

## lib/main/fast-power\_trace.ath

```

1  # setup-trace, a procedure For tracing an eval execution using code
2  # compiled from rewrite rules. 'G' ("Gas") can be used to control
3  # number of trace steps via the number of 'G's around a term to be
4  # evaluated. The following function inserts one G so the rule
5  # effectively becomes a rule for G rather than for the root function
6  # of the rule, and evaluation of the compiled code takes one step and
7  # stops.
8
9  declare G: (T) [T] -> T
10
11 define (setup-trace E) :=
12   match E {
13     (forall (some-list L) (= left right)) => (forall* L (= (G left) right))
14     | (forall (some-list L) (iff left right)) => (forall* L (iff (G left) right))
15     | (forall (some-list L) (if cond (= left right))) =>
16       (forall* L (if cond (= (G left) right)))
17     | (forall (some-list L) (if cond (iff left right))) =>
18       (forall* L (if cond (iff (G left) right)))
19   }
20
21 # Example:
22
23 #-----
24 load "fast-power"
25
26 #-----
27 extend-module Monoid {
28
29   extend-module fast-power {
30
31     define trace-axioms := (map setup-trace axioms)
32
33   } # fast-power
34 } # Monoid
35
36 module Test1 {
37
38   open Monoid
39
40   datatype Exp := xone | a | (* Exp Exp)
41
42   define M1 := (renaming [+ * <0> xone])
43
44   assert (M1 fast-power.trace-axioms)
45
46   (red-code G)
47
48   (reduce (G (fast-power a (S S S S S S S S S S zero))))
49   (reduce (G G (fast-power a (S S S S S S S S S S zero))))
50   (reduce (G G G (fast-power a (S S S S S S S S S S zero))))
51   (reduce (G G G G (fast-power a (S S S S S S S S S S zero))))
52   (reduce (G G G G G (fast-power a (S S S S S S S S S S zero))))
53   (reduce (G G G G G G (fast-power a (S S S S S S S S S S zero))))
54   (reduce (G G G G G G G (fast-power a (S S S S S S S S S S zero))))
55
56   define (gas-up t k) :=
57     match k {
58       (S k) => (gas-up (G t) k)
59       | zero => t
60     };
61
62   define (run f n m format) :=
63     letrec {loop := lambda (k)
64       let {t := (reduce (gas-up (f a (int->nat n)) (int->nat k)))};
65       _ := (print "k = " k " " " (format t) "\n")}
66     match t {
67       ((some-symbol f) (some-list args)) =>

```

```

68         check {
69             (equal? f *) => ()
70             | (negate (equal? k m)) => (loop (plus k 1))
71             | else => ()
72         }
73     }}
74     (loop 1)
75
76
77 (run fast-power 10 8 id)
78
79 (run fast-power 10 1 id)
80
81 define (display t) :=
82   letrec {count := lambda (t)
83     match t {
84       (++ x y) => (plus (count x) (count y))
85       | _ => 1
86     };
87     v->s := val->string}
88   match t {
89     (fpp_1 x n) =>
90       (join "(fpp_1 a^" (v->s (count x)) " " (v->s (nat->int n)) ")")
91     | (fpp_2 x n) =>
92       (join "(fpp_2 a^" (v->s (count x)) " " (v->s (nat->int n)) ")")
93     | (pap_1 x y n) =>
94       (join "(pap_1 a^" (v->s (count x)) " a^" (v->s (count y)) " "
95         (v->s (nat->int n)) ")")
96     | (pap_2 x y n) =>
97       (join "(pap_2 a^" (v->s (count x)) " a^" (v->s (count y)) " "
98         (v->s (nat->int n)) ")")
99     | _ => (join "a^" (v->s (count t)) "\n")
100   }
101
102 (run fast-power 13 8 display)
103
104 (run fast-power 10 8 display)
105
106 (run fast-power 100 20 display)
107
108 (run fast-power 1000 20 display)
109
110 # Instead of starting from the beginning for each k, start from the
111 # (k-1)st term:
112
113 define (run1 f n m format) :=
114   letrec {loop := lambda (t k)
115     let {t := (reduce (G t));
116         _ := (print "k = " k " " (format t) "\n")}
117     match t {
118       ((some-symbol f) (some-list args)) =>
119         check {
120             (equal? f *) => ()
121             | (negate (equal? k m)) => (loop t (plus k 1))
122             | else => ()
123         }
124     }}
125     (loop (f a (int->nat n)) 1)
126
127 (run1 fast-power 13 8 display)
128
129 (run1 fast-power 10 8 display)
130
131 (run1 fast-power 100 20 display)
132
133 (run1 fast-power 1000 20 display)
134
135 (run1 fast-power 4095 40 display)
136
137 } # Test1

```