Programming Languages (TC-2006)

Higher-order functions in Racket

José Carlos Ortiz Bayliss
jcobayliss@tec.mx

Xavier Fernando C. Sánchez Díaz
sax@tec.mx

Contents

Tecnológico de Monterrey

Functions in functional languages

- Functions can be passed as arguments to other functions.
- Functions can be the result of a function.
- Functions can also be part of a data structure.

Evaluation order

- In general, Racket expressions are evaluated in applicative order.
- Special forms such as `lambda`, `if` and `cond` are exceptions to this rule. The implementation of `if` checks to see whether the first argument evaluates to `#t` . If so, it returns the value of the second argument, without evaluating the third argument. Otherwise it returns the value of the third argument, without evaluating the second.

Functions as arguments

You can create functions that receive functions as arguments:

```
(define (myApply function a b)
   (function a b)
)
```

- `(myApply + 8 5)` $\Rightarrow$ `13`
- `(myApply * 8 5)` $\Rightarrow$ `40`
- `(myApply remainder 8 5)` $\Rightarrow$ `3`

Applying a function to each element in a list: `map`

```
(map function arg1 arg2 ...argn)
```

- Returns a list with the result of applying the function provided as argument to each element in the input lists.

For example:

- `(map sqrt '(4 9 16 25 36))` ⇒ `'(2 3 4 5 6)`
- `(map + '(1 2 3) '(4 5 6))` ⇒ `'(5 7 9)`
- `(map list '(1 2 3) '(4 5 6))` ⇒ `'((1 4) (2 5) (3 6))`

Tecnológico de Monterrey

Calculate the transpose of a matrix

```
(define (transpose matrix)
   (cond
      ((null? (car matrix)) null)
      (else (cons (map car matrix) (transpose (map cdr
         matrix))))
   )
)
```

- `(transpose '((10 4 8) (4 7 10)))` ⇒ `'((10 4) (4 7) (8 10))`

Tecnológico de Monterrey

Applying a function to each element in a list: `for-each`

```
(for-each function arg)
```

- Applies the function provided as argument to each element in the input list.
- This method does not produce a list.

For example:

- `(for-each display '("hello" " " "world" "\n"))` ⇒
  `hello world`

Tecnológico de Monterrey

Applying a function to each element in a list: `apply`

```
(apply function arg1 arg2 ...argn)
```

- Evaluates the function on each element in the list resulting from calling `(append (list arg1 arg2 ...) argn)`.
- The last argument must be a list.

For example:

- `(apply + '(4 10 17 3))` $\Rightarrow$ `34`
- `(apply + 4 10 '(17 3))` $\Rightarrow$ `34`

Tecnológico de Monterrey

Functions that return functions

You can create functions that return functions:

```
(define (get-function option)
   (cond
      ((= option 1) +)
      ((= option 2) -)
      ((= option 3) *)
      ((= option 4) /)
   )
)
```

- `((get-function 1) 10 3)` ⇒ `13`
- `((get-function 2) 10 3)` ⇒ `7`
- `((get-function 3) 10 3)` ⇒ `30`
- `((get-function 4) 10 3)` ⇒ `10/3`

Tecnológico de Monterrey

Lists of functions

You can create lists of functions:

```
(define functions (list + - * /))
```

- `((car functions) 10 3)` $\Rightarrow$ `13`
- `((cadr functions) 10 3)` $\Rightarrow$ `7`
- `((caddr functions) 10 3)` $\Rightarrow$ `30`
- `((cadddr functions) 10 3)` $\Rightarrow$ `10/3`

The benefit of using higher-order functions: many functions based on one

You can use a generic function:

```
(define (fold op base x)
   (if (null? x)
      base
      (op (car x) (fold op base (cdr x)))
   )
)
```

To produce various implementations:

- `(define (sum x) (fold + 0 x))`
- `(define (mul x) (fold * 1 x))`
- `(define (concat x y) (fold cons x y))`

Tecnológico de Monterrey

Special form `lambda`

The special form `lambda` is used to define a function that has no name.

```
(lambda (parameters) body)
```

The, we can also define functions in Racket by using `lambda`:

```
(define pow2 (lambda (x) (* x x)))
```

Which is equivalent to:

```
(define (pow2 x) (* x x))
```

Tecnológico de Monterrey

Currying

Currying is the technique of translating the evaluation of a function that takes multiple arguments into evaluating a sequence of functions, each with a single argument.

Standard form:

```
((lambda (x y) (+ x y)) 5 6)
```

Curried version:

```
(((lambda (x) (lambda (y) (+ x y))) 5) 6)
```

Tecnológico de Monterrey