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 = "), …)))

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
```