lib/main/fast-power_trace.ath

```
# setup-trace, a procedure For tracing an eval execution using code
# compiled from rewrite rules.  'G' ("Gas") can be used to control
# number of trace steps via the number of 'G's around a term to be
# evaluated. The following function inserts one G so the rule
# effectively becomes a rule for G rather than for the root function
# of the rule, and evaluation of the compiled code takes one step and
# stops.

declare G: (T) [T] -> T

define (setup-trace E) :=
  match E {
    (forall (some-list L) (= left right)) => (forall* L (= (G left) right))
  |  (forall (some-list L) (iff left right)) => (forall* L (iff (G left) right))
  | (forall (some-list L) (if cond (= left right))) =>
        (forall* L (if cond (= (G left) right)))
  | (forall (some-list L) (if cond (iff left right))) =>
        (forall* L (if cond (iff (G left) right)))
  }

# Example:

#--------------------------------------------------------------------------
load "fast-power"

#--------------------------------------------------------------------------
extend-module Monoid {

extend-module fast-power {

define trace-axioms := (map setup-trace axioms)

} # fast-power
} # Monoid

module Test1 {

open Monoid

datatype Exp := xone | a | (* Exp Exp)

define M1 := (renaming [+ *  <0> xone])

assert (M1 fast-power.trace-axioms)

(red-code G)

(reduce (G (fast-power a (S S S S S S S S S zero))))
(reduce (G G (fast-power a (S S S S S S S S S zero))))
(reduce (G G G (fast-power a (S S S S S S S S S zero))))
(reduce (G G G G (fast-power a (S S S S S S S S S zero))))
(reduce (G G G G G (fast-power a (S S S S S S S S S zero))))
(reduce (G G G G G G (fast-power a (S S S S S S S S S zero))))
(reduce (G G G G G G G (fast-power a (S S S S S S S S S zero))))

define (gas-up t k) :=
  match k {
    (S k) => (gas-up (G t) k)
  | zero => t
  };

define (run f n m format) :=
  letrec {loop := lambda (k)
                    let {t := (reduce (gas-up (f a (int->nat n)) (int->nat k)));
                         _ := (print "k = " k "   " (format t) "\n")}
                    match t {
                      ((some-symbol f) (some-list args)) =>
```

```
                         check {
                           (equal? f *) => ()
                         | (negate (equal? k m)) => (loop (plus k 1))
                         | else => ()
                         }
                      }}
  (loop 1)


(run fast-power 10 8 id)

(run fast-power 10 1 id)

define (display t) :=
 letrec {count := lambda (t)
                      match t {
                        (++ x y) => (plus (count x) (count y))
                      | _ => 1
                      };
        v->s := val->string}
 match t {
   (fpp_1 x n) =>
      (join "(fpp_1 a^" (v->s (count x)) " " (v->s (nat->int n)) ")")
 | (fpp_2 x n) =>
      (join "(fpp_2 a^" (v->s (count x)) " " (v->s (nat->int n)) ")")
 | (pap_1 x y n) =>
      (join "(pap_1 a^" (v->s (count x)) " a^" (v->s (count y)) " "
            (v->s (nat->int n)) ")")
 | (pap_2 x y n) =>
      (join "(pap_2 a^" (v->s (count x)) " a^" (v->s (count y)) " "
            (v->s (nat->int n)) ")")
 | _ => (join "a^" (v->s (count t)) "\n")
 }

(run fast-power 13 8 display)

(run fast-power 10 8 display)

(run fast-power 100 20 display)

(run fast-power 1000 20 display)

# Instead of starting from the beginning for each k, start from the
# (k-1)st term:

define (run1 f n m format) :=
  letrec {loop := lambda (t k)
                      let {t := (reduce (G t));
                           _ := (print "k = " k "    " (format t) "\n")}
                      match t {
                        ((some-symbol f) (some-list args)) =>
                         check {
                           (equal? f *) => ()
                         | (negate (equal? k m)) => (loop t (plus k 1))
                         | else => ()
                         }
                      }}
  (loop (f a (int->nat n)) 1)

(run1 fast-power 13 8 display)

(run1 fast-power 10 8 display)

(run1 fast-power 100 20 display)

(run1 fast-power 1000 20 display)

(run1 fast-power 4095 40 display)

} # Test1
```