PROLOG NOTES #6

Syntax

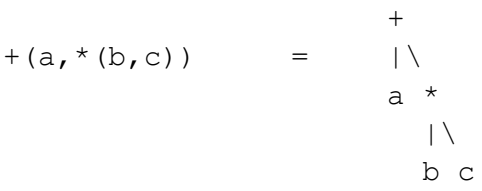
How elements of a language can be put together

Term

Building block of Prolog programs

- constant = names a specific object or relationship
  - number
  - atom = starts with lower-case letter, can use or start with signs, or enclosed in ‘ ‘
    ?- and :- are also atoms
- variable = things we do not want or cannot at the time of writing the program
  start with upper-case letter or _ ; single _ represents an anonymous variable i.e. variable whose name will never be used, it saves the need to invent unique variable names for single-use, every time _ is used it represents a different variable
- structure = “compound term”, single object consisting of components; components can be made up of other components; structures are made of a functor constant followed by components enclosed in parenthesis; think of functor as data type
  ex. owns(john, book(wuthering_heights, author(bronte, emily)))
  structures can participate in question answering ex. ?- owns(john, book(A,B)).
  predicates and facts are structures!

Trees can represent structures - the functor is the root and the components are leaves:



NOTE: You can create a directed acyclic graph DAG ex. f(X,g(X,a))

Rule

head :- body Where head is a term and body is made of conjunction of goals which are terms.

Fact

Rule with no body is a fact.

Predicate

The functor constant of the head of a rule plus the arity (number of components)
Two predicates with the same constant but different arity are different.

Lists

Ordered sequence of elements, any practical structures can be represented with lists
It can be an empty lists [] or something with head and tail and tail is a list (including an empty list).

```
[a]         is    .(a,[])
[a,b,c]     is    .(a,.(b,.(c,[])))          .-.-.-[]
                                             | | |
                                             a b c
```

Lists are manipulated by splitting the head and tail:

[a,b,c] → [a|[b,c]]
[a,b,c] → [a,b|[c]]
[a,b,c] → [a,b,c|[]]

NOTE: [white|horse] is legal and not a list, also can be represented as .(white|horse)

Unification

try to make two thing equal if possible, represented by infix =
If X and Y are not instantiated then X=Y will co-refer to the same thing.

Important Built-In Predicates

=:=	same number
=/=	different number
//	integer quotient
mod	integer division reminder

NOTE: arithmetic is not performed in Prolog until “is” is used and all values on the right-hand side must be known at this time.

write(X)	write something on the terminal
read(X)	read a term from the terminal (must be followed by .)
consult(Fn)	load and compilea Prolog program
true	do succeed
fail	do fail
var(X)	is X an uninstantiated variable?
nonvar(X)	is X an instantiated variable?
atom(X)	is X an atom?
number(X)	is X an integer number?
atomic(X)	is X an atom or a number?

asserta(C)	dynamically add a clause at the beginning of the program
assertz(C)	dynamically add a clause at the end of the program
retract(C)	dynamically remove a clausefrom the program
clause(H,B)	find a matching clause in the current program with head H and body B
listing(C)	find all clauses with a given predicate symbol
functor(T,F,A)	retrieve the functor F and arity A from structure T
arg(N,T,A)	retriever Nth argument A for structure T
X=..L	turn a structure into a list ex. a(b,c)=..[a,b,c]
	convert a goal into a list of functor and arguments
repeat	repeat and pause
,	AND for goals
;	OR for goals
call(X)	execute/resolve X
\+	NOT
==	test if variables are coreferring

Logic Programming and Prolog

Logic programming = stating what is true and asking for conclusions; declarative; no control over execution is used/required.

Prolog can be both logic programming and procedural language, depending on the chosen principles/standards. To keep Prolog as logic programming as possible, keep non-logical parts (built-in library predicates and ! for example) in helper predicates and use the helper predicates to build logic programs, avoid state testing predicates (for example var()), avoid assert/retract predicates.