Some Thoughts on Directions for R

Luke Tierney
Department of Statistics & Actuarial Science
University of Iowa

Introduction

- The S language has had enormous impact on statistics
- R has become a major implementation and extension
  - a good fit for many users' needs
  - excellent platform for distributing new ideas
- R is a great framework for language research
  - open source
  - large user base
  - good user feedback mechanisms

Some Issues to Consider

- object-oriented programming
- performance improvement
- persistent storage issues
- language enhancements

Object Oriented Programming

- A Really Good Idea
- started in our neighborhood
- a niche for a while
- now very much mainstream

Object Oriented Programming (cont.)

- means many things to many people
- some variations:
  - single or multiple inheritance
  - single or multiple dispatch
- single is simplest (to use and implement)
- multiple inheritance can be ignored
- multiple dispatch usually cannot

Generic Functions and Multiple Dispatch

- multiple dispatch usually implies generic functions
- there may be alternatives:

standard disp.	generic fcn.
`x.print`	`print(x)`
`(x,d).plot`	`plot(x,d)`

- generic functions
  - are useful with single dispatch (S3)
  - seem natural in a function-oriented language
  - seem odd to Java/C++ programmers

Generic Functions and Multiple Dispatch (cont.)

- multiple dispatch is more powerful
  - no need for visitor pattern, double dispatch
- generic functions separate behavior from structure
  - can be useful (eliminates need for *traits*)
  - can make it hard to manage code structure

A Simple Generic Functions Example

- A simple binary tree interface:

```
right(x), left(x),...
```

A Simple Generic Functions Example

- A simple binary tree interface:

```
right(x), left(x),...
```

- A simple quiz interface:

```
right(x), wrong(x),...
```

A Simple Generic Functions Example

- A simple binary tree interface:

```
right(x), left(x),...
```

- A simple quiz interface:

```
right(x), wrong(x),...
```

- Two `right`'s make a ...?
- This is a non-issue in single dispatch languages.

Costs/Benefits of Generic Functions

- benefits of multiple dispatch/generic functions are clear
- greatest cost: loss of organizational structure
- possible solution: interface packages
  - defines, documents basic generic functions (`right`,`left`)
  - maybe some generic operations (`depthFirst`)
  - implementations import interface
  - clients import interface
- clearly separates `bintree::right`, `quiz::right`

Costs/Benefits of Generic Functions (cont.)

- some complaints that R is not Java or C++ are misguided
- but there is some merit to some of these concerns
- multiple dispatch is very valuable when it is needed
- it is not needed most of the time

Costs/Benefits of Generic Functions (cont.)

- Complaints about R's approach to OOP will not go away
- We will need to either:
  - develop suitable language constructs and conventions
  - educate users in the effective use of generic functions

  to reduce the conceptual costs of generic functions
- or: we may want to consider alternate OO strategies
  - one based on `(x,d).plot` style
  - a framework to support multiple strategies

Performance Improvement

- Many computations are essentially instantaneous
- Some are not
- Strategies for speeding up those that are not:
  - profiling and recoding
  - explicit parallelization (e.g. `snow`)
  - automated recoding (compilation)
  - implicit parallelization

Compilation and Implicit Parallelization

- A simple example:

```
exp(-0.5*((x-mu)/sigma)^2)
```

- some cost factors:
  - interpreter overhead
  - allocation of intermediate values

Compilation and Implicit Parallelization (cont.)

- A simple stack-based assembly language:

```
push3 sigma, x, mu
sub
div
square
scale -0.5
exp
```

- at most three allocations if `push3` is smart

Compilation and Implicit Parallelization (cont.)

- multiprocessor machines likely to become widely available
- push3 can distribute data among several threads
- once data is distributed, the rest can be run in parallel

```
                push3 sigma, x, mu
sub             | sub             | sub
div             | div             | div
square          | square          | square
scale -0.5      | scale -0.5      | scale -0.5
exp             | exp             | exp
```

Persistent Storage Issues

- R currently has multiple persistent storage approaches:
  - lazy loading data bases
  - RObjectTables
  - Parsed description files
  - SQL data base interfaces
- It would be useful to explore unifying these approaches
  - could support alternate back ends for R's own needs
  - would reduce maintenance overhead
  - might improve performance for all uses

Language Enhancements

Some possible language enhancements to explore:

- more general `for` loop; iterators
- python/ruby-style generators
- threads
- declarations
- mutable objects, with freezing
- predicate dispatch
- callable objects

Conclusions

- The S language is very much alive and evolving
- R is an exciting framework for learning and developing new ideas
- This talk has outlined a small set of possible areas for further exploration
- These and other explorations should keep us busy for many years to come.