Installing s(CASP) on SWI-Prolog (for Windows)

Joaquín Arias[1], Sarat-Chandra Varanasi[2] and Gopal Gupta[2]

[1]*CETINIA, Universidad Rey Juan Carlos, Madrid, Spain*
[2]*University of Texas at Dallas, Richardson, USA*

Abstract
This paper presents a short tutorial on how to install s(CASP), under SWI-Prolog, on Windows, highlighting the different options available thanks to the integration of s(CASP) with SWI-Prolog The most important aspect of s(CASP) under SWI-Prolog is its integration as a library and the opportunity to use the online tool swish, available at https://swish.swi-prolog.org/

Keywords
Answer Set Programming, Constraint, Goal-directed, s(CASP), Installation, SWI-Prolog

1. Installation for Windows

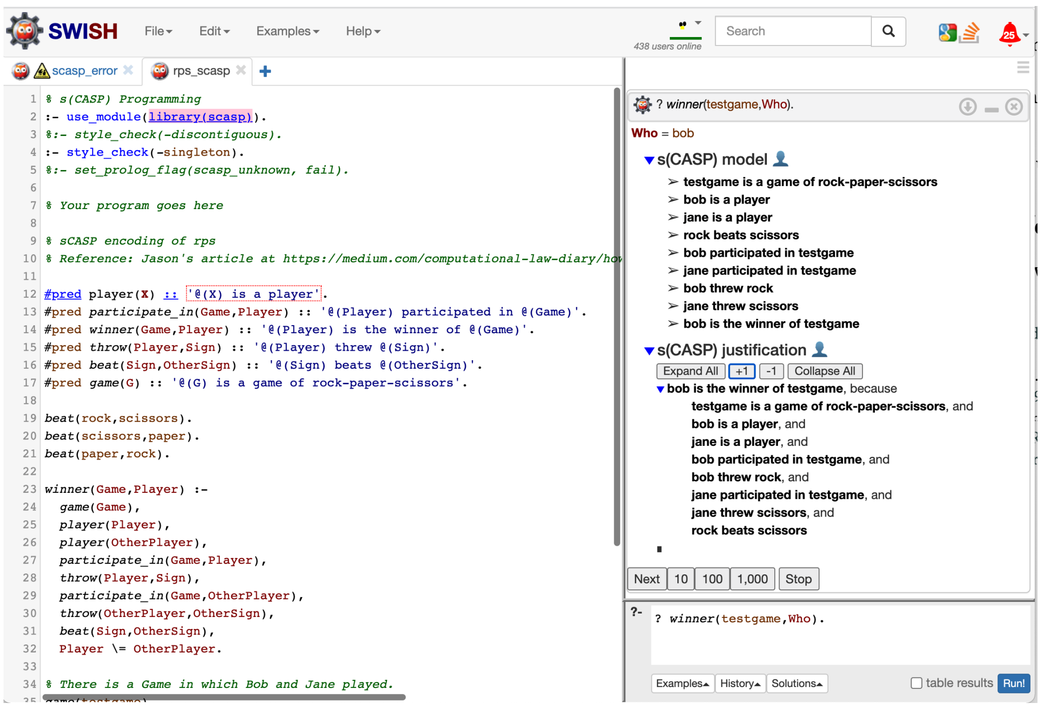
s(CASP) [1] is a novel non-monotonic reasoner, developed by Joaquín Arias in collaboration with IMDEA Software Institute and the University of Texas at Dallas. It is a re-implementation of s(ASP) [2] by Kyle Marple et al, extended with constraints. The s(CASP) and s(ASP) systems are, essentially, goal-directed implementations of *answer set programming* [3], with and without constraint solving over reals, respectively.

This tutorial explains step-by-step, how to install (as of December 2021) a re-implementation of s(CASP) by Jan Wielemaker to run s(CASP) under SWI-Prolog [4]. Thanks to this SWI-Prolog re-incarnation of s(CASP), now we are able to install and use s(CASP) in three ways:

1. *As a standalone executable program* using SWI-Prolog. Its behaviour follows previous versions of s(ASP) and s(CASP).
2. *As a sub-program* in a more complex Prolog program, i.e., we are allowed to include s(CASP)) as a library in another Prolog program.
3. *Through a Web-interface* of SWI-Prolog called *SWISH* (https://swish.swi-prolog.org/p/rps_scasp.pl), an online environment for teaching and exchanging ideas. Therefore, no installation is needed. Following image correspond to the example that is pre-populated in SWISH. We can run run the query for this program in the box on the right, as shown below:

joaquin.arias@urjc.es (J. Arias); sarat-chandra.varanasi@utdallas.edu (S. Varanasi); gupta@utdallas.edu (G. Gupta)
http://www.ia.urjc.es/GIA/joaquin-arias/ (J. Arias); https://personal.utdallas.edu/~gupta/ (G. Gupta)
0000-0003-4148-311X (J. Arias); 0000-0002-4620-4266 (S. Varanasi); 0000-0001-9727-0362 (G. Gupta)

CEUR Workshop Proceedings (CEUR-WS.org)



```
% s(CASP) Programming
:- use_module(library(scasp)).
%:- style_check(-discontiguous).
:- style_check(-singleton).
%:- set_prolog_flag(scasp_unknown, fail).

% Your program goes here

% sCASP encoding of rps
% Reference: Jason's article at https://medium.com/computational-law-diary/hov

#pred player(X) :: '@(X) is a player'.
#pred participate_in(Game,Player) :: '@(Player) participated in @(Game)'.
#pred winner(Game,Player) :: '@(Player) is the winner of @(Game)'.
#pred throw(Player,Sign) :: '@(Player) threw @(Sign)'.
#pred beat(Sign,OtherSign) :: '@(Sign) beats @(OtherSign)'.
#pred game(G) :: '@(G) is a game of rock-paper-scissors'.

beat(rock,scissors).
beat(scissors,paper).
beat(paper,rock).

winner(Game,Player) :-
  game(Game),
  player(Player),
  player(OtherPlayer),
  participate_in(Game,Player),
  throw(Player,Sign),
  participate_in(Game,OtherPlayer),
  throw(OtherPlayer,OtherSign),
  beat(Sign,OtherSign),
  Player \= OtherPlayer.

% There is a Game in which Bob and Jane played.
```

```
? winner(testgame,Who).
Who = bob
▼ s(CASP) model
  ➢ testgame is a game of rock-paper-scissors
  ➢ bob is a player
  ➢ jane is a player
  ➢ rock beats scissors
  ➢ bob participated in testgame
  ➢ jane participated in testgame
  ➢ bob threw rock
  ➢ jane threw scissors
  ➢ bob is the winner of testgame
▼ s(CASP) justification
  ▼ bob is the winner of testgame, because
      testgame is a game of rock-paper-scissors, and
      bob is a player, and
      jane is a player, and
      bob participated in testgame, and
      bob threw rock, and
      jane participated in testgame, and
      jane threw scissors, and
      rock beats scissors
  .

?- ? winner(testgame,Who).
```

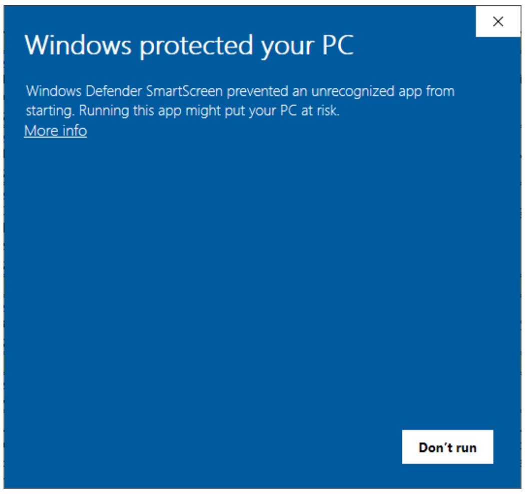
Next we explain how to install and use s(CASP) under SWI-Prolog for Windows.

2. Installing SWI-Prolog for Windows

The installation of SWI-Prolog under Windows is straightforward.[1]

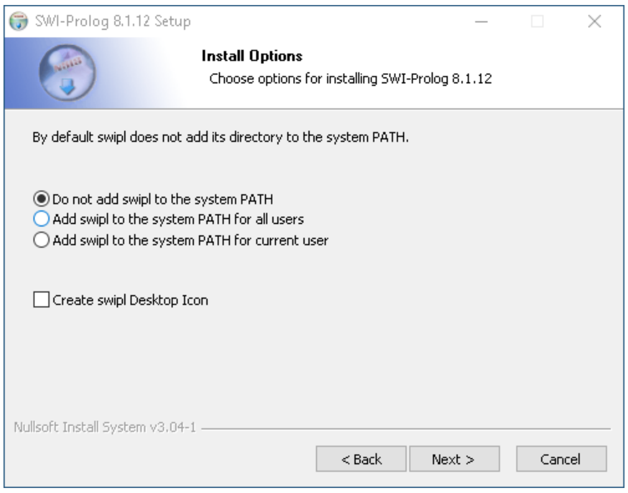
1. Visit the SWI-Prolog website https://www.swi-prolog.org/ and select the menu: `DOWNLOAD/Swi-Prolog`.
2. Then, follow the link "`Development release`" and download one of the first two binaries (i) `SWI-Prolog 8.12.1 for Microsoft Windows (64 bit)` or (ii) `SWI-Prolog 8.12.1 for Microsoft Windows (32 bit)`.
3. To download the selected binary you have to click "`I understand`' in the notification page and the final link will be activated.
   This notification page explains that during the installation process the Windows Security Dialog will appear and you will have to go through it:

[1]Instructions extracted from https://swi-prolog.discourse.group/t/install-swi-prolog-development-version-on-windows-10/1131 on December 2021.


Windows protected your PC

Windows Defender SmartScreen prevented an unrecognized app from starting. Running this app might put your PC at risk.
More info

4. It is important, when you reach the Install Options window, that you select the correct options. We recommend:
    - Add swipl to the system PATH for all users
    - Create swipl Desktop Icon



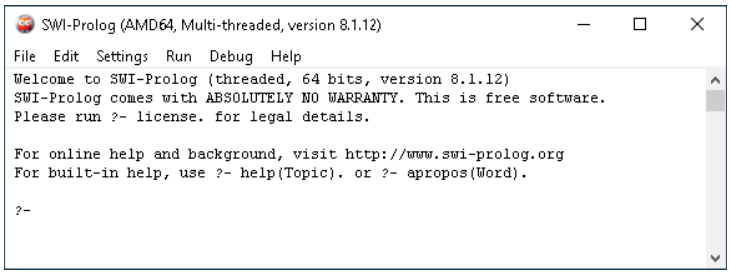
Install Options
Choose options for installing SWI-Prolog 8.1.12

By default swipl does not add its directory to the system PATH.

[✓ Do not add swipl to the system PATH]
Add swipl to the system PATH for all users
Add swipl to the system PATH for current user

☐ Create swipl Desktop Icon

5. Check that the installation is done by clicking the SWI-Prolog button. If you see the SWI-Prolog console below, the installation is done!!!


SWI-Prolog (AMD64, Multi-threaded, version 8.1.12)
File Edit Settings Run Debug Help

```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.1.12)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?-
```

Example 1. *To invoke the s(CASP) package (included by default in the Windows version of SWI-Prolog), let us consider* `pq_package.pl`*:*

```
:- use_module(library(scasp)). %% include the scasp package.
:- style_check(-singleton).    %% remove warning due to singletons.

p(X) :- not q(X).
q(X) :- not p(X).
```

First, we run the swi interpreter, then, consult/load the file `pq_package` *and finally, we can invoke any query using the symbol '?', e.g.,:*

```
$ swipl
?- [pq_swi].
?- ? p(X).
% s(CASP) model
{ p(a),      not q(a)
},
% s(CASP) justification
...
```

References

[1] J. Arias, M. Carro, E. Salazar, K. Marple, G. Gupta, Constraint Answer Set Programming without Grounding, Theory and Practice of Logic Programming 18 (2018) 337–354. doi:10.1017/S1471068418000285.

[2] K. Marple, E. Salazar, G. Gupta, Computing Stable Models of Normal Logic Programs Without Grounding, arXiv 1709.00501 (2017). URL: http://arxiv.org/abs/1709.00501. arXiv:1709.00501.

[3] M. Gelfond, Y. Kahl, Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach, Cambridge University Press, 2014.

[4] J. Wielemaker, J. Arias, G. Gupta, s(CASP) for SWI-Prolog, in: ICLP Workshops, volume 2970 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2021, p. 4.