# testR – R language test driven specification

Petr Maj*, Tomas Kalibera, Jan Vitek
Purdue University, West Lafayette, IN, contact authors: peta.maj82@gmail.com

## Motivation

Every computer language that seeks widespread adoption and large user base requires a formal language specification. Language references, such as the ECMAScript reference for Javascript[1] are usually used to provide the formal language specification as these documents are relatively easy for programmers and virtual machine (VM) implementers to understand. But while such a reference can be easily created for a standardized language like JavaScript, creating such document for highly evolving and community driven language like R is extremely time consuming and bound to be not correlated to the latest language development.

## Introducing testR

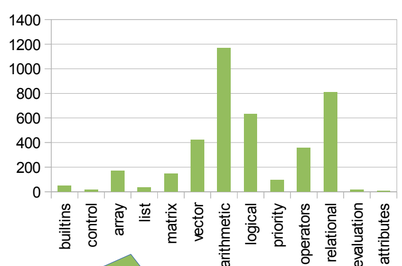
TestR provides a different approach to the specification problem. It consists of a growing number of relatively simple tests that aim to completely cover the behavior of the R core language with all its features and corner cases exposed.

Tests are structured according to the language features and are written in such way they read almost like a manual, complete with code examples. Maintaining the suite in sync with latest language development is as easy as running the suite and attending the failures.

TestR has also the potential to help VM developers. The suite can be used to measure GNU-R compatibility, or validate new commits in the regression mode. It is simple and extensible enough to either be integrated to, or make a test suite for nearly any R VM greatly simplifying the development efforts.

## Current State

The test suite currently consists of about 4K tests covering vector, array, matrix and list data types, arithmetic, relational and logic operators, operator priorities, few builtins and some other language features, such as attributes. We currently support GNU-R, FastR and Renjin VMs for tests and benchmarking.




## Implementation

A single TestR test consists of a header defining the test name and its properties and the test code, which is an R code. A simple preprocessor allows automating repetitive tasks by simple generics.

TestR commands begin with **#!**. They can only appear in test headers.

Each test is given a name that describes its function, This allows the tests to easily form human readable description

test header

```
#! testName
#!g OP = ( + # - # * # / )
#!g RESULT(OP) = ( 8 # 4 # 12 # 3)
#!t @RESULT
a <- 6
b <- 2
a @OP b
```

test body

Example header commands:

| | |
|---|---|
| **g** | generic argument |
| **t** | expected result |
| **w** | expected warning |
| **e** | expected error |
| **o** | expected output |
| **t** | test on/off for different targets |

New commands can be added by creating simple python functions.

Generic usage, this will be replaced by the current value of the generic. Generics can also be used in the header commands as shown in the **#!t** check above.

When a generic is specified, a new test is created for each of its values. If multiple generics are present, all possible combinations of their values are created.

This is not true for dependent generics. The dependent generic changes each time its original generic does. In the example, RESULT is dependent generic and changes with the OP generic always providing the correct value for the result of the operation.

While the generics can be used to create powerful super tests, to increase readability, these are not used and the bulk of the suite consists of simple tests aimed at a single feature. As an example the test below documents the inconsistent propagation of NAs in arithmetic operations:

```
#! NA propagation in arithmetic operators
#!g OP = (+ # - # * # /)
#!t NA NA
NA @OP NaN
NaN @OP NA
```

| Win32 | Win64 | Linux32 | Linux64 |
|---|---|---|---|
| NA NA | NA NaN | NA NA | NA NaN |

Another interesting failure found is the wrong default value for the exact argument for [[]] matching. Two tests and their outputs are shown below displaying the difference.

```
#! partial matching exact default value is NA
#!o TRUE
#!w partial match of 'b' to 'bar'
a <- list(foo=1, bar=2, foobar=3)
a[["b"]] == 2
```

```
logical(0)
```

```
a = list(foo=1, bar=2, foobar=3)
a[["b", exact=NA]] == 2
```

```
TRUE
Warning message:
  In a[['b', exact=NA]] : partial match of 'b' to 'bar'
```

## Framework

TestR itself is written in Python and great emphasis has been placed on its modularity. New targets (R VMs) can be added with a few lines as well as new modules to gather and analyze the results.TestR can be used in almost any test oriented task from regressions to profiling and benchmarking R on multiple VMs.

```
#! timer module example
#!g SIZE = (10 # 100 # 1000 # 10000 # 100000)
f <- function(a,b) {
  a + b * a / (b-a)
}
c <- 0
for (i in 1:@SIZE) {
  c <- f(c,c+i)
}
```

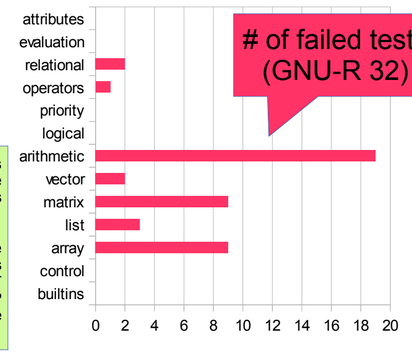
output of module **timer**

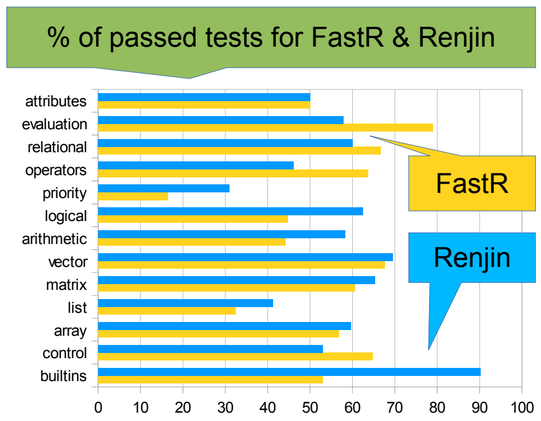
```
test name                                | gnur_linux  | fastr       | fastr_graal | renjin
-----------------------------------------------------------------------------------------
timer module example [SIZE=10]           | 0.183993    | 1.242291    | 0.682417    | 3.248864
timer module example [SIZE=100]          | 0.186549    | 1.253723    | 0.701230    | 3.320019
timer module example [SIZE=1000]         | 0.189415    | 1.258490    | 0.711636    | 3.325598
timer module example [SIZE=10000]        | 0.209359    | 1.260419    | 0.725398    | 3.617592
timer module example [SIZE=100000]       | 0.452572    | 1.272221    | 0.733940    | 3.809238
```

## Findings

Since the tests are created to conform to the current R manual[2], we present two different graphs: The first one shows the absolute number of tests failed (examples of these tests are displayed on this poster) by GNU-R (Linux64bit, version 2.15.2 (2012-10-26)) per test groups.

The second graph is percents of tests passed by our implementation, FastR (Linux) and Renjin (0.7.0) showing the degree of their implementation.




## Conclusions

TestR is an approach to formal language specification using tests to map the features. It also provides compatibility and regression suite for the VM developers further helping with the adoption of the language.

Future work will include increasing the language coverage and possibly tests of non-core modules.

TestR and FastR can be obtained from:

**http://github.com/allr/testr.git**

**http://github.com/allr/fastr.git**

## References

**1)** ECMAScript reference can be obtained from **http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf**

**2)** R manual and reference can be obtained from: **http://cran.r-project.org/doc/manuals/r-release/R-lang.html**