package: ofp 0.5.0

Object-Functional Programming

Charlotte Maia

June 12, 2010

This vignette is the third vignette in the ofp package, an R package for object oriented R programming, with a dual focus on enhanced S3 programming and object-functional programming. Here, we first consider the notion of object-functional programming, then consider enhanced functions.

Introduction

In the first vignette, we considered the idea of combining object oriented programming with computational mathematics, and sought to create an object oriented system within R, to support computational mathematics. In the second vignette, we considered environments, and how they support rich and powerful data structures.

Now we consider object-functional programming, a programming paradigm that combines the strengths of object oriented programming with the strengths of functional programming. Object-functional programming builds on the ideas in those earlier vignettes. It provides us with an extremely powerful system for using object oriented programming with computational mathematics. It also makes use of function environments.

The current view of the author, is that object-functional programming revolves around four concepts:

1. Functions are objects. They can have and do anything that an object can have and do, constructors, inheritance and attributes.
2. Functions are the most important kind of objects. Object-functional programs should be designed to emphasize functions.
3. Functions can be evaluated directly.
4. Functions can return functions.

In principle, we allow both functions and methods. However, given that S3 methods are functions, that principle is a trivial one. Here, we do not require that functions are strictly functions in the mathematical sense, however this is still a desirable property. i.e. We allow functions to modify state, or to return different values given the same arguments.

To support these ideas, we consider enhanced functions. These are extended R functions, that are given their own environment (which we shall regard as a container object). We can assign values to the container object, and regard those values as function attributes.

A similar idea is seen in R's splinefun and ecdf functions.

Enhanced Functions

Enhanced functions are created with the FUNCTION function. Creating FUNCTIONs is similar to creating VECTORs described in the first vignette. Rather than providing a seed vector, we provide a seed function, along with any attributes that we require. Here, is an example, for a lookup function.

```
> #first, a suitable data structure to look things up in
> key = LETTERS [1:6]
> value = c ("A's value", "B's value", "C's value",
         "D's value", "E's value", "F's value")
> table = data.frame (key, value, stringsAsFactors=FALSE)
> table
  key     value
1   A A's value
2   B B's value
3   C C's value
4   D D's value
5   E E's value
6   F F's value


> #second, the function itself
> f = function (key) table [match (key, d [,1]), 2]
> lookup = FUNCTION (f, d=table)


> #calling the function
> lookup ("D")
[1] "D's value"
```

Sometimes me may wish to have a function, where an attribute name is the same as an argument name. Probably not the best design pattern. However it can still be achieved using a self reference.

```
> f = function (x) .$x + x
> f = FUNCTION (f, x=10)
> f (2)
[1] 12
```

Noting that we can print the function and access the attributes directly

```
> f
FUNCTION (x)
.$x + x
attributes:
x
> f$x
[1] 10
```

Extending Functions

Extending a function could mean different things. It could mean extending it's class attribute and giving it further attributes. It could mean changing or extending the body of the function. It could even mean changing or extending it's attribute list.

Here, we regard extending a function, as a combination of extending it's class attribute, potentially giving it more attributes, and potentially changing the body of the function. If we do not wish to change the body of the function, then we can use the extend function in the usual way.

```
> f = function (x) x
> linef1 = extend (FUNCTION (f), "line")
> linef1
FUNCTION (x)
x
```

However, if we do indeed wish to change the body, then we need the extendf function, which is the same as the extend function, except that the third argument is a function with the new body.

```
> f = function (x) a + b * x
> linef2 = extendf (linef1, "fancyline", f, a=0, b=1)
> linef2
FUNCTION (x)
a + b * x
attributes:
a b
```

Function Constructors

It's possible to create function classes, with function constructors. These can also be considered function-valued functions. The previous example can be reformulated in several ways.

```
> quad = function (a=0, b=1, c=0)
 {        f = function (x) a + b * x + c * x^2
         extend (FUNCTION (f), "quad", a, b, c)
 }


> q = quad (0, 0, 1)


> q (1:10)
 [1]   1   4   9  16  25  36  49  64  81 100
```

S3 Methods

We can create S3 methods for functions, in the usual way.

```
> summary.quad = function (q, ...)
         cat ("quad object\na:", q$a, "b:", q$b, "c:", q$c, "\n")


> summary (q)
quad object
a: 0 b: 0 c: 1
```

Other Methods

It's possible for a function to contain other functions as attributes. It's also possible to set the environment of those functions to be equal to the primary function's environment.

```
> f = function (x) g (x)
> g = function (x) 2 * x + k
> f = FUNCTION (f, g, k=2)
> environment (f$g) = environment (f)


> f (4)

[1] 10
```