Advances in integrating statistical inference

Nicos Angelopoulos[1] Samer Abdallah[2] and Georgios Giamas[1]

[1] Department of Surgery and Cancer, Division of Cancer, Imperial College London, Hammersmith Hospital Campus, Du Cane Road, London W12 ONN, UK.
[2] Department of Computer Science, University College London Gower Street, London WC1E 6BT, UK.

Abstract. We present recent developments on the syntax of *Real*, a library for interfacing two Prolog systems to the statistical language *R*. We focus on the changes in Prolog syntax within SWI-Prolog that accommodate greater syntactic integration, enhanced user experience and improved features for web-services. We recount the full syntax and functionality of *Real* as well as presenting sister packages which include Prolog code interfacing a number of common and useful tasks that can be delegated to *R*. We argue that *Real* is a powerful extension to logic programming, providing access to a popular statistical system that has complementary strengths in areas such as machine learning, statistical inference and visualisation. Furthermore, *Real* has a central role to play in the uptake of computational biology and bioinformatics as application areas for research in logic programming.

1 Introduction

Real is a low level interface between Prolog and *R* (Angelopoulos et al., 2013). It enables the user to call *R* functions on Prolog data and communicate the results back to the logic system. The library works on two open source systems: YAP (Costa et al., 2012) and SWI-Prolog (Wielemaker et al., 2012) whose *C* language interface is compatible (Wielemaker and Costa, 2011). Since its first introduction *Real* has evolved and has exerted some influence in advances to Prolog syntax. Furthermore, it has been used in a number of projects and in the process acquired a number of sister libraries that depend on it to deliver Prolog interfaces to useful tasks that can be best be dealt by existing *R* code. *Real* has thus be shown to be a useful and well integrated Prolog library that can provide access to the wealth of open source code available in *R* which is often accompanied by published scientific papers.

Here we focus on describing the full syntax of *Real 1.4* and its role in recent developments with syntactic changes in *SWI-7*. The changes in both systems have made the integration of *R* code into Prolog more natural and unobtrusive. Changes in the library itself had to be made to accommodate transition to the new Prolog syntax while preserving compatibility with traditional implementations.

Real gives access to *R* libraries that can complement Prolog's weaknesses in areas such as statistical inference and visualisation. With the library installed it

is straight forward with a basic grasp of *R* to call its functions on Prolog data. However for users with no prior exposure to *R* there still might be a barrier. To address this, and in order to increase general usability of the library a number of sister packages have been developed. We highlight some of the predicates that enable access to *R* code without any knowledge of *R*.

Central application areas in the inception of *Real* and its recent advances, has been the areas of bioinformatics and computational biology. The sister libraries we describe here have evolved in addressing real world bioinformatics tasks in the context of a variety of projects: (Zhang et al., 2015; MacIntyre et al., 2015; Stebbing et al., 2015). The main thesis of this paper is that Prolog can play a central role as a unifying platform in research in bioinformatics, taking advantage of its strong grip in knowledge representation and reasoning and in combinations with recent advances with *Real* and web programming (Wielemaker et al., 2008; Lager and Wielemaker, 2014).

2 *Real*

Here we first present the innovations of *Real 1.4* before we summarise its overall syntax and usage with particular focus on new features.

2.1 Innovations

In terms of syntax, *Real* faced three major clashes between Prolog notation and syntax acceptable to *R*. Those were the use of '.' in *R* identifiers, the use of double quotes (' " ') to represent strings and the representation of terms with 0 arity '$foo()$'. In previous versions the library was able to bypass those by employing a number of indirect techniques concentrating on keeping as faithful as possible to the original syntax. Briefly,

- operator '..' was used to construct arity 2 terms that were behind the scenes converted to a Prolog atom interpreted as an *R* identifier (Prolog term $my..variable$ was translated to *R* variable $my.variable$, $my..variable \rightarrow my.variable$).
- operator + on non numerical values was used to convert atoms and code lists to strings ($+foo \rightarrow "foo"$)
- with the newly, at the time, introduced block operator '()' it was possible to parse $foo`()'$ as $foo()$

With *Real* in mind, *SWI-7* (Wielemaker, 2014) introduced syntax that legalised all of the above constructs, as well as the implementation of lists as primary data structures (as oppose to ./2 terms). Dots in atoms and the use of double quotes are now controlled by global flags, the former's default being off and the latter's being on. *Real* has been adapted to utilise the new changes in a backwards compatible manner. All of the following are now valid *Real* syntax mapping to the corresponding *R* constructs, proviso of the appropriate global flags been enabled,

- func.foo(a,b,c)
- write.csv("to_file.csv", x)
- foo()

Under the bonnet, list representations were additionally generalised to accommodate the new data type. Also in the *C* interface, *Real 1.4* includes improvements in that it can be employed within a web-service, thus allowing the *R*-server thread to be an arbitrary one. This is of particular interest, as in itself *R* is single threaded.

A final innovation at the syntactic level has been the introduction of 'NA' values in the interface. In *R*, NA values stand for not available or unknown value placeholders. Prolog does not internally support such values, but the interface enables mapping of such values within arithmetic vectors and matrices to '$NaN'. When passing numeric data from Prolog to *R* in addition to $NaN, the empty atom ('') is also translated to *R* 's NA value.

Taken together these innovations allow a tighter and smoother integration of *R* code and enable Prolog programmers to tap in the wealth of statistical functions implemented in *R*.

2.2 Communication with *R*

The bulk of the communication with *R* is via a single predicate $\leftarrow /2$ which is also defined as an infix operator. This is an alternative assignment operator in *R*. Within *Real* it can be used to transfer data between *R* and Prolog, to apply, in an in-line fashion, *R* functions to Prolog data as well as destructively assigning values to *R* variables. Disambiguation clearly distinguishes the different modes, which can be summarised by:

$$+Rexpr \leftarrow +Rexpr$$
$$-PlVar \leftarrow +Rexpr$$
$$+Rexpr \leftarrow +PlData$$

When the LHS of the operator is a uninstantiated variable, the second mode is assumed, where the value of *Rexpr* is passed to $PlVar$ after it has been evaluated in *R*. When the RHS is a c/n term or a list then the third mode is applied and the data in the RHS is transferred to the LHS *Rexpr* (usually an *R* variable).

The following examples show how to: transfer Prolog data to *R* and back (1), transfer Prolog data to *R* and get the result of applying a function to the data in the new *R* variable (2) and demonstrating how to apply an *R* function on Prolog data without the use of an explicit *R* variable (3).

$$?- a \leftarrow [1,2,3], A \leftarrow a. \tag{1}$$

$$A = [1,2,3].$$

Indicator	Operator Symbol		Description
r/1	`<-`	$\leftarrow$	evaluate R expression (no return value)
r/2	`<-`	$\leftarrow$	main communication to R library
r_new/1	`<<-`	$\twoheadleftarrow$	argument is a fresh variable
`<<-`/2	`<<-`	$\twoheadleftarrow$	r/2 but with error if R variable exists
r_call/2	`<-C++O`	$\leftarrow$ ++	r/1,2 with options (O)
r_library/1			load R library in a hookable manner
r_start/0			start the connection to R
r_stop/0			stop the connection to R
r_remove/1			remove R variable
r_thread_loop/0			start an R thread server
r_serve/0			serve all R expressions on queue thread

Table 1. Library's main predicates

$$?- a \leftarrow [1,2,3], Mean \leftarrow mean(a). \tag{2}$$

$$Mean = 2.0.$$

$$?- Mean \leftarrow mean([1,2,3]). \tag{3}$$

$$Mean = 2.0.$$

2.3 Real's predicates

Real 1.4 adopts the convention of a uniform prefix to all the library predicates. The full list of *Real*'s predicates along with the associated operators and brief descriptions are shown in Table 1. New additions include a hookable locator for R libraries, web server support, intuitive syntax for non-destructive assignment and a generic predicate for mixing Prolog and R options and directing output to graphic devices.

With new predicate *r_library/1* the user can load the standard R libraries in their local installation. In addition, the predicate can be directed to user specified locations where local, possibly, changed sources of such libraries can be loaded preferentially. The flexibility allows for (a) specific code to be loaded only known to *Real* thus living the remainder of the R installation intact, and (b) user code that can be made available and can work either with the distributed version while having extra functionality when used with the altered sources.

Real is inherently single threaded. To support the use of *Real* in multi-threaded applications, in particular in web servers built on SWI Prolog's HTTP libraries (Wielemaker et al., 2008), *Real 1.4* allows a single designated *Real* server thread to be started, which then takes over the task of executing or evaluating

R commands or expressions. Then, when the *←/1* and *←/2* predicates are used on any other thread, the requests are redirected to the *Real* server thread and the results awaited. Communication is handled synchronously using SWI Prolog queues.

This system was implemented to support an application in the area of large scale computational musicology, the *Digital Music Laboratory*, which is built on SWI-Prolog's semantic web server *Cliopatria*. Here, *Real* is used both for general numerical computations and the generation of high-quality scalable vector graphics. In comparison with previous versions of the system which used Matlab's engine API to communicate with a separate *Matlab* process, the lower overhead of communicating with *Real*'s in-process embedded *R* yields much better performance when numerous relatively small computations are required.

As *R* supports destructive assignment, it can be the case that the programmer might unwittingly overwrite variables already in the working space. To ease and provide visual cues of the fact that a variable is fresh in a specific context, we introduced operators *↞/2* and *↞/1* and predicate *r_new/1*. The first ensures that its first argument (an *R* variable) did not exist prior to assigning to it some new values. The second removes its arguments from the *R* work-space and the third fails if its argument is already a known *R* variable.

Integral to the *R* language design and practice is the use of options that control the details of function calls. These are = pairs of argument name to values, which more often than not do not have to be present at invocation. When not present, default values supplied by the function developers are used. Similarly but not as widely used is the use of list of terms that control calls to Prolog predicates. By convention an options list is placed at the last argument of a predicate and commonly contains a number of single arity terms. *Real* now provides a uniform way to marry the two conventions and a flexible way of handling options addressed to Prolog predicates accessing *R* functions. In addition, a number of standard tasks have been incorporated to a new interface predicate:

$$r\_call(Func, Opts).$$

which can also be accessed as

$$\leftarrow Func \;\; +\!\!+ \;\; Opts$$

$Func$ is a compound term which is translated to an *R* function call and $Opts$ can be a combination of: (a) *=/2* terms, which are added to $Func$, (b) options controlling *r_call/2*'s own execution and (c) Prolog style options which can influence the caller's behaviour but are ignored in the *R* call. Some of *r_call/2* options are:

rvar(Rvar) when given call becomes: $Rvar \leftarrow Fcall$
rmv(Rmv=false) removes *Rvar* after end of call
stem(Stem=real_plot) stem to use for output files
outputs(Outs=false) a list of output devices
debug(Dbg=false) sets debug(real) for the duration of call
fcall(FinCall) returns the term constructed after *=/2* additions
post_call(Post) call this after the function call

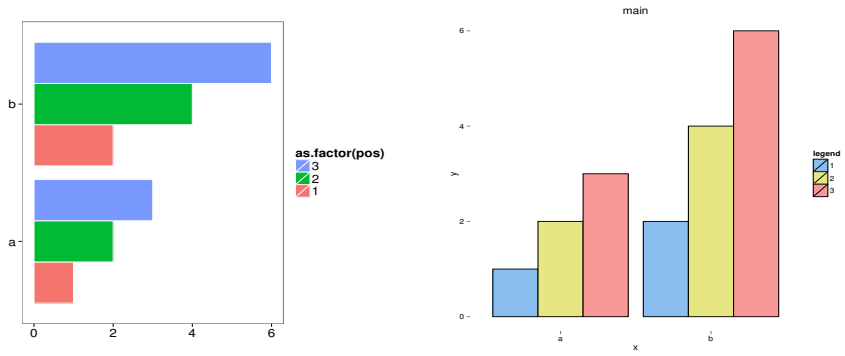

Fig. 1. ggplot2 based bar plots. Left: with default options. Right: a number of options have altered elements of the plot.

3 Associated packages

3.1 *b_real*

b_real is a library based on *Real* which contains a collection of predicates that aim to provide a Prolog based interface to a number of simple tasks. The target audience is Prolog users that have no previous experience with *R*. The predicates described here can use the basic functionality of the underlying *R* functions and can adjust some of the behaviour entirely in Prolog, while allowing arbitrary option passing to users with some familiarity with *R*.

Bar plots are basic plots that can present comparative information in a intuitive manner. Here we present a Prolog interface to *ggplot2* (Wickham, 2009). In its most general form, predicate *gg_bar_plot/2* displays a number of grouped measurements such as, for instance, the cpu-timings of a number of machine learning algorithms ran on a number of datasets. The following query, produces the plot in the LHS of Fig. 3.1.

$$?- Pairs = [a-[1,2,3], b-[2,4,6]],\ gg\_bar\_plot(Pairs, [\,]). \tag{4}$$

ggplot2 is a complex piece of software able to display many types of plots while *gg_bar_plot/2* only accessing the bar plotting part. Within this, a number of plot elements can be controlled with Prolog options passed in the second argument. The following query changes elements such as the colour of the drawing pen (black) the labels (x,y and main), legend title and fill colours, producing the plot in the RHS of Fig. 3.1.

$$\begin{aligned} ?- Pairs &= [a-[1,2,3], b-[2,4,6]], \\ Opts &= [\, geom\_bar\_draw\_colour(black), \\ & fill\_colours([\text{"}skyblue2\text{"}, \text{"}khaki2\text{"}, \text{"}\#FB9A99\text{"}]), \end{aligned} \tag{5}$$

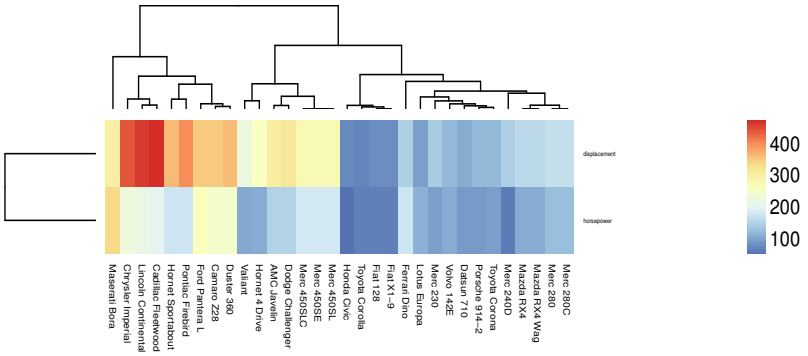

Fig. 2. Heatmap generation with aheatmap() from package NMF.

$$flip(false), labels(x, y, main),$$
$$legend\_title(legend)],$$
$$gg\_bar\_plot(Pairs, Opts).$$

Heatmap functions are ubiquitous in *R*. *b_real* provides a Prolog interface to the *aheatmap* library. In addition to some simple option mapping *aheatmap/2* provides polymorphic support for the first argument which could be a matrix *R* variable or a Prolog representation of one. The following code uses the *mtcars* example dataset, from which it plots a heatmap of two variables: *hp* (horsepower) and *disp* (displacement).

$$\begin{aligned} ?- MtC \leftarrow as.list(mtcars), memberchk(hp = HP, MtC), \\ memberchk(disp = Disp, MtC), x \leftarrow [HP, Disp], \\ rownames(x) \leftarrow c(\text{"}horsepower\text{"}, \text{"}displacement\text{"}), \\ <- aheatmap(x). \end{aligned} \tag{6}$$

3.2 wgraph

R has a number of plotting functions for drawing graphs formed of nodes and edges. Two of these are *igraph()* and *qgraph()*. The latter being based on the former with some extra options and facilities for grouping nodes. The Prolog pack *wgraph* provides a uniform Prolog interface to these *R* libraries. A plot

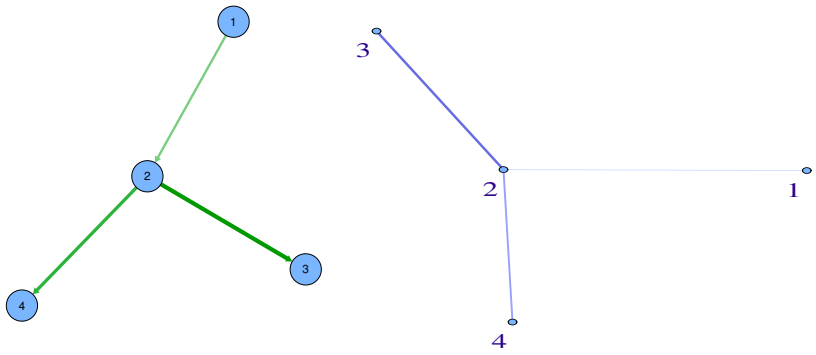

Fig. 3. Graphs generated by *wgraph_plot/2*. Left: plot uses default rendering with *qgraph()* call. Right: render changed to *igraph()* and a number of options specialised the output.

with the default renderings can be easily drawn from a list representing the graph connections and the weights on the edges:

$$\begin{aligned} ?- G = [1-2:200, 2-3:400, 2-4:300], \\ wgraph\_plot(G, [\,]). \end{aligned} \tag{7}$$

A set of Prolog options that control the choice of the drawing function and basic parameters of the graph, and which work irrespective of the drawing function can be provided in the second argument of *wgraph_plot/2*. In the following example *igraph()* is passed the size of nodes to use, the degree at which the node labels should be displayed and the distance of the label from the node edge. The resulting graph is shown in the RHS of Fig. 3.

$$\begin{aligned} ?- G = [1-2:200, 2-3:400, 2-4:300], \\ Opts = [\,plotter(igraph), label\_distance(-1), \\ label\_degree(2), node\_size(4)\,], \\ wgraph\_plot(G, Opts). \end{aligned} \tag{8}$$

3.3 Availability

The three libraries discussed here, (*Real*, *b_real* and *wgraph*) are available as SWI-Prolog packages[3] which can be installed easily from within SWI-Prolog. To download and install *Real* the user needs to query with:

$$?- install\_pack(real). \tag{9}$$

[3] `http://swi-prolog.org/pack/list`

4 Conclusions

We presented a number of recent advances in *Real* and in particular shown how developments in Prolog syntax have made *Real* syntax blend naturally into Prolog code. The resulting syntax provides a powerful platform for accessing the extensive collection of freely available *R* code. As a consequence *Real* can have a strong positive influence into the penetration of Prolog to new application areas such as bioinformatics and machine learning. With version 1.4 *Real* has reached a new level of maturity including facilities for using *R* in web-servers. In addition we highlighted some predicates from two sister packages. As with *Real* itself, these are freely available and can be easily installed via the SWI-Prolog package manager. In the future we plan to work towards suggesting internal ways for Prolog to work better, or more confluent to *R*, with *NA* values and infinity.

Real has been used in a number of projects in the area of bioinformatics and has a steady stream of downloads via SWI-Prolog's package manager. With the enhanced level of integration, *Real* is becoming a powerful hybrid programming language.

Bibliography

Nicos Angelopoulos, Vitor Santos Costa, Joao Azevedo, Jan Wielemaker, Rui Camacho, and Lodewyk Wessels. Integrative functional statistics in logic programming. In *Proc. of Practical Aspects of Declarative Languages*, volume 7752 of *LNCS*, pages 190–205, Rome, Italy, Jan. 2013. URL `http://stoics.org.uk/~nicos/sware/real/`.

Vítor Santos Costa, Ricardo Rocha, and Luís Damas. The YAP Prolog system. *Theory and Practice of Logic Programming*, 12:5–34, 1 2012. ISSN 1475-3081.

Torbjorn Lager and Jan Wielemaker. Pengines: Web logic programming made easy. In *International Conference of Logic Programming*, 2014.

David MacIntyre, Manju Chandiramani, Yun S Lee, Lindsay Kindinger, Ann Smith, Nicos Angelopoulos, Benjamin C. Lehne, Shankari Arulkumaran, Richard Brown, Tiong Ghee Teoh, Elaine Holmes, Jeremy K. Nicholson, Julian Marchesi, and Phillip R. Bennett. The vaginal microbiome during pregnancy and the postpartum period in a european population. *Scientific Reports*, 5:Article number: 8988, 2015. URL `http://www.nature.com/srep/2015/150311/srep08988/full/srep08988.html`.

Justin Stebbing, Hua Zhang, , Yichen Xu, Adam Sanit Nicos Angelopoulos, and Georgios Giamas. Global mapping of tyrosine kinase signalling. *Journal Title*, 2015. Accepted for publication.

Hadley Wickham. *ggplot2: elegant graphics for data analysis*. Springer New York, 2009. ISBN 978-0-387-98140-6. URL `http://had.co.nz/ggplot2/book`.

Jan Wielemaker. SWI-Prolog ODBC interface, 2014. URL `http://www.swi-prolog.org/pldoc/package/odbc.html`.

Jan Wielemaker and Vítor Santos Costa. On the portability of Prolog applications. In *Practical aspects of Declarative Languages*, pages 69–83, 2011.

Jan Wielemaker, Zhisheng Huang, and Lourens van der Meij. SWI-Prolog and the web. *TPLP*, 8(3):363–392, 2008.

Jan Wielemaker, Tom Schrijvers, Markus Triska, and Torbjörn Lager. SWI-Prolog. *Theory and Practice of Logic Programming*, 12(1-2):67–96, 2012. ISSN 1471-0684.

Hua Zhang, Nicos Angelopoulos, Yichen Xu, Arnhild Grothey, Joao Nunes, Justin Stebbing, and Georgios Giamas. Proteomic profile of KSR1-regulated signaling in response to genotoxic agents in breast cancer. *Breast Cancer Research and Treatment*, 2015. URL `http://link.springer.com/article/10.1007/s10549-015-3443-y`.