```
 1: (* $Id: mathfns-trace.ml,v 1.1 2012-02-07 19:44:53-08 - - $ *)
 2:
 3: (*
 4: * Power and square root functions.
 5: * Uses tail recursive accumulator-style coding.
 6: *)
 7:
 8: let even number = number mod 2 = 0
 9: let frexp' = frexp (* So we can trace these functions. *)
10: let ldexp' (frac, expt) = ldexp frac expt
11:
12: let rec power' (base, expt, result) = match expt with
13:     | 0                    -> result
14:     | expt when even expt -> power' (base *. base, expt / 2, result)
15:     | expt                 -> power' (base, expt - 1, base *. result)
16:
17: let power (base, expt) =
18:     if expt < 0 then power' (1. /. base, - expt, 1.)
19:                 else power' (base, expt, 1.)
20:
21: let rec sqrt' (number, approx) =
22:     let next = (approx +. number /. approx) /. 2.
23:     in  if abs_float (next -. approx) /. approx <= epsilon_float *. 2.
24:         then approx
25:         else sqrt' (number, next)
26:
27: let sqrt number =
28:     if number < 0.
29:     then raise (Invalid_argument ("sqrt of " ^ string_of_float number))
30:     else let frac, expt = frexp' number
31:          in  sqrt' (number, ldexp' (frac, expt / 2))
32:
33: ;;
34: #trace power  ;;
35: #trace power' ;;
36: #trace sqrt   ;;
37: #trace frexp' ;;
38: #trace ldexp' ;;
39: #trace sqrt'  ;;
40:
```

```
 1: bash-1$ ocaml
 2:         OCaml version 4.02.1
 3:
 4: # #use "mathfns-trace.ml";;
 5: val even : int -> bool = <fun>
 6: val frexp' : float -> float * int = <fun>
 7: val ldexp' : float * int -> float = <fun>
 8: val power' : float * int * float -> float = <fun>
 9: val power : float * int -> float = <fun>
10: val sqrt' : float * float -> float = <fun>
11: val sqrt : float -> float = <fun>
12: power is now traced.
13: power' is now traced.
14: sqrt is now traced.
15: frexp' is now traced.
16: ldexp' is now traced.
17: sqrt' is now traced.
18: # power (2., 12);;
19: power <-- (2., 12)
20: power' <-- (2., 12, 1.)
21: power' <-- (4., 6, 1.)
22: power' <-- (16., 3, 1.)
23: power' <-- (16., 2, 16.)
24: power' <-- (256., 1, 16.)
25: power' <-- (256., 0, 4096.)
26: power' --> 4096.
27: power' --> 4096.
28: power' --> 4096.
29: power' --> 4096.
30: power' --> 4096.
31: power' --> 4096.
32: power --> 4096.
33: - : float = 4096.
34: # power (11., 11);;
35: power <-- (11., 11)
36: power' <-- (11., 11, 1.)
37: power' <-- (11., 10, 11.)
38: power' <-- (121., 5, 11.)
39: power' <-- (121., 4, 1331.)
40: power' <-- (14641., 2, 1331.)
41: power' <-- (214358881., 1, 1331.)
42: power' <-- (214358881., 0, 285311670611.)
43: power' --> 285311670611.
44: power' --> 285311670611.
45: power' --> 285311670611.
46: power' --> 285311670611.
47: power' --> 285311670611.
48: power' --> 285311670611.
49: power' --> 285311670611.
50: power --> 285311670611.
51: - : float = 285311670611.
52: #
```

```
 1: bash-1$ ocaml
 2:         OCaml version 4.02.1
 3:
 4: # #use "mathfns-trace.ml";;
 5: val even : int -> bool = <fun>
 6: val frexp' : float -> float * int = <fun>
 7: val ldexp' : float * int -> float = <fun>
 8: val power' : float * int * float -> float = <fun>
 9: val power : float * int -> float = <fun>
10: val sqrt' : float * float -> float = <fun>
11: val sqrt : float -> float = <fun>
12: power is now traced.
13: power' is now traced.
14: sqrt is now traced.
15: frexp' is now traced.
16: ldexp' is now traced.
17: sqrt' is now traced.
18: # sqrt 2.;;
19: sqrt <-- 2.
20: frexp' <-- 2.
21: frexp' --> (0.5, 2)
22: ldexp' <-- (0.5, 1)
23: ldexp' --> 1.
24: sqrt' <-- (2., 1.)
25: sqrt' <-- (2., 1.5)
26: sqrt' <-- (2., 1.41666666666666652)
27: sqrt' <-- (2., 1.41421568627450966)
28: sqrt' <-- (2., 1.41421356237468987)
29: sqrt' <-- (2., 1.41421356237309492)
30: sqrt' --> 1.41421356237309492
31: sqrt' --> 1.41421356237309492
32: sqrt' --> 1.41421356237309492
33: sqrt' --> 1.41421356237309492
34: sqrt' --> 1.41421356237309492
35: sqrt' --> 1.41421356237309492
36: sqrt --> 1.41421356237309492
37: - : float = 1.41421356237309492
38: # sqrt 1000.;;
39: sqrt <-- 1000.
40: frexp' <-- 1000.
41: frexp' --> (0.9765625, 10)
42: ldexp' <-- (0.9765625, 5)
43: ldexp' --> 31.25
44: sqrt' <-- (1000., 31.25)
45: sqrt' <-- (1000., 31.625)
46: sqrt' <-- (1000., 31.6227766798419)
47: sqrt' <-- (1000., 31.6227766016837926)
48: sqrt' --> 31.6227766016837926
49: sqrt' --> 31.6227766016837926
50: sqrt' --> 31.6227766016837926
51: sqrt' --> 31.6227766016837926
52: sqrt --> 31.6227766016837926
53: - : float = 31.6227766016837926
54: #
```