gKnit - Ruby and R Knitting with Galaaz in GraalVM

Rodrigo Botafogo

19 October 2018

1 Introduction

The idea of "literate programming" was first introduced by Donald Knuth in the 1980's. The main intention of this approach was to develop software interspersing macro snippets, traditional source code, and a natural language such as English that could be compiled into executable code and at the same time easily read by a human developer. According to Knuth "The practitioner of literate programming can be regarded as an essayist, whose main concern is with exposition and excellence of style."

The idea of literate programming envolved into the idea of reproducible research, in which all the data, software code, documentation, graphics etc. needed to reproduce the research and its reports could be included in a single document or set of documents that when distributed to peers could be rerun generating the same output and reports.

The R community has put a great deal of effort in reproducible research. In 2002, Sweave was introduced and it allowed mixing R code with Latex generating hight quality PDF documents. Those documents could include the code, the result of executing the code, graphics and text. This contained the whole narrative to reproduce the research. But Sweave had many problems and in 2012, Knitr, developed by Yihui Xie from RStudio was released, solving many of the long lasting problems from Sweave and including in one single package many extensions and add-on packages that were necessary for Sweave.

With Knitr, R markdown was also developed, an extension the the Markdown format. With R markdown and Knitr it is possible to generate reports in a multitude of formats such as HTML, markdown, Latex, PDF, dvi, etc. R markdown also allows the use of multiple programming languages in the same document. In R markdown text is interspersed with code chunks that can be executed and both the code as the result of executing the code can become part of the final report. Although R markdown allows multiple programming languages in the same document, only R and Python (with the reticulate package) can persist variables between chunks. For other languages, such as Ruby, every chunk will start a new process and thus all data is lost between chunks, unless it is somehow stored in a data file that is read by the next chunk.

Being able to persist data between chunks is critical for literate programming otherwise the flow of the narrative is lost by all the effort of having to save data and then reload it. Probably, because of this impossibility, it is very rare to see any R markdown document document in the Ruby community.

In the Python community, the same effort to have code and text in an integrated environment started also on the first decade of 2000. In 2006 iPython 0.7.2 was released. In 2014, Fernando Pérez, spun off project Jupyter from iPython creating a web-based interactive computation environment. Jupyter can now be used with many languages, including Ruby with the iruby gem (https://github.com/SciRuby/iruby). I am not sure if multiple languages can be used in a Jupyter notebook.

2 gKnitting a Document

This document describes gKnit. gKnit uses Knitr and R markdown to knit a document in Ruby or R and output it in any of the available formats for R markdown. The only difference between gKnit and normal Knitr documents is that gKnit runs atop of GraalVM, and Galaaz (an integration library between Ruby and R). Another blog post on Galaaz and its integration with ggplot2 can be found at: https://towardsdatascience.com/ruby-plotting-with-galaaz-an-example-of-tightly-coupling-ruby-and-r-in-graalvm-520b69e21021. With Galaaz, gKnit can knit documents in Ruby and R and both Ruby and R execute on the same process and memory, variables, classes, etc. will be preserved between chunks of code.

This is not a blog post on rmarkdown, and the interested user is directed to

- https://rmarkdown.rstudio.com/ or
- https://bookdown.org/yihui/rmarkdown/ for detailed information on its capabilities and use.

Here, we will describe quickly the main aspects of R markdown, so the user can start gKnitting Ruby and R documents quickly.

2.1 The Yaml header

An R markdown document should start with a Yaml header and be stored in a file with '.Rmd' extension. This document has the following header for gKitting an HTML document.

```
---
title: "gKnit - Ruby and R Knitting with Galaaz in GraalVM"
author: "Rodrigo Botafogo"
tags: [Galaaz, Ruby, R, TruffleRuby, FastR, GraalVM, knitr, gknit]
date: "29 October 2018"
output:
  html_document:
    keep_md: true
---
```

For more information on the options in the Yaml header, check https://bookdown.org/yihui/rmarkdown/html-document.html.

2.2 R Markdown formatting

Document formating can be done with simple markups such as:

2.2.1 Headers

```
# Header 1

## Header 2

### Header 3
```

2.2.2 Lists

```
Unordered lists:

* Item 1
* Item 2
    + Item 2a
    + Item 2b
```

```
Ordered Lists

1. Item 1
2. Item 2
3. Item 3
    + Item 3a
    + Item 3b
```

Please, go to https://rmarkdown.rstudio.com/authoring_basics.html, for more R markdown formating.

2.3 Code Chunks

Running and executing Ruby and R code is actually what really interests us is this blog. Inserting a code chunk is done by adding code in a block delimited by three back ticks followed by a block with the engine name (r, ruby, rb, include, others), an optional chunk_label and optional options, as shown bellow:

~~~
```{engine_name [chunk_label], [chunk_options]}
```
~~~

for instance, let's add an R chunk to the document labeled 'first_r_chunk'. In this case, the code should not be shown in the document, so the option 'echo=FALSE' was added.

~~~
```{r first_r_chunk, echo = FALSE}
```
~~~

A description of the available chunk options can be found in the documentation cited above.

For including a Ruby chunk, just change the name of the engine to ruby as follows:

~~~
```{ruby first_ruby_chunk}
```
~~~

In this example, the ruby chunk is called 'first_ruby_chunk'. One important aspect of chunk labels is that they cannot be duplicate. If a chunk label is duplicate, the knitting will stop with an error.

2.3.1 R chunks

Let's now add an R chunk to this document. In this example, a vector 'r_vec' is created and a new function 'redundat_sum' is defined. The chunk specification is

~~~
```{r data_creation}
r_vec <- c(1, 2, 3, 4, 5)

redef_sum <- function(...) {
  Reduce(sum, as.list(...))
}
```
~~~

and this is how it will look like once executed. From now on, we will not show the chunk definition any longer.

```r
r_vec <- c(1, 2, 3, 4, 5)

redef_sum <- function(...) {
  Reduce(sum, as.list(...))
}
```

We can, possibly in another chunk, access the vector and call the function as follows:

```r
print(r_vec)
```

```
## [1] 1 2 3 4 5
```

```
print(redef_sum(r_vec))
```

```
## [1] 15
```

2.3.2 Ruby chunks

In the same way that an R chunk was created, let's now create a Ruby chunk. One important aspect of Ruby is that in Ruby every evaluation of a chunk occurs on its own local scope, so, creating a variable in a chunk will be out of scope in the next chunk. To make sure that variables are available between chunks, they should be made global.

In this chunk, variable '$a', '$b' and '$c' are standard Ruby variables and '$vec' and '$vec2' are two vectors created by a call to FastR. It should be clear that there is no requirement in gknit to call or use R functions. gKnit will knit standard Ruby code, or even general text without code.

```
$a = [1, 2, 3]
$b = "US$ 250.000"
$c = "Inline text in a Heading"

$vec = R.c(1, 2, 3)
$vec2 = R.c(10, 20, 30)
```

In this next block, variables '$a', '$vec' and '$vec2' are used and printed.

```
puts $a
puts $vec * $vec2
```

```
## 1
## 2
## 3
## [1] 10 40 90
```

2.3.3 Accessing R from Ruby

One of the nice aspects of Galaaz on GraalVM, is that variables and functions defined in R, can be easily accessed from Ruby. This next chunk, reads data from R and uses the 'redef_fun' function defined previously. To access an R variable from Ruby the '~' function shoud be applied to the Ruby symbol representing the R variable. Since the R variable is called 'r_vec', in Ruby, the symbol to acess it is ':r_vec' and thus '~:r_vec' retrieves the value of the variable.

```
puts ~:r_vec
```

```
## [1] 1 2 3 4 5
```

In order to call an R function, the 'R.' module is used as follows

```
puts R.redef_sum($vec)
```

```
## [1] 6
```

2.3.4 Inline Ruby code

Knitr allows inserting R inline by adding ‘r code‘ . Unfortunately, this is not possible with Ruby code as there is no provision in knitr for adding this kind of inline engine. However, gKnit allows adding inline Ruby code with the ‘rb’ engine. The following text will create and inline Ruby text:

~~~
This is some text with inline Ruby accessing variable \$b which has value:
```{rb puts $b}
~~~
and is followed by some other text!
```

The result of executing the above ckunk is the following sentence with inline Ruby code

This is some text with inline Ruby accessing variable $b which has value: US$ 250.000 and is followed by some other text!

In an inline block, it is possible to execute multiple Ruby statements by adding a semicolom between them:

```
Multiple statements in the 'rb' engine use semicolom:
```{rb puts $a, puts $b}
```
```

Multiple statements in the ‘rb’ engine use semicolom: 1 2 3 US$ 250.000

2.3.5 Inline text in a Heading

Sometimes one wants to add an inline text in a heading. To do that in Ruby the whole heading needs to be returned by the inline Ruby engine. For example the heading above, was created by the following chunk:

```
```{rb puts "### #{$c}"}
```
```

Remember that variable ‘$¸ was defined in a previous Ruby chunk and is now being used to create the section heading for this section.

2.3.6 Including Ruby files

R is a language that was created to be easy and fast for statisticians to use. It was not a language to be used for developing large systems. Of course, there are large systems and libraries in R, but the focus of the language is for developing statistical models and distribute that to peers.

Ruby on the other hand, is a language for large software development. Systems written in Ruby will have dozens or hundreds of files. In order to document a large system with literate programming we cannot expect the developer to add all the files in a single ‘.Rmd’ file. gKnit provides the ‘include’ chunk engine to include a Ruby file as if it had being typed in the ‘.Rmd’ file.

To include a file the following chunk should be created, whre is the name of the file to be include and where the extension, if it is ‘.rb’, does not need to be added. If the ‘relative’ option is not included, then it is treated as TRUE. When ‘relative’ is true, ‘require_relative’ semantics is used to load the file, when false, Ruby’s $LOAD_PATH is searched to find the file and it is ’require’d.

```
```{include <filename>, relative = <TRUE/FALSE>}
```
```
```

Here we include file ‘model.rb’ which is in the same directory of this blog. This code uses R ‘caret’ package to split a dataset in a train and test sets.

~~~
```{include model}
```
~~~

```
require 'galaaz'

# Loads the R 'caret' package.  If not present, installs it
R.install_and_loads 'caret'

class Model

  attr_reader :data
  attr_reader :test
  attr_reader :train

  #==========================================================
  #
  #==========================================================

  def initialize(data, percent_train:, seed: 123)

    R.set__seed(seed)
    @data = data
    @percent_train = percent_train
    @seed = seed

  end

  #==========================================================
  #
  #==========================================================

  def partition(field)

    train_index =
      R.createDataPartition(@data.send(field), p: @percet_train,
                            list: false, times: 1)
    @train = @data[train_index, :all]
    @test = @data[-train_index, :all]

  end

end
```

```ruby
mtcars = ~:mtcars
model = Model.new(mtcars, percent_train: 0.8)
model.partition(:mpg)
puts model.train.head
puts model.test.head
```

```
##                    mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
```

```
## Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C         17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
##               mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4     21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag 21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Valiant       18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360    14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 230      22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 450SE    16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
```

2.3.7 Documenting Gems

```
# Copyright (c) 2017 Oracle and/or its affiliates. All rights reserved. This
# code is released under a tri EPL/GPL/LGPL license. You can use it,
# redistribute it and/or modify it under the terms of the:
#
# Eclipse Public License version 1.0, or
# GNU General Public License version 2, or
# GNU Lesser General Public License version 2.1.

warn "#{File.basename(__FILE__)}: warning: callcc is obsolete; use Fiber instead"

class Continuation
  def initialize
    @fiber = Fiber.current
  end

  def call
    if Fiber.current != @fiber
      raise 'continuation called across fiber'
    end
    raise 'Continuations are unsupported on TruffleRuby'
  end
end

module Kernel
  def callcc
    yield Continuation.new
  end
  module_function :callcc
end
```

2.4 Converting to PDF

One of the beauties of knitr is that the same input can be converted to many different outputs. One very useful format, is, of course, PDF. In order to converted an R markdown file to PDF it is necessary to have LaTeX installed on the system. We will not explain here how to install LaTeX as there are plenty of documents on the web showing how to proceed.

gKnit comes with a simple LaTeX style file for gknitting this blog as a PDF document. Here is the Yaml header to generate this blog in PDF format instead of HTML:

```
---
title: "gKnit - Ruby and R Knitting with Galaaz in GraalVM"
author: "Rodrigo Botafogo"
tags: [Galaaz, Ruby, R, TruffleRuby, FastR, GraalVM, knitr, gknit]
date: "29 October 2018"
```

```
output:
  pdf_document:
    includes:
      in_header: ["../../sty/galaaz.sty"]
    number_sections: yes
---
```

3 Conclusion

4 Installing gKnit

4.1 Prerequisites

- GraalVM (>= rc7)
- TruffleRuby
- FastR

The following R packages will be automatically installed when necessary, but could be installed prior to using gKnit if desired:

- ggplot2
- gridExtra
- knitr

Installation of R packages requires a development environment and can be time consuming. In Linux, the gnu compiler and tools should be enough. I am not sure what is needed on the Mac.

4.2 Preparation

- gem install galaaz

4.3 Usage

- gknit [filename]