Output

```
---------------------------------------------------------------------
       HOL-4 [Kananaskis 11 (stdknl, built Thu May 05 10:43:02 2016)]

       For introductory HOL help, type: help "hol";
       To exit type <Control>-D
---------------------------------------------------------------------
[extending loadPath with Holmakefile INCLUDES variable]
[extending loadPath with Holmakefile PRE_INCLUDES variable]
[Use-ing configuration file /Users/josephchan/.hol-config.sml]
[In non-standard heap: /Users/josephchan/work/hol/aks/theories/algebra-heap]
> >
val _ = HOL_Interactive.toggle_quietdec();

val _ = load "lcsymtacs";
open lcsymtacs;

(* open dependent theories *)

val _ = load "primesTheory";
open primesTheory;

open arithmeticTheory dividesTheory logrootTheory;

val _ = HOL_Interactive.toggle_quietdec();
>
(* Press SPACE bar to continue *)
(* ------------------------------------------------------------------------- *)
(* Primality Testing based on SQRT                                           *)
(* ------------------------------------------------------------------------- *)

(* Enable Unicode display *)
set_trace "Unicode" 1;
val it = (): unit
>
(* For divides -- one way *)
val _ = set_mapped_fixity {
            term_name = "divides",
                  fixity = Infix(NONASSOC, 450),
                   tok = UTF8.chr 0x2223
          };
# # # # > (* For not divides -- another way *)
(* val _ = overload_on (UTF8.chr 0x2224, ``\m n. ~(m divides n)``); *)
(* val _ = set_fixity (UTF8.chr 0x2224) (Infix(NONASSOC, 450)); *)

(* For SQRT *)
val _ = overload_on (UTF8.chr 0x221A, ``SQRT``);
>
(* Primality Testing by factors up to square root. *)

(* Theorem:
   Given a number p, it is prime iff it has no divisors up to SQRT p.
*)
(* Sounds reasonable? *)

(* Formulate the theorem *)
g `!p. prime p <=> !q. q <= SQRT p ==> ~(q divides p)`;
# # # val it =
   Proof manager status: 1 proof.
1. Incomplete goalstack:
     Initial goal:


     ∀p. prime p ⇔ ∀q. q ≤ √ p ⇒ ¬(q ∣ p)

:
   proofs
>
(* First, separate the if-part and only-if parts: *)
e (rw[EQ_IMP_THM]); (* >> *)
OK..
<<HOL message: Initialising SRW simpset ... done>>
2 subgoals:
```

```
val it =

  ∀q. q ≤ √ p ⇒ ¬(q | p)
------------------------------------
prime p


  0.  prime p
  1.  q ≤ √ p
------------------------------------
¬(q | p)

2 subgoals
:
   proof
>
(* This gives two cases, work on the first one *)
(* Very odd to prove a negation, so try the following: *)
e (spose_not_then strip_assume_tac); (* by contradiction *)
OK..
1 subgoal:
val it =

  0.  prime p
  1.  q ≤ √ p
  2.  q | p
------------------------------------
F
:
   proof
>
(* Recall the prime definition: *)
prime_def;
val it =
   ⊢ ∀a. prime a ⇔ a ≠ 1 ∧ ∀b. b | a ⇒ (b = a) ∨ (b = 1):
   thm
>
(* This helps us to assert: *)
e (`(q = p) \/ (q = 1)` by metis_tac[prime_def]); (* >> *)
OK..
metis: r[+0+11]+0+0+0+0+0+0+0+0+0+0+1#
2 subgoals:
val it =

  0.  prime p
  1.  q ≤ √ p
  2.  q | p
  3.  q = 1
------------------------------------
F


  0.  prime p
  1.  q ≤ √ p
  2.  q | p
  3.  q = p
------------------------------------
F

2 subgoals
:
   proof
>
(* This gives two subcases, one for each possibility. *)

(* The first case: q = p, gives p <= √ p *)
(* Usually SQRT gives a smaller value, but here a value is bounded by its own square root! *)
(* This is quite unusual, and the only possible values for p will be given by: *)
SQRT_GE_SELF;
val it =
   ⊢ ∀n. n ≤ √ n ⇔ (n = 0) ∨ (n = 1):
   thm
>
(* But we also have: *)
```

```
ONE_LT_PRIME;
val it = ⊢ ∀p. prime p ⇒ 1 < p: thm
>
(* so we can derive a contradiction by: *)
e (`1 < p` by rw[ONE_LT_PRIME]);
OK..
1 subgoal:
val it =

  0.  prime p
  1.  q ≤ √ p
  2.  q | p
  3.  q = p
  4.  1 < p
------------------------------------
F
:
   proof
>
e (`p <> 0 /\ p <> 1` by decide_tac);
OK..
1 subgoal:
val it =

  0.  prime p
  1.  q ≤ √ p
  2.  q | p
  3.  q = p
  4.  1 < p
  5.  p ≠ 0
  6.  p ≠ 1
------------------------------------
F
:
   proof
>
(* Now derive the contradiction. *)
e (metis_tac[SQRT_GE_SELF]); (* << *)
OK..
metis: r[+0+11]+0+1+0+0+0+0+0+0+0+0+1#

Goal proved.
 [.......] ⊢ F

Goal proved.
 [.....] ⊢ F

Goal proved.
 [....] ⊢ F

Remaining subgoals:
val it =

  0.  prime p
  1.  q ≤ √ p
  2.  q | p
  3.  q = 1
------------------------------------
F
:
   proof
>

(* Putting q = 1 into q | p gives 1 | p. This is valid, due to: *)
ONE_DIVIDES_ALL;
val it = ⊢ ∀a. 1 | a: thm
>
(* Putting q = 1 into q <= √ p gives 1 <= √ p. This is also valid, due to SQRT is monotonic and: *)
ONE_LT_PRIME;
val it = ⊢ ∀p. prime p ⇒ 1 < p: thm
>
(* There is no hope of deriving a contradiction. *)
(* The theorem as stated is false: testing should start with 1 < q *)
(* Throw away the theorem. *)
```

```
drop();
OK..
val it = There are currently no proofs.: proofs
> (* Reformulate the theorem. *)
g `!p. prime p <=> !q. 1 < q /\ q <= SQRT p ==> ~(q divides p)`;
val it =
   Proof manager status: 1 proof.
1. Incomplete goalstack:
     Initial goal:


     ∀p. prime p ⇔ ∀q. 1 < q ∧ q ≤ √ p ⇒ ¬(q ❘ p)

:
   proofs
>
(* Repeat the previous steps: *)
e (rw[EQ_IMP_THM]); (* >> *)
OK..
2 subgoals:
val it =

  ∀q. 1 < q ∧ q ≤ √ p ⇒ ¬(q ❘ p)
------------------------------------
prime p


  0.  prime p
  1.  1 < q
  2.  q ≤ √ p
------------------------------------
¬(q ❘ p)

2 subgoals
:
   proof
>
e (spose_not_then strip_assume_tac); (* by contradiction *)
OK..
1 subgoal:
val it =

  0.  prime p
  1.  1 < q
  2.  q ≤ √ p
  3.  q ❘ p
------------------------------------
F
:
   proof
>
e (`q <> 1` by decide_tac);
OK..
1 subgoal:
val it =

  0.  prime p
  1.  1 < q
  2.  q ≤ √ p
  3.  q ❘ p
  4.  q ≠ 1
------------------------------------
F
:
   proof
>
(* Now we can assert q <> 1, due to 1 < q. *)
e (`q = p` by metis_tac[prime_def]);
OK..
metis: r[+0+12]+0+0+0+0+0+0+0+0+0+0+0+1#
1 subgoal:
val it =

  0.  prime p
```

```
  1.  1 < q
  2.  q ≤ √ p
  3.  q | p
  4.  q ≠ 1
  5.  q = p
------------------------------------
F
:
   proof
>
e (`1 < p` by rw[ONE_LT_PRIME]);
OK..
1 subgoal:
val it =

  0.  prime p
  1.  1 < q
  2.  q ≤ √ p
  3.  q | p
  4.  q ≠ 1
  5.  q = p
  6.  1 < p
------------------------------------
F
:
   proof
>
e (`p <> 0 /\ p <> 1` by decide_tac);
OK..
1 subgoal:
val it =

  0.  prime p
  1.  1 < q
  2.  q ≤ √ p
  3.  q | p
  4.  q ≠ 1
  5.  q = p
  6.  1 < p
  7.  p ≠ 0
  8.  p ≠ 1
------------------------------------
F
:
   proof
>
e (metis_tac[SQRT_GE_SELF]); (* << *)
OK..
metis: r[+0+11]+0+1+0+0+0+0+0+0+0+0+1#

Goal proved.
 [.........] ⊢ F

Goal proved.
 [........] ⊢ F

Goal proved.
 [......] ⊢ F

Goal proved.
 [.....] ⊢ F

Goal proved.
 [....] ⊢ F

Goal proved.
 [...] ⊢ ¬(q | p)

Remaining subgoals:
val it =

  ∀q. 1 < q ∧ q ≤ √ p ⇒ ¬(q | p)
------------------------------------
prime p
```

```
:
   proof
>
(* case: ∀q. 1 < q ∧ q ≤ √ p ==> ¬(q | p) ==> prime p *)
(* Expand this by definition of prime to see what needs to be proved: *)
e (rw[prime_def]); (* >> *)
OK..
2 subgoals:
val it =

  0.  ∀q. 1 < q ∧ q ≤ √ p ⇒ ¬(q | p)
  1.  b | p
------------------------------------
(b = p) ∨ (b = 1)


  ∀q. 1 < q ∧ q ≤ √ p ⇒ ¬(q | p)
------------------------------------
p ≠ 1

2 subgoals
:
   proof
>
(* case: ∀q. 1 < q ∧ q ≤ √ p ==> ¬(q | p) ==> p <> 1 *)
(* By contradiction again. *)
e (spose_not_then strip_assume_tac); (* by contradiction *)
OK..
1 subgoal:
val it =

  0.  ∀q. 1 < q ∧ q ≤ √ p ⇒ ¬(q | p)
  1.  p = 1
------------------------------------
F
:
   proof
>
(* Since √ 1 = 1 *)
EVAL ``SQRT 1``;
val it = ⊢ √ 1 = 1: thm
> (* Putting p = 1 reduces the first condition to: ∀q. 1 < q /\ q ≤ 1 ==> ¬(q | 1) *)
(* This is true by default, because the pre-condition needs 1 < q /\ q < 2 -- there is no such q! *)
(* Again, no hope to get a contradiction. *)
(* The stated theorem is false: p = 1 will be excluded, need to check p > 1. *)
(* Throw away the theorem. *)
drop();
OK..
val it = There are currently no proofs.: proofs
> (* Reformulate the theorem. *)
g `!p. prime p <=> 1 < p /\ !q. 1 < q /\ q <= SQRT p ==> ~(q divides p)`;
val it =
   Proof manager status: 1 proof.
1. Incomplete goalstack:
     Initial goal:


     ∀p. prime p ⇔ 1 < p ∧ ∀q. 1 < q ∧ q ≤ √ p ⇒ ¬(q | p)

:
   proofs
>
(* Repeat the previous steps: *)
e (rw[EQ_IMP_THM]); (* >> 3 *)
OK..
3 subgoals:
val it =

  0.  1 < p
  1.  ∀q. 1 < q ∧ q ≤ √ p ⇒ ¬(q | p)
------------------------------------
prime p
```

```
  0.  prime p
  1.  1 < q
  2.  q ≤ √ p
------------------------------------
¬(q | p)


  prime p
------------------------------------
1 < p

3 subgoals
:
   proof
>
(* There are 3 subgoals, the first one is easy: *)
(* case: prime p ==> 1 < p *)
e (rw[ONE_LT_PRIME]); (* << *)
OK..

Goal proved.
 [.] ⊢ 1 < p

Remaining subgoals:
val it =

  0.  1 < p
  1.  ∀q. 1 < q ∧ q ≤ √ p ⇒ ¬(q | p)
------------------------------------
prime p


  0.  prime p
  1.  1 < q
  2.  q ≤ √ p
------------------------------------
¬(q | p)

2 subgoals
:
   proof
>
(* case: prime p /\ 1 < q /\ q <= √ p ==> ¬(q | p) *)
(* We've seen this before. *)
e (spose_not_then strip_assume_tac); (* by contradiction *)
OK..
1 subgoal:
val it =

  0.  prime p
  1.  1 < q
  2.  q ≤ √ p
  3.  q | p
------------------------------------
F
:
   proof
>
e (`q <> 1` by decide_tac); (* since 1 < q *)
OK..
1 subgoal:
val it =

  0.  prime p
  1.  1 < q
  2.  q ≤ √ p
  3.  q | p
  4.  q ≠ 1
------------------------------------
F
:
   proof
>
```

```
e (`q = p` by metis_tac[prime_def]); (* since q divides p *)
OK..
metis: r[+0+12]+0+0+0+0+0+0+0+0+0+0+0+1#
1 subgoal:
val it =

  0.  prime p
  1.  1 < q
  2.  q ≤ √ p
  3.  q | p
  4.  q ≠ 1
  5.  q = p
------------------------------------
F
:
   proof
>
(* This makes p <= SQRT p, prepare to contradict SQRT_GE_SELF. *)
e (`1 < p` by rw[ONE_LT_PRIME]);
OK..
1 subgoal:
val it =

  0.  prime p
  1.  1 < q
  2.  q ≤ √ p
  3.  q | p
  4.  q ≠ 1
  5.  q = p
  6.  1 < p
------------------------------------
F
:
   proof
>
e (`p <> 0 /\ p <> 1` by decide_tac); (* since 1 < p *)
OK..
1 subgoal:
val it =

  0.  prime p
  1.  1 < q
  2.  q ≤ √ p
  3.  q | p
  4.  q ≠ 1
  5.  q = p
  6.  1 < p
  7.  p ≠ 0
  8.  p ≠ 1
------------------------------------
F
:
   proof
>
e (metis_tac[SQRT_GE_SELF]); (* << SQRT_GE_SELF would give: (p = 0) or (p = 1) *)
OK..
metis: r[+0+11]+0+1+0+0+0+0+0+0+0+0+1#

Goal proved.
 [.........] ⊢ F

Goal proved.
 [.......] ⊢ F

Goal proved.
 [......] ⊢ F

Goal proved.
 [.....] ⊢ F

Goal proved.
 [....] ⊢ F

Goal proved.
```

```
  [...] ⊢ ¬(q | p)

Remaining subgoals:
val it =

   0.  1 < p
   1.  ∀q. 1 < q ∧ q ≤ √ p ⇒ ¬(q | p)
------------------------------------
prime p
:
   proof
>
(* case: 1 < p /\ ∀q. 1 < q ∧ q ≤ √ p ==> ¬(q | p) ==> prime p *)
(* Expand by prime definition. *)
e (rw[prime_def]);
OK..
1 subgoal:
val it =

   0.  1 < p
   1.  ∀q. 1 < q ∧ q ≤ √ p ⇒ ¬(q | p)
   2.  b | p
------------------------------------
(b = p) ∨ (b = 1)
:
   proof
>
(* goal: b divides p ==> (b = p) \/ (b = 1) *)
(* By contradiction, *)
e (spose_not_then strip_assume_tac); (* by contradiction *)
OK..
1 subgoal:
val it =

   0.  1 < p
   1.  ∀q. 1 < q ∧ q ≤ √ p ⇒ ¬(q | p)
   2.  b | p
   3.  b ≠ p
   4.  b ≠ 1
------------------------------------
F
:
   proof
>
(* What is divides? *)
divides_def;
val it =
   ⊢ ∀a b. a | b ⇔ ∃q. b = q * a:
   thm
>
(* So apply this definition. *)
e (`?a. p = a * b` by rw[GSYM divides_def]);
OK..
1 subgoal:
val it =

   0.  1 < p
   1.  ∀q. 1 < q ∧ q ≤ √ p ⇒ ¬(q | p)
   2.  b | p
   3.  b ≠ p
   4.  b ≠ 1
   5.  p = a * b
------------------------------------
F
:
   proof
>
(* With p expressed as a product of two factors, we can use: *)
two_factors_property;
val it =
   ⊢ ∀n a b. (n = a * b) ⇒ a ≤ √ n ∨ b ≤ √ n:
   thm
>
```

```
(* pause *)
(* Assert this result by two_factors_property *)
e (`a <= SQRT p \/ b <= SQRT p` by rw[two_factors_property]); (* >> *)
OK..
2 subgoals:
val it =

  0.  1 < p
  1.  ∀q. 1 < q ∧ q ≤ √ p ⇒ ¬(q | p)
  2.  b | p
  3.  b ≠ p
  4.  b ≠ 1
  5.  p = a * b
  6.  b ≤ √ p
------------------------------------
F


  0.  1 < p
  1.  ∀q. 1 < q ∧ q ≤ √ p ⇒ ¬(q | p)
  2.  b | p
  3.  b ≠ p
  4.  b ≠ 1
  5.  p = a * b
  6.  a ≤ √ p
------------------------------------
F

2 subgoals
:
   proof
>
(* This gives 2 subcases. *)
(* First case: a <= √ p ==> F *)
(* Clearly, a | p, by definition. *)
e (`a divides p` by metis_tac[divides_def, MULT_COMM]);
OK..
metis: r[+0+12]+0+0+0+0+0+0+0+0+1+0+1#
1 subgoal:
val it =

  0.  1 < p
  1.  ∀q. 1 < q ∧ q ≤ √ p ⇒ ¬(q | p)
  2.  b | p
  3.  b ≠ p
  4.  b ≠ 1
  5.  p = a * b
  6.  a ≤ √ p
  7.  a | p
------------------------------------
F
:
   proof
>
(* The aim is to put a as q in the implication, to have ¬(a | p), a contradiction. *)
(* Need to have 1 < a to fit into implication. *)
e (`a <> 0` by metis_tac[MULT, DECIDE``~(1 < 0)``]); (* since p = a * b, and 1 < p *)
OK..
metis: r[+0+13]+1+0+0+0+0+0#
1 subgoal:
val it =

  0.  1 < p
  1.  ∀q. 1 < q ∧ q ≤ √ p ⇒ ¬(q | p)
  2.  b | p
  3.  b ≠ p
  4.  b ≠ 1
  5.  p = a * b
  6.  a ≤ √ p
  7.  a | p
  8.  a ≠ 0
------------------------------------
F
:
```

```
   proof
>
e (`a <> 1` by metis_tac[MULT_LEFT_1]); (* since p = a * b, and b ≠ p *)
OK..
metis: r[+0+12]+0+0+0+0+0+0#
1 subgoal:
val it =

  0.  1 < p
  1.  ∀q. 1 < q ∧ q ≤ √ p ⇒ ¬(q | p)
  2.  b | p
  3.  b ≠ p
  4.  b ≠ 1
  5.  p = a * b
  6.  a ≤ √ p
  7.  a | p
  8.  a ≠ 0
  9.  a ≠ 1
------------------------------------
F
:
   proof
>
e (`1 < a` by decide_tac); (* since a ≠ 0, a ≠ 1 *)
OK..
1 subgoal:
val it =

  0.  1 < p
  1.  ∀q. 1 < q ∧ q ≤ √ p ⇒ ¬(q | p)
  2.  b | p
  3.  b ≠ p
  4.  b ≠ 1
  5.  p = a * b
  6.  a ≤ √ p
  7.  a | p
  8.  a ≠ 0
  9.  a ≠ 1
  10.  1 < a
------------------------------------
F
:
   proof
>
e (metis_tac[]); (* << by putting q = a in implication *)
OK..
metis: r[+0+12]+0+0+0+0+0+0+0+0+0+0+1#

Goal proved.
 [...........] ⊢ F

Goal proved.
 [..........] ⊢ F

Goal proved.
 [.........] ⊢ F

Goal proved.
 [........] ⊢ F

Goal proved.
 [.......] ⊢ F

Remaining subgoals:
val it =

  0.  1 < p
  1.  ∀q. 1 < q ∧ q ≤ √ p ⇒ ¬(q | p)
  2.  b | p
  3.  b ≠ p
  4.  b ≠ 1
  5.  p = a * b
  6.  b ≤ √ p
```

```
------------------------------------
F
:
   proof
>
(* Second case: b <= √ p ==> F *)
(* Already has b | p. *)
(* The aim is to put b as q in the implication, to have ¬(b | p), a contradiction. *)
(* Need to have 1 < b to fit into implication. *)
e (`b <> 0` by metis_tac[MULT_0, DECIDE``~(1 < 0)``]); (* since p = a * b, and 1 < p *)
OK..
metis: r[+0+11]+1+0+0#
1 subgoal:
val it =

  0.  1 < p
  1.  ∀q. 1 < q ∧ q ≤ √ p ⇒ ¬(q | p)
  2.  b | p
  3.  b ≠ p
  4.  b ≠ 1
  5.  p = a * b
  6.  b ≤ √ p
  7.  b ≠ 0
------------------------------------
F
:
   proof
>
e (`1 < b` by decide_tac); (* since b ≠ 0, b ≠ 1 *)
OK..
1 subgoal:
val it =

  0.  1 < p
  1.  ∀q. 1 < q ∧ q ≤ √ p ⇒ ¬(q | p)
  2.  b | p
  3.  b ≠ p
  4.  b ≠ 1
  5.  p = a * b
  6.  b ≤ √ p
  7.  b ≠ 0
  8.  1 < b
------------------------------------
F
:
   proof
>
e (metis_tac[]); (* << by putting q = b in implication *)
OK..
metis: r[+0+10]+0+0+0+0+0+0+0+0+0+1#

Goal proved.
 [.........] ⊢ F

Goal proved.
 [........] ⊢ F

Goal proved.
 [.......] ⊢ F

Goal proved.
 [......] ⊢ F

Goal proved.
 [.....] ⊢ F

Goal proved.
 [...] ⊢ (b = p) ∨ (b = 1)

Goal proved.
 [..] ⊢ prime p
val it =
   Initial goal proved.
```

```
⊢ ∀p. prime p ⇔ 1 < p ∧ ∀q. 1 < q ∧ q ≤ √ p ⇒ ¬(q | p):
   proof
>
(* Save the the theorem by providing a name, then drop the proof. *)
val prime_by_sqrt_factors = save_thm("prime_by_sqrt_factors", top_thm());
val prime_by_sqrt_factors =
   ⊢ ∀p. prime p ⇔ 1 < p ∧ ∀q. 1 < q ∧ q ≤ √ p ⇒ ¬(q | p):
   thm
>
drop();
OK..
val it = There are currently no proofs.: proofs
> (* Retrieve the theorem from library by name: *)
prime_by_sqrt_factors;
val it =
   ⊢ ∀p. prime p ⇔ 1 < p ∧ ∀q. 1 < q ∧ q ≤ √ p ⇒ ¬(q | p):
   thm
>

(* That's how an interactive theorem-prover works! *)
(* Bye! *)

Session Terminated.
- Goodbye.
```

Input

Type your input here.

Info

[load script=[primes-demo.hol]] wait [0 ms]