Prolog Cheat Sheet

Prolog programs consist of comments and period terminated terms.

Comments

```
% end of line comment
/* C style comment */
/** invokes the structured comment mechanism, pldoc */
%! so does this
%% so does this
```

Special

colon, period, parens, comment chars, quasiquote delimiters, [] (empty list)
! is "Cut", discard choice points.

Terms

Every unique term has a different place in standard sort order.
Variables < *Numbers* < *Strings* < *Atoms* < *Compound Terms*

Atoms

Fundamental keyword like atomic object.
print out as themselves, so most string constants

- Any sequence of letters, numbers, and _ that starts with a lowercase letter
- any sequence of specials, from `#$&*+-./:<=>?@\^~` (also lots of unicode characters!)
- Any sequence of characters between single quotes, with escapes like \n \' The quotes only exist at read time

```
by_convention_atoms_are_like_this
```

Variables

Thing whose value may or may not be known. More like ‘unknowns’ from algebra class.
Variables sequence of upper and lowercase letters and underscores. Start with uppercase letter.

```
ByConventionVariablesAreCamelCase
```

Underscore

Special variable, each underscore gives a new variable whose binding is discarded. _X is an ordinary variable, but it's value is not reported in the top level.

Codes String

pre 7.0 and in many books, anything in double quotes. Now anything in back quotes.
Only exist at read time - become a **list** of integer utf-8 code points.

```
?- X = `hello`.
X = [104, 101, 108, 108, 111].
```

Real String

Introduced with rev. 7.0, non-interned "real" strings, stored compactly.

Integer

Integers up to $2^{3+2^{32}}$, 0'a ascii value of a, 0xFF

Float

Rational

?- X is 22 rdiv 7.
X = 22 rdiv 7.
?- X is 8 rdiv 2.
X = 4.

Prolog Cheat Sheet

Compound
like C structs. Can be called.
<atom> **(** <term> , <term>, ...**)**
Compounds can be written as prefix, infix, postfix, or distfix operators by defining operators.
Thus

```
?- write_canonical(2 + 3).
+(2,3)
```

List
convert to a list of compounds at read time. [] is special. pre 7.0 was '.'(a,[]).

```
?- X = '[|]'(a,[]).
X = [a].
```

Lists can have a 'rest' section that unifies with the remainder of list

```
?- [A, B | C] = [a,b,c,d].
A = a,
B = b,
C = [c, d].
```

The univ operator =.. converts between lists and compounds

```
?- some_compound(a, foo(b)) =.. X.
X = [some_compound, a, foo(b)].
```

Dicts
Added 7.0,
<tag_atom>{ <key_atom> : <term> , ... }
key-value store. Has some special syntax for extracting values.

Operators (abbrev. list)

?- not an operator, the prompt for the interactor
:- "Neck", the right side implies the left. As a prefix operator, is a directive.
--> DCG - definite clause grammars use this instead of neck.
= Unify
== can unify - don't do it, just succeed if you can
=:= numerically equal
=\= numerically unequal
\= cannot unify
=< , < , > , >= inequality ops (note that <= and => not defined)
-> if (this is more like C's ?: operator, beginners should avoid)
; disjunction (OR) - also best avoided by beginners- note that conjunction binds more tightly than disjunction, so

```
something(),
(
        try_a()
 ;
        try_b()
),
something_else()
```

, conjunction (AND)
(a,b,c) is a so called 'round list', it's a list of ','(a,','(b,c)) - avoid unless you're making a directive

Prolog Cheat Sheet

=@=, @< etc. compare standard order
-,/, = have other meanings, but also often used to make key-value pairs
+ - * / << ^ arithmetic is evaluated by the **is** operator, not the **=** operator
mod, rem remainder
rdiv rational division
div integer division
dynamic, meta_predicate, etc - many directives are defined as operators, so one can write
:- dynamic foo/2,bar/3.
:< sub dict unification
>:< dict unification
#= #< etc These are constraint operators

Signatures
<name>/<arity>
<module>:<name>/<arity>
<name>//<arity> - a DCG with arity semantic arguments.

Common predicates
lists - member/2 length/2 append/3, append/2,
append/3
is/2
call/1, call/n

Help
doc_server(5000). Starts localhost server that serves docs.
The swi-prolog website http://swi-prolog.org