTRO_THM` $\vdash \forall t_1\, t_2.\, t_1 \supset (t_2 \supset t_1 \wedge t_2)$

`BOOL_EQ_DISTINCT` $\vdash \neg(\mathsf{T} = \mathsf{F}) \wedge \neg(\mathsf{F} = \mathsf{T})$

`COND_CLAUSES` $\vdash \forall t_1\, t_2.\, ((\mathsf{T} \Rightarrow t_1 \mid t_2) = t_1) \wedge ((\mathsf{F} \Rightarrow t_1 \mid t_2) = t_2)$

`COND_ID` $\vdash \forall b\, t.\, (b \Rightarrow t \mid t) = t$

`CONJ_ASSOC` $\vdash \forall t_1\, t_2\, t_3.\, t_1 \wedge t_2 \wedge t_3 = (t_1 \wedge t_2) \wedge t_3$

`CONJ_SYM` $\vdash \forall t_1\, t_2.\, t_1 \wedge t_2 = t_2 \wedge t_1$

`DE_MORGAN_THM` $\vdash \forall t_1\, t_2.\, (\neg(t_1 \wedge t_2) = \neg t_1 \vee \neg t_2) \wedge (\neg(t_1 \vee t_2) = \neg t_1 \wedge \neg t_2)$

`DISJ_ASSOC` $\vdash \forall t_1\, t_2\, t_3.\, t_1 \vee t_2 \vee t_3 = (t_1 \vee t_2) \vee t_3$

`DISJ_SYM` $\vdash \forall t_1\, t_2.\, t_1 \vee t_2 = t_2 \vee t_1$

`EQ_CLAUSES` $\vdash \forall t.\, ((\mathsf{T} = t) = t) \wedge ((t = \mathsf{T}) = t) \wedge ((\mathsf{F} = t) = \neg t) \wedge ((t = \mathsf{F}) = \neg t)$

`EQ_EXT` $\vdash \forall f\, g.\, (\forall x.\, f\, x = g\, x) \supset (f = g)$

`EQ_REFL` $\vdash \forall x.\, x = x$

`EQ_SYM` $\vdash \forall x\, y.\, (x = y) \supset (y = x)$

`EQ_TRANS` $\vdash \forall x\, y\, z.\, (x = y) \wedge (y = z) \supset (x = z)$

`EXCLUDED_MIDDLE` $\vdash \forall t.\, t \vee \neg t$

`EXISTS_SIMP` $\vdash \forall t.\, (\exists x.\, t) = t$

`FALSITY` $\vdash \forall t.\, \mathsf{F} \supset t$

`FORALL_SIMP` $\vdash \forall t.\, (\forall x.\, t) = t$

`F_IMP` $\vdash \forall t.\, \neg t \supset (t \supset \mathsf{F})$

`IMP_CLAUSES` $\vdash \forall t.\, (\mathsf{T} \supset t = t) \wedge (t \supset \mathsf{T} = \mathsf{T}) \wedge (\mathsf{F} \supset t = \mathsf{T}) \wedge (t \supset t = \mathsf{T}) \wedge$
$(t \supset \mathsf{F} = \neg t)$

`IMP_F` $\vdash \forall t.\, (t \supset \mathsf{F}) \supset \neg t$

`LEFT_AND_OVER_OR` $\vdash \forall t_1\, t_2\, t_3.\, t_1 \wedge (t_2 \vee t_3) = t_1 \wedge t_2 \vee t_1 \wedge t_3$

`LEFT_OR_OVER_AND` $\vdash \forall t_1\, t_2\, t_3.\, t_1 \vee t_2 \wedge t_3 = (t_1 \vee t_2) \wedge (t_1 \vee t_3)$

`NOT_AND` $\vdash \neg(t \wedge \neg t)$

`NOT_CLAUSES` $\vdash (\forall t.\, \neg(\neg t) = t) \wedge (\neg \mathsf{T} = \mathsf{F}) \wedge (\neg \mathsf{F} = \mathsf{T})$

`NOT_F` $\vdash \forall t.\, \neg t \supset (t = \mathsf{F})$

`NOT_IMP` $\vdash \forall t_1\, t_2.\, \neg(t_1 \supset t_2) = t_1 \wedge \neg t_2$

`OR_CLAUSES` $\vdash \forall t.\, (\mathsf{T} \vee t = \mathsf{T}) \wedge (t \vee \mathsf{T} = \mathsf{T}) \wedge (\mathsf{F} \vee t = t) \wedge (t \vee \mathsf{F} = t) \wedge (t \vee t = t)$

`OR_ELIM_THM` $\vdash \forall t\, t_1\, t_2.\, t_1 \vee t_2 \supset ((t_1 \supset t) \supset ((t_2 \supset t) \supset t))$

`OR_IMP_THM` $\vdash \forall t_1\, t_2.\, (t_1 = t_2 \vee t_1) = t_2 \supset t_1$

`OR_INTRO_THM1` $\vdash \forall t_1\, t_2.\, t_1 \supset t_1 \vee t_2$

`OR_INTRO_THM2` $\vdash \forall t_1\, t_2.\, t_2 \supset t_1 \vee t_2$

`REFL_CLAUSE` $\vdash \forall x.\, (x = x) = \mathsf{T}$

`RIGHT_AND_OVER_OR` $\vdash \forall t_1\, t_2\, t_3.\, (t_2 \vee t_3) \wedge t_1 = t_2 \wedge t_1 \vee t_3 \wedge t_1$

`RIGHT_OR_OVER_AND` $\vdash \forall t_1\, t_2\, t_3.\, t_2 \wedge t_3 \vee t_1 = (t_2 \vee t_1) \wedge (t_3 \vee t_1)$

`TRUTH` $\vdash \mathsf{T}$

4.4 Theorems about the type one

The type one contains only one object.

one_Axiom	$\vdash \forall e.\,(\exists\forall fn.\,fn\,\mathsf{one} = e)$
one_DEF	$\vdash \mathsf{one} = (\varepsilon x.\,\mathsf{T})$
one_TY_DEF	$\vdash \exists rep.\,\mathsf{TYPE\_DEFINITION}\,(\lambda b.\,b)\,rep$
one_axiom	$\vdash \forall f\,g.\,f = g$
one	$\vdash \forall v.\,v = \mathsf{one}$

4.5 Theorems about combinators

Definitions

I_DEF	$\vdash \mathsf{I} = \mathsf{S\,K\,K}$
K_DEF	$\vdash \mathsf{K} = (\lambda x\,y.\,x)$
S_DEF	$\vdash \mathsf{S} = (\lambda f\,g\,x.\,f\,x\,(g\,x))$
o_DEF	$\vdash \forall f\,g.\,(f \circ g) = (\lambda x.\,f\,(g\,x))$

Theorems

I_THM	$\vdash \forall x.\,\mathsf{I}\,x = x$
I_o_ID	$\vdash \forall f.\,((\mathsf{I} \circ f) = f) \wedge ((f \circ \mathsf{I}) = f)$
K_THM	$\vdash \forall x\,y.\,\mathsf{K}\,x\,y = x$
S_THM	$\vdash \forall f\,g\,x.\,\mathsf{S}\,f\,g\,x = f\,x\,(g\,x)$
o_ASSOC	$\vdash \forall f\,g\,h.\,(f \circ g \circ h) = ((f \circ g) \circ h)$
o_THM	$\vdash \forall f\,g\,x.\,\$\circ\,f\,g\,x = f\,(g\,x)$

4.6 Theorems about the type pair

The definition of the type `pair` is in the theory `bool`

Definitions

CURRY_DEF	$\vdash \forall f\,x\,y.\,\mathsf{CURRY}\,f\,x\,y = f\,(x, y)$
UNCURRY_DEF	$\vdash \forall f\,x\,y.\,\mathsf{UNCURRY}\,f\,(x, y) = f\,x\,y$
SND_DEF	$\vdash \forall p.\,\mathsf{SND}\,p = (\varepsilon y.\,(\exists x.\,\mathsf{MK\_PAIR}\,x\,y = \mathsf{REP\_prod}\,p))$
FST_DEF	$\vdash \forall p.\,\mathsf{FST}\,p = (\varepsilon x.\,(\exists y.\,\mathsf{MK\_PAIR}\,x\,y = \mathsf{REP\_prod}\,p))$
COMMA_DEF	$\vdash \forall x\,y.\,x, y = (\varepsilon p.\,\mathsf{REP\_prod}\,p = \mathsf{MK\_PAIR}\,x\,y)$
REP_prod	$\vdash \mathsf{REP\_prod} = (\varepsilon rep.\,(\forall p'\,p''.\,(rep\,p' = rep\,p'') \supset (p' = p'')) \wedge (\forall p.\,\mathsf{IS\_PAIR}\,p = (\exists p'.\,p = rep\,p')))$
prod_TY_DEF	$\vdash \exists rep.\,\mathsf{TYPE\_DEFINITION}\,\mathsf{IS\_PAIR}\,rep$
IS_PAIR_DEF	$\vdash \forall p.\,\mathsf{IS\_PAIR}\,p = (\exists x\,y.\,p = \mathsf{MK\_PAIR}\,x\,y)$
MK_PAIR_DEF	$\vdash \forall x\,y.\,\mathsf{MK\_PAIR}\,x\,y = (\lambda a\,b.\,(a = x) \wedge (b = y))$

Theorems

`FST` $\vdash \forall x\, y.\, \mathsf{FST}\,(x, y) = x$

`PAIR_EQ` $\vdash \forall x\, y\, a\, b.\, (x, y = a, b) = (x = a) \wedge (y = b)$

`PAIR_EXISTS` $\vdash \exists p.\, \mathsf{IS\_PAIR}\, p$

`PAIR` $\vdash \forall x.\, \mathsf{FST}\, x, \mathsf{SND}\, x = x$

`SND` $\vdash \forall x\, y.\, \mathsf{SND}\,(x, y) = y$

4.7 Theorems about disjoint sums

Definitions

`INL_DEF` $\vdash \forall e.\, \mathsf{INL}\, e = \mathsf{ABS\_sum}\,(\lambda b\, x\, y.\, (x = e) \wedge b)$

`INR_DEF` $\vdash \forall e.\, \mathsf{INR}\, e = \mathsf{ABS\_sum}\,(\lambda b\, x\, y.\, (y = e) \wedge \neg b)$

`ISL` $\vdash (\forall x.\, \mathsf{ISL}\,(\mathsf{INL}\, x)) \wedge (\forall y.\, \neg \mathsf{ISL}\,(\mathsf{INR}\, y))$

`ISR` $\vdash (\forall x.\, \mathsf{ISR}\,(\mathsf{INR}\, x)) \wedge (\forall y.\, \neg \mathsf{ISR}\,(\mathsf{INL}\, y))$

`IS_SUM_REP` $\vdash \forall f.\, \mathsf{IS\_SUM\_REP}\, f = (\exists v_1\, v_2.\, (f = (\lambda b\, x\, y.\, (x = v_1) \wedge b)) \vee (f = (\lambda b\, x\, y.\, (y = v_2) \wedge \neg b)))$

`OUTL` $\vdash \forall x.\, \mathsf{OUTL}\,(\mathsf{INL}\, x) = x$

`OUTR` $\vdash \forall x.\, \mathsf{OUTR}\,(\mathsf{INR}\, x) = x$

`sum_ISO_DEF` $\vdash (\forall a.\, \mathsf{ABS\_sum}\,(\mathsf{REP\_sum}\, a) = a) \wedge (\forall r.\, \mathsf{IS\_SUM\_REP}\, r = (\mathsf{REP\_sum}\,(\mathsf{ABS\_sum}\, r) = r))$

`sum_TY_DEF` $\vdash \exists rep.\, \mathsf{TYPE\_DEFINITION}\, \mathsf{IS\_SUM\_REP}\, rep$

Theorems

`INL` $\vdash \forall x.\, \mathsf{ISL}\, x \supset (\mathsf{INL}\,(\mathsf{OUTL}\, x) = x)$

`INR` $\vdash \forall x.\, \mathsf{ISR}\, x \supset (\mathsf{INR}\,(\mathsf{OUTR}\, x) = x)$

`ISL_OR_ISR` $\vdash \forall x.\, \mathsf{ISL}\, x \vee \mathsf{ISR}\, x$

`sum_Axiom` $\vdash \forall f\, g.\, (\exists! h.\, (\forall x.\, h\,(\mathsf{INL}\, x) = f\, x) \wedge (\forall x.\, h\,(\mathsf{INR}\, x) = g\, x))$

`sum_axiom` $\vdash \forall f\, g.\, (\exists! h.\, ((h \circ \mathsf{INL}) = f) \wedge ((h \circ \mathsf{INR}) = g))$

4.8 Theorems about the type `num` and arithmetic

Definitions

`ADD` $\vdash (\forall n.\, 0 + n = n) \wedge (\forall m\, n.\, \mathsf{SUC}\, m + n = \mathsf{SUC}\,(m + n))$

`DIVISION` $\vdash \forall n.\, 0 < n \supset (\forall k.\, (k = (k\, \mathsf{DIV}\, n) \times n + k\, \mathsf{MOD}\, n) \wedge k\, \mathsf{MOD}\, n < n)$

`EXP` $\vdash (\forall m.\, (m)\, \mathsf{EXP}\,(0) = 1) \wedge (\forall m\, n.\, (m)\, \mathsf{EXP}\,(\mathsf{SUC}\, n) = m \times (m)\, \mathsf{EXP}\,(n))$

`GREATER_OR_EQ` $\vdash \forall m\, n.\, m \geq n = m > n \vee (m = n)$

`GREATER` $\vdash \forall m\, n.\, m > n = n < m$

`IS_NUM_REP` $\vdash \forall m.\, \mathsf{IS\_NUM\_REP}\, m = (\forall P.\, P\, \mathsf{ZERO\_REP} \wedge (\forall n.\, P\, n \supset P\,(\mathsf{SUC\_REP}\, n)) \supset P\, m)$

`LESS_OR_EQ` $\vdash \forall m\, n.\, m \leq n = m < n \vee (m = n)$

LESS $\vdash \forall m\, n.\, m < n = (\exists P.\, (\forall n'.\, P\,(\mathsf{SUC}\,n') \supset P\,n') \wedge P\,m \wedge \neg P\,n)$

MULT $\vdash (\forall n.\, 0 \times n = 0) \wedge (\forall m\, n.\, \mathsf{SUC}\,m \times n = m \times n + n)$

PRE_DEF $\vdash \forall m.\, \mathsf{PRE}\,m = ((m = 0) \Rightarrow 0 \mid (\varepsilon n.\, m = \mathsf{SUC}\,n))$

PRIM_REC_FUN $\vdash \forall x\, f.\, \mathsf{PRIM\_REC\_FUN}\,x\,f = \mathsf{SIMP\_REC}\,(\lambda n.\, x)\,(\lambda fun\, n.\, f\,(fun\,(\mathsf{PRE}\,n))\,n)$

PRIM_REC $\vdash \forall x\, f\, m.\, \mathsf{PRIM\_REC}\,x\,f\,m = \mathsf{PRIM\_REC\_FUN}\,x\,f\,m\,(\mathsf{PRE}\,m)$

SIMP_REC_FUN $\vdash \forall x\, f\, n.\, \mathsf{SIMP\_REC\_FUN}\,x\,f\,n = (\varepsilon fun.\, \mathsf{SIMP\_REC\_REL}\,fun\,x\,f\,n)$

SIMP_REC_REL $\vdash \forall fun\, x\, f\, n.\, \mathsf{SIMP\_REC\_REL}\,fun\,x\,f\,n = (fun\,0 = x) \wedge (\forall m.\, m < n \supset (fun\,(\mathsf{SUC}\,m) = f\,(fun\,m)))$

SIMP_REC $\vdash \forall x\, f\, n.\, \mathsf{SIMP\_REC}\,x\,f\,n = \mathsf{SIMP\_REC\_FUN}\,x\,f\,(\mathsf{SUC}\,n)\,n$

SUB $\vdash (\forall m.\, 0 - m = 0) \wedge (\forall m\, n.\, \mathsf{SUC}\,m - n = ((m < n) \Rightarrow 0 \mid \mathsf{SUC}\,(m - n)))$

SUC_DEF $\vdash \forall m.\, \mathsf{SUC}\,m = \mathsf{ABS\_num}\,(\mathsf{SUC\_REP}\,(\mathsf{REP\_num}\,m))$

SUC_REP_DEF $\vdash \mathsf{SUC\_REP} = (\varepsilon f.\, \mathsf{ONE\_ONE}\,f \wedge \neg \mathsf{ONTO}\,f)$

ZERO_DEF $\vdash 0 = \mathsf{ABS\_num}\,\mathsf{ZERO\_REP}$

ZERO_REP_DEF $\vdash \mathsf{ZERO\_REP} = (\varepsilon x.\, (\forall y.\, \neg(x = \mathsf{SUC\_REP}\,y)))$

num_ISO_DEF $\vdash (\forall a.\, \mathsf{ABS\_num}\,(\mathsf{REP\_num}\,a) = a) \wedge (\forall r.\, \mathsf{IS\_NUM\_REP}\,r = (\mathsf{REP\_num}\,(\mathsf{ABS\_num}\,r) = r))$

num_TY_DEF $\vdash \exists rep.\, \mathsf{TYPE\_DEFINITION}\,\mathsf{IS\_NUM\_REP}\,rep$

Theorems

ADD1 $\vdash \forall m.\, \mathsf{SUC}\,m = m + 1$

ADD_0 $\vdash \forall m.\, m + 0 = m$

ADD_ASSOC $\vdash \forall m\, n\, p.\, m + n + p = (m + n) + p$

ADD_CLAUSES $\vdash (0 + m = m) \wedge (m + 0 = m) \wedge (\mathsf{SUC}\,m + n = \mathsf{SUC}\,(m + n)) \wedge (m + \mathsf{SUC}\,n = \mathsf{SUC}\,(m + n))$

ADD_EQ_0 $\vdash \forall m\, n.\, (m + n = 0) = (m = 0) \wedge (n = 0)$

ADD_EQ_SUB $\vdash \forall m\, n\, p.\, n \leq p \supset ((m + n = p) = (m = p - n))$

ADD_INV_0_EQ $\vdash \forall m\, n.\, (m + n = m) = (n = 0)$

ADD_INV_0 $\vdash \forall m\, n.\, (m + n = m) \supset (n = 0)$

ADD_SUB $\vdash \forall a\, c.\, (a + c) - c = a$

ADD_SUC $\vdash \forall m\, n.\, \mathsf{SUC}\,(m + n) = m + \mathsf{SUC}\,n$

ADD_SYM $\vdash \forall m\, n.\, m + n = n + m$

CANCEL_SUB $\vdash \forall p\, n\, m.\, p \leq n \wedge p \leq m \supset ((n - p = m - p) = (n = m))$

DA $\vdash \forall k\, n.\, 0 < n \supset (\exists r\, q.\, (k = q \times n + r) \wedge r < n)$

DIV_LESS_EQ $\vdash \forall n.\, 0 < n \supset (\forall k.\, k\ \mathsf{DIV}\ n \leq k)$

DIV_MULT $\vdash \forall n\, r.\, r < n \supset (\forall q.\, (q \times n + r)\ \mathsf{DIV}\ n = q)$

DIV_UNIQUE $\vdash \forall n\, k\, q.\, (\exists r.\, (k = q \times n + r) \wedge r < n) \supset (k\ \mathsf{DIV}\ n = q)$

EQ_LESS $\vdash \forall n.\, (\mathsf{SUC}\,m = n) \supset m < n$

EQ_MONO_ADD_EQ $\vdash \forall m\, n\, p.\, (m + p = n + p) = (m = n)$

EXP_ADD $\vdash \forall p\, q\, n.\, (n)\ \mathsf{EXP}\ ((p + q)) = (n)\ \mathsf{EXP}\ (p) \times (n)\ \mathsf{EXP}\ (q)$

FUN_EQ_LEMMA $\vdash \forall f\, x_1\, x_2.\, f\,x_1 \wedge \neg f\,x_2 \supset \neg(x_1 = x_2)$

IMP_DISJ_THM $\vdash \forall t_1\, t_2.\, t_1 \supset t_2 = \neg t_1 \vee t_2$

INDUCTION $\vdash \forall P.\, P\,0 \wedge (\forall n.\, P\,n \supset P\,(\mathsf{SUC}\,n)) \supset (\forall n.\, P\,n)$

INV_PRE_EQ $\vdash \forall m\, n.\, 0 < m \wedge 0 < n \supset ((\mathsf{PRE}\,m = \mathsf{PRE}\,n) = (m = n))$

INV_PRE_LESS_EQ $\vdash \forall n.\, 0 < n \supset (\forall m.\, \mathsf{PRE}\,m \leq \mathsf{PRE}\,n = m \leq n)$

INV_PRE_LESS $\vdash \forall m\, n.\, 0 < m \wedge 0 < n \supset (\mathsf{PRE}\,m < \mathsf{PRE}\,n = m < n)$

INV_SUC_EQ $\vdash \forall m\, n.\, (\mathsf{SUC}\,m = \mathsf{SUC}\,n) = (m = n)$

INV_SUC $\vdash \forall m\, n.\, (\mathsf{SUC}\,m = \mathsf{SUC}\,n) \supset (m = n)$

LEFT_ADD_DISTRIB $\vdash \forall m\, n\, p.\, p \times (m + n) = p \times m + p \times n$

LESS_0_0 $\vdash 0 < \mathsf{SUC}\,0$

LESS_0_CASES $\vdash \forall m.\, (0 = m) \vee 0 < m$

LESS_0 $\vdash \forall n.\, 0 < \mathsf{SUC}\,n$

LESS_ADD_1 $\vdash \forall m\, n.\, n < m \supset (\exists p.\, m = n + p + 1)$

LESS_ADD_NONZERO $\vdash \forall m\, n.\, \neg(n = 0) \supset m < m + n$

LESS_ADD_SUC $\vdash \forall m\, n.\, m < m + \mathsf{SUC}\, n$

LESS_ADD $\vdash \forall m\, n.\, n < m \supset (\exists p.\, p + n = m)$

LESS_ANTISYM $\vdash \forall m\, n.\, \neg(m < n \wedge n < m)$

LESS_CASES_IMP $\vdash \forall m\, n.\, \neg(m < n) \wedge \neg(m = n) \supset n < m$

LESS_CASES $\vdash \forall m\, n.\, m < n \vee n \leq m$

LESS_EQUAL_ANTISYM $\vdash \forall n\, m.\, n \leq m \wedge m \leq n \supset (n = m)$

LESS_EQ_ADD_SUB $\vdash \forall c\, b.\, c \leq b \supset (\forall a.\, (a + b) - c = a + (b - c))$

LESS_EQ_ADD $\vdash \forall m\, n.\, m \leq m + n$

LESS_EQ_ANTISYM $\vdash \forall m\, n.\, \neg(m < n \wedge n \leq m)$

LESS_EQ_IMP_LESS_SUC $\vdash \forall n\, m.\, n \leq m \supset n < \mathsf{SUC}\, m$

LESS_EQ_LESS_EQ_MONO $\vdash \forall m\, n\, p\, q.\, m \leq p \wedge n \leq q \supset m + n \leq p + q$

LESS_EQ_MONO_ADD_EQ $\vdash \forall m\, n\, p.\, m + p \leq n + p = m \leq n$

LESS_EQ_MONO $\vdash \forall n\, m.\, \mathsf{SUC}\, n \leq \mathsf{SUC}\, m = n \leq m$

LESS_EQ_REFL $\vdash \forall m.\, m \leq m$

LESS_EQ_SUB_LESS $\vdash \forall a\, b.\, b \leq a \supset (\forall c.\, a - b < c = a < b + c)$

LESS_EQ_SUC_REFL $\vdash \forall m.\, m \leq \mathsf{SUC}\, m$

LESS_EQ_TRANS $\vdash \forall m\, n\, p.\, m \leq n \wedge n \leq p \supset m \leq p$

LESS_EQ $\vdash \forall m\, n.\, m < n = \mathsf{SUC}\, m \leq n$

LESS_EXP_SUC_MONO
$\vdash \forall n\, m.\, (\mathsf{SUC}\,(\mathsf{SUC}\, m))\ \mathsf{EXP}\ (n) < (\mathsf{SUC}\,(\mathsf{SUC}\, m))\ \mathsf{EXP}\ (\mathsf{SUC}\, n)$

LESS_IMP_LESS_ADD $\vdash \forall n\, m.\, n < m \supset (\forall p.\, n < m + p)$

LESS_IMP_LESS_OR_EQ $\vdash \forall m\, n.\, m < n \supset m \leq n$

LESS_LEMMA1 $\vdash \forall m\, n.\, m < \mathsf{SUC}\, n \supset (m = n) \vee m < n$

LESS_LEMMA2 $\vdash \forall m\, n.\, (m = n) \vee m < n \supset m < \mathsf{SUC}\, n$

LESS_LESS_SUC $\vdash \forall m\, n.\, \neg(m < n \wedge n < \mathsf{SUC}\, m)$

LESS_MOD $\vdash \forall n\, k.\, k < n \supset (k\ \mathsf{MOD}\ n = k)$

LESS_MONO_ADD_EQ $\vdash \forall m\, n\, p.\, m + p < n + p = m < n$

LESS_MONO_ADD_INV $\vdash \forall m\, n\, p.\, m + p < n + p \supset m < n$

LESS_MONO_ADD $\vdash \forall m\, n\, p.\, m < n \supset m + p < n + p$

LESS_MONO_EQ $\vdash \forall m\, n.\, \mathsf{SUC}\, m < \mathsf{SUC}\, n = m < n$

LESS_MONO_MULT $\vdash \forall m\, n\, p.\, m \leq n \supset m \times p \leq n \times p$

LESS_MONO_REV $\vdash \forall m\, n.\, \mathsf{SUC}\, m < \mathsf{SUC}\, n \supset m < n$

LESS_MONO $\vdash \forall m\, n.\, m < n \supset \mathsf{SUC}\, m < \mathsf{SUC}\, n$

LESS_MULT_MONO $\vdash \forall m\, i\, n.\, \mathsf{SUC}\, n \times m < \mathsf{SUC}\, n \times i = m < i$

LESS_NOT_EQ $\vdash \forall m\, n.\, m < n \supset \neg(m = n)$

LESS_NOT_SUC $\vdash \forall m\, n.\, m < n \wedge \neg(n = \mathsf{SUC}\, m) \supset \mathsf{SUC}\, m < n$

LESS_OR_EQ_ADD $\vdash \forall n\, m.\, n < m \vee (\exists p.\, n = p + m)$

LESS_OR $\vdash \forall m\, n.\, m < n \supset \mathsf{SUC}\, m \leq n$

LESS_REFL $\vdash \forall n.\, \neg(n < n)$

LESS_SUB_ADD_LESS $\vdash \forall n\, m\, i.\, i < n - m \supset i + m < n$

LESS_SUC_EQ_COR $\vdash \forall m\, n.\, m < n \wedge \neg(\mathsf{SUC}\, m = n) \supset \mathsf{SUC}\, m < n$

LESS_SUC_IMP $\vdash \forall m\, n.\, m < \mathsf{SUC}\, n \supset (\neg(m = n) \supset m < n)$

LESS_SUC_NOT $\vdash \forall m\, n.\, m < n \supset \neg(n < \mathsf{SUC}\, m)$

LESS_SUC_REFL $\vdash \forall n.\, n < \mathsf{SUC}\, n$

LESS_SUC_SUC $\vdash \forall m.\, m < \mathsf{SUC}\, m \wedge m < \mathsf{SUC}\,(\mathsf{SUC}\, m)$

LESS_SUC $\vdash \forall m\, n.\, m < n \supset m < \mathsf{SUC}\, n$

LESS_THM $\vdash \forall m\, n.\, m < \mathsf{SUC}\, n = (m = n) \vee m < n$

LESS_TRANS $\vdash \forall m\, n\, p.\, m < n \wedge n < p \supset m < p$

MOD_EQ_0 $\vdash \forall n.\, 0 < n \supset (\forall k.\, (k \times n)\ \mathsf{MOD}\ n = 0)$

MOD_MOD $\vdash \forall n.\, 0 < n \supset (\forall k.\, (k\ \mathsf{MOD}\ n)\ \mathsf{MOD}\ n = k\ \mathsf{MOD}\ n)$

MOD_MULT $\vdash \forall n\, r.\, r < n \supset (\forall q.\, (q \times n + r)\ \mathsf{MOD}\ n = r)$

MOD_ONE $\vdash \forall k.\, k\ \mathsf{MOD}\ \mathsf{SUC}\, 0 = 0$

MOD_PLUS $\vdash \forall n.\, 0 < n \supset (\forall j\, k.\, (j\, \mathsf{MOD}\, n + k\, \mathsf{MOD}\, n)\, \mathsf{MOD}\, n = (j+k)\, \mathsf{MOD}\, n)$

MOD_TIMES $\vdash \forall n.\, 0 < n \supset (\forall q\, r.\, (q \times n + r)\ \mathsf{MOD}\ n = r\ \mathsf{MOD}\ n)$

MOD_UNIQUE $\vdash \forall n\, k\, r.\, (\exists q.\, (k = q \times n + r) \wedge r < n) \supset (k\ \mathsf{MOD}\ n = r)$

MULT_0 $\vdash \forall m.\, m \times 0 = 0$

MULT_ASSOC $\vdash \forall m\, n\, p.\, m \times n \times p = (m \times n) \times p$

MULT_CLAUSES $\vdash \forall m\, n.\, (0 \times m = 0) \wedge (m \times 0 = 0) \wedge (1 \times m = m) \wedge (m \times 1 = m) \wedge (\mathsf{SUC}\, m \times n = m \times n + n) \wedge (m \times \mathsf{SUC}\, n = m + m \times n)$

MULT_EXP_MONO $\vdash \forall p\, q\, n\, m.\, (n \times (\mathsf{SUC}\, q)\, \mathsf{EXP}\, (p) = m \times (\mathsf{SUC}\, q)\, \mathsf{EXP}\, (p)) = (n = m)$

MULT_LEFT_1 $\vdash \forall m.\, 1 \times m = m$

MULT_MONO_EQ $\vdash \forall m\, i\, n.\, (\mathsf{SUC}\, n \times m = \mathsf{SUC}\, n \times i) = (m = i)$

MULT_RIGHT_1 $\vdash \forall m.\, m \times 1 = m$

MULT_SUC_EQ $\vdash \forall p\, m\, n.\, (n \times \mathsf{SUC}\, p = m \times \mathsf{SUC}\, p) = (n = m)$

MULT_SUC $\vdash \forall m\, n.\, m \times \mathsf{SUC}\, n = m + m \times n$

MULT_SYM $\vdash \forall m\, n.\, m \times n = n \times m$

NOT_EXP_0 $\vdash \forall m\, n.\, \neg((\mathsf{SUC}\, n)\ \mathsf{EXP}\ (m) = 0)$

NOT_LESS_0 $\vdash \forall n.\, \neg(n < 0)$

NOT_LESS_EQ $\vdash \forall m\, n.\, (m = n) \supset \neg(m < n)$

NOT_LESS $\vdash \forall m\, n.\, \neg(m < n) = n \leq m$

NOT_ODD_EQ_EVEN $\vdash \forall n\, m.\, \neg(\mathsf{SUC}\, (n + n) = m + m)$

NOT_SUC_LESS_EQ $\vdash \forall n\, m.\, \neg(\mathsf{SUC}\, n \leq m) = m \leq n$

NOT_SUC $\vdash \forall n.\, \neg(\mathsf{SUC}\, n = 0)$

ODD_OR_EVEN $\vdash \forall n.\, (\exists m.\, (n = \mathsf{SUC}\, (\mathsf{SUC}\, 0) \times m) \vee (n = \mathsf{SUC}\, (\mathsf{SUC}\, 0) \times m + 1))$

OR_LESS $\vdash \forall m\, n.\, \mathsf{SUC}\, m \leq n \supset m < n$

PRE_SUB1 $\vdash \forall m.\, \mathsf{PRE}\, m = m - 1$

PRE_SUB $\vdash \forall m\, n.\, \mathsf{PRE}\, (m - n) = \mathsf{PRE}\, m - n$

PRE_SUC_EQ $\vdash \forall m\, n.\, 0 < n \supset ((m = \mathsf{PRE}\, n) = (\mathsf{SUC}\, m = n))$

PRE $\vdash (\mathsf{PRE}\, 0 = 0) \wedge (\forall m.\, \mathsf{PRE}\, (\mathsf{SUC}\, m) = m)$

PRIM_REC_EQN $\vdash \forall x\, f.\, (\forall n.\, \mathsf{PRIM\_REC\_FUN}\, x\, f\, 0\, n = x) \wedge (\forall m\, n.\, \mathsf{PRIM\_REC\_FUN}\, x\, f\, (\mathsf{SUC}\, m)\, n = f\, (\mathsf{PRIM\_REC\_FUN}\, x\, f\, m\, (\mathsf{PRE}\, n))\, n)$

PRIM_REC_THM $\vdash \forall x\, f.\, (\mathsf{PRIM\_REC}\, x\, f\, 0 = x) \wedge (\forall m.\, \mathsf{PRIM\_REC}\, x\, f\, (\mathsf{SUC}\, m) = f\, (\mathsf{PRIM\_REC}\, x\, f\, m)\, m)$

RIGHT_ADD_DISTRIB $\vdash \forall m\, n\, p.\, (m + n) \times p = m \times p + n \times p$

RIGHT_SUB_DISTRIB $\vdash \forall m\, n\, p.\, (m - n) \times p = m \times p - n \times p$

SIMP_REC_EXISTS $\vdash \forall x\, f\, n.\, (\exists fun.\, \mathsf{SIMP\_REC\_REL}\, fun\, x\, f\, n)$

SIMP_REC_FUN_LEMMA $\vdash (\exists fun.\, \mathsf{SIMP\_REC\_REL}\, fun\, x\, f\, n) = (\mathsf{SIMP\_REC\_FUN}\, x\, f\, n\, 0 = x) \wedge (\forall m.\, m < n \supset (\mathsf{SIMP\_REC\_FUN}\, x\, f\, n\, (\mathsf{SUC}\, m) = f\, (\mathsf{SIMP\_REC\_FUN}\, x\, f\, n\, m)))$

SIMP_REC_THM $\vdash \forall x\, f.\, (\mathsf{SIMP\_REC}\, x\, f\, 0 = x) \wedge (\forall m.\, \mathsf{SIMP\_REC}\, x\, f\, (\mathsf{SUC}\, m) = f\, (\mathsf{SIMP\_REC}\, x\, f\, m))$

SUB_0 $\vdash \forall m.\, (0 - m = 0) \wedge (m - 0 = m)$

SUB_ADD $\vdash \forall m\, n.\, n \leq m \supset ((m - n) + n = m)$

SUB_CANCEL $\vdash \forall p\, n\, m.\, n \leq p \wedge m \leq p \supset ((p - n = p - m) = (n = m))$

SUB_EQUAL_0 $\vdash \forall c.\, c - c = 0$

SUB_EQ_0 $\vdash \forall m\, n.\, (m - n = 0) = m \leq n$

SUB_EQ_EQ_0 $\vdash \forall m\, n.\, (m - n = m) = (m = 0) \vee (n = 0)$

SUB_LESS_0 $\vdash \forall n\, m.\, m < n = 0 < n - m$

SUB_LESS_EQ_ADD $\vdash \forall m\, p.\, m \leq p \supset (\forall n.\, p - m \leq n = p \leq m + n)$

SUB_LESS_EQ $\vdash \forall n\, m.\, n - m \leq n$

SUB_LESS_OR $\vdash \forall m\, n.\, n < m \supset n \leq m - 1$

SUB_MONO_EQ $\vdash \forall n\, m.\, \mathsf{SUC}\, n - \mathsf{SUC}\, m = n - m$

SUB_PLUS $\vdash \forall a\, b\, c.\, a - (b + c) = (a - b) - c$

SUB_SUB $\vdash \forall b\, c.\, c \leq b \supset (\forall a.\, a - (b - c) = (a + c) - b)$

SUC_ID $\vdash \forall n.\, \neg(\mathsf{SUC}\, n = n)$

SUC_LESS $\vdash \forall m\, n.\, \mathsf{SUC}\, m < n \supset m < n$

SUC_NOT $\vdash \forall n.\, \neg(0 = \mathsf{SUC}\, n)$

SUC_SUB1 $\vdash \forall m.\, \mathsf{SUC}\, m - 1 = m$

TIMES2 $\vdash \forall n.\, 2 \times n = n + n$

WOP $\vdash \forall P.\, (\exists n.\, P\, n) \supset (\exists n.\, P\, n \wedge (\forall m.\, m < n \supset \neg P\, m))$

ZERO_LESS_EQ $\vdash \forall n.\, 0 \leq n$

ZERO_LESS_EXP $\vdash \forall m\, n.\, 0 < (\mathsf{SUC}\, n)\, \mathsf{EXP}\, (m)$

ZERO_MOD $\vdash \forall n.\, 0 < n \supset (0\, \mathsf{MOD}\, n = 0)$

num_Axiom $\vdash \forall e\, f.\, (\exists \forall fn.\, (fn\, 0 = e) \wedge (\forall n.\, fn\, (\mathsf{SUC}\, n) = f\, (fn\, n)\, n))$

num_CASES $\vdash \forall m.\, (m = 0) \vee (\exists n.\, m = \mathsf{SUC}\, n)$

4.9 Theorems about trees

Definitions

HT $\vdash \forall t.\, \mathsf{HT}\, t = (\varepsilon n.\, \mathsf{bht}\, n\, t \wedge (\forall m.\, m < n \supset \neg \mathsf{bht}\, m\, t))$

Is_ltree $\vdash \forall t\, l.\, \mathsf{Is\_ltree}\, (t, l) = (\mathsf{Size}\, t = \mathsf{LENGTH}\, l)$

Is_tree_REP $\vdash \mathsf{Is\_tree\_REP} =$
$(\lambda t.\, (\forall P.\, (\forall tl.\, \mathsf{EVERY}\, P\, tl \supset P\, (\mathsf{node\_REP}\, tl)) \supset P\, t))$

Node $\vdash \forall v\, tl.\, \mathsf{Node}\, v\, tl = \mathsf{ABS\_ltree}\, (\mathsf{node}\, (\mathsf{MAP}\, (\mathsf{FST} \circ$
$\mathsf{REP\_ltree})\, tl), \mathsf{CONS}\, v\, (\mathsf{FLAT}\, (\mathsf{MAP}\, (\mathsf{SND} \circ \mathsf{REP\_ltree})\, tl)))$

Size $\vdash \mathsf{Size} = (\varepsilon fn.\, (\forall tl.\, fn\, (\mathsf{node}\, tl) = \mathsf{SUC}\, (\mathsf{SUM}\, (\mathsf{MAP}\, fn\, tl))))$

bht $\vdash \mathsf{bht} =$
$\mathsf{PRIM\_REC}\, (\lambda tr.\, tr = \mathsf{node}\, [\,])\, (\lambda res\, n\, tr.\, (\exists trl.\, (tr = \mathsf{node}\, trl) \wedge$
$\mathsf{EVERY}\, res\, trl))$

dest_node $\vdash \forall t.\, \mathsf{dest\_node}\, t = (\varepsilon p.\, t = \mathsf{node}\, p)$

ltree_ISO_DEF $\vdash (\forall a.\, \mathsf{ABS\_ltree}\, (\mathsf{REP\_ltree}\, a) = a) \wedge$
$(\forall r.\, \mathsf{Is\_ltree}\, r = (\mathsf{REP\_ltree}\, (\mathsf{ABS\_ltree}\, r) = r))$

ltree_TY_DEF $\vdash \exists rep.\, \mathsf{TYPE\_DEFINITION}\, \mathsf{Is\_ltree}\, rep$

node_REP $\vdash (\mathsf{node\_REP}\, [\,] = 0) \wedge (\forall h\, t.\, \mathsf{node\_REP}\, (\mathsf{CONS}\, h\, t) = \mathsf{SUC}\, (h + h) \times$
$(2)\, \mathsf{EXP}\, (\mathsf{node\_REP}\, t))$

node $\vdash \forall tl.\, \mathsf{node}\, tl = \mathsf{ABS\_tree}\, (\mathsf{node\_REP}\, (\mathsf{MAP}\, \mathsf{REP\_tree}\, tl))$

tree_ISO_DEF $\vdash (\forall a.\, \mathsf{ABS\_tree}\, (\mathsf{REP\_tree}\, a) = a) \wedge$
$(\forall r.\, \mathsf{Is\_tree\_REP}\, r = (\mathsf{REP\_tree}\, (\mathsf{ABS\_tree}\, r) = r))$

tree_TY_DEF $\vdash \exists rep.\, \mathsf{TYPE\_DEFINITION}\, \mathsf{Is\_tree\_REP}\, rep$

trf $\vdash \forall n\, f.\, \mathsf{trf}\, n\, f =$
$(\varepsilon fn.\, (\forall trl.\, \mathsf{HT}\, (\mathsf{node}\, trl) \leq n \supset (fn\, (\mathsf{node}\, trl) = f\, (\mathsf{MAP}\, fn\, trl))))$

Theorems

Node_11 $\vdash \forall v_1\, v_2\, trl_1\, trl_2.\, (\mathsf{Node}\, v_1\, trl_1 = \mathsf{Node}\, v_2\, trl_2) = (v_1 = v_2) \wedge$
$(trl_1 = trl_2)$

Node_onto $\vdash \forall l.\, (\exists v\, trl.\, l = \mathsf{Node}\, v\, trl)$

ltree_Axiom $\vdash \forall f.\, (\exists \forall fn.\, (\forall v\, tl.\, fn\, (\mathsf{Node}\, v\, tl) = f\, (\mathsf{MAP}\, fn\, tl)\, v\, tl))$

ltree_Induct $\vdash \forall P.\, (\forall t.\, \mathsf{EVERY}\, P\, t \supset (\forall h.\, P\, (\mathsf{Node}\, h\, t))) \supset (\forall l.\, P\, l)$

node_11 $\vdash \forall tl_1\, tl_2.\, (\mathsf{node}\, tl_1 = \mathsf{node}\, tl_2) = (tl_1 = tl_2)$

tree_Axiom $\vdash \forall f.\, (\exists \forall fn.\, (\forall tl.\, fn\, (\mathsf{node}\, tl) = f\, (\mathsf{MAP}\, fn\, tl)\, tl))$

tree_Induct $\vdash \forall P.\, (\forall tl.\, \mathsf{EVERY}\, P\, tl \supset P\, (\mathsf{node}\, tl)) \supset (\forall t.\, P\, t)$

4.10 Theorems about the type `lists`

Definitions

`AP` $\vdash (\forall l.\, \mathsf{AP}\,[\,]\,l = [\,]) \wedge (\forall h\,t\,l.\, \mathsf{AP}\,(\mathsf{CONS}\,h\,t)\,l = \mathsf{CONS}\,(h\,(\mathsf{HD}\,l))\,(\mathsf{AP}\,t\,(\mathsf{TL}\,l)))$

`APPEND` $\vdash (\forall l.\, \mathsf{APPEND}\,[\,]\,l = l) \wedge (\forall l_1\,l_2\,h.\, \mathsf{APPEND}\,(\mathsf{CONS}\,h\,l_1)\,l_2 = \mathsf{CONS}\,h\,(\mathsf{APPEND}\,l_1\,l_2))$

`CONS_DEF` $\vdash \forall h\,t.\, \mathsf{CONS}\,h\,t = \mathsf{ABS\_list}\,((\lambda m.\,((m = 0) \Rightarrow h \mid \mathsf{FST}\,(\mathsf{REP\_list}\,t)\,(\mathsf{PRE}\,m))), \mathsf{SUC}\,(\mathsf{SND}\,(\mathsf{REP\_list}\,t)))$

`EL` $\vdash (\forall l.\, \mathsf{EL}\,0\,l = \mathsf{HD}\,l) \wedge (\forall l\,n.\, \mathsf{EL}\,(\mathsf{SUC}\,n)\,l = \mathsf{EL}\,n\,(\mathsf{TL}\,l))$

`EVERY_DEF` $\vdash (\forall P.\, \mathsf{EVERY}\,P\,[\,] = \mathsf{T}) \wedge (\forall P\,h\,t.\, \mathsf{EVERY}\,P\,(\mathsf{CONS}\,h\,t) = P\,h \wedge \mathsf{EVERY}\,P\,t)$

`FLAT` $\vdash (\mathsf{FLAT}\,[\,] = [\,]) \wedge (\forall h\,t.\, \mathsf{FLAT}\,(\mathsf{CONS}\,h\,t) = \mathsf{APPEND}\,h\,(\mathsf{FLAT}\,t))$

`HD` $\vdash \forall h\,t.\, \mathsf{HD}\,(\mathsf{CONS}\,h\,t) = h$

`IS_list_REP` $\vdash \forall r.\, \mathsf{IS\_list\_REP}\,r = (\exists f\,n.\, r = (\lambda m.\,((m < n) \Rightarrow f\,m \mid (\varepsilon x.\,\mathsf{T}))), n)$

`LENGTH` $\vdash (\mathsf{LENGTH}\,[\,] = 0) \wedge (\forall h\,t.\, \mathsf{LENGTH}\,(\mathsf{CONS}\,h\,t) = \mathsf{SUC}\,(\mathsf{LENGTH}\,t))$

`MAP` $\vdash (\forall f.\, \mathsf{MAP}\,f\,[\,] = [\,]) \wedge (\forall f\,h\,t.\, \mathsf{MAP}\,f\,(\mathsf{CONS}\,h\,t) = \mathsf{CONS}\,(f\,h)\,(\mathsf{MAP}\,f\,t))$

`NIL_DEF` $\vdash [\,] = \mathsf{ABS\_list}\,((\lambda n.\,(\varepsilon e.\,\mathsf{T})), 0)$

`NULL_DEF` $\vdash (\mathsf{NULL}\,[\,] = \mathsf{T}) \wedge (\forall h\,t.\, \mathsf{NULL}\,(\mathsf{CONS}\,h\,t) = \mathsf{F})$

`PART` $\vdash (\forall l.\, \mathsf{PART}\,[\,]\,l = [\,]) \wedge (\forall n\,t\,l.\, \mathsf{PART}\,(\mathsf{CONS}\,n\,t)\,l = \mathsf{CONS}\,(\mathsf{FST}\,(\mathsf{SPLIT}\,n\,l))\,(\mathsf{PART}\,t\,(\mathsf{SND}\,(\mathsf{SPLIT}\,n\,l))))$

`SPLIT` $\vdash (\forall l.\, \mathsf{SPLIT}\,0\,l = [\,], l) \wedge (\forall n\,l.\, \mathsf{SPLIT}\,(\mathsf{SUC}\,n)\,l = \mathsf{CONS}\,(\mathsf{HD}\,l)\,(\mathsf{FST}\,(\mathsf{SPLIT}\,n\,(\mathsf{TL}\,l))), \mathsf{SND}\,(\mathsf{SPLIT}\,n\,(\mathsf{TL}\,l)))$

`SUM` $\vdash (\mathsf{SUM}\,[\,] = 0) \wedge (\forall h\,t.\, \mathsf{SUM}\,(\mathsf{CONS}\,h\,t) = h + \mathsf{SUM}\,t)$

`TL` $\vdash \forall h\,t.\, \mathsf{TL}\,(\mathsf{CONS}\,h\,t) = t$

`list_ISO_DEF` $\vdash (\forall a.\, \mathsf{ABS\_list}\,(\mathsf{REP\_list}\,a) = a) \wedge (\forall r.\, \mathsf{IS\_list\_REP}\,r = (\mathsf{REP\_list}\,(\mathsf{ABS\_list}\,r) = r))$

`list_TY_DEF` $\vdash \exists rep.\, \mathsf{TYPE\_DEFINITION}\,\mathsf{IS\_list\_REP}\,rep$

Theorems

`APPEND_ASSOC` $\vdash \forall l_1\,l_2\,l_3.\, \mathsf{APPEND}\,l_1\,(\mathsf{APPEND}\,l_2\,l_3) = \mathsf{APPEND}\,(\mathsf{APPEND}\,l_1\,l_2)\,l_3$

`CONS_11` $\vdash \forall h\,t\,h'\,t'.\, (\mathsf{CONS}\,h\,t = \mathsf{CONS}\,h'\,t') = (h = h') \wedge (t = t')$

`CONS` $\vdash \forall l.\, \neg\mathsf{NULL}\,l \supset (\mathsf{CONS}\,(\mathsf{HD}\,l)\,(\mathsf{TL}\,l) = l)$

`EQ_LIST` $\vdash \forall h_1\,h_2.\, (h_1 = h_2) \supset (\forall l_1\,l_2.\, (l_1 = l_2) \supset (\mathsf{CONS}\,h_1\,l_1 = \mathsf{CONS}\,h_2\,l_2))$

`EVERY_CONJ` $\vdash \forall l.\, \mathsf{EVERY}\,(\lambda x.\, P\,x \wedge Q\,x)\,l = \mathsf{EVERY}\,P\,l \wedge \mathsf{EVERY}\,Q\,l$

`EVERY_EL` $\vdash \forall l\,P.\, \mathsf{EVERY}\,P\,l = (\forall n.\, n < \mathsf{LENGTH}\,l \supset P\,(\mathsf{EL}\,n\,l))$

`LENGTH_APPEND` $\vdash \forall l_1\,l_2.\, \mathsf{LENGTH}\,(\mathsf{APPEND}\,l_1\,l_2) = \mathsf{LENGTH}\,l_1 + \mathsf{LENGTH}\,l_2$

`LENGTH_CONS` $\vdash \forall l\,n.\, (\mathsf{LENGTH}\,l = \mathsf{SUC}\,n) = (\exists h\,l'.\, (\mathsf{LENGTH}\,l' = n) \wedge (l = \mathsf{CONS}\,h\,l'))$

`LENGTH_EQ_CONS` $\vdash \forall P\,n.\, (\forall l.\, (\mathsf{LENGTH}\,l = \mathsf{SUC}\,n) \supset P\,l) = (\forall l.\, (\mathsf{LENGTH}\,l = n) \supset (\lambda l.\, (\forall x.\, P\,(\mathsf{CONS}\,x\,l)))\,l)$

`LENGTH_EQ_NIL` $\vdash \forall P.\, (\forall l.\, (\mathsf{LENGTH}\,l = 0) \supset P\,l) = P\,[\,]$

`LENGTH_MAP` $\vdash \forall l\,f.\, \mathsf{LENGTH}\,(\mathsf{MAP}\,f\,l) = \mathsf{LENGTH}\,l$

`LENGTH_NIL` $\vdash \forall l.\, (\mathsf{LENGTH}\,l = 0) = (l = [\,])$

`LIST_NOT_EQ` $\vdash \forall l_1\,l_2.\, \neg(l_1 = l_2) \supset (\forall h_1\,h_2.\, \neg(\mathsf{CONS}\,h_1\,l_1 = \mathsf{CONS}\,h_2\,l_2))$

`MAP_APPEND` $\vdash \forall f\,l_1\,l_2.\, \mathsf{MAP}\,f\,(\mathsf{APPEND}\,l_1\,l_2) = \mathsf{APPEND}\,(\mathsf{MAP}\,f\,l_1)\,(\mathsf{MAP}\,f\,l_2)$

`MAP_o` $\vdash \forall f\,g.\, \mathsf{MAP}\,(f \circ g) = (\mathsf{MAP}\,f \circ \mathsf{MAP}\,g)$

`NOT_CONS_NIL` $\vdash \forall h\,t.\, \neg(\mathsf{CONS}\,h\,t = [\,])$

`NOT_EQ_LIST` $\vdash \forall h_1\,h_2.\, \neg(h_1 = h_2) \supset (\forall l_1\,l_2.\, \neg(\mathsf{CONS}\,h_1\,l_1 = \mathsf{CONS}\,h_2\,l_2))$

`NOT_NIL_CONS` $\vdash \forall h\, t.\, \neg([\,] = \mathsf{CONS}\, h\, t)$

`NULL` $\vdash \mathsf{NULL}\, [\,] \wedge (\forall h\, t.\, \neg \mathsf{NULL}\, (\mathsf{CONS}\, h\, t))$

`list_Axiom` $\vdash \forall x\, f.\, (\exists ! fn.\, (fn\, [\,] = x) \wedge (\forall h\, t.\, fn\, (\mathsf{CONS}\, h\, t) = f\, (fn\, t)\, h\, t))$

`list_CASES` $\vdash \forall l.\, (l = [\,]) \vee (\exists t\, h.\, l = \mathsf{CONS}\, h\, t)$

`list_INDUCT` $\vdash \forall P.\, P\, [\,] \wedge (\forall t.\, P\, t \supset (\forall h.\, P\, (\mathsf{CONS}\, h\, t))) \supset (\forall l.\, P\, l)$

4.11 Theorems used to define types

`ABS_REP_THM` $\vdash \forall P.\, (\exists rep.\, \mathsf{TYPE\_DEFINITION}\, P\, rep) \supset$
$(\exists rep\, abs.\, (\forall a.\, abs\, (rep\, a) = a) \wedge (\forall r.\, P\, r = (rep\, (abs\, r) = r)))$

`exists_TRP` $\vdash \forall P.\, (\exists v.\, P\, v\, [\,]) \supset (\exists t.\, \mathsf{TRP}\, P\, t)$

`TRP_DEF` $\vdash \forall P.\, \mathsf{TRP}\, P = (\varepsilon trp.\, (\forall v\, tl.\, trp\, (\mathsf{Node}\, v\, tl) = P\, v\, tl \wedge$
$\mathsf{EVERY}\, trp\, tl))$

`TRP` $\vdash \forall P\, v\, tl.\, \mathsf{TRP}\, P\, (\mathsf{Node}\, v\, tl) = P\, v\, tl \wedge \mathsf{EVERY}\, (\mathsf{TRP}\, P)\, tl$

`TY_DEF_THM` $\vdash \forall REP\, ABS\, P.\, (\forall a.\, ABS\, (REP\, a) = a) \wedge$
$(\forall r.\, \mathsf{TRP}\, P\, r = (REP\, (ABS\, r) = r)) \supset$
$(\forall f.\, (\exists ! fn.\, (\forall v\, tl.\, P\, v\, (\mathsf{MAP}\, REP\, tl) \supset$
$(fn\, (ABS\, (\mathsf{Node}\, v\, (\mathsf{MAP}\, REP\, tl))) = f\, (\mathsf{MAP}\, fn\, tl)\, v\, tl))))$

`TYPE_DEFINITION` $\vdash \forall P\, rep.\, \mathsf{TYPE\_DEFINITION}\, P\, rep =$
$(\forall x'\, x''.\, (rep\, x' = rep\, x'') \supset (x' = x'')) \wedge$
$(\forall x.\, P\, x = (\exists x'.\, x = rep\, x'))$

5 Libraries

This section lists some of the HOL libraries. Each library is shown in a subsection. A library can be loaded into HOL by the command

```
load_library`xxx`;;
```

where `xxx` is the name of the library. The auto-loading of theorems, etc., is depended on the setup of the library itself. Most of the libraries do set this up properly. If this is not set up, you have to call `autoload_theory` to do it.

If a library contains a single theory, the name of the subsection would read 'The theory xxx' where 'xxx' is both the name pf the library and the theory. If a library contains more than one theory, then the subsection name will read 'The Library xxx', and each theory within the library is listed separately under the heading 'The theory yyy'.

5.1 The library `prog_logic88`

5.1.1 The theory `dijkstra`

Parents

`HOL string halts_thms`

Constants

WEAKER	`":(* -> bool) -> ((* -> bool) -> bool)"`
WEAKEST	`":((* -> bool) -> bool) -> (* -> bool)"`
WP	`":((string -> num) # (string -> num) -> bool) # ((string -> num) -> bool) -> ((string -> num) -> bool)"`
WLP	`":((string -> num) # (string -> num) -> bool) # ((string -> num) -> bool) -> ((string -> num) -> bool)"`
WP1	`":((string -> num) # (string -> num) -> bool) # ((string -> num) -> bool) -> ((string -> num) -> bool)"`
WLP1	`":((string -> num) # (string -> num) -> bool) # ((string -> num) -> bool) -> ((string -> num) -> bool)"`
ITER	`":num -> (((string -> num) -> bool) # ((string -> num) # (string -> num) -> bool) -> ((string -> num) # (string -> num) -> bool))"`
ITER_WP	`":num -> (((string -> num) -> bool) -> (((string -> num) # (string -> num) -> bool) -> (((string -> num) -> bool) -> ((string -> num) -> bool))))"`
ITER_WLP	`":num -> (((string -> num) -> bool) -> (((string -> num) # (string -> num) -> bool) -> (((string -> num) -> bool) -> ((string -> num) -> bool))))"`

Definitions

`WEAKER`	$\vdash \forall p'\, p.\, \mathsf{WEAKER}\, p'\, p = (\forall s.\, p\, s \supset p'\, s)$
`WEAKEST`	$\vdash \forall P.\, \mathsf{WEAKEST}\, P = (\varepsilon p.\, P\, p \wedge (\forall p'.\, P\, p' \supset \mathsf{WEAKER}\, p\, p'))$
`WP`	$\vdash \forall c\, q.\, \mathsf{WP}\,(c, q) = \mathsf{WEAKEST}\,(\lambda p.\, \mathsf{T\_SPEC}\,(p, c, q))$
`WLP`	$\vdash \forall c\, q.\, \mathsf{WLP}\,(c, q) = \mathsf{WEAKEST}\,(\lambda p.\, \mathsf{MK\_SPEC}\,(p, c, q))$
`WP1`	$\vdash \forall c\, q\, s.\, \mathsf{WP1}\,(c, q)\, s = (\exists s'.\, c\,(s, s')) \wedge (\forall s'.\, c\,(s, s') \supset q\, s')$
`WLP1`	$\vdash \forall c\, q\, s.\, \mathsf{WLP1}\,(c, q)\, s = (\forall s'.\, c\,(s, s') \supset q\, s')$
`ITER`	$\vdash (\mathsf{ITER}\, 0 = (\lambda(b, c)\, s, s'.\, \neg b\, s \wedge (s = s'))) \wedge (\forall n.\, \mathsf{ITER}\,(\mathsf{SUC}\, n) = (\lambda(b, c)\, s, s'.\, b\, s \wedge \mathsf{MK\_SEQ}\,(c, \mathsf{ITER}\, n\,(b, c))\,(s, s')))$
`ITER_WP`	$\vdash (\forall b\, c\, p\, s.\, \mathsf{ITER\_WP}\, 0\, b\, c\, p\, s = \neg b\, s \wedge p\, s) \wedge (\forall n\, b\, c\, p\, s.\, \mathsf{ITER\_WP}\,(\mathsf{SUC}\, n)\, b\, c\, p\, s = b\, s \wedge \mathsf{WP}\,(c, \mathsf{ITER\_WP}\, n\, b\, c\, p)\, s)$

`ITER_WLP` $\vdash (\forall b\,c\,p\,s.\ \mathsf{ITER\_WLP}\ 0\ b\,c\,p\,s = \neg b\,s \supset p\,s) \wedge$
$(\forall n\,b\,c\,p\,s.\ \mathsf{ITER\_WLP}\,(\mathsf{SUC}\,n)\,b\,c\,p\,s =$
$b\,s \supset \mathsf{WLP}\,(c, \mathsf{ITER\_WLP}\,n\,b\,c\,p)\,s)$

Theorems

`WEAKER_ANTISYM` $\vdash \forall p_1\,p_2.\ \mathsf{WEAKER}\,p_1\,p_2 \wedge \mathsf{WEAKER}\,p_2\,p_1 \supset (p_1 = p_2)$

`WEAKEST_UNIQUE_LEMMA` $\vdash \forall P\,p_1\,p_2.\,(P\,p_1 \wedge (\forall p'.\ P\,p' \supset \mathsf{WEAKER}\,p_1\,p')) \wedge$
$P\,p_2 \wedge (\forall p'.\ P\,p' \supset \mathsf{WEAKER}\,p_2\,p') \supset (p_1 = p_2)$

`WEAKEST_UNIQUE`
$\vdash \forall P\,p.\ P\,p \wedge (\forall p'.\ P\,p' \supset \mathsf{WEAKER}\,p\,p') \supset (p = \mathsf{WEAKEST}\ P)$

`WP1_T_SPEC` $\vdash \forall c\,q.\ \mathsf{T\_SPEC}\,(\mathsf{WP1}\,(c,q), c, q)$

`WLP1_MK_SPEC` $\vdash \forall c\,q.\ \mathsf{MK\_SPEC}\,(\mathsf{WLP1}\,(c,q), c, q)$

`WP1_WEAKEST` $\vdash \forall p\,c\,q.\ \mathsf{T\_SPEC}\,(p,c,q) \supset \mathsf{WEAKER}\,(\mathsf{WP1}\,(c,q))\,p$

`WLP1_WEAKEST` $\vdash \forall p\,c\,q.\ \mathsf{MK\_SPEC}\,(p,c,q) \supset \mathsf{WEAKER}\,(\mathsf{WLP1}\,(c,q))\,p$

`WP_WP1` $\vdash \forall c\,q.\ \mathsf{WP}\,(c,q) = \mathsf{WP1}\,(c,q)$

`WLP_WLP1` $\vdash \forall c\,q.\ \mathsf{WLP}\,(c,q) = \mathsf{WLP1}\,(c,q)$

`WP_THM` $\vdash \forall c\,q.\ \mathsf{WP}\,(c,q) = (\lambda s.\,(\exists s'.\ c\,(s,s')) \wedge (\forall s'.\ c\,(s,s') \supset q\,s'))$

`WLP_THM` $\vdash \forall c\,q.\ \mathsf{WLP}\,(c,q) = (\lambda s.\,(\forall s'.\ c\,(s,s') \supset q\,s'))$

`WP_T_SPEC` $\vdash \forall c\,q.\,(\exists s.\ \mathsf{WP}\,(c,q)\,s) \supset \mathsf{T\_SPEC}\,(\mathsf{WP}\,(c,q), c, q)$

`WLP_MK_SPEC` $\vdash \forall c\,q.\,(\exists s.\ \mathsf{WLP}\,(c,q)\,s) \supset \mathsf{MK\_SPEC}\,(\mathsf{WLP}\,(c,q), c, q)$

`WP_WEAKEST` $\vdash \forall p\,c\,q.\ \mathsf{T\_SPEC}\,(p,c,q) \supset \mathsf{WEAKER}\,(\mathsf{WP}\,(c,q))\,p$

`WLP_WEAKEST` $\vdash \forall p\,c\,q.\ \mathsf{MK\_SPEC}\,(p,c,q) \supset \mathsf{WEAKER}\,(\mathsf{WLP}\,(c,q))\,p$

`T_SPEC_WP` $\vdash \forall p\,c\,q.\ \mathsf{T\_SPEC}\,(p,c,q) = (\forall s.\ p\,s \supset \mathsf{WP}\,(c,q)\,s)$

`MK_SPEC_WLP` $\vdash \forall p\,c\,q.\ \mathsf{MK\_SPEC}\,(p,c,q) = (\forall s.\ p\,s \supset \mathsf{WLP}\,(c,q)\,s)$

`WP_PROP1` $\vdash \forall c.\ \mathsf{WP}\,(c, (\lambda s.\,\mathsf{F})) = (\lambda s.\,\mathsf{F})$

`WLP_PROP1` $\vdash \forall c.\ \mathsf{WLP}\,(c, (\lambda s.\,\mathsf{F})) = (\lambda s.\,\neg(\exists s'.\ c\,(s,s')))$

`WP_PROP2` $\vdash \forall p\,q\,c.\,(\forall s.\ p\,s \supset q\,s) \supset (\forall s.\ \mathsf{WP}\,(c,p)\,s \supset \mathsf{WP}\,(c,q)\,s)$

`WLP_PROP2` $\vdash \forall p\,q\,c.\,(\forall s.\ p\,s \supset q\,s) \supset (\forall s.\ \mathsf{WLP}\,(c,p)\,s \supset \mathsf{WLP}\,(c,q)\,s)$

`WP_PROP3` $\vdash \forall p\,q\,c\,s.\ \mathsf{WP}\,(c,p)\,s \wedge \mathsf{WP}\,(c,q)\,s = \mathsf{WP}\,(c, (\lambda s.\ p\,s \wedge q\,s))\,s$

`WLP_PROP3` $\vdash \forall p\,q\,c\,s.\ \mathsf{WLP}\,(c,p)\,s \wedge \mathsf{WLP}\,(c,q)\,s = \mathsf{WLP}\,(c, (\lambda s.\ p\,s \wedge q\,s))\,s$

`WP_PROP4` $\vdash \forall p\,q\,c\,s.\ \mathsf{WP}\,(c,p)\,s \vee \mathsf{WP}\,(c,q)\,s \supset \mathsf{WP}\,(c, (\lambda s.\ p\,s \vee q\,s))\,s$

`WLP_PROP4` $\vdash \forall p\,q\,c\,s.\ \mathsf{WLP}\,(c,p)\,s \vee \mathsf{WLP}\,(c,q)\,s \supset \mathsf{WLP}\,(c, (\lambda s.\ p\,s \vee q\,s))\,s$

`WP_PROP4'` $\vdash \forall p\,q\,c.\ \mathsf{DET}\ c \supset (\forall s.\ \mathsf{WP}\,(c,p)\,s \vee \mathsf{WP}\,(c,q)\,s = \mathsf{WP}\,(c, (\lambda s.\ p\,s \vee$
$q\,s))\,s)$

`WLP_PROP4'` $\vdash \forall p\,q\,c.\ \mathsf{DET}\ c \supset (\forall s.\ \mathsf{WLP}\,(c,p)\,s \vee$
$\mathsf{WLP}\,(c,q)\,s = \mathsf{WLP}\,(c, (\lambda s.\ p\,s \vee q\,s))\,s)$

`SKIP_WP` $\vdash \forall p.\ \mathsf{WP}\,(\mathsf{MK\_SKIP}, p) = p$

`SKIP_WLP` $\vdash \forall p.\ \mathsf{WLP}\,(\mathsf{MK\_SKIP}, p) = p$

`ABORT_WP` $\vdash \forall p.\ \mathsf{WP}\,(\mathsf{MK\_ABORT}, p) = (\lambda s.\,\mathsf{F})$

`ABORT_WLP` $\vdash \forall p.\ \mathsf{WLP}\,(\mathsf{MK\_ABORT}, p) = (\lambda s.\,\mathsf{T})$

`ASSIGN_WP` $\vdash \forall x\,f\,p.\ \mathsf{WP}\,(\mathsf{MK\_ASSIGN}\,(x,f), p) = (\lambda s.\ p\,(\mathsf{BND}\,(f\,s)\,x\,s))$

`ASSIGN_WLP` $\vdash \forall x\,f\,p.\ \mathsf{WLP}\,(\mathsf{MK\_ASSIGN}\,(x,f), p) = (\lambda s.\ p\,(\mathsf{BND}\,(f\,s)\,x\,s))$

`SEQ_WP` $\vdash \forall c_1\,c_2\,p.\ \mathsf{DET}\ c_1 \supset$
$(\forall s.\ \mathsf{WP}\,(\mathsf{MK\_SEQ}\,(c_1,c_2), p)\,s = \mathsf{WP}\,(c_1, \mathsf{WP}\,(c_2,p))\,s)$

`SEQ_WLP` $\vdash \forall c_1\,c_2\,p\,s.\ \mathsf{WLP}\,(\mathsf{MK\_SEQ}\,(c_1,c_2), p)\,s = \mathsf{WLP}\,(c_1, \mathsf{WLP}\,(c_2,p))\,s$

`IF1_WP` $\vdash \forall c\,b\,p\,s.\ \mathsf{WP}\,(\mathsf{MK\_IF1}\,(b,c), p)\,s = (b\,s \Rightarrow \mathsf{WP}\,(c,p)\,s \mid p\,s)$

`IF1_WLP` $\vdash \forall c\,b\,p\,s.\ \mathsf{WLP}\,(\mathsf{MK\_IF1}\,(b,c), p)\,s = (b\,s \Rightarrow \mathsf{WLP}\,(c,p)\,s \mid p\,s)$

`IF2_WP` $\vdash \forall c_1\,c_2\,p\,s.\ \mathsf{WP}\,(\mathsf{MK\_IF2}\,(b,c_1,c_2), p)\,s =$
$(b\,s \Rightarrow \mathsf{WP}\,(c_1,p)\,s \mid \mathsf{WP}\,(c_2,p)\,s)$

`IF2_WLP` $\vdash \forall c_1\,c_2\,p\,s.\ \mathsf{WLP}\,(\mathsf{MK\_IF2}\,(b,c_1,c_2), p)\,s =$
$(b\,s \Rightarrow \mathsf{WLP}\,(c_1,p)\,s \mid \mathsf{WLP}\,(c_2,p)\,s)$

`ITER_CLAUSES` $\vdash (\mathsf{ITER}\ 0\,(b,c)\,(s,s') = \neg b\,s \wedge (s = s')) \wedge$
$(\mathsf{ITER}\,(\mathsf{SUC}\,n)\,(b,c)\,(s,s') = b\,s \wedge \mathsf{MK\_SEQ}\,(c, \mathsf{ITER}\,n\,(b,c))\,(s,s'))$

`WHILE_ITER1` $\vdash \mathsf{MK\_WHILE}\,(b,c)\,(s,s') \supset (\exists n.\ \mathsf{ITER}\,n\,(b,c)\,(s,s'))$

`WHILE_ITER2` $\vdash \forall n\,s\,s'.\ \mathsf{ITER}\,n\,(b,c)\,(s,s') \supset \mathsf{MK\_WHILE}\,(b,c)\,(s,s')$

`WHILE_ITER` $\vdash \mathsf{MK\_WHILE}\,(b,c)\,(s,s') = (\exists n.\ \mathsf{ITER}\,n\,(b,c)\,(s,s'))$

`ITER_UNIQUE` $\vdash \mathsf{DET}\, c \supset (\forall n\, s\, s'.\, \mathsf{ITER}\, n\, (b, c)\, (s, s') \supset (\forall n'\, s''.\, \mathsf{ITER}\, n'\, (b, c)\, (s, s'') \supset (n = n')))$

`DET_ITER` $\vdash \mathsf{DET}\, c \supset (\forall n.\, \mathsf{DET}\, (\mathsf{ITER}\, n\, (b, c)))$

`WP_ITER` $\vdash \mathsf{DET}\, c \supset (\forall n\, s.\, \mathsf{WP}\, (\mathsf{ITER}\, n\, (b, c), p)\, s = \mathsf{ITER\_WP}\, n\, b\, c\, p\, s)$

`WHILE_WP` $\vdash \forall c.\, \mathsf{DET}\, c \supset (\forall b\, p\, s.\, \mathsf{WP}\, (\mathsf{MK\_WHILE}\, (b, c), p)\, s = (\exists n.\, \mathsf{ITER\_WP}\, n\, b\, c\, p\, s))$

`WLP_ITER` $\vdash \forall n\, s.\, \mathsf{WLP}\, (\mathsf{ITER}\, n\, (b, c), p)\, s = \mathsf{ITER\_WLP}\, n\, b\, c\, p\, s$

`WHILE_WLP` $\vdash \forall c\, b\, p\, s.\, \mathsf{WLP}\, (\mathsf{MK\_WHILE}\, (b, c), p)\, s = (\forall n.\, \mathsf{ITER\_WLP}\, n\, b\, c\, p\, s)$

5.1.2 The theory `dynamic_logic`

Parents

`HOL string halts_logic dijkstra`

Constants

`->` `":((string -> num) -> bool) -> (((string -> num) -> bool) -> ((string -> num) -> bool))"`

Box `":((string -> num) # (string -> num) -> bool) -> (((string -> num) -> bool) -> ((string -> num) -> bool))"`

Diamond `":((string -> num) # (string -> num) -> bool) -> (((string -> num) -> bool) -> ((string -> num) -> bool))"`

Valid `":((string -> num) -> bool) -> bool"`

Infixes

`->` `":((string -> num) -> bool) -> (((string -> num) -> bool) -> ((string -> num) -> bool))"`

Definitions

`Box` $\vdash \forall c\, p\, s.\, \mathsf{Box}\, c\, p\, s = (\forall s'.\, c\, (s, s') \supset p\, s')$

`Diamond` $\vdash \forall c\, p\, s.\, \mathsf{Diamond}\, c\, p\, s = \neg \mathsf{Box}\, c\, (\lambda s'.\, \neg p\, s')\, s$

`Valid` $\vdash \forall p.\, \mathsf{Valid}\, p = (\forall s.\, p\, s)$

`-->` $\vdash \forall p\, q.\, p --> q = (\lambda s.\, p\, s \supset q\, s)$

Theorems

`Box_MK_SPEC` $\vdash \mathsf{Valid}\, (\mathsf{Box}\, c\, p) = \mathsf{MK\_SPEC}\, ((\lambda s.\, \mathsf{T}), c, p)$

`Diamond_HALTS` $\vdash \mathsf{DET}\, c \supset (\mathsf{Valid}\, (\mathsf{Diamond}\, c\, p) = \mathsf{T\_SPEC}\, ((\lambda s.\, \mathsf{T}), c, p))$

`Diamond_WP` $\vdash \mathsf{DET}\, c \supset (\mathsf{WP}\, (c, q)\, s = \mathsf{Diamond}\, c\, q\, s)$

`Box_MK_SPEC1` $\vdash \mathsf{Valid}\, (p --> \mathsf{Box}\, c\, q) = \mathsf{MK\_SPEC}\, (p, c, q)$

`Diamond_HALTS1` $\vdash \mathsf{DET}\, c \supset (\mathsf{Valid}\, (p --> \mathsf{Diamond}\, c\, q) = \mathsf{T\_SPEC}\, (p, c, q))$

`DET_THM` $\vdash \forall c\, q.\, \mathsf{DET}\, c \supset \mathsf{Valid}\, (\mathsf{Diamond}\, c\, q --> \mathsf{Box}\, c\, q)$

5.1.3 The theory `halts`

Parents

`HOL string semantics hoare_thms`

Constants

HALTS `":((string -> num) -> bool) -> (((string -> num) # (string -> num) -> bool) -> bool)"`

DEC_SEQ `":num -> ((string -> num) -> (((string -> num) # (string -> num) -> bool) -> (((string -> num) -> bool) -> (string -> num))))"`

Definitions

`HALTS` $\vdash \forall p\, c.\, \mathsf{HALTS}\, p\, c = (\forall s.\, p\, s \supset (\exists s'.\, c\,(s, s')))$

`DEC_SEQ` $\vdash (\forall s\, c\, b.\, \mathsf{DEC\_SEQ}\, 0\, s\, c\, b = s) \wedge (\forall n\, s\, c\, b.\, \mathsf{DEC\_SEQ}\,(\mathsf{SUC}\, n)\, s\, c\, b = \mathbf{let} y = \mathsf{DEC\_SEQ}\, n\, s\, c\, b \mathbf{in} (\neg b\, y \Rightarrow y \mid (\varepsilon z.\, c\,(y, z))))$

Theorems

`ASSIGN_HALTS` $\vdash \forall p\, x\, f.\, \mathsf{HALTS}\, p\,(\mathsf{MK\_ASSIGN}\,(x, f))$

`SEQ_HALTS` $\vdash \forall p\, c_1\, c_2\, q.\, \mathsf{HALTS}\, p\, c_1 \wedge \mathsf{MK\_SPEC}\,(p, c_1, q) \wedge \mathsf{HALTS}\, q\, c_2 \supset \mathsf{HALTS}\, p\,(\mathsf{MK\_SEQ}\,(c_1, c_2))$

`IF1_HALTS` $\vdash \forall p\, c\, b.\, \mathsf{HALTS}\,(\lambda s.\, p\, s \wedge b\, s)\, c \supset \mathsf{HALTS}\, p\,(\mathsf{MK\_IF1}\,(b, c))$

`IF2_HALTS` $\vdash \forall p\, c_1\, c_2\, b.\, \mathsf{HALTS}\,(\lambda s.\, p\, s \wedge b\, s)\, c_1 \wedge \mathsf{HALTS}\,(\lambda s.\, p\, s \wedge \neg b\, s)\, c_2 \supset \mathsf{HALTS}\, p\,(\mathsf{MK\_IF2}\,(b, c_1, c_2))$

`PRE_STRENGTH_HALTS` $\vdash \forall p\, p'\, c.\, (\forall s.\, p'\, s \supset p\, s) \wedge \mathsf{HALTS}\, p\, c \supset \mathsf{HALTS}\, p'\, c$

`SPEC_LEMMA1` $\vdash (\exists x.\, (\forall n.\, \mathsf{MK\_SPEC}\,((\lambda s.\, p\, s \wedge b\, s \wedge (s\, x = n)), c, (\lambda s.\, p\, s \wedge s\, x < n)))) \supset \mathsf{MK\_SPEC}\,((\lambda s.\, p\, s \wedge b\, s), c, p)$

`SEQ_LEMMA1` $\vdash (\forall n.\, b\,(\mathsf{DEC\_SEQ}\, n\, s\, c\, b)) \wedge \mathsf{MK\_SPEC}\,((\lambda s.\, p\, s \wedge b\, s), c, p) \wedge \mathsf{HALTS}\,(\lambda s.\, p\, s \wedge b\, s)\, c \wedge p\, s \supset (\forall m.\, p\,(\mathsf{DEC\_SEQ}\, m\, s\, c\, b) \wedge c\,(\mathsf{DEC\_SEQ}\, m\, s\, c\, b, \mathsf{DEC\_SEQ}\,(\mathsf{SUC}\, m)\, s\, c\, b))$

`SEQ_LEMMA2` $\vdash (\forall n.\, b\,(\mathsf{DEC\_SEQ}\, n\, s\, c\, b)) \wedge (\exists x.\, (\forall n.\, \mathsf{MK\_SPEC}\,((\lambda s.\, p\, s \wedge b\, s \wedge (s\, x = n)), c, (\lambda s.\, p\, s \wedge s\, x < n)))) \wedge \mathsf{HALTS}\,(\lambda s.\, p\, s \wedge b\, s)\, c \wedge p\, s \supset (\exists x.\, (\forall m.\, \mathsf{DEC\_SEQ}\, m\, s\, c\, b\, x > \mathsf{DEC\_SEQ}\,(\mathsf{SUC}\, m)\, s\, c\, b\, x))$

`WF_LEMMA` $\vdash (\forall n.\, f\, n > f\,(\mathsf{SUC}\, n)) \supset (\forall n\, m.\, f\, m > n)$

`WF_THM` $\vdash \neg(\exists f.\, (\forall n.\, f\, n > f\,(\mathsf{SUC}\, n)))$

`SEQ_LEMMA3` $\vdash (\exists x.\, (\forall n.\, \mathsf{MK\_SPEC}\,((\lambda s.\, p\, s \wedge b\, s \wedge (s\, x = n)), c, (\lambda s.\, p\, s \wedge s\, x < n)))) \wedge \mathsf{HALTS}\,(\lambda s.\, p\, s \wedge b\, s)\, c \wedge p\, s \supset \neg(\forall n.\, b\,(\mathsf{DEC\_SEQ}\, n\, s\, c\, b))$

`SEQ_LEMMA4` $\vdash (\exists x.\, (\forall n.\, \mathsf{MK\_SPEC}\,((\lambda s.\, p\, s \wedge b\, s \wedge (s\, x = n)), c, (\lambda s.\, p\, s \wedge s\, x < n)))) \wedge \mathsf{HALTS}\,(\lambda s.\, p\, s \wedge b\, s)\, c \wedge p\, s \supset (\exists n.\, \neg b\,(\mathsf{DEC\_SEQ}\, n\, s\, c\, b))$

`SEQ_LEMMA5` $\vdash (\exists x.\, (\forall n.\, \mathsf{MK\_SPEC}\,((\lambda s.\, p\, s \wedge b\, s \wedge (s\, x = n)), c, (\lambda s.\, p\, s \wedge s\, x < n)))) \wedge \mathsf{HALTS}\,(\lambda s.\, p\, s \wedge b\, s)\, c \wedge p\, s \supset (\exists n.\, \neg b\,(\mathsf{DEC\_SEQ}\, n\, s\, c\, b) \wedge (\forall m.\, m < n \supset b\,(\mathsf{DEC\_SEQ}\, m\, s\, c\, b)))$

`SEQ_LEMMA6` $\vdash \forall n.\, \mathsf{DEC\_SEQ}\, n\,(\mathsf{DEC\_SEQ}\,(\mathsf{SUC}\, 0)\, s\, c\, b)\, c\, b = \mathsf{DEC\_SEQ}\,(\mathsf{SUC}\, n)\, s\, c\, b$

`SEQ_LEMMA7` $\vdash \mathsf{MK\_SPEC}\,((\lambda s.\, p\, s \wedge b\, s), c, p) \wedge \mathsf{HALTS}\,(\lambda s.\, p\, s \wedge b\, s)\, c \supset (\forall n\, s.\, p\, s \wedge \neg b\,(\mathsf{DEC\_SEQ}\, n\, s\, c\, b) \wedge (\forall m.\, m < n \supset b\,(\mathsf{DEC\_SEQ}\, m\, s\, c\, b)) \supset \mathsf{MK\_FINITE\_WHILE}\,(\mathsf{SUC}\, n)\,(b, c)\,(s, \mathsf{DEC\_SEQ}\, n\, s\, c\, b))$

`WHILE_HALTS` $\vdash \forall b\, c\, x.\, (\forall n.\, \mathsf{MK\_SPEC}\,((\lambda s.\, p\, s \wedge b\, s \wedge (s\, x = n)), c, (\lambda s.\, p\, s \wedge s\, x < n))) \wedge \mathsf{HALTS}\,(\lambda s.\, p\, s \wedge b\, s)\, c \supset \mathsf{HALTS}\, p\,(\mathsf{MK\_WHILE}\,(b, c))$

5.1.4 The theory `halts_logic`

Parents

`HOL string hoare_thms halts_thms`

5.1.5 The theory `halts_thms`

Parents

`HOL string hoare_thms halts`

Constants

T_SPEC `":((string -> num) -> bool) # (((string -> num) # (string -> num) -> bool) # ((string -> num) -> bool)) -> bool"`

Definitions

`T_SPEC` $\vdash \forall p\, c\, q.\, \mathsf{T\_SPEC}\,(p, c, q) = \mathsf{MK\_SPEC}\,(p, c, q) \wedge \mathsf{HALTS}\, p\, c$

Theorems

`ASSIGN_T_THM` $\vdash \forall p\,x\,f.\, \mathsf{T\_SPEC}\,((\lambda s.\, p\,(\mathsf{BND}\,(f\,s)\,x\,s)), \mathsf{MK\_ASSIGN}\,(x,f), p)$

`SEQ_T_THM` $\vdash \forall p\,q\,r\,c_1\,c_2.\, \mathsf{T\_SPEC}\,(p, c_1, q) \wedge \mathsf{T\_SPEC}\,(q, c_2, r) \supset \mathsf{T\_SPEC}\,(p, \mathsf{MK\_SEQ}\,(c_1, c_2), r)$

`IF1_T_THM` $\vdash \forall p\,q\,c\,b.\, \mathsf{T\_SPEC}\,((\lambda s.\, p\,s \wedge b\,s), c, q) \wedge (\forall s.\, p\,s \wedge \neg b\,s \supset q\,s) \supset \mathsf{T\_SPEC}\,(p, \mathsf{MK\_IF1}\,(b, c), q)$

`IF2_T_THM` $\vdash \forall p\,q\,c_1\,c_2\,b.\, \mathsf{T\_SPEC}\,((\lambda s.\, p\,s \wedge b\,s), c_1, q) \wedge \mathsf{T\_SPEC}\,((\lambda s.\, p\,s \wedge \neg b\,s), c_2, q) \supset \mathsf{T\_SPEC}\,(p, \mathsf{MK\_IF2}\,(b, c_1, c_2), q)$

`PRE_STRENGTH_T_THM` $\vdash \forall p\,p'\,q\,c.\, (\forall s.\, p'\,s \supset p\,s) \wedge \mathsf{T\_SPEC}\,(p, c, q) \supset \mathsf{T\_SPEC}\,(p', c, q)$

`POST_WEAK_T_THM`
$\vdash \forall p\,q\,q'\,c.\, \mathsf{T\_SPEC}\,(p, c, q) \wedge (\forall s.\, q\,s \supset q'\,s) \supset \mathsf{T\_SPEC}\,(p, c, q')$

`WHILE_T_THM` $\vdash \forall p\,c\,b.\, (\forall n.\, \mathsf{T\_SPEC}\,((\lambda s.\, p\,s \wedge b\,s \wedge (s\,x = n)), c, (\lambda s.\, p\,s \wedge s\,x < n))) \supset \mathsf{T\_SPEC}\,(p, \mathsf{MK\_WHILE}\,(b, c), (\lambda s.\, p\,s \wedge \neg b\,s))$

5.1.6 The theory `hoare_thms`

Parents

`HOL string semantics`

Theorems

`ASSIGN_THM` $\vdash \forall p\,x\,f.\, \mathsf{MK\_SPEC}\,((\lambda s.\, p\,(\mathsf{BND}\,(f\,s)\,x\,s)), \mathsf{MK\_ASSIGN}\,(x,f), p)$

`BND_THM1` $\vdash \forall e\,x\,s.\, \mathsf{BND}\,e\,x\,s\,x = e$

`BND_THM2` $\vdash \forall e\,x\,s\,y.\, \neg(y = x) \supset (\mathsf{BND}\,e\,x\,s\,y = s\,y)$

`SEQ_THM` $\vdash \forall p\,q\,r\,c_1\,c_2.\, \mathsf{MK\_SPEC}\,(p, c_1, q) \wedge \mathsf{MK\_SPEC}\,(q, c_2, r) \supset \mathsf{MK\_SPEC}\,(p, \mathsf{MK\_SEQ}\,(c_1, c_2), r)$

`IF1_THM` $\vdash \forall p\,q\,c\,b.\, \mathsf{MK\_SPEC}\,((\lambda s.\, p\,s \wedge b\,s), c, q) \wedge (\forall s.\, p\,s \wedge \neg b\,s \supset q\,s) \supset \mathsf{MK\_SPEC}\,(p, \mathsf{MK\_IF1}\,(b, c), q)$

`IF2_THM` $\vdash \forall p\,q\,c_1\,c_2\,b.\, \mathsf{MK\_SPEC}\,((\lambda s.\, p\,s \wedge b\,s), c_1, q) \wedge \mathsf{MK\_SPEC}\,((\lambda s.\, p\,s \wedge \neg b\,s), c_2, q) \supset \mathsf{MK\_SPEC}\,(p, \mathsf{MK\_IF2}\,(b, c_1, c_2), q)$

`PRE_STRENGTH_THM`
$\vdash \forall p\,p'\,q\,c.\, (\forall s.\, p'\,s \supset p\,s) \wedge \mathsf{MK\_SPEC}\,(p, c, q) \supset \mathsf{MK\_SPEC}\,(p', c, q)$

`POST_WEAK_THM`
$\vdash \forall p\,q\,q'\,c.\, \mathsf{MK\_SPEC}\,(p, c, q) \wedge (\forall s.\, q\,s \supset q'\,s) \supset \mathsf{MK\_SPEC}\,(p, c, q')$

`MK_FINITE_WHILE_CLAUSES` $\vdash (\mathsf{MK\_FINITE\_WHILE}\,0\,(p,c)\,(s,s') = \mathsf{F}) \wedge (\mathsf{MK\_FINITE\_WHILE}\,(\mathsf{SUC}\,n)\,(p,c)\,(s,s') = \mathsf{MK\_IF1}\,(p, \mathsf{MK\_SEQ}\,(c, \mathsf{MK\_FINITE\_WHILE}\,n\,(p,c)))\,(s,s'))$

`WHILE_THM` $\vdash \forall p\,c\,b.\, \mathsf{MK\_SPEC}\,((\lambda s.\, p\,s \wedge b\,s), c, p) \supset \mathsf{MK\_SPEC}\,(p, \mathsf{MK\_WHILE}\,(b, c), (\lambda s.\, p\,s \wedge \neg b\,s))$

5.1.7 The theory `semantics`

Parents

`HOL string`

Constants

DET	`":((string -> num) # (string -> num) -> bool) -> bool"`
MK_SKIP	`":(string -> num) # (string -> num) -> bool"`
MK_ABORT	`":(string -> num) # (string -> num) -> bool"`
BND	`":num -> (string -> ((string -> num) -> (string -> num)))"`
MK_ASSIGN	`":string # ((string -> num) -> num) -> ((string -> num) # (string -> num) -> bool)"`
MK_IF1	`":((string -> num) -> bool) # ((string -> num) # (string -> num) -> bool) -> ((string -> num) # (string -> num) -> bool)"`
MK_IF2	`":((string -> num) -> bool) # (((string -> num) # (string -> num) -> bool) # ((string -> num) # (string -> num) -> bool)) -> ((string -> num) # (string -> num) -> bool)"`

MK_SEQ `":((string -> num) # (string -> num) -> bool) # ((string -> num) # (string -> num) -> bool) -> ((string -> num) # (string -> num) -> bool)"`

MK_SEQL `":((string -> num) # (string -> num) -> bool)list -> ((string -> num) # (string -> num) -> bool)"`

MK_FINITE_WHILE `":num -> (((string -> num) -> bool) # ((string -> num) # (string -> num) -> bool) -> ((string -> num) # (string -> num) -> bool))"`

MK_WHILE `":((string -> num) -> bool) # ((string -> num) # (string -> num) -> bool) -> ((string -> num) # (string -> num) -> bool)"`

MK_SPEC `":((string -> num) -> bool) # (((string -> num) # (string -> num) -> bool) # ((string -> num) -> bool)) -> bool"`

MK_ASSERT `":((string -> num) -> bool) -> ((string -> num) # (string -> num) -> bool)"`

MK_INVARIANT `":((string -> num) -> bool) -> ((string -> num) # (string -> num) -> bool)"`

MK_VARIANT `":((string -> num) -> num) -> ((string -> num) # (string -> num) -> bool)"`

Definitions

`DET` $\vdash \forall c.\, \mathsf{DET}\, c = (\forall s\, s_1\, s_2.\, c\,(s, s_1) \wedge c\,(s, s_2) \supset (s_1 = s_2))$

`MK_SKIP` $\vdash \forall s\, s'.\, \mathsf{MK\_SKIP}\,(s, s') = (s = s')$

`MK_ABORT` $\vdash \forall s\, s'.\, \mathsf{MK\_ABORT}\,(s, s') = \mathsf{F}$

`BND` $\vdash \forall n\, x\, s.\, \mathsf{BND}\, n\, x\, s = (\lambda z.\, ((z = x) \Rightarrow n \mid s\, z))$

`MK_ASSIGN` $\vdash \forall x\, f\, s\, s'.\, \mathsf{MK\_ASSIGN}\,(x, f)\,(s, s') = (s' = \mathsf{BND}\,(f\, s)\, x\, s)$

`MK_IF1` $\vdash \forall p\, c\, s\, s'.\, \mathsf{MK\_IF1}\,(p, c)\,(s, s') = (p\, s \Rightarrow c\,(s, s') \mid (s = s'))$

`MK_IF2` $\vdash \forall p\, c_1\, c_2\, s\, s'.\, \mathsf{MK\_IF2}\,(p, c_1, c_2)\,(s, s') = (p\, s \Rightarrow c_1\,(s, s') \mid c_2\,(s, s'))$

`MK_SEQ` $\vdash \forall c_1\, c_2\, s\, s'.\, \mathsf{MK\_SEQ}\,(c_1, c_2)\,(s, s') = (\exists s''.\, c_1\,(s, s'') \wedge c_2\,(s'', s'))$

`MK_SEQL` $\vdash (\mathsf{MK\_SEQL}\,[\,] = (\lambda s, s'.\, s = s')) \wedge$
$(\forall c\, cl.\, \mathsf{MK\_SEQL}\,(\mathsf{CONS}\, c\, cl) = \mathsf{MK\_SEQ}\,(c, \mathsf{MK\_SEQL}\, cl))$

`MK_FINITE_WHILE` $\vdash (\mathsf{MK\_FINITE\_WHILE}\, 0 = (\lambda (p, c)\, s, s'.\, \mathsf{F})) \wedge$
$(\forall n.\, \mathsf{MK\_FINITE\_WHILE}\,(\mathsf{SUC}\, n) =$
$(\lambda p, c.\, \mathsf{MK\_IF1}\,(p, \mathsf{MK\_SEQ}\,(c, \mathsf{MK\_FINITE\_WHILE}\, n\,(p, c)))))$

`MK_WHILE` $\vdash \forall p\, c\, s\, s'.$
$\mathsf{MK\_WHILE}\,(p, c)\,(s, s') = (\exists n.\, \mathsf{MK\_FINITE\_WHILE}\, n\,(p, c)\,(s, s'))$

`MK_SPEC` $\vdash \forall p\, c\, q.\, \mathsf{MK\_SPEC}\,(p, c, q) = (\forall s_1\, s_2.\, p\, s_1 \wedge c\,(s_1, s_2) \supset q\, s_2)$

`MK_ASSERT` $\vdash \forall p\, s_1\, s_2.\, \mathsf{MK\_ASSERT}\, p\,(s_1, s_2) = p\, s_1 \wedge (s_1 = s_2)$

`MK_INVARIANT` $\vdash \forall p\, s_1\, s_2.\, \mathsf{MK\_INVARIANT}\, p\,(s_1, s_2) = p\, s_1 \wedge (s_1 = s_2)$

`MK_VARIANT` $\vdash \forall p\, s_1\, s_2.\, \mathsf{MK\_VARIANT}\, p\,(s_1, s_2) = p\, s_1 > p\, s_2 \wedge (s_1 = s_2)$

5.2 The Theory `sets`

This is the new sets theory which has special syntax supported by the HOL parser. For example, `{ }` represents the empty set and the set of first three natural number can be written as `{0,1,2}`.

This library contains several conversions for use with sets. There is also a set induction tactic `SET_INDUCT_TAC`. They are list after the listing of the theory.

Parents

`HOL`

Types

`":(*)set"`

Constants

IN	`":* -> ((*)set -> bool)"`
SUBSET	`":(*)set -> ((*)set -> bool)"`
PSUBSET	`":(*)set -> ((*)set -> bool)"`
UNION	`":(*)set -> ((*)set -> (*)set)"`
INTER	`":(*)set -> ((*)set -> (*)set)"`
DIFF	`":(*)set -> ((*)set -> (*)set)"`
INSERT	`":* -> ((*)set -> (*)set)"`
DELETE	`":(*)set -> (* -> (*)set)"`
CHF	`":(*)set -> (* -> bool)"`
SPEC	`":(* -> bool) -> (*)set"`
GSPEC	`":(** -> * # bool) -> (*)set"`
EMPTY	`":(*)set"`
UNIV	`":(*)set"`
DISJOINT	`":(*)set -> ((*)set -> bool)"`
CHOICE	`":(*)set -> *"`
REST	`":(*)set -> (*)set"`
SING	`":(*)set -> bool"`
IMAGE	`":(* -> **) -> ((*)set -> (**)set)"`
FINITE	`":(*)set -> bool"`
CARD	`":(*)set -> num"`
INFINITE	`":(*)set -> bool"`

Infixes

IN	`":* -> ((*)set -> bool)"`
SUBSET	`":(*)set -> ((*)set -> bool)"`
PSUBSET	`":(*)set -> ((*)set -> bool)"`
UNION	`":(*)set -> ((*)set -> (*)set)"`
INTER	`":(*)set -> ((*)set -> (*)set)"`
DIFF	`":(*)set -> ((*)set -> (*)set)"`
INSERT	`":* -> ((*)set -> (*)set)"`
DELETE	`":(*)set -> (* -> (*)set)"`

Definitions

`CARD_DEF`	$\vdash (\mathsf{CARD}\,\{\,\} = 0) \wedge (\forall s.\, \mathsf{FINITE}\, s \supset$ $(\forall x.\, \mathsf{CARD}\,(x\ \mathsf{INSERT}\ s) = (x\ \mathsf{IN}\ s \Rightarrow \mathsf{CARD}\, s \mid \mathsf{SUC}\,(\mathsf{CARD}\, s))))$
`CHOICE_DEF`	$\vdash \forall s.\, \neg(s = \{\,\}) \supset \mathsf{CHOICE}\, s\ \mathsf{IN}\ s$
`DELETE_DEF`	$\vdash \forall s\, x.\, s\ \mathsf{DELETE}\ x = s\ \mathsf{DIFF}\ \{x\}$
`DIFF_DEF`	$\vdash \forall s\, t.\, s\ \mathsf{DIFF}\ t = \{x \mid x\ \mathsf{IN}\ s \wedge \neg x\ \mathsf{IN}\ t\}$
`DISJOINT_DEF`	$\vdash \forall s\, t.\, \mathsf{DISJOINT}\, s\, t = (s\ \mathsf{INTER}\ t = \{\,\})$
`EMPTY_DEF`	$\vdash \{\,\} = \mathsf{SPEC}\,(\lambda x.\, \mathsf{F})$
`FINITE_DEF`	$\vdash \forall s.\, \mathsf{FINITE}\, s = (\forall P.\, P\,\{\,\} \wedge$ $(\forall s'.\, P\, s' \supset (\forall e.\, P\,(e\ \mathsf{INSERT}\ s'))) \supset P\, s)$
`GSPEC_DEF`	$\vdash \forall f.\, \mathsf{GSPEC}\, f = \mathsf{SPEC}\,(\lambda x.\, (\exists y.\, x, \mathsf{T} = f\, y))$
`IMAGE_DEF`	$\vdash \forall f\, s.\, \mathsf{IMAGE}\, f\, s = \{f\, x \mid x\ \mathsf{IN}\ s\}$
`INFINITE_DEF`	$\vdash \forall s.\, \mathsf{INFINITE}\, s = \neg \mathsf{FINITE}\, s$
`INSERT_DEF`	$\vdash \forall x\, s.\, x\ \mathsf{INSERT}\ s = \{y \mid (y = x) \vee y\ \mathsf{IN}\ s\}$
`INTER_DEF`	$\vdash \forall s\, t.\, s\ \mathsf{INTER}\ t = \{x \mid x\ \mathsf{IN}\ s \wedge x\ \mathsf{IN}\ t\}$
`IN_DEF`	$\vdash \forall x\, s.\, x\ \mathsf{IN}\ s = \mathsf{CHF}\, s\, x$
`PSUBSET_DEF`	$\vdash \forall s\, t.\, s\ \mathsf{PSUBSET}\ t = s\ \mathsf{SUBSET}\ t \wedge \neg(s = t)$
`REST_DEF`	$\vdash \forall s.\, \mathsf{REST}\, s = s\ \mathsf{DELETE}\ \mathsf{CHOICE}\, s$
`SING_DEF`	$\vdash \forall s.\, \mathsf{SING}\, s = (\exists x.\, s = \{x\})$
`SUBSET_DEF`	$\vdash \forall s\, t.\, s\ \mathsf{SUBSET}\ t = (\forall x.\, x\ \mathsf{IN}\ s \supset x\ \mathsf{IN}\ t)$
`UNION_DEF`	$\vdash \forall s\, t.\, s\ \mathsf{UNION}\ t = \{x \mid x\ \mathsf{IN}\ s \vee x\ \mathsf{IN}\ t\}$

`UNIV_DEF` $\vdash \mathsf{UNIV} = \mathsf{SPEC}\,(\lambda x.\,\mathsf{T})$

`set_ISO_DEF` $\vdash (\forall a.\,\mathsf{SPEC}\,(\mathsf{CHF}\,a) = a) \wedge (\forall r.\,(\lambda p.\,\mathsf{T})\,r = (\mathsf{CHF}\,(\mathsf{SPEC}\,r) = r))$

`set_TY_DEF` $\vdash \exists rep.\,\mathsf{TYPE\_DEFINITION}\,(\lambda p.\,\mathsf{T})\,rep$

Theorems

`ABSORPTION` $\vdash \forall x\,s.\,x\,\mathsf{IN}\,s = (x\,\mathsf{INSERT}\,s = s)$

`CARD_DELETE`
$\vdash \forall s.\,\mathsf{FINITE}\,s \supset (\forall x.\,\mathsf{CARD}\,(s\,\mathsf{DELETE}\,x) = (x\,\mathsf{IN}\,s \Rightarrow (\mathsf{CARD}\,s - 1) \mid \mathsf{CARD}\,s))$

`CARD_EMPTY` $\vdash \mathsf{CARD}\,\{\,\} = 0$

`CARD_EQ_0` $\vdash \forall s.\,\mathsf{FINITE}\,s \supset ((\mathsf{CARD}\,s = 0) = (s = \{\,\}))$

`CARD_INSERT` $\vdash \forall s.\,\mathsf{FINITE}\,s \supset$
$(\forall x.\,\mathsf{CARD}\,(x\,\mathsf{INSERT}\,s) = (x\,\mathsf{IN}\,s \Rightarrow \mathsf{CARD}\,s \mid \mathsf{SUC}\,(\mathsf{CARD}\,s)))$

`CARD_INTER_LESS_EQ` $\vdash \forall s.\,\mathsf{FINITE}\,s \supset (\forall t.\,\mathsf{CARD}\,(s\,\mathsf{INTER}\,t) \leq \mathsf{CARD}\,s)$

`CARD_PSUBSET` $\vdash \forall s.\,\mathsf{FINITE}\,s \supset (\forall t.\,t\,\mathsf{PSUBSET}\,s \supset \mathsf{CARD}\,t < \mathsf{CARD}\,s)$

`CARD_SING` $\vdash \forall x.\,\mathsf{CARD}\,\{x\} = 1$

`CARD_SUBSET` $\vdash \forall s.\,\mathsf{FINITE}\,s \supset (\forall t.\,t\,\mathsf{SUBSET}\,s \supset \mathsf{CARD}\,t \leq \mathsf{CARD}\,s)$

`CARD_UNION` $\vdash \forall s.\,\mathsf{FINITE}\,s \supset (\forall t.\,\mathsf{FINITE}\,t \supset (\mathsf{CARD}\,(s\,\mathsf{UNION}\,t) +$
$\mathsf{CARD}\,(s\,\mathsf{INTER}\,t) = \mathsf{CARD}\,s + \mathsf{CARD}\,t))$

`CHOICE_INSERT_REST` $\vdash \forall s.\,\neg(s = \{\,\}) \supset (\mathsf{CHOICE}\,s\,\mathsf{INSERT}\,\mathsf{REST}\,s = s)$

`CHOICE_NOT_IN_REST` $\vdash \forall s.\,\neg\mathsf{CHOICE}\,s\,\mathsf{IN}\,\mathsf{REST}\,s$

`CHOICE_SING` $\vdash \forall x.\,\mathsf{CHOICE}\,\{x\} = x$

`COMPONENT` $\vdash \forall x.\,x\,\mathsf{IN}\,(x\,\mathsf{INSERT}\,s)$

`DECOMPOSITION` $\vdash \forall s\,x.\,x\,\mathsf{IN}\,s = (\exists t.\,(s = x\,\mathsf{INSERT}\,t) \wedge \neg x\,\mathsf{IN}\,t)$

`DELETE_COMM` $\vdash \forall x\,y\,s.\,(s\,\mathsf{DELETE}\,x)\,\mathsf{DELETE}\,y = (s\,\mathsf{DELETE}\,y)\,\mathsf{DELETE}\,x$

`DELETE_DELETE` $\vdash \forall x\,s.\,(s\,\mathsf{DELETE}\,x)\,\mathsf{DELETE}\,x = s\,\mathsf{DELETE}\,x$

`DELETE_EQ_SING` $\vdash \forall s\,x.\,x\,\mathsf{IN}\,s \supset ((s\,\mathsf{DELETE}\,x = \{\,\}) = (s = \{x\}))$

`DELETE_INSERT` $\vdash \forall x\,y\,s.\,(x\,\mathsf{INSERT}\,s)\,\mathsf{DELETE}\,y =$
$((x = y) \Rightarrow s\,\mathsf{DELETE}\,y \mid x\,\mathsf{INSERT}\,(s\,\mathsf{DELETE}\,y))$

`DELETE_NON_ELEMENT` $\vdash \forall x\,s.\,\neg x\,\mathsf{IN}\,s = (s\,\mathsf{DELETE}\,x = s)$

`DELETE_SUBSET` $\vdash \forall x\,s.\,(s\,\mathsf{DELETE}\,x)\,\mathsf{SUBSET}\,s$

`DIFF_DIFF` $\vdash \forall s\,t.\,(s\,\mathsf{DIFF}\,t)\,\mathsf{DIFF}\,t = s\,\mathsf{DIFF}\,t$

`DIFF_EMPTY` $\vdash \forall s.\,s\,\mathsf{DIFF}\,\{\,\} = s$

`DIFF_EQ_EMPTY` $\vdash \forall s.\,s\,\mathsf{DIFF}\,s = \{\,\}$

`DIFF_INSERT` $\vdash \forall s\,t\,x.\,s\,\mathsf{DIFF}\,(x\,\mathsf{INSERT}\,t) = (s\,\mathsf{DELETE}\,x)\,\mathsf{DIFF}\,t$

`DIFF_UNIV` $\vdash \forall s.\,s\,\mathsf{DIFF}\,\mathsf{UNIV} = \{\,\}$

`DISJOINT_EMPTY_REFL` $\vdash \forall s.\,(s = \{\,\}) = \mathsf{DISJOINT}\,s\,s$

`DISJOINT_INSERT` $\vdash \forall x\,s\,t.\,\mathsf{DISJOINT}\,(x\,\mathsf{INSERT}\,s)\,t = \mathsf{DISJOINT}\,s\,t \wedge \neg x\,\mathsf{IN}\,t$

`DISJOINT_SING_EMPTY` $\vdash \forall x.\,\mathsf{DISJOINT}\,\{x\}\,\{\,\}$

`DISJOINT_SYM` $\vdash \forall s\,t.\,\mathsf{DISJOINT}\,s\,t = \mathsf{DISJOINT}\,t\,s$

`EMPTY_DELETE` $\vdash \forall x.\,\{\,\}\,\mathsf{DELETE}\,x = \{\,\}$

`EMPTY_DIFF` $\vdash \forall s.\,\{\,\}\,\mathsf{DIFF}\,s = \{\,\}$

`EMPTY_NOT_UNIV` $\vdash \neg(\{\,\} = \mathsf{UNIV})$

`EMPTY_SUBSET` $\vdash \forall s.\,\{\,\}\,\mathsf{SUBSET}\,s$

`EMPTY_UNION` $\vdash \forall s\,t.\,(s\,\mathsf{UNION}\,t = \{\,\}) = (s = \{\,\}) \wedge (t = \{\,\})$

`EQUAL_SING` $\vdash \forall x\,y.\,(\{x\} = \{y\}) = (x = y)$

`EQ_UNIV` $\vdash (\forall x.\,x\,\mathsf{IN}\,s) = (s = \mathsf{UNIV})$

`EXTENSION` $\vdash \forall s\,t.\,(s = t) = (\forall x.\,x\,\mathsf{IN}\,s = x\,\mathsf{IN}\,t)$

`FINITE_DELETE` $\vdash \forall x\,s.\,\mathsf{FINITE}\,(s\,\mathsf{DELETE}\,x) = \mathsf{FINITE}\,s$

`FINITE_DIFF` $\vdash \forall s.\,\mathsf{FINITE}\,s \supset (\forall t.\,\mathsf{FINITE}\,(s\,\mathsf{DIFF}\,t))$

`FINITE_EMPTY` $\vdash \mathsf{FINITE}\,\{\,\}$

`FINITE_INDUCT` $\vdash \forall P.\,P\,\{\,\} \wedge (\forall s.\,\mathsf{FINITE}\,s \wedge$
$P\,s \supset (\forall e.\,\neg e\,\mathsf{IN}\,s \supset P\,(e\,\mathsf{INSERT}\,s))) \supset (\forall s.\,\mathsf{FINITE}\,s \supset P\,s)$

`FINITE_INSERT` $\vdash \forall x\,s.\,\mathsf{FINITE}\,(x\,\mathsf{INSERT}\,s) = \mathsf{FINITE}\,s$

`FINITE_ISO_NUM` $\vdash \forall s.\, \mathsf{FINITE}\, s \supset (\exists f.\, (\forall n\, m.\, n < \mathsf{CARD}\, s \wedge m < \mathsf{CARD}\, s \supset ((f\, n = f\, m) \supset (n = m))) \wedge (s = \{f\, n \mid n < \mathsf{CARD}\, s\}))$

`FINITE_PSUBSET_INFINITE` $\vdash \forall s.\, \mathsf{INFINITE}\, s = (\forall t.\, \mathsf{FINITE}\, t \supset (t\, \mathsf{SUBSET}\, s \supset t\, \mathsf{PSUBSET}\, s))$

`FINITE_PSUBSET_UNIV` $\vdash \mathsf{INFINITE}\, \mathsf{UNIV} = (\forall s.\, \mathsf{FINITE}\, s \supset s\, \mathsf{PSUBSET}\, \mathsf{UNIV})$

`FINITE_SING` $\vdash \forall x.\, \mathsf{FINITE}\, \{x\}$

`FINITE_UNION` $\vdash \forall s\, t.\, \mathsf{FINITE}\, (s\, \mathsf{UNION}\, t) = \mathsf{FINITE}\, s \wedge \mathsf{FINITE}\, t$

`GSPECIFICATION` $\vdash \forall sp\, v.\, v\, \mathsf{IN}\, \mathsf{GSPEC}\, sp = (\exists y.\, v, \mathsf{T} = sp\, y)$

`IMAGE_11_INFINITE` $\vdash \forall f.\, (\forall x\, y.\, (f\, x = f\, y) \supset (x = y)) \supset (\forall s.\, \mathsf{INFINITE}\, s \supset \mathsf{INFINITE}\, (\mathsf{IMAGE}\, f\, s))$

`IMAGE_COMPOSE` $\vdash \forall f\, g\, s.\, \mathsf{IMAGE}\, (f \circ g)\, s = \mathsf{IMAGE}\, f\, (\mathsf{IMAGE}\, g\, s)$

`IMAGE_DELETE` $\vdash \forall f\, x\, s.\, \neg x\, \mathsf{IN}\, s \supset (\mathsf{IMAGE}\, f\, (s\, \mathsf{DELETE}\, x) = \mathsf{IMAGE}\, f\, s)$

`IMAGE_EMPTY` $\vdash \forall f.\, \mathsf{IMAGE}\, f\, \{\,\} = \{\,\}$

`IMAGE_FINITE` $\vdash \forall s.\, \mathsf{FINITE}\, s \supset (\forall f.\, \mathsf{FINITE}\, (\mathsf{IMAGE}\, f\, s))$

`IMAGE_ID` $\vdash \forall s.\, \mathsf{IMAGE}\, (\lambda x.\, x)\, s = s$

`IMAGE_INSERT` $\vdash \forall f\, x\, s.\, \mathsf{IMAGE}\, f\, (x\, \mathsf{INSERT}\, s) = f\, x\, \mathsf{INSERT}\, \mathsf{IMAGE}\, f\, s$

`IMAGE_INTER` $\vdash \forall f\, s\, t.\, \mathsf{IMAGE}\, f\, (s \mathsf{INTER} t) \mathsf{SUBSET} (\mathsf{IMAGE}\, f\, s \mathsf{INTER} \mathsf{IMAGE}\, f\, t)$

`IMAGE_IN` $\vdash \forall x\, s.\, x\, \mathsf{IN}\, s \supset (\forall f.\, f\, x\, \mathsf{IN}\, \mathsf{IMAGE}\, f\, s)$

`IMAGE_SUBSET` $\vdash \forall s\, t.\, s\, \mathsf{SUBSET}\, t \supset (\forall f.\, \mathsf{IMAGE}\, f\, s\, \mathsf{SUBSET}\, \mathsf{IMAGE}\, f\, t)$

`IMAGE_UNION` $\vdash \forall f\, s\, t.\, \mathsf{IMAGE}\, f\, (s\, \mathsf{UNION}\, t) = \mathsf{IMAGE}\, f\, s\, \mathsf{UNION}\, \mathsf{IMAGE}\, f\, t$

`INFINITE_DIFF_FINITE` $\vdash \forall s\, t.\, \mathsf{INFINITE}\, s \wedge \mathsf{FINITE}\, t \supset \neg(s\, \mathsf{DIFF}\, t = \{\,\})$

`INFINITE_SUBSET` $\vdash \forall s.\, \mathsf{INFINITE}\, s \supset (\forall t.\, s\, \mathsf{SUBSET}\, t \supset \mathsf{INFINITE}\, t)$

`INFINITE_UNIV` $\vdash \mathsf{INFINITE}\, \mathsf{UNIV} = (\exists f.\, (\forall x\, y.\, (f\, x = f\, y) \supset (x = y)) \wedge (\exists y.\, (\forall x.\, \neg(f\, x = y))))$

`INSERT_COMM` $\vdash \forall x\, y\, s.\, x\, \mathsf{INSERT}\, (y\, \mathsf{INSERT}\, s) = y\, \mathsf{INSERT}\, (x\, \mathsf{INSERT}\, s)$

`INSERT_DELETE` $\vdash \forall x\, s.\, x\, \mathsf{IN}\, s \supset (x\, \mathsf{INSERT}\, (s\, \mathsf{DELETE}\, x) = s)$

`INSERT_DIFF` $\vdash \forall s\, t\, x.\, (x \mathsf{INSERT} s) \mathsf{DIFF} t = (x \mathsf{IN} t \Rightarrow s \mathsf{DIFF} t \mid x \mathsf{INSERT} (s \mathsf{DIFF} t))$

`INSERT_INSERT` $\vdash \forall x\, s.\, x\, \mathsf{INSERT}\, (x\, \mathsf{INSERT}\, s) = x\, \mathsf{INSERT}\, s$

`INSERT_INTER` $\vdash \forall x\, s\, t.\, (x\, \mathsf{INSERT}\, s)\, \mathsf{INTER}\, t = (x\, \mathsf{IN}\, t \Rightarrow x\, \mathsf{INSERT}\, (s\, \mathsf{INTER}\, t) \mid s\, \mathsf{INTER}\, t)$

`INSERT_SING_UNION` $\vdash \forall s\, x.\, x\, \mathsf{INSERT}\, s = \{x\}\, \mathsf{UNION}\, s$

`INSERT_SUBSET` $\vdash \forall x\, s\, t.\, (x\, \mathsf{INSERT}\, s)\, \mathsf{SUBSET}\, t = x\, \mathsf{IN}\, t \wedge s\, \mathsf{SUBSET}\, t$

`INSERT_UNION_EQ` $\vdash \forall x\, s\, t.\, (x\, \mathsf{INSERT}\, s)\, \mathsf{UNION}\, t = x\, \mathsf{INSERT}\, (s\, \mathsf{UNION}\, t)$

`INSERT_UNION` $\vdash \forall x\, s\, t.\, (x\, \mathsf{INSERT}\, s)\, \mathsf{UNION}\, t = (x\, \mathsf{IN}\, t \Rightarrow s\, \mathsf{UNION}\, t \mid x\, \mathsf{INSERT}\, (s\, \mathsf{UNION}\, t))$

`INSERT_UNIV` $\vdash \forall x.\, x\, \mathsf{INSERT}\, \mathsf{UNIV} = \mathsf{UNIV}$

`INTER_ASSOC` $\vdash \forall s\, t\, u.\, (s\, \mathsf{INTER}\, t)\, \mathsf{INTER}\, u = s\, \mathsf{INTER}\, (t\, \mathsf{INTER}\, u)$

`INTER_COMM` $\vdash \forall s\, t.\, s\, \mathsf{INTER}\, t = t\, \mathsf{INTER}\, s$

`INTER_EMPTY` $\vdash (\forall s.\, \{\,\}\, \mathsf{INTER}\, s = \{\,\}) \wedge (\forall s.\, s\, \mathsf{INTER}\, \{\,\} = \{\,\})$

`INTER_FINITE` $\vdash \forall s.\, \mathsf{FINITE}\, s \supset (\forall t.\, \mathsf{FINITE}\, (s\, \mathsf{INTER}\, t))$

`INTER_IDEMPOT` $\vdash \forall s.\, s\, \mathsf{INTER}\, s = s$

`INTER_OVER_UNION` $\vdash \forall s\, t\, u.\, s\, \mathsf{UNION}\, (t\, \mathsf{INTER}\, u) = (s\, \mathsf{UNION}\, t)\, \mathsf{INTER}\, (s\, \mathsf{UNION}\, u)$

`INTER_SUBSET` $\vdash (\forall s\, t.\, (s\, \mathsf{INTER}\, t)\, \mathsf{SUBSET}\, s) \wedge (\forall s\, t.\, (t\, \mathsf{INTER}\, s)\, \mathsf{SUBSET}\, s)$

`INTER_UNIV` $\vdash (\forall s.\, \mathsf{UNIV}\, \mathsf{INTER}\, s = s) \wedge (\forall s.\, s\, \mathsf{INTER}\, \mathsf{UNIV} = s)$

`IN_DELETE_EQ` $\vdash \forall s\, x\, x'.\, (x\, \mathsf{IN}\, s = x'\, \mathsf{IN}\, s) = (x\, \mathsf{IN}\, (s\, \mathsf{DELETE}\, x') = x'\, \mathsf{IN}\, (s\, \mathsf{DELETE}\, x))$

`IN_DELETE` $\vdash \forall s\, x\, y.\, x\, \mathsf{IN}\, (s\, \mathsf{DELETE}\, y) = x\, \mathsf{IN}\, s \wedge \neg(x = y)$

`IN_DIFF` $\vdash \forall s\, t\, x.\, x\, \mathsf{IN}\, (s\, \mathsf{DIFF}\, t) = x\, \mathsf{IN}\, s \wedge \neg x\, \mathsf{IN}\, t$

`IN_DISJOINT` $\vdash \forall s\, t.\, \mathsf{DISJOINT}\, s\, t = \neg(\exists x.\, x\, \mathsf{IN}\, s \wedge x\, \mathsf{IN}\, t)$

`IN_IMAGE` $\vdash \forall y\, s\, f.\, y\, \mathsf{IN}\, \mathsf{IMAGE}\, f\, s = (\exists x.\, (y = f\, x) \wedge x\, \mathsf{IN}\, s)$

`IN_INFINITE_NOT_FINITE` $\vdash \forall s\, t.\, \mathsf{INFINITE}\, s \wedge \mathsf{FINITE}\, t \supset (\exists x.\, x\, \mathsf{IN}\, s \wedge \neg x\, \mathsf{IN}\, t)$

`IN_INSERT` $\vdash \forall x\, y\, s.\, x\, \mathsf{IN}\, (y\, \mathsf{INSERT}\, s) = (x = y) \vee x\, \mathsf{IN}\, s$

IN_INTER $\vdash \forall s\, t\, x.\, x\ \mathsf{IN}\ (s\ \mathsf{INTER}\ t) = x\ \mathsf{IN}\ s \wedge x\ \mathsf{IN}\ t$

IN_SING $\vdash \forall x\, y.\, x\ \mathsf{IN}\ \{y\} = (x = y)$

IN_UNION $\vdash \forall s\, t\, x.\, x\ \mathsf{IN}\ (s\ \mathsf{UNION}\ t) = x\ \mathsf{IN}\ s \vee x\ \mathsf{IN}\ t$

IN_UNIV $\vdash \forall x.\, x\ \mathsf{IN}\ \mathsf{UNIV}$

MEMBER_NOT_EMPTY $\vdash \forall s.\, (\exists x.\, x\ \mathsf{IN}\ s) = \neg(s = \{\,\})$

NOT_EMPTY_INSERT $\vdash \forall x\, s.\, \neg(\{\,\} = x\ \mathsf{INSERT}\ s)$

NOT_EMPTY_SING $\vdash \forall x.\, \neg(\{\,\} = \{x\})$

NOT_EQUAL_SETS $\vdash \forall s\, t.\, \neg(s = t) = (\exists x.\, x\ \mathsf{IN}\ t = \neg x\ \mathsf{IN}\ s)$

NOT_INSERT_EMPTY $\vdash \forall x\, s.\, \neg(x\ \mathsf{INSERT}\ s = \{\,\})$

NOT_IN_EMPTY $\vdash \forall x.\, \neg x\ \mathsf{IN}\ \{\,\}$

NOT_IN_FINITE $\vdash \mathsf{INFINITE}\ \mathsf{UNIV} = (\forall s.\, \mathsf{FINITE}\ s \supset (\exists x.\, \neg x\ \mathsf{IN}\ s))$

NOT_PSUBSET_EMPTY $\vdash \forall s.\, \neg s\ \mathsf{PSUBSET}\ \{\,\}$

NOT_SING_EMPTY $\vdash \forall x.\, \neg(\{x\} = \{\,\})$

NOT_UNIV_PSUBSET $\vdash \forall s.\, \neg \mathsf{UNIV}\ \mathsf{PSUBSET}\ s$

PSUBSET_FINITE $\vdash \forall s.\, \mathsf{FINITE}\ s \supset (\forall t.\, t\ \mathsf{PSUBSET}\ s \supset \mathsf{FINITE}\ t)$

PSUBSET_INSERT_SUBSET
$\vdash \forall s\, t.\, s\ \mathsf{PSUBSET}\ t = (\exists x.\, \neg x\ \mathsf{IN}\ s \wedge (x\ \mathsf{INSERT}\ s)\ \mathsf{SUBSET}\ t)$

PSUBSET_IRREFL $\vdash \forall s.\, \neg s\ \mathsf{PSUBSET}\ s$

PSUBSET_MEMBER $\vdash \forall s\, t.\, s\ \mathsf{PSUBSET}\ t = s\ \mathsf{SUBSET}\ t \wedge (\exists y.\, y\ \mathsf{IN}\ t \wedge \neg y\ \mathsf{IN}\ s)$

PSUBSET_TRANS $\vdash \forall s\, t\, u.\, s\ \mathsf{PSUBSET}\ t \wedge t\ \mathsf{PSUBSET}\ u \supset s\ \mathsf{PSUBSET}\ u$

PSUBSET_UNIV $\vdash \forall s.\, s\ \mathsf{PSUBSET}\ \mathsf{UNIV} = (\exists x.\, \neg x\ \mathsf{IN}\ s)$

REST_PSUBSET $\vdash \forall s.\, \neg(s = \{\,\}) \supset \mathsf{REST}\ s\ \mathsf{PSUBSET}\ s$

REST_SING $\vdash \forall x.\, \mathsf{REST}\ \{x\} = \{\,\}$

REST_SUBSET $\vdash \forall s.\, \mathsf{REST}\ s\ \mathsf{SUBSET}\ s$

SET_CASES $\vdash \forall s.\, (s = \{\,\}) \vee (\exists x\, t.\, (s = x\ \mathsf{INSERT}\ t) \wedge \neg x\ \mathsf{IN}\ t)$

SING_DELETE $\vdash \forall x.\, \{x\}\ \mathsf{DELETE}\ x = \{\,\}$

SING_FINITE $\vdash \forall s.\, \mathsf{SING}\ s \supset \mathsf{FINITE}\ s$

SING_IFF_CARD1 $\vdash \forall s.\, \mathsf{SING}\ s = (\mathsf{CARD}\ s = 1) \wedge \mathsf{FINITE}\ s$

SING_IFF_EMPTY_REST $\vdash \forall s.\, \mathsf{SING}\ s = \neg(s = \{\,\}) \wedge (\mathsf{REST}\ s = \{\,\})$

SING $\vdash \forall x.\, \mathsf{SING}\ \{x\}$

SPECIFICATION $\vdash \forall f\, x.\, x\ \mathsf{IN}\ \mathsf{SPEC}\ f = f\ x$

SUBSET_ANTISYM $\vdash \forall s\, t.\, s\ \mathsf{SUBSET}\ t \wedge t\ \mathsf{SUBSET}\ s \supset (s = t)$

SUBSET_DELETE $\vdash \forall x\, s\, t.\, s\ \mathsf{SUBSET}\ (t\ \mathsf{DELETE}\ x) = \neg x\ \mathsf{IN}\ s \wedge s\ \mathsf{SUBSET}\ t$

SUBSET_EMPTY $\vdash \forall s.\, s\ \mathsf{SUBSET}\ \{\,\} = (s = \{\,\})$

SUBSET_FINITE $\vdash \forall s.\, \mathsf{FINITE}\ s \supset (\forall t.\, t\ \mathsf{SUBSET}\ s \supset \mathsf{FINITE}\ t)$

SUBSET_INSERT_DELETE
$\vdash \forall x\, s\, t.\, s\ \mathsf{SUBSET}\ (x\ \mathsf{INSERT}\ t) = (s\ \mathsf{DELETE}\ x)\ \mathsf{SUBSET}\ t$

SUBSET_INSERT $\vdash \forall x\, s.\, \neg x\ \mathsf{IN}\ s \supset (\forall t.\, s\ \mathsf{SUBSET}\ (x\ \mathsf{INSERT}\ t) = s\ \mathsf{SUBSET}\ t)$

SUBSET_INTER_ABSORPTION $\vdash \forall s\, t.\, s\ \mathsf{SUBSET}\ t = (s\ \mathsf{INTER}\ t = s)$

SUBSET_REFL $\vdash \forall s.\, s\ \mathsf{SUBSET}\ s$

SUBSET_TRANS $\vdash \forall s\, t\, u.\, s\ \mathsf{SUBSET}\ t \wedge t\ \mathsf{SUBSET}\ u \supset s\ \mathsf{SUBSET}\ u$

SUBSET_UNION_ABSORPTION $\vdash \forall s\, t.\, s\ \mathsf{SUBSET}\ t = (s\ \mathsf{UNION}\ t = t)$

SUBSET_UNION $\vdash (\forall s\, t.\, s\ \mathsf{SUBSET}\ (s\ \mathsf{UNION}\ t)) \wedge (\forall s\, t.\, s\ \mathsf{SUBSET}\ (t\ \mathsf{UNION}\ s))$

SUBSET_UNIV $\vdash \forall s.\, s\ \mathsf{SUBSET}\ \mathsf{UNIV}$

UNION_ASSOC $\vdash \forall s\, t\, u.\, (s\ \mathsf{UNION}\ t)\ \mathsf{UNION}\ u = s\ \mathsf{UNION}\ (t\ \mathsf{UNION}\ u)$

UNION_COMM $\vdash \forall s\, t.\, s\ \mathsf{UNION}\ t = t\ \mathsf{UNION}\ s$

UNION_EMPTY $\vdash (\forall s.\, \{\,\}\ \mathsf{UNION}\ s = s) \wedge (\forall s.\, s\ \mathsf{UNION}\ \{\,\} = s)$

UNION_IDEMPOT $\vdash \forall s.\, s\ \mathsf{UNION}\ s = s$

UNION_OVER_INTER
$\vdash \forall s\, t\, u.\, s\ \mathsf{INTER}\ (t\ \mathsf{UNION}\ u) = (s\ \mathsf{INTER}\ t)\ \mathsf{UNION}\ (s\ \mathsf{INTER}\ u)$

UNION_UNIV $\vdash (\forall s.\, \mathsf{UNIV}\ \mathsf{UNION}\ s = \mathsf{UNIV}) \wedge (\forall s.\, s\ \mathsf{UNION}\ \mathsf{UNIV} = \mathsf{UNIV})$

UNIV_NOT_EMPTY $\vdash \neg(\mathsf{UNIV} = \{\,\})$

UNIV_SUBSET $\vdash \forall s.\, \mathsf{UNIV}\ \mathsf{SUBSET}\ s = (s = \mathsf{UNIV})$

Conversions for sets

- `FINITE_CONV "FINITE {x1,...,xn}" :conv`

$$\vdash \mathsf{FINITE}\,\{x_1,\ldots,x_n\} = \mathsf{T}$$

- `DELETE_CONV conv "{x1,...,xn} DELETE x" :conv -> conv`

$$\vdash \{x_1,\ldots,x_n\}\,\mathsf{DELETE}\,x = \{x_i,\ldots,x_k\}$$

  where for all x_j in $\{x_i,\ldots,x_k\}$ either `conv` proves $\vdash x_j = x$ or x_j is syntactically identical to x, and for all x_j in $\{x_1,\ldots,x_n\}$ and *not* in $\{x_i,\ldots,x_k\}$ `conv` proves $\vdash (x_j = x) = \mathsf{F}$.

- `IMAGE_CONV conv iconv "IMAGE f {x1,...,xn}" :conv -> conv -> conv`

$$\vdash \mathsf{IMAGE}\,f\,\{x_1,\ldots,x_n\} = \{y_1,\ldots,y_n\}$$

  where `conv` proves $\vdash f\,x_i = y_i$ for all $1 \leq i \leq n$.

- `IN_CONV conv "x IN {x1,...,xn}" :conv -> conv`

  $\vdash x\,\mathsf{IN}\,\{x_1,\ldots,x_n\} = \mathsf{T}$ if x is syntactically identical to x_i for some i, or if `conv` proves $\vdash (x = x_i) = \mathsf{T}$ for some i where $1 \leq i \leq n$.

  $\vdash x\,\mathsf{IN}\,\{x_1,\ldots,x_n\} = \mathsf{F}$ if `conv` proves $\vdash (x = x_i) = \mathsf{F}$ for all i where $1 \leq i \leq n$

- `INSERT_CONV conv "x INSERT S" :conv -> conv`

$$\vdash x\,\mathsf{INSERT}\,S = S$$

  if `IN_CONV conv` proves that $\vdash x\,\mathsf{IN}\,S = \mathsf{T}$, otherwise fail

- `SET_SPEC_CONV "t IN {v | P[v] }" :conv`

$$\vdash t\,\mathsf{IN}\;\{v \mid P[v]\} = P[t/v]$$

- `SET_SPEC_CONV "t IN {tm[x1,...,xn] | P[x1,...,xn] }" :conv`

$$\vdash t\,\mathsf{IN}\;\{tm[x_1,\ldots,x_n] \mid P[x_1,\ldots,x_n]\} = \exists x_1 \ldots x_n.t = tm[x_1,\ldots,x_n] \wedge P[x_1,\ldots,x_n]$$

- `UNION_CONV conv "{x1,...,xn} UNION S" :conv -> conv`

$$\vdash \{x_1,\ldots,x_n\}\,\mathsf{UNION}\,S = x_i\,\mathsf{INSERT}\,\ldots(x_k\,\mathsf{INSERT}\,S)$$

  where for all x_j in $\{x_1,\ldots,x_n\}$ but *not* in $\{x_i,\ldots,x_k\}$, `IN_CONV conv` proves that $\vdash x_j \mathsf{IN} S = \mathsf{T}$.

Tactic for type `set`

- `SET_INDUCT_TAC :tactic`

$$\frac{\{A\}\,\forall s.\mathsf{FINITE}\,s \supset P\,s}{\{A\}\,P\,\{\,\} \qquad \{A\}\,P\,(x\,\mathsf{INSERT}\,t)}$$

  and the following are added to the assumptions of the inductive subgoal: $\mathsf{FINITE} t$, $P\,s$ and $\neg x\,\mathsf{IN}\,t$

5.3 The Library `string`

5.3.1 The theory `ascii`

Parents

`HOL`

Types

`":ascii"`

Constants

REP_ascii	`":ascii -> (bool # (bool # (bool # (bool # (bool # (bool # (bool # bool))))))) ltree"`
ABS_ascii	`":(bool # (bool # (bool # (bool # (bool # (bool # (bool # bool))))))) ltree -> ascii"`
ASCII	`":bool -> (bool -> (bool -> (bool -> (bool -> (bool -> (bool -> (bool -> ascii)))))))"`

Definitions

`ascii_TY_DEF` $\vdash \exists rep.\,$ TYPE_DEFINITION (TRP $(\lambda v\, tl.\,(\exists b_0\, b_1\, b_2\, b_3\, b_4\, b_5\, b_6\, b_7.\, v = b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7) \wedge$ (LENGTH $tl = 0$))) rep

`ascii_ISO_DEF` $\vdash (\forall a.\,$ ABS_ascii (REP_ascii $a) = a) \wedge$ $(\forall r.\,$ TRP $(\lambda v\, tl.\,(\exists b_0\, b_1\, b_2\, b_3\, b_4\, b_5\, b_6\, b_7.$ $v = b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7) \wedge$ (LENGTH $tl = 0$)) $r =$ (REP_ascii (ABS_ascii $r) = r$))

`ASCII_DEF` $\vdash \forall b_0\, b_1\, b_2\, b_3\, b_4\, b_5\, b_6\, b_7.\,$ ASCII $b_0\, b_1\, b_2\, b_3\, b_4\, b_5\, b_6\, b_7 =$ ABS_ascii (Node $(b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7)$ [ ])

Theorems

`ascii_Axiom` $\vdash \forall f.\,(\exists! fn.\,(\forall b_0\, b_1\, b_2\, b_3\, b_4\, b_5\, b_6\, b_7.$ fn (ASCII $b_0\, b_1\, b_2\, b_3\, b_4\, b_5\, b_6\, b_7) = f\, b_0\, b_1\, b_2\, b_3\, b_4\, b_5\, b_6\, b_7))$

`ascii_Induct` $\vdash \forall P.$ $(\forall b_0\, b_1\, b_2\, b_3\, b_4\, b_5\, b_6\, b_7.\, P$ (ASCII $b_0\, b_1\, b_2\, b_3\, b_4\, b_5\, b_6\, b_7)) \supset (\forall a.\, P\, a)$

`ascii_CASES` $\vdash \forall a.\,(\exists b_0\, b_1\, b_2\, b_3\, b_4\, b_5\, b_6\, b_7.\, a =$ ASCII $b_0\, b_1\, b_2\, b_3\, b_4\, b_5\, b_6\, b_7)$

`ASCII_11` $\vdash \forall b_0\, b_1\, b_2\, b_3\, b_4\, b_5\, b_6\, b_7\, b_0'\, b_1'\, b_2'\, b_3'\, b_4'\, b_5'\, b_6'\, b_7'.$ (ASCII $b_0\, b_1\, b_2\, b_3\, b_4\, b_5\, b_6\, b_7 =$ ASCII $b_0'\, b_1'\, b_2'\, b_3'\, b_4'\, b_5'\, b_6'\, b_7') = (b_0 = b_0') \wedge (b_1 = b_1') \wedge (b_2 = b_2') \wedge (b_3 = b_3') \wedge (b_4 = b_4') \wedge (b_5 = b_5') \wedge (b_6 = b_6') \wedge (b_7 = b_7')$

5.3.2 The theory `string`

Parents

`HOL ascii`

Types

`":string"`

Constants

REP_string	`":string -> (one + ascii)ltree"`
ABS_string	`":(one + ascii)ltree -> string"`
‘‘	`":string"`
STRING	`":ascii -> (string -> string)"`

Definitions

`string_TY_DEF` $\vdash \exists rep.\,$ TYPE_DEFINITION (TRP $(\lambda v\, tl.\,(v =$ INL one$) \wedge$ (LENGTH $tl = 0) \vee (\exists a.\, v =$ INR $a) \wedge$ (LENGTH $tl =$ SUC 0))) rep

`string_ISO_DEF` $\vdash (\forall a.\,$ ABS_string (REP_string $a) = a) \wedge$ $(\forall r.\,$ TRP $(\lambda v\, tl.\,(v =$ INL one$) \wedge$(LENGTH $tl = 0)\vee(\exists a.\, v =$ INR $a)\wedge$ (LENGTH $tl =$ SUC 0)) $r =$ (REP_string (ABS_string $r) = r$))

`‘‘_DEF` $\vdash$ ‘‘ = ABS_string (Node (INL one) [ ])

`STRING_DEF` $\vdash \forall a\, s.\,$ STRING $a\, s =$ ABS_string (Node (INR a) [REP_string s])

Theorems

`string_Axiom`
$\vdash \forall e\, f.\, (\exists\forall fn.\, (fn\ `` = e) \land (\forall a\, s.\, fn\, (\mathsf{STRING}\, a\, s) = f\, (fn\, s)\, a\, s))$

`string_Induct` $\vdash \forall P.\, P\ `` \land (\forall s.\, P\, s \supset (\forall a.\, P\, (\mathsf{STRING}\, a\, s))) \supset (\forall s.\, P\, s)$

`string_CASES` $\vdash \forall s.\, (s = ``) \lor (\exists s'\, a.\, s = \mathsf{STRING}\, a\, s')$

`STRING_11` $\vdash \forall a\, s\, a'\, s'.\, (\mathsf{STRING}\, a\, s = \mathsf{STRING}\, a'\, s') = (a = a') \land (s = s')$

`NOT_STRING_EMPTY` $\vdash \forall a\, s.\, \neg(`` = \mathsf{STRING}\, a\, s)$