lib/basic/testing-aids.ath

```
# Methods for use in testing proof code.

# The following methods allow execution to continue even if (!test) fails.
# Unfortunately, the error message generated when (!test) fails is
# not printed.  For those that are reported to fail, enter
# (!test) at the interactive command prompt to see the error message.

(define (proof-test test id )
  (dlet ((_ (print (join "\nProof test " id ", should succeed:")))
         (attempt
          (dtry (!test)
                (dlet ((_ (print (join "\nTest " id " FAILED!\n"))))
                  (!true-intro)))))
    (dcheck ((equal? attempt true)
             (!true-intro))
            (else
             (dlet ((_ (print (join "\nTest " id " succeeded, as expected.\n"))))
               (!claim attempt))))))

(define (negative-proof-test test id reason)
  (dlet ((_ (print (join "\nNegative proof test " id ", should fail, since " reason ".\n")))
         (attempt
          (dtry (!test)
                (dlet ((_ (print (join "\nThere was an error, as expected, in test " id ".\n"))))
                  (!true-intro)))))
    (dcheck ((equal? attempt true)
             (!true-intro))
            (else
             (dlet ((_ (print (join "\nOOPS! - Unexpectedly, proof test " id " succeeded!\n"))))
               (!claim attempt))))))

# Methods for use in testing expression code (i.e., term or sentence)

define result := (cell true)
define test-cases := (cell [])
define test-failures := (cell [])

define test :=
 lambda (id test expected)
  let {_ := (set! test-cases (add id (ref test-cases)));
       _ := (set! result 'failed);
       _ := (process-input-from-string
              (join "(try (set! result " test ") (set! result 'failed))"));
       _ := (print "\n----------------------------------------\n")}
    check {(equal? (ref result) 'failed) =>
             let {_ := (print "\nTest " id ": " test
                              "\nError: THE TEST FAILED!\n")}
               (set! test-failures (add id (ref test-failures)))
          | else =>
             let {_ := (print "\nTest " id ": " test
                              "\ncompleted execution, returning: ");
                  _ := (write (ref result))}
               check {(equal? expected 'none) => ()
                     | (equal? (ref result) expected) =>
                         (print "\nas expected.\n")
                     | else =>
                         let {_ := (print "\nbut the expected result was: ");
                              _ := (write expected);
                              _ := (print "\nError: THE TEST FAILED!\n")}
                           (set! test-failures (add id (ref test-failures)))
                     }
          }

define test-summary :=
 lambda ()
   let {L := (rev (ref test-cases));
        F := (rev (ref test-failures));
        _ := (print "\n\n====================================================\n\n")}
     check {(equal? (length F) 0) =>
            (print "All" (length L) "tests succeeded.\n")
           | else =>
              let {_ := (print "Of" (length L) "tests:   " L "\n")}
                (print " " (length F) "FAILED:   " F "\n")}

define run-tests :=
 lambda (K)
   letrec {loop :=
            lambda (L)
              match L {
                (split [_id _test] rest) =>
                  let {_ := (test _id _test 'none)}
                    (loop rest)
              | [] => (print "========================================================\n\n")};
          _ := (print "========================================================\n\n");
          _ := (print "Running" ((length K) div 2) "tests")}
    (loop K)

()
```