Postfix

The Team of Fu

September 19, 2013

Contents

1 Introduction

Remark This is a literate program. [1] Source code *and* PDF documentation spring from the same, plain-text source files.

```
(defproject ex1 "0.1.0-SNAPSHOT"
  :description "Project Fortune's Postfix Expression Evaluator"
  :url "http://example.com/TODO"
  :license {:name "TODO"
            :url "TODO"}
  :dependencies [[org.clojure/clojure  "1.5.1"]
                 ]
  :repl-options {:init-ns ex1.core})
```

[1] See `http://en.wikipedia.org/wiki/Literate_programming`.

2 The Namespace

```
(ns ex1.core)
```

3 Functions

This *postfix* function receives a sequence of expressions es. It produces a reduction of a binary function f over the empty vector [ ] and es. f receives a vector a and an expression e. a implements a stack. If e is a function, f replaces the top two elements r and l of a with $e(l, r)$, the application of function e to those arguments. Otherwise, f just *cons*-es e to the front of a. r and l appear in opposite order in a to the order that e receives them; while reducing arguments left-to-right in es, *postfix* reverses them when *cons*-ing to the front of the stack.

```
(defn postfix [& es]
  (reduce
    (fn f [a-vec e]
      (if (fn? e)
          (let [[r l & m] a-vec]
            (cons (e l r) m))
          (cons e a-vec)))
    []
    es))
```

```
(defn zero-out-divide-by-zero [fn l r]
  ; if function is division (/) and right-hand operand is 0
  (if (and
        (= fn /)
        (some #{r} [0 0.0 0M]))
    ; return operand
    r
    ; otherwise, execute function
    (fn l r)))

(defn binary-apply-fn [fn r l]
  (apply map (partial zero-out-divide-by-zero fn)
    (list r l)))

(defn postfix-collections [& e]
  (mapcat identity
    (reduce #(if (fn? %2)
               (let [[r l & m]%]
                 (cons (binary-apply-fn %2 l r) m))
               (cons %2 %))[]e)))
```

4 Core Unit-Test File

```
(ns ex1.core-test
  (:require [ex1.core :refer :all]
            [clojure.test :refer :all]
  ))



(deftest null-test
  (testing "null test"
    (is (= 1 1))))
```

5 A REPL-based Solution

To run the REPL for interactive programming and testing in org-mode, take the following steps:

1. Set up emacs and nRepl (TODO: explain; automate)
2. Edit your init.el file as follows (TODO: details)
3. Start nRepl while visiting the actual |project-clj| file.
4. Run code in the org-mode buffer with `C-c C-c`; results of evaluation are placed right in the buffer for inspection; they are not copied out to the PDF file.