Learn about closures and trampolines

Andrea Giugliano

November 2, 2017

Contents

1 DONE learn about lexical scope in Elisp

```
(setq test (lexical-let ((foo "bar"))
             (lambda ()
               foo)))

(funcall test)

(let ((foo "something-else"))
  (funcall test))
```

2 DONE learn about trampolines

Based on: `http://www.datchley.name/recursion-tail-calls-and-trampolines/`

Elisp does not have optimization for tail recursion. For this issue one can use an higher order function to avoid overflowing the stack:

```
(defun range (s e &optional res)
  (let ((res (cons s res)))
   (if (eql s e)
      (reverse res)
      (range (if (< s e) (+ 1 s) (- 1 s)) e res))))

(range 1 4)
```

For big numbers this fails:

```
(range 1 31181)
```

Important: the following apply only if lexical binding is enabled (we need closures):

```
(setq lexical-binding t)
```

We can introduce the trampoline function:

```
(defun trampoline (fn &rest args)
  (progn
    (message "%s" fn)
    (setf v (apply fn args))
    (while (functionp v)
      (message "%s" v)
      (setf v (funcall v)))
    v))
```

Note that we keep applying function calls (that do not accumulate in the stack) until we get a non callable value.

We need to modify the range function, as it must return something that we can execute singularly instead its recursive call:

```
(defun range (s e &optional res)
  (let ((res (cons s res)))
    (if (eql s e)
        (reverse res)
      (lambda ()
        (range
         (if (< s e) (+ 1 s) (- 1 s))
         e
         res)))))

(trampoline #'range 1 4)
```

So we have basically split recursive calls in independent function calls (and saved our stack).