Prolog Cheat Sheet

Interactive Prolog Setup

In Prolog, one declares a relationship `r(x0, x1, ..., xn)` to be true for the declared `xi` —with a change of perspective any of the `xi` can be considered 'input' and the rest considered 'output'.

```
(use-package prolog)


;; Obtain "swipl" interpreter.
(async-shell-command "brew install swi-prolog")

;; alhassy-air:~ musa$ swipl --version
;; SWI-Prolog version 8.0.2 for x86_64-darwin
```

The following did not work for me :'(—so I made my own lolz.

```
(use-package ob-prolog)
(org-babel-do-load-languages
 'org-babel-load-languages
 '((prolog . t)))

(use-package ediprolog)
```

Here's my current setup:

```
(local-set-key (kbd "<f6>") (lambda () (interactive)
  "
    org-babel-tangle the whole file, then execute the final query
    in the current SRC block.

    If the query mentions the variable 'X', then show all possible solutions
    followed by 'false'. Usually one presses ';' to see other solutions,
    but in Emacs this only shows one futher solution then terminates.
    We get around this by executing essentially
    "forall(your-query-with-X, writeln(X)).''
    This prints all solutions X to your query.

    If you want to use a variable but don't want to see all solutions,
    then avoid using 'X'; e.g., use 'Y' ^_^.
  "
  (-let [kill-buffer-query-functions nil]
  (ignore-errors
     (switch-to-buffer "*Prolog*")
     (kill-buffer "*Prolog*"))

  ;; Get final query in current source block
  (search-forward "#+END_SRC")
  (search-backward "% ?-")
  ;; Copy line, without killing it.
  (setq xx (thing-at-point 'line t))

  (async-shell-command (format "swipl -s %s" (car (org-babel-tangle))) "*Prolog*")
  (other-window 1)

  ;; Paste the final query
  (setq xx (s-chop-prefix "% ?- " xx))
  (when (s-contains? "X" xx)
    (setq xx (concat "writeln(\"X =\"), forall(" (s-replace "." ", writeln(X))." xx

  (insert xx)
  (comint-send-input nil t) ;; Send it, i.e., "press enter at prompt".

  ;; Insert query again, but do not send, in case user wishes to change it.
  (insert xx)
  (previous-line) (end-of-line)

)))
```

For example:

```
magicNumber(7).
magicNumber(9).
magicNumber(42).

% ?- magicNumber(8).
% ?- magicNumber(X).
```

Or

```
main :- write('Hello, world!').

% ?- main.
```

ToDo: Get the query, replace (,) with space, split to words, filter those that start with a capital letter, these are the variables. Execute: forall((query) , (write("Var0 = "), write(Var0), write(" "), write("Var1 = "), ...)))

Nullary Facts

We declare relations by having them begin with a lowercase letter; variables are distinguished by starting with a capital letter.

```
jasim_is_nice.

% ?- jasim_is_nice. % true: We declared it so.

it_is_raining. /* Another fact of our world */

% ?- it_is_raining. % true

eats(fred, mangoes).
eats(bob, apples).
eats(fred, oranges).
```

```
% ?- eats(bob, apples). % true

% Which foods are eaten by fred?
% ?- eats(fred, what). % false; 'what' is name!
% ?- eats(fred, What). % mangoes oranges
```

Here's a cute one:

```
% All men are mortal.
mortal(X) :- man(X).

% Socrates is a man.
man(socrates).

% Hence, he's expected to be mortal.
% ?- mortal(socrates). % true

% What about Plato?
% ?- mortal(plato). % false, plato's not a man.

% Let's fix that.
man(plato).

% Who is mortal?
% ?- mortal(X). %  socrates plato
```

Lists

Lists are enclosed in brackets, separated by commas, and constructed out of cons "|".

```
% ?- ["one", two, 3] = [Head|Tail].
```

See here for the list library, which includes:

member(element, list)
append(list1, list2, lists12)
prefix(part, whole)
nth0(index, list, element)
last(list, element)
length(list, number)
reverse(list1, list2)
permutation(list1, list2)
sum_{list}(list, number)
max_{list}(list, number)
$\text{is}_{\text{set}}(\text{list}_{\text{maybenoduplicates}})$

Core Ideas

Prolog's execution model is based on 4 building blocks: Logical or, Logical and, term rewriting, and unification.

Unification Can the given terms be made to represent the same structure?

- ◇ This is how type inference is made to work in all (?) languages.

Backtracking When a choice in unification causes it to fail, go back to the most recent choice point and select the next avialable choice.

Unification performs no simplification, whence no arithmetic. This means, for example, we can form pairs by sticking an infix operator between two items; moreover we can form distinct kinds of pairs by using different operators:

```
?- C + "nice" = woah + Z.
C = woah,
Z = "nice".

% '+' and '/' are different, so no way to make these equal.
?- C + "nice" = woah / Z.
false.
```

Basics

- ◇ Unification only tries to make both sides of an equality true by binding free variables to expressions. It does not do any arithmetic.
- ◇ Use `is` to perform arithmetic.

```
% ?- X = 3 + 2.  %% X = 3 + 2
% ?- X is 3 + 2. %% X = 5
```

`Print` predicate always succeeds, never binds any variables, and prints out its parameter as a side effect.

- ◇ Conjunction: `p(X), q(X)` means "let `X` be *a* solution to `p`, then use it in query `q`."
  - ○ Operational semantics: Let `X` be the first solution declared, found, for `p`, then try `q`; if it fails, then *backtrack* and pick the next declared solution to `p`, if any, and repeat until `q` succeeds.
    For example, `p(X), print(X), fail.` gets a solution to `p`, prints it, then fails thereby necessitating a backtrack to obtain a different solution `X` for `p`, then repeats. In essence, this is prints all solutions to `p` —a so-called "fail driven loop".

For example,

```
yum(pie).
yum(apples).
yum(maths).

% ?- yum(Y), writeln(Y), fail. % pie apples maths false.
```

Use built-ins `var` and `novar` to check if a variable is free or bound.

```
% ?- var(Y).  % true
% ?- Y = 2, var(Y). % false
% ?- Y = 2, novar(Y). % true
```

Adminstriva

- ◇ Write a prolog program as a text file with a .pl ending. For example, program.pl.
- ◇ Open SWI-Prolog by invoking swipl in the terminal.
- ◇ In SWI-Prolog, type [program] to load the program, i.e. the file name in brackets, but without the ending.

- In order to query the loaded program, type goals and watch the output.
- To exit SWI-Prolog, type halt..

Alternatively, you can also load the program by passing its name as a parameter to SWI-Prolog: swipl -s program.pl.

Reads

Organised in terms of length:

- Introduction to logic programming with Prolog –12 minute read.
- Introduction to Prolog —with interactive quizzes
- prolog :- tutorial
- `https://learnxinyminutes.com/docs/prolog/`