A Critique of $R$
from the Perspective of Programming Language Theory

Sungwoo Park
Pohang University of Science and Technology, Korea

Japanese $R$ Use $R$'s meeting
Dec 8, 2006
First Encounter with R

• A regional workshop on R in May, 2006
  – motto:
    "Don't teach SAS. Teach R instead."

• An invited talk at the workshop
  – supposed to say "SAS is bad. R is good."
  – actually said "SAS is really bad, R is also bad."

• R seemed to have quite a few flaws in its design.
'Towards 2020 Science'

• A report on
  "the role and future of science over the next 14 years"
  – by the 2020 Science Group
    • over 30 scientists elected for their expertise
    • met over an intense 3 days in July 2005
  – 86 pages
  – sponsored by Microsoft
Towards 2020 Science: A Draft Roadmap
"Many niche areas of software development exist where alternatives and/or enhancements of managed platforms are deployed and used by scientists, including ... and the language R."
R Dissected

• Popularity of R in the statistics community
  – statistical computing
  – high level graphics
    
    "Many users will come to R mainly for its graphical facilities." – An introduction to R

• R as a hybrid language
Caveat

• A technical debate on
  "Why is your programming language good/bad?"
  \approx

  A religious debate on
  "What is the best religion?"

  \Rightarrow \text{Take this presentation with a grain of salt.}

• One thing is certain, however:
  "More features do not always mean
  a better programming language."
Outline

• Introduction ✓
  • Programming paradigm for R
    – Imperative language?
    – Functional language?
    – Both?
    – Or neither?
  • Lexical scoping
  • Further analysis
  • A functional language for R users
  • Conclusion
Imperative vs Functional

<table>
<thead>
<tr>
<th>Imperative languages</th>
<th>Functional languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Everything denotes a command.</td>
<td>• Everything denotes a value.</td>
</tr>
<tr>
<td>• Variables are mutable.</td>
<td>• Variables are immutable.</td>
</tr>
<tr>
<td>• Functions are not first-class objects.</td>
<td>• <strong>Functions as first-class objects.</strong></td>
</tr>
</tbody>
</table>

Functions are first-class objects in R.
Does this mean that R is a functional language?
Imperative Languages

• A program consists of **commands**.
  – command = “do something”

• Nothing wrong:

```plaintext
if (x == 1)
  x ← x + 1
else
  x ← x - 1
```

• Nothing wrong either:

```plaintext
if (x == 1)
  x ← x + 1
```
Functional Languages

• A program consists of expressions.
  – expression = “obtain a value”

• Nothing wrong:
  
  \[
  \text{if} \ (x \ == \ 1) \\
  \quad x \ + \ 1 \\
  \text{else} \\
  \quad x \ - \ 1
  \]

• But this does not make sense:
  
  \[
  \text{if} \ (x \ == \ 1) \\
  \quad x \ + \ 1 \\
  \quad \text{What is the value if } x \neq 1?
  \]
R: Not Functional

> foo
> function (x) {
>   if (x <= 0) 1
> }
> foo (0)
[1] 1
> foo (1)
> foo (0) + foo (1)
numeric(0)

if (\(x \leq 0\)) 1 is not an expression: it does not always evaluate to a value.
foo is not a function: it is not defined on positive integers.

numeric(0) is what?
Variable Binding

> x = 1 + 1
> x
[1] 2

• A variable \( x \) is “\textbf{bound}” to value 2.
• From now on, any occurrence of \( x \) is replaced by 2.

> y = x + x
> y
[1] 4
Variables are NOT Variable?

**Imperative languages**
- The contents of a variable can change.
  
  ```
  \texttt{> x <- 0}
  \texttt{> x}
  \texttt{[1] 0}
  \texttt{> x <- 1}
  \texttt{> x}
  \texttt{[1] 1}
  ```

**Functional languages**
- The contents of a variable *never* change.
  
  ⇒ You **cannot** assign a new value to a variable.
  
  • Surprise?
  ⇒ nothing special in functional languages

So, R is an imperative language?
References in Functional Languages

• There are assignments, but not to variables.
  ⇒ assignments to **references**.

• Reference (≈ pointer in C)
  – points to a heap cell.

- `val x = ref 0;`  // initialization
- `val x = ref 0 : int ref`
- `!x;`            // dereferencing
- `val it = 0 : int`
- `x := 1;`        // assignment
- `val it = () : unit`
- `!x;`            // dereferencing
- `val it = 1 : int`
R: Neither Functional Nor Imperative

Imperative languages
- Everything denotes a command.
- **Variables are mutable.**
- Functions are not first-class objects.

Functional languages
- Everything denotes a value.
- Variables are immutable.
- **Functions as first-class objects**

- Functions are first-class objects, but no clear definition of `commands` or `expressions` no distinction between `variables` and `references`
- A fatal design decision
  ⇒ engenders many idiosyncrasies in the definition.
Outline

• Introduction ✓
• Programming paradigm for R ✓
• Lexical scoping
• Further analysis
• A functional language for R users
• Conclusion
Lexical Scoping

- Uses bindings that are active at the time of creating a function.

```r
y ← 0
foo ←
  function () {
    y ← 100
    function (x) x + y
  }

> foo () (0)
[1] 100
```

- Useful in R because functions are first-class objects.
- Unfortunately R fails to implement lexical scoping correctly.
No Lexical Scoping

- R
  ```
  > x ← 1
  > foo ← function (y) x + y
  > x ← 100
  > foo (0)
  [1] 100
  ```

- Standard ML
  ```
  val x = 1
  val foo = fn y => x + y
  val x = 100
  - foo 0;
  val it = 1 : int
  ```

- Dynamic scoping at the top-level
- lexical scoping at inner levels
  ⇒ for the sake of compatibility?
\( x \leftarrow 1 \)

```r
foo \leftarrow 

function (y) {
  if (y < 10)
    x \leftarrow x + 1
}

> foo (0)
> x
[1] 2
```

\( x \leftarrow 1 \)

```r
foo \leftarrow 

function (y) {
  if (y < 10)
    x \leftarrow x + 1
}

> foo (0)
> x
[1] 1
```
Special Top Level?

"While purely functional languages do not allow assignment, they allow it at top-level; otherwise the user could not define new functions."

– Lexical Scope and Statistical Computing

⇒ Wrong!

– There is nothing special for the top level.
– assignment at the top level?
  • No, it's just a binding.

• Due to failure to distinguish between variables and references, or bindings and assignments.
Lexical Scoping in CS

"Although the usual definition of static or lexical scope in computer science is that ..., this definition is not specific enough. Computer scientists tend not to differentiate as finely because their concerns are different."

– Lexical Scope and Statistical Computing

⇒ This is absolutely wrong.

\[
\begin{align*}
\text{expression} & \quad e ::= x \mid \lambda x : A. e \mid e \; e \\
\text{environment} & \quad \eta ::= \cdot \mid \eta, x \leftrightarrow v \\
\frac{x \leftrightarrow v \in \eta}{\eta \vdash x \leftrightarrow v} & \quad \text{Var}_e \\
\frac{\eta \vdash \lambda x : A. e \leftrightarrow [\eta, \lambda x : A. e]}{\eta \vdash [\eta', \lambda x : A. e]} & \quad \text{Lam}_e \\
\frac{\eta \vdash e_1 \leftrightarrow [\eta', \lambda x : A. e]}{\eta \vdash e_2 \leftrightarrow v_2} & \quad \eta', x \leftrightarrow v_2 \vdash e \leftrightarrow v \\
\frac{\eta \vdash e_1 \; e_2 \leftrightarrow v}{\eta \vdash e_1 \; e_2 \leftrightarrow v} & \quad \text{App}_e
\end{align*}
\]
Outline

• Introduction ✓
• Programming paradigm for R ✓
• Lexical scoping ✓
• **Further analysis**
• A functional language for R users
• Conclusion
Dynamic type binding
- An R object can change its type during the computation.
  \[ x \leftarrow c(1.0, 2.0, 3.0) \]
  \[ x \leftarrow 47 \]
  - `typeof` returns the type of an R object.
    - `symbol, pairlist, closure, environment`

Is it good? ⇒ philosophical debate
- dynamic type binding is good for:
  - quick, small programming tasks
- static type binding is good for:
  - large programming tasks
Complex Semantics

- **Ex. Section 3.4 Indexing in R Language Definition**

\[ x[i] \]
\[ x[i, j] \]
\[ x[[i]] \]
\[ x[[i, j]] \]
\[ x$a \]
\[ x"a" \]

- Why on earth such "a" complex semantics for statistical computing?
So Many Complex/Special Cases

- From *R Language Definition*
  - "Another more *subtle* difference is ..."
  - "... evaluated in some *unexpected* cases."
  - "... can lead to *surprises*."
  - "*In a very few* cases, ..."
  - "... in certain (*rather* rare) circumstances, ..."
  - "... are treated *specially*."
  - "... should be done *with caution*."
  - "*A couple of* *special* rules apply, though:"
  - "... *is not guaranteed* to hold in all implementations."
  - "*is not generally* handled correctly."
  - "*The special exception* for ... *is admittedly peculiar*."

26
Evolution or Degeneration?

"R appears to be working fine."
"??? seems often useful, so let's add it to R."
"Now ??? is available, but there is something fishy going on."

• Example of ??? = first-class functions

  "This ability is rarely used even though it is potentially very powerful." – Lexical Scope and Statistical Computing
  – incorporating first-class functions
    without expressions and bindings
    ⇒ fitting a square peg into a round hole

• The worst example of ??? is yet to come, however.
Lazy Evaluation

"A policy of lazy arguments is very useful because ... This can be very useful for specifying functions or models in symbolic form."
– R: A Language for Data Analysis and Graphics

• Evaluation strategy of R
  – eager evaluation for built-in functions: fully evaluate arguments
  – lazy evaluation for promise objects: evaluate only when necessary

• Yes, lazy evaluation is a great idea.
  – Ex. Haskell
  – But only if all functions are pure mathematical functions.

• Lazy evaluation + computational effects ⇒ total complete mess
  – computational effects (= side effects)
    • plot, print, vector update, assignments
  – Functions in R are not mathematical functions anyway.
  – Solution from programming language theory = monad

• Besides lazy evaluation in R is not really lazy evaluation!
Meta-programming in R

- **quote** creates unevaluated expressions.
- **eval** treats programs as data.

```r
> e <- quote (2 + 2)
> v <- eval (e)
```

- Useful constructs? Yes!
  - implementing compilers, staged computation, and so on
- But do you really need **quote**, **eval**, **deparse**, **substitute** for statistical computing?

"More frequently, one wants to ... in order to deparse it and use it for **labeling plots**, for instance." – R Language definition

⇒ launching a nuclear missile to kill a fly
Why Not Use First-Class Functions?

• A weird program exploiting lazy evaluation and `eval`
curve ← function (expr, from, to) {
  x ← seq (from, to, length=500)
  y ← eval (substitute (expr))
  plot(x, y, type="l")
}
curve (x^2 - 1, -2, 2)  This function call does not make sense. ⇒ misunderstanding of lazy evaluation!

• A quick fix = use a first-class function
curve ← function (f, from, to) …
curve (function (x) x^2 - 1, -2, 2)
Other Minor (Yet Serious) Points

• Maintaining state within functions
  "The ability to preserve state information between function invocations is a \textbf{very useful} feature ..."
  – \textit{R: A Language for Data Analysis and Graphics}
  ⇒ a trivial exercise in functional programming

• Confusion between definition and implementation
  "To understand completely the rules ..., the reader needs to be familiar with the notion of an evaluation frame."
  – \textit{An Introduction to R}
  – Specific implementation strategies are taken as part of the definition.
    • environment, closure, call stack, evaluation frame, ...
Outline

• Introduction ✓
• Programming paradigm for R ✓
• Lexical scoping ✓
• Further analysis ✓
• A functional language for R users
• Conclusion
Next Generation R?

• Claim
  1. Admit it or not, R is an ill-designed language.
  2. Nevertheless, R is too juicy to give up:
     • statistical computing
     • high level graphics
  3. R shares a lot in common with functional languages.

• Plan
  – extend an existing functional language with an interface to the R base library.
Objective CAML with R

• Objective CAML
  – industrial strength functional language
  – rough speed comparison
    • nearly as fast as, or sometimes faster than, C
    • consistently faster than C++
    • about 10 times faster than Matlab
  – strong type system (based on type theory)
    • significantly less development time than in C
    • more reliable code than in C
  – huge library contributed by users
  – free!

• Let's develop an Objective CAML interface to R!
Preliminary Results

```
open r;;  ;(open r modules defined in r.init)
init ();  ;(initialize)
let z1 = array.create 100 0.0;;
  ;(assign the array values)
  for i = 0 to 19 - 1 do
    for j = 0 to 19 - 1 do
      let idx = i + 10 * j in
      z1.(idx) <- (float_of_int idx) *. 0.1 *. (float_of_int [j]) *. 1.57
      done
    done;
    done;

  ;(get their sin values)
  let 2 = r.sin z1;;

  ;(draw 3D graph)
  r_persp 10 18 2;;
  pause ();;;
```
Outline

• Introduction ✓
• Programming paradigm for R ✓
• Lexical scoping ✓
• Further analysis ✓
• A functional language for R users ✓
• Conclusion
Summary

• R is great!
  – library for statistical computing
  – library for publication quality graphics
  – the whole statistics community actively contributing new libraries

• R is an ill-designed language, however.

• So, it's time to act.
  – just use programming language theory!
Thanks a lot!