lib/main/fast-power_unittest.ath

```
1
2  load "fast-power"
3
4  module Test1 {
5  define M1 := no-renaming
6
7  assert (theory-axioms Monoid.theory)
8
9  (!prove-property Monoid.fast-power.pap_1-correctness0 M1 Monoid.theory)
10 (!prove-property Monoid.fast-power.pap_1-correctness M1 Monoid.theory)
11 (!prove-property Monoid.fast-power.fpp_1-correctness0 M1 Monoid.theory)
12 (!prove-property Monoid.fast-power.fpp_2-correctness M1 Monoid.theory)
13 (!prove-property Monoid.fast-power.fpp_1-correctness M1 Monoid.theory)
14 (!prove-property Monoid.fast-power.correctness M1 Monoid.theory)
15
16 } # Test1
17
18
19 #-----------------------------------------------------------------------
20 load "list-of"
21
22 module Test2 {
23
24 declare list-pap_1, list-pap_2: (T) [(List T) (List T) N] -> (List T)
25 declare list-fpp_1, list-fpp_2: (T) [(List T) N] -> (List T)
26
27 declare join*: (T) [(List T) N] -> (List T)
28 declare fast-join*: (T) [(List T) N] -> (List T)
29
30 define M1 :=
31   (renaming |{Monoid.+* := join*, Monoid.+ := List.join, Monoid.<0> := nil,
32              Monoid.fast-power := fast-join*, Monoid.pap_1 := list-pap_1,
33              Monoid.pap_2 := list-pap_2, Monoid.fpp_1 := list-fpp_1,
34              Monoid.fpp_2 := list-fpp_2}|)
35
36 # Define join* and fast-join* as instances of abstract functions:
37
38 assert (M1 (join [Monoid.Power.right-zero Monoid.Power.right-nonzero]
39                  (rev Monoid.fast-power.axioms)))
40
41 (!prove-property Monoid.fast-power.correctness M1 'Monoid)
42
43 expand-input fast-join* [(alist->clist char-ord) int->nat]
44
45 transform-output eval [(clist->alist char)]
46
47 (print (eval (fast-join* "Hello " 6)))
48
49 (print (eval (fast-join* "All work and no play makes Jack a dull boy.\n" 13)))
50
51 (print (eval (fast-join* " x " 33)))
52
53 } # Test2
54
55 #-----------------------------------------------------------------------
56 load "nat-power"
57
58 module Test3 {
59 extend-module N {
60   declare pap_1, pap_2: [N N N] -> N
61   declare fpp_1, fpp_2: [N N] -> N
62   declare fast-power: [N N] -> N
63 } # N
64
65 define M1 := (renaming
66               |{Monoid.+* := N.**, Monoid.+ := N.*,  Monoid.<0> := N.one,
67                 Monoid.pap_1 := N.pap_1, Monoid.pap_2 := N.pap_2,
```

```
                Monoid.fpp_1 := N.fpp_1, Monoid.fpp_2 := N.fpp_2,
                Monoid.fast-power := N.fast-power}|)

# Define the functions in N as instances of the abstract functions

assert (M1 (rev Monoid.fast-power.axioms))

(!prove-property Monoid.fast-power.correctness M1 Monoid.theory)

expand-input N.fast-power [int->nat int->nat]

expand-input N.** [int->nat int->nat]

transform-output eval [nat->int]

(eval (N.** 4 3))        # expand-input works for this

#(eval (N.fast-power 4 3))  # but not for this  ?????
(eval (N.fast-power (S S S S zero) (S S S zero)))

# (eval (N.fast-power (S S S S S S S zero) (S S S S S S S zero)))

} # Test3

#-------------------------------------------------------------------------

module Test4 {
declare **: (T) [T N] -> T

set-precedence ** 400

define M1 := (renaming |{Monoid.+* := **}|)

assert (M1 (rev (theory-axioms MM.theory)))

(!prove-property Monoid.fast-power.correctness M1 MM.theory)

} # Test4

#-------------------------------------------------------------------------

module Test5 {
declare **: [Real N] -> Real

set-precedence ** 400

declare pap_1, pap_2: [Real Real N] -> Real
declare fpp_1, fpp_2: [Real N] -> Real
declare fast-power: [Real N] -> Real

define M1 := (renaming |{Monoid.+* := **, Monoid.+ := *, Monoid.<0> := 1,
                         Monoid.pap_1 := pap_1, Monoid.pap_2 := pap_2,
                         Monoid.fpp_1 := fpp_1, Monoid.fpp_2 := fpp_2,
                         Monoid.fast-power := fast-power}|)

assert (M1 (rev Monoid.fast-power.axioms))

(eval (fast-power 3 zero))

(eval (fast-power 3 (S S S S zero)))

expand-input fast-power [id int->nat]

(eval (fast-power 3 5))

(eval (fast-power 7 7))

(eval (fast-power 2 20))

} # Test 5
```