SWI-Prolog markdown

Emmanuel BATT

28/09/2020

Contents

Introduction

swiplr is an R wrapper for Prolog. I use it only with SWI-Prolog but you may succeed to use it with other versions of Prolog (not tested).

Install

swiplr is using your SWI-Prolog binary. You shall install it first on your operating system. Use the command below for a debian (or ubuntu) based linux. For other operating systems go to swi-prolog download page.

```
sudo apt-get install -qq -y swi-prolog
```

Then you can install the package from github using devtools. If you do not have devtools, install it first with `install.packages("devtools")`

```
devtools::install_github("https://github.com/battmanux/swiplr.git")
```

```
## Skipping install of 'swiplr' from a github remote, the SHA1 (ffbed9ad) has not changed since last ins
##   Use `force = TRUE` to force installation
```

Congratulations! You are ready to use *swiplr*.

```
library(swiplr)
```

In case you would like to use another Prolog binary, you can force to binary path with `options(swipl_binary = "prolog")`. *swipl_binary* can also be an absolute path to your own version of swipl. For instance */usr/local/bin/swipl*

Simple usage

Return booleans

```
bar.

?- bar.
```

Query: bar

TRUE

```
foo(bar).

?- foo(foo).
```

Query: foo(foo)

FALSE

Return data.frame

```
male(joe).
male(bob).
female(olivia).
female(mia).

parent(joe, bob).
parent(joe, olivia).
parent(mia, bob).
parent(mia, olivia).


children(X, Y)   :- parent(X, Y).
sibling(X1, X2, P)  :- parent(X1, P), parent(X2, P), dif(X1,X2).

?- sibling(SIBLING_1, SIBLING_2, PARENT)
```

Query: sibling(SIBLING_1, SIBLING_2, PARENT)

SIBLING_1	SIBLING_2	PARENT
joe	mia	bob
joe	mia	olivia
mia	joe	bob
mia	joe	olivia

Hiding columns

You can add _ at the end of a variable to remove it from result table.

```
male(joe).
```

```
male(bob).
female(olivia).
female(mia).

parent(joe, bob).
parent(joe, olivia).
parent(mia, bob).
parent(mia, olivia).


children(X, Y)   :- parent(X, Y).
sibling(X1, X2, P)  :- parent(X1, P), parent(X2, P), dif(X1,X2).

?- sibling(SIBLING_1, SIBLING_2, PARENT_), female(PARENT_)
```

Query: sibling(SIBLING_1, SIBLING_2, PARENT_), female(PARENT_)

SIBLING_1	SIBLING_2
joe	mia
mia	joe

By default we return anly the 10 first rows. Use `maxnsols` to increase the table side.

we set the maxnsols=20 variable to this chunk header

```
foo(X) :- member(X, [1,2,3,4,5,6,7,8,9,10,11,12]).

?- foo(Number)
```

Query: foo(Number)

Number
1
2
3
4
5
6
7
8
9
10
11
12

Exchange data with R

Lets define a few variables

```
some_name <- "john"

value_list <- paste0("bar_", 1:3)

named_list <- list(
  var1 = list(
    field1 = "sömé àçênts",
  var2 = "bar0")
)

some_table <- r_to_pro(iris[sample(1:150, 4),])
```

We can use them from prolog

One can use `whisker::whisker.render` syntax. By default whisker applies html escaping on the generated text. To prevent this use {{{variable}}} (triple) in stead of {{variable}}.

we give the name `simple_foo_chunk` to this chunk so that we can use its output in R

```
% Inject the content of R some_name variable
foo({{{ some_name }}}).


?- foo(BAR).
```

Query: foo(BAR)

BAR
john

We can also inject more complex data structures

```
foo('{{{ named_list$var1$field1 }}}').

% Iterates through the list of values in R value_list vector
{{{#value_list}}}
  foo({{{.}}}).
{{{/value_list}}}

% Iterate through a table (list of rows)
% use r_to_pro() to convert data.frame to the right format
{{{#some_table}}}
  some_table({{{Species}}},{{{Sepal_Length}}},{{{Sepal_Width}}}).
{{{/some_table}}}

?- foo(FOO)
?- some_table(Species, Sepal_Length, Sepal_Width)
```

Query: foo(FOO)

FOO
sömé àçênts
bar_1
bar_2
bar_3

Query: some_table(Species, Sepal_Length, Sepal_Width)

Species	Sepal_Length	Sepal_Width
virginica	6.4	3.2
versicolor	6.9	3.1
virginica	7.2	3.6
virginica	7.1	3

We can use prolog results from R

And then get back the last chunk values in R with the `prolog_output` variable.

```
prolog_output$result_1
```

```
##           FOO
## 1 sömé àçênts
## 2       bar_1
## 3       bar_2
## 4       bar_3
```

If we give a label or a name to a Prolog chunk, its output will be saved into an R variable with the same same:

```
str(simple_foo_chunk)
```

```
## 'data.frame':    1 obs. of  1 variable:
##  $ BAR: chr "john"
```

Use prolog ouside of Rmd files

You may want to directly call prolog from an R script (maybe shiny). You can achieve this by calling `pl_eval`:

This is an R chunk

```
my_value <- "open_bar"

my_result <- pl_eval(
  data = .GlobalEnv,
  body = "foo(bar1). foo(bar2). foo({{ my_value }}).",
  query = "foo(Foo)")

print(my_result)
```

```
##        Foo
## 1     bar1
```

```
## 2     bar2
## 3 open_bar
```

Build a database over several chunks

This `database` describe some predicates: **we set the label=database variable to this chunk header**

```
foo(bar_from_database_chunk_1).
foo(bar_from_database_chunk_2).

?- foo(BAR)
```

Query: foo(BAR)

BAR
bar_from_database_chunk_1
bar_from_database_chunk_2

This chunk load the database chunk, add facts and query it. Note that with RStudio you can use chunk names completion that appears after typing the $ character.

Query at the end of the chunk is optional. You can define a chunk of data without it.

we set the label=extended variable to this chunk header

```
{{{ .swiplr_chunks$database }}}

foo(bar_from_extended_chunk_3).
```

Chunks with eval=FALSE will be added to the chunk list but results will not be displayed.

we set the label=extended_bis variable to this chunk header

```
{{{ .swiplr_chunks$extended }}}

foo(bar_from_extended_chunk_4).

?- foo(X).
```

Lets query the entire database. Note that you can rerun intermediate chunks without rerunning the entire document.

```
{{{ .swiplr_chunks$extended_bis }}}

foo(bar_from_unnamed_chunk_5).

?- foo(X).
```

Query: foo(X)

X
bar_from_database_chunk_1
bar_from_database_chunk_2
bar_from_extended_chunk_3
bar_from_extended_chunk_4
bar_from_unnamed_chunk_5

However you must evaluate a prolog named database chunk after change so that *swiplr* knows its content.

Advanced result formating

It is often needed to look at results as sparse table or connected graph. We have here two experimental ways of doing it.

Lets consider a simple database

```
cell(cat,legs,yes).
cell(cat,tail,yes).
cell(dog,legs,yes).
cell(dog,tail,yes).
cell(chicken,wings,yes).
cell(snake,legs,no).
cell(X,mamal,yes) :- member(X,[cat,dog,chicken]).
```

As a sparse table

Whenever you would like to see results with one entity per line and attributes in columns, you can build a query with the three following keywords: ENTITY, COLUMN_HEADER, CELL.

Each result produces one cell in the table: - ENTITY: will be the row label - COLUMN_HEADER: will be the column header - CELL: will be the content of the cell.

Make sure you have `tidyr` package installed.

```
table_query(
  body = .swiplr_chunks$sparse_table,
  query = "cell(ENTITY, COLUMN_HEADER, CELL)")
```

```
## # A tibble: 4 x 5
##   ENTITY  legs  tail  wings mamal
##   <chr>   <chr> <chr> <chr> <chr>
## 1 cat     "yes" "yes" ""    "yes"
## 2 dog     "yes" "yes" ""    "yes"
## 3 chicken ""    ""    "yes" "yes"
## 4 snake   "no"  ""    ""    ""
```

As a chart

You way also need to represent results as interconections in a graph. - FROM: name of the source node - TO: name of the target node - LINK: name of the link

Each result will be displayed as an oriented link in the graph

Make sure you have `visNetwork` package installed.

```
plot_query(
  body = .swiplr_chunks$sparse_table,
  query = "cell(FROM, TO, LINK)")
```

This produces an interactive HTML chart like this:

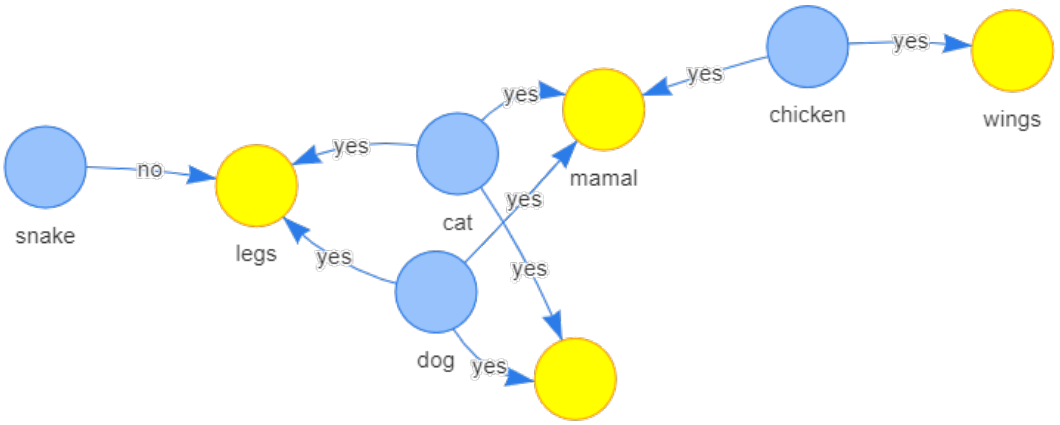

Figure 1: graph

Degraded modes

Infinite loop

In case you define an infinit-loop in Prolog, or your query takes too long to process, there is a time limit protection of 10s. You can change the duration with *timeout*. Remove *eval=F* to try.

```
s1(A) :- s2(A).
s2(A) :- s1(A).

?- s1(X)
```

Verbose mode

In case you do not get the expected result, you can activate the *verbose* mode.

Remove *eval=F* to try.

```
foo(ee)
bar(ee,a) :- foo(cc).

?- bar(A, B)
```

You will get the Prolog source file and Prolog output in R Console output. *swiplr* will load the source file in swi-prolog and add several main_[action] predicates.

Advances features

Check query duration

Use *mode="duration"*. You will get the result in milliseconds.

```
loop([], 0).

loop([[a,D]|L], C) :-
  C>0,
  D is C-1,
  loop(L, D).

?- loop(L, 100000)
```

Query: loop(L, 100000)

90

Profile

Use *mode="profile"*. You will get the result in milliseconds.

```
loop([], 0).

loop([[a,D]|L], C) :-
  C>0,
  D is C-1,
  loop(L, D).

?- loop(L, 10)
```

Query: loop(L, 10)

L
c(c(a, 9), c(a, 8), c(a, 7), c(a, 6), c(a, 5), c(a, 4), c(a, 3), c(a, 2), c(a, 1), c(a, 0))

trace

You can trace calls and activate verbose mode. Doing so you will see output in Console.

```
foo(one).
bar(two).
bar(X) :- foo(X).

:- trace(bar/1).
:- trace(foo/1).

?- bar(X).
```

Query: bar(X)

X
two
one

Strange cases

```
foo(no_space).
foo('with space').
foo([this, is, a, list]).
foo([this, is, a, 'list with space']).
foo([this, is, a, [list, [of, list]]]).
foo(this(has(pred))).
foo(this(has('pred with space'))).
foo('even , } { } ) or ( works in atom').

?- foo(X)
```

Query: foo(X)

X
no_space
with space
c(this, is, a, list)
c(this, is, a, “list with space”)
c(this, is, a, c(list, c(of, list)))
this(has(pred))
this(has(“pred with space”))
even , } { }) or (works in atom

known limitations

```
should_we_calculate(case1, 1+1).
should_we_calculate(case2, 2*5).
should_we_calculate(case3, 2/5).
should_we_calculate(case4, 2-5).
should_we_calculate(but5, 5**2).
should_we_calculate(but6, sqrt(9)).
should_we_calculate(bug, c(9, 3)).
% this is due to collision with R c() function

?- should_we_calculate(CASE, RESULT)
```

Query: should_we_calculate(CASE, RESULT)

CASE	RESULT
case1	2
case2	10
case3	0.4

CASE	RESULT
case4	-3
but5	5^2
but6	sqrt(9)
bug	9,3

Build an app

```
run :- writeln('hello world').

?- qsave_program('app.bin', [goal(run), stand_alone(true), foreign(save)]).
```

Query: qsave_program('app.bin', [goal(run), stand_alone(true), foreign(save)])

TRUE

```
./app.bin
```

```
## hello world
```