HOL Light Very Quick Reference

compiled by John Harrison, mangled by Freek Wiedijk

Theorems (type thm)

```
ADD1                        |- SUC m = m + 1
ADD_AC                      |- m + n = n + m /\ (m + n) + p = m + n + p /\ m + n + p = n + m + p
ADD_ASSOC                   |- m + n + p = (m + n) + p
ADD_CLAUSES                 |- (!n. 0 + n = n) /\ (!m. m + 0 = m) /\ (!m n. SUC m + n = SUC (m + n)) /\ (!m n. m + SUC n = SUC (m + n))
ADD_SUB                     |- (m + n) - n = m
ADD_SYM                     |- m + n = n + m
ALL                         |- (ALL P [] <=> T) /\ (ALL P (CONS h t) <=> P h /\ ALL P t)
ALL2                        |- (ALL2 P [] [] <=> T) /\ ... /\ (ALL2 P (CONS h1 t1) (CONS h2 t2) <=> P h1 h2 /\ ALL2 P t1 t2)
APPEND                      |- (!l. APPEND [] l = l) /\ (!h t l. APPEND (CONS h t) l = CONS h (APPEND t l))
ARITH                       |- (NUMERAL 0 = 0 /\ BIT0 _0 = _0) /\ ((!n. SUC (NUMERAL n) = NUMERAL (SUC n)) /\ ...
ARITH_EQ                    |- (!m n. NUMERAL m = NUMERAL n <=> m = n) /\ (_0 = _0 <=> T) /\ ...
CARD_CLAUSES                |- CARD {} = 0 /\ (!x s. FINITE s ==> CARD (x INSERT s) = (if x IN s then CARD s else SUC (CARD s)))
CART_EQ                     |- x = y <=> (!i. 1 <= i /\ i <= dimindex UNIV ==> x $ i = y $ i)
CONJ_ASSOC                  |- t1 /\ t2 /\ t3 <=> (t1 /\ t2) /\ t3
DE_MORGAN_THM               |- (~(t1 /\ t2) <=> ~t1 \/ ~t2) /\ (~(t1 \/ t2) <=> ~t1 /\ ~t2)
DIVISION                    |- ~(n = 0) ==> m = m DIV n * n + m MOD n /\ m MOD n < n
ETA_AX                      |- (\x. t x) = t
EVEN                        |- (EVEN 0 <=> T) /\ (!n. EVEN (SUC n) <=> ~EVEN n)
EXISTS_REFL                 |- ?x. x = a
EXP                         |- (!m. m EXP 0 = 1) /\ (!m n. m EXP SUC n = m * m EXP n)
EXTENSION                   |- s = t <=> (!x. x IN s <=> x IN t)
FACT                        |- FACT 0 = 1 /\ (!n. FACT (SUC n) = SUC n * FACT n)
FINITE_INDUCT_STRONG        |- P {} /\ (!x s. P s /\ ~(x IN s) /\ FINITE s ==> P (x INSERT s)) ==> (!s. FINITE s ==> P s)
FINITE_NUMSEG               |- FINITE (m .. n)
FINITE_RULES                |- FINITE {} /\ (!x s. FINITE s ==> FINITE (x INSERT s))
FINITE_SUBSET               |- FINITE t /\ s SUBSET t ==> FINITE s
FORALL_PAIR_THM             |- (!p. P p) <=> (!p1 p2. P (p1,p2))
FUN_EQ_THM                  |- f = g <=> (!x. f x = g x)
GE                          |- m >= n <=> n <= m
HAS_SIZE                    |- s HAS_SIZE n <=> FINITE s /\ CARD s = n
HD                          |- HD (CONS h t) = h
IMP_IMP                     |- p ==> q ==> r <=> p /\ q ==> r
IN                          |- x IN P <=> P x
IN_DELETE                   |- x IN s DELETE y <=> x IN s /\ ~(x = y)
IN_ELIM_THM                 |- (!P x. x IN GSPEC (\v. P (SETSPEC v)) <=> P (\p t. p /\ x = t)) /\ ...
IN_IMAGE                    |- y IN IMAGE f s <=> (?x. y = f x /\ x IN s)
IN_INSERT                   |- x IN y INSERT s <=> x = y \/ x IN s
IN_INTER                    |- x IN s INTER t <=> x IN s /\ x IN t
IN_NUMSEG                   |- p IN m .. n <=> m <= p /\ p <= n
IN_SING                     |- x IN {y} <=> x = y
IN_UNION                    |- x IN s UNION t <=> x IN s \/ x IN t
IN_UNIV                     |- x IN UNIV
LAMBDA_BETA                 |- 1 <= i /\ i <= dimindex UNIV ==> (lambda) g $ i = g i
LAST                        |- LAST (CONS h t) = (if t = [] then h else LAST t)
LE                          |- (!m. m <= 0 <=> m = 0) /\ (!m n. m <= SUC n <=> m = SUC n \/ m <= n)
LEFT_ADD_DISTRIB            |- m * (n + p) = m * n + m * p
LEFT_IMP_EXISTS_THM         |- (?x. P x) ==> Q <=> (!x. P x ==> Q)
LENGTH                      |- LENGTH [] = 0 /\ (!h t. LENGTH (CONS h t) = SUC (LENGTH t))
LENGTH_APPEND               |- LENGTH (APPEND l m) = LENGTH l + LENGTH m
LE_0                        |- 0 <= n
LE_ADD                      |- m <= m + n
LE_EXISTS                   |- m <= n <=> (?d. n = m + d)
LE_MULT_LCANCEL             |- m * n <= m * p <=> m = 0 \/ n <= p
LE_REFL                     |- n <= n
LE_TRANS                    |- m <= n /\ n <= p ==> m <= p
LT                          |- (!m. m < 0 <=> F) /\ (!m n. m < SUC n <=> m = n \/ m < n)
LT_0                        |- 0 < SUC n
LT_REFL                     |- ~(n < n)
MEM                         |- (MEM x [] <=> F) /\ (MEM x (CONS h t) <=> x = h \/ MEM x t)
MEMBER_NOT_EMPTY            |- (?x. x IN s) <=> ~(s = {})
MONO_EXISTS                 |- (!x. P x ==> Q x) ==> (?x. P x) ==> (?x. Q x)
MONO_FORALL                 |- (!x. P x ==> Q x) ==> (!x. P x) ==> (!x. Q x)
MULT_AC                     |- m * n = n * m /\ (m * n) * p = m * n * p /\ m * n * p = n * m * p
MULT_ASSOC                  |- m * n * p = (m * n) * p
MULT_CLAUSES                |- (!n. 0 * n = 0) /\ ... /\ (!m n. m * SUC n = m + m * n)
MULT_SYM                    |- m * n = n * m
NOT_CONS_NIL                |- ~(CONS h t = [])
NOT_EXISTS_THM              |- ~(?x. P x) <=> (!x. ~P x)
NOT_FORALL_THM              |- ~(!x. P x) <=> (?x. ~P x)
NOT_IMP                     |- ~(t1 ==> t2) <=> t1 /\ ~t2
NOT_IN_EMPTY                |- ~(x IN {})
NOT_LE                      |- ~(m <= n) <=> n < m
NOT_LT                      |- ~(m < n) <=> n <= m
NOT_SUC                     |- ~(SUC n = 0)
PAIR_EQ                     |- x,y = a,b <=> x = a /\ y = b
PRE                         |- PRE 0 = 0 /\ (!n. PRE (SUC n) = n)
```

```
REAL_ABS_MUL                    |- abs (x * y) = abs x * abs y
REAL_ABS_NEG                    |- abs (--x) = abs x
REAL_ABS_NUM                    |- abs (&n) = &n
REAL_ABS_POS                    |- &0 <= abs x
REAL_ABS_POW                    |- abs (x pow n) = abs x pow n
REAL_ADD_ASSOC                  |- x + y + z = (x + y) + z
REAL_ADD_LID                    |- &0 + x = x
REAL_ADD_LINV                   |- --x + x = &0
REAL_ADD_RID                    |- x + &0 = x
REAL_ADD_SYM                    |- x + y = y + x
REAL_ENTIRE                     |- x * y = &0 <=> x = &0 \/ y = &0
REAL_EQ_IMP_LE                  |- x = y ==> x <= y
REAL_INV_MUL                    |- inv (x * y) = inv x * inv y
REAL_LET_TRANS                  |- x <= y /\ y < z ==> x < z
REAL_LE_LMUL                    |- &0 <= x /\ y <= z ==> x * y <= x * z
REAL_LE_LT                      |- x <= y <=> x < y \/ x = y
REAL_LE_REFL                    |- x <= x
REAL_LE_SQUARE                  |- &0 <= x * x
REAL_LE_TOTAL                   |- x <= y \/ y <= x
REAL_LTE_TRANS                  |- x < y /\ y <= z ==> x < z
REAL_LT_01                      |- &0 < &1
REAL_LT_DIV                     |- &0 < x /\ &0 < y ==> &0 < x / y
REAL_LT_IMP_LE                  |- x < y ==> x <= y
REAL_LT_IMP_NZ                  |- &0 < x ==> ~(x = &0)
REAL_LT_LE                      |- x < y <=> x <= y /\ ~(x = y)
REAL_LT_MUL                     |- &0 < x /\ &0 < y ==> &0 < x * y
REAL_LT_REFL                    |- ~(x < x)
REAL_LT_TRANS                   |- x < y /\ y < z ==> x < z
REAL_MUL_AC                     |- m * n = n * m /\ (m * n) * p = m * n * p /\ m * n * p = n * m * p
REAL_MUL_ASSOC                  |- x * y * z = (x * y) * z
REAL_MUL_LID                    |- &1 * x = x
REAL_MUL_LINV                   |- ~(x = &0) ==> inv x * x = &1
REAL_MUL_LZERO                  |- &0 * x = &0
REAL_MUL_RID                    |- x * &1 = x
REAL_MUL_RINV                   |- ~(x = &0) ==> x * inv x = &1
REAL_MUL_RZERO                  |- x * &0 = &0
REAL_MUL_SYM                    |- x * y = y * x
REAL_NEGNEG                     |- -- --x = x
REAL_NEG_NEG                    |- -- --x = x
REAL_NOT_LE                     |- ~(x <= y) <=> y < x
REAL_NOT_LT                     |- ~(x < y) <=> y <= x
REAL_OF_NUM_ADD                 |- &m + &n = &(m + n)
REAL_OF_NUM_EQ                  |- &m = &n <=> m = n
REAL_OF_NUM_LE                  |- &m <= &n <=> m <= n
REAL_OF_NUM_LT                  |- &m < &n <=> m < n
REAL_OF_NUM_MUL                 |- &m * &n = &(m * n)
REAL_OF_NUM_POW                 |- &x pow n = &(x EXP n)
REAL_POS                        |- &0 <= &n
REAL_POW_2                      |- x pow 2 = x * x
REAL_POW_ADD                    |- x pow (m + n) = x pow m * x pow n
REAL_SUB_0                      |- x - y = &0 <=> x = y
REAL_SUB_LDISTRIB               |- x * (y - z) = x * y - x * z
REAL_SUB_LE                     |- &0 <= x - y <=> y <= x
REAL_SUB_LT                     |- &0 < x - y <=> y < x
REAL_SUB_REFL                   |- x - x = &0
REAL_SUB_RZERO                  |- x - &0 = x
RIGHT_ADD_DISTRIB               |- (m + n) * p = m * p + n * p
RIGHT_FORALL_IMP_THM            |- (!x. P ==> Q x) <=> P ==> (!x. Q x)
SKOLEM_THM                      |- (!x. ?y. P x y) <=> (?y. !x. P x (y x))
SUBSET                          |- s SUBSET t <=> (!x. x IN s ==> x IN t)
SUC_INJ                         |- SUC m = SUC n <=> m = n
TL                              |- TL (CONS h t) = t
TRUTH                           |- T
```

Inference rules (return type `thm`)

`AC th tm`	Prove equivalence by associativity and commutativity
`AP_TERM tm th`	From `|- s = t` to `|- f s = f t`
`AP_THM th tm`	From `|- f = g` to `|- f x = g x`
`ARITH_RULE tm`	Linear arithmetic prover over N
`ASSUME tm`	Generate trivial theorem `p |- p`
`BETA_RULE th`	Reduce all beta-redexes in theorem
`CONJ th th`	From `|- p` and `|- q` to `|- p /\ q`
`CONJUNCT1 th`	From `|- p /\ q` to `|- p`
`CONJUNCT2 th`	From `|- p /\ q` to `|- q`
`CONV_RULE conv th`	Apply conversion to conclusion of theorem
`DISCH tm th`	From `p |- q` to `|- p ==> q`
`DISCH_ALL th`	From `p1, ..., pn |- q` to `|- p1 ==> ... ==> pn ==> q`
`EQT_ELIM th`	From `|- p <=> T` to `|- p`
`EQT_INTRO th`	From `|- p` to `|- p <=> T`
`EQ_MP th th`	From `|- p <=> q` and `|- p` to `|- q`
`GEN tm th`	From `|- p[x]` to `|- !x. p[x]`
`GENL[tm] th`	From `|- p[x1,...,xn]` to `|- !x1 .. xn. p[x1,...,xn]`
`GEN_ALL th`	From `|- p[x1,...,xn]` to `|- !x1 .. xn. p[x1,...,xn]`, all variables
`GEN_REWRITE_RULE cnvn [th] th`	Rewrite conclusion of theorem using precise depth conversion
`GSYM th`	Switch topmost equations, e.g. from `|- !x. s[x] = t[x]` to `!x. t[x] = s[x]`
`INST[tm,tm] th`	Instantiate `|- p[x1,...xn]` to `|- p[t1,...,tn]`
`INT_ARITH tm`	Linear arithmetic prover over Z
`INT_OF_REAL_THM th`	Map universal theorem from R to analog over Z
`ISPEC tm th`	From `|- !x. p[x]` to `|- p[t]` with type instantiation
`ISPECL[tm] th`	From `|- !x1 .. xn. p[x1,...,xn]` to `|- p[t1,...,tn]` with type instantiation
`MATCH_MP th th`	From `|- p ==> q` and `|- p'` to `|- q'`, instantiating first theorem to match
`MK_COMB(th,th)`	From `|- f = g` and `|- x = y` to `|- f(x) = g(y)`
`MP th th`	From `|- p ==> q` and `|- p` to `|- q`, no matching
`ONCE_REWRITE_RULE[th] th`	Rewrite conclusion of theorem once at topmost subterms
`PART_MATCH tmfn th tm`	Instantiate theorem by matching part of it to a term
`PROVE_HYP th th`	From `|- p` and `p |- q` to `|- q`
`REAL_ARITH tm`	Linear arithmetic prover over R
`REFL tm`	Produce trivial theorem `|- t = t`
`REWRITE_RULE[th] th`	Rewrite conclusion of theorem with equational theorems
`SPEC tm th`	From `|- !x. p[x]` to `|- p[t]`
`SPECL[tm] th`	From `|- !x1 .. xn. p[x1,...,xn]` to `|- p[t1,...,tn]`
`SPEC_ALL th`	From `|- !x1 .. xn. p[x1,...,xn]` to `|- p[x1,...,xn]`
`SYM th`	From `|- s = t` to `|- t = s`
`TAUT tm`	Prove propositional tautology like ‘`p /\ q ==> p`‘
`TRANS th th`	From `|- s = t` and `|- t = u` and `|- s = u`
`UNDISCH th`	From `|- p ==> q` to `p |- q`

Inference rule with return type `thm list`

`CONJUNCTS th`	From `|- p1 /\ ... /\ pn` to `[|- p1; ...; |- pn]`

Conversions (type `conv = term -> thm`)

`BETA_CONV tm`	Reduce toplevel beta-redex `|- (\x. s[x]) t = s[t]`
`CONTRAPOS_CONV`	From ‘`p ==> q`‘ give `|- (p ==> q) <=> (~q ==> ~p)`
`GEN_BETA_CONV`	Reduce general beta-redex like `|- (\(x,y). p[x,y]) (a,b) = p[a,b]`
`GEN_REWRITE_CONV cnvn [th]`	Rewriting conversion using precise depth conversion
`NUM_REDUCE_CONV`	Evaluate numerical expressions over N like ‘`2 + 7 DIV (FACT 3)`‘
`conv ORELSEC conv`	Try to apply one conversion and if it fails, apply the other
`REAL_RAT_REDUCE_CONV`	Evaluate numerical expressions over R like ‘`&22 / &7 - &3 * &1`‘
`REWRITE_CONV[th]`	Conversion to rewrite a term `t` to `t'` giving `|- t = t'`
`REWR_CONV th`	Conversion to rewrite a term `t` once at top level giving `|- t = t'`
`SYM_CONV`	Conversion to switch equations once `|- P[s = t] <=> P[t = s]`
`conv THENC conv`	Apply one conversion then the other
`TOP_DEPTH_CONV conv`	Apply conversion once to top-level terms

Conversionals (type `conv -> conv`)

`BINDER_CONV`	Apply conversion to body of quantifier etc.
`LAND_CONV`	Apply conversion to LHS of binary operator, e.g. ‘`s`‘ in ‘`s + t`‘
`ONCE_DEPTH_CONV`	Apply conversion to first possible subterms top-down
`RAND_CONV`	Apply conversion to rand of combination, e.g. `x` in `f(x)`
`RATOR_CONV`	Apply conversion to rator of combination, e.g. `f` in `f(x)`

Tactics (return type `tactic`)

`ABBREV_TAC tm`	Introduce abbreviation for t, from ?- p[t] to t = x ?- p[x]
`ABS_TAC`	From ?- (\x. s[x]) = (\x. t[x]) to ?- s[x] = t[x]
`ALL_TAC`	Tactic with no effect
`ANTS_TAC`	From ?- (p ==> q) ==> r to ?- p and ?- q ==> r
`AP_TERM_TAC`	From ?- f s = f t to ?- s = t
`AP_THM_TAC`	From ?- f x = g x to ?- f = g
`ARITH_TAC`	Tactic to solve linear arithmetic over N
`ASM_CASES_TAC tm`	Split ?- q into p ?- q and ~p ?- q
`ASM_MESON_TAC[th]`	Tactic for first-order logic including assumptions
`ASM_REWRITE_TAC[th]`	Rewrite goal by theorems including assumptions
`ASM_SIMP_TAC[th]`	Simplify goal by theorems including assumptions
`BETA_TAC`	Reduce all beta-redexes in conclusion of goal
`COND_CASES_TAC`	From ?- P[if p then x else y] to p ?- p[x] and ~p ?- p[y]
`CONJ_TAC`	Split ?- p /\ q into ?- p and ?- q
`CONV_TAC conv`	Apply conversion to conclusion of goal
`DISCH_TAC`	From ?- p ==> q to p ?- q
`DISCH_THEN ttac`	From ?- p ==> q to ?- q after using |- p
`DISJ1_TAC`	From ?- p \/ q to ?- p
`DISJ2_TAC`	From ?- p \/ q to ?- q
`EQ_TAC`	Split ?- p <=> q into ?- p ==> q and ?- q ==> p
`EVERY_ASSUM ttac`	Apply function to each assumption of goal
`EXISTS_TAC tm`	From ?- ?x. p[x] to ?- p[t]
`EXPAND_TAC s`	Expand an abbreviation in a goal
`FIRST_ASSUM ttac`	Apply function to first possible assumption of goal
`FIRST_X_ASSUM ttac`	Apply function to and remove first possible assumption of goal
`GEN_REWRITE_TAC cnvn [th]`	Rewrite conclusion of goal using precise depth conversion
`GEN_TAC`	From ?- !x. p[x] to ?- p[x]
`INDUCT_TAC`	Apply ordinary mathematical induction to goal
`LIST_INDUCT_TAC`	Apply list induction to goal
`MAP_EVERY atac [a]`	Map tactic-producing function over a list of arguments, apply in sequence
`MESON_TAC[th]`	Solve goal using first-order automation, ignoring assumptions
`ONCE_REWRITE_TAC[th]`	Rewrite conclusion of goal once at topmost subterms
`tac ORELSE tac`	Try to apply one tactic and if it fails, apply the other
`POP_ASSUM ttac`	Remove first assumption of goal and apply function to it
`POP_ASSUM_LIST tltac`	Remove assumptions of goal and apply function to it
`REAL_ARITH_TAC`	Tactic to solve linear arithmetic over R
`REFL_TAC`	Solve trivial goal ?- t = t
`REPEAT tac`	Apply a tactic repeatedly until it fails
`REWRITE_TAC[th]`	Rewrite conclusion of goal with equational theorems
`RULE_ASSUM_TAC thfn`	Apply inference rule to all hypotheses of goal
`SET_TAC[th]`	Solve trivial set-theoretic goal like ‘x IN (x INSERT s)‘
`SIMP_TAC[th]`	Simplify goal by theorems ignoring assumptions
`SPEC_TAC(tm,tm)`	From ?- p[t] to ?- !x. p[x]
`STRIP_TAC`	Break down goal, ?- p /\ q to ?- p and ?- q etc. etc.
`SUBGOAL_THEN tm ttac`	Split off a separate subgoal
`TRY tac`	Try a tactic but do nothing if it fails
`tac THEN tac`	Apply one tactic then the other to all resulting subgoals
`tac THENL [tac]`	Apply one tactic then second list to corresponding subgoals
`UNDISCH_TAC tm`	From p ?- q to ?- p ==> q
`USE_THEN s ttac`	Apply function to assumption with particular label
`X_GEN_TAC tm`	From ?- !x. p[x] to ?- p[y] with specified ‘y‘

Theorem-tactics (type `thm_tactic = thm -> tactic`)

`ACCEPT_TAC`	Solve goal ?- p by theorem |- p
`ANTE_RES_THEN ttac`	Using |- p ==> q in goal p ?- r apply theorem-tactic to |- q
`ASSUME_TAC`	Given |- p, from ?- q to p ?- q, no label on new assumption
`CHOOSE_THEN ttac`	Using |- ?x. p[x] apply theorem-tactic to |- p[x]
`CONJUNCTS_THEN ttac`	Using |- p /\ q apply theorem-tactic to |- p and |- q
`CONJUNCTS_THEN2 ttac ttac`	Using |- p /\ q apply respective theorem-tactics to |- p and |- q
`DISJ_CASES_TAC`	Use |- p \/ q, from ?- r to p ?- r and q ?- r
`DISJ_CASES_THEN ttac`	Use |- p \/ q, apply theorem-tactic to |- p and |- q separately
`LABEL_TAC s`	Given |- p, from ?- q to p ?- q, labelling new assumption "s"
`MATCH_ACCEPT_TAC`	From |- p[x1,...,xn] solve goal ?- p[t1,...,tn] that’s an instance
`MATCH_MP_TAC`	Use |- p ==> q to go from ?- q’ to ?- p’, instantiation theorem to match
`MP_TAC`	Use |- p to go from ?- q to ?- p ==> q
`REPEAT_TCL ttacfn ttac`	Apply theorem-tactical repeatedly until it fails
`STRIP_ASSUME_TAC`	Break theorem down into pieces and add them as assumptions
`SUBST1_TAC`	Substitute equation in conclusion of goal, no matching
`SUBST_ALL_TAC`	Substitute equation in hypotheses and conclusion of goal, no matching
`X_CHOOSE_TAC tm`	From |- ?x. p[x] and ?- q to p[y] ?- q, specified y
`X_CHOOSE_THEN tm ttac`	From |- ?x. p[x] apply theorem-tactic to |- p[y], specified y

```
tm                              : term
[tm]                            : term list
(tm,tm)                         : term * term
[tm,tm]                         : (term * term) list
tmfn                            : term -> term
th                              : thm
[th]                            : thm list
(th,th)                         : thm * thm
thfn                            : thm -> thm
conv                            : conv
cnvn                            : conv -> conv
tac                             : tactic
[tac]                           : tactic list
ttac                            : thm_tactic = thm -> tactic
tltac                           : thm list -> tactic
ttacfn                          : thm_tactical = thm_tactic -> thm_tactic
atac                            : 'a -> tactic
[a]                             : 'a list
s                               : string
```