Prolog

Prolog is a programming language for PROgramming with LOGic

- It doesn't mean just being sensible and making a logical design
- It means expressing your program in terms of facts and deduction rules
- No loops, no functions of the sort we're used to
- Just predicates, terms, and definitions
- It makes use of
  Unification
  Resolution
  Predicate calculus (parts of it anyway) and
  Backtracking
- Available on rabbit.eng.miami.edu
- But download your own: https://www.swi-prolog.org/download/stable
- Under Settings menu select "User init file …".
- If it asks about creating it, say yes.
- Insert into it

```
:- working_directory(_, 'D:\\prolog').
```

- but of course choose your own working directory instead of C:\\prolog
- Save and exit. Exit from prolog and restart it.

Backward Chaining

- Instead of Forward Chaining which goes from known facts to conculsions
- We start from the Goal (the question) and work backwards
- Building an (almost) ever growing list of things that we need to be true
- e.g., the goal is Criminal(W)
- that becomes
  american(W) ∧ weapon(Y) ∧ hostile(Z) ∧ sold(W, Y, Z)
- that becomes
  weapon(Y) ∧ hostile(Z) ∧ sold(colw, Y, Z)
- that becomes
  missile(Y) ∧ hostile(Z) ∧ sold(colw, Y, Z)
- that becomes
  hostile(Z) ∧ sold(colw, m1, Z)
- that becomes
  enemy(Z, america) ∧ sold(colw, m1, Z)
- that becomes
  sold(colw, m1, nk)
- that becomes
  missile(m1) ∧ owns(nk, m1)
- that becomes
  owns(nk, m1)
- that becomes
  (nothing)
- Which means that we succeeded
- And along the way, we learned that
  { W→colw, X→W, Y→m1, Z→nk }

- W is the only variable that appeared in the question, so the answer is
  W = colw

Syntax

- As before, little letters for predicates and functions, big letters for variables
- Predicates separated by commas are to be anded together
  each predicate must be made true in turn, left to right
- Predicates separated by semicolons are to be ored together
  try each in turn until one comes out as true
- Not doesn't really work very well, but we get by without it
- Colon-dash :- means ⇐, it is how predicates are defined
  A dot . appears at the end of a definition (called a Rule)
  hates(X, Y) :- cat(Y), mouse(X).
  hates(X, Y) :- dog(Y).
- Two definitions for the same predicate means they are ored together
  that's why we don't see semicolons so much
- A predicate on its own is just a statement of fact
  cat(tom).
  is-something(X).

Here is a complete program:

```
cat(tom).
dog(spike).
mouse(jerry).
mouse(jerrys-nephew).
would-eat(X, Y) :- cat(X), mouse(Y).
hates(X, Y) :- would-eat(Y, X).
hates(X, Y) :- cat(X), dog(Y).
```

We run it by entering a predicate, which is treated as a query

hates(jerrys-nephew, tom).
  says Yes
hates(tom, jerry)
  Fails
hates(X, Y)
  lists all valuations for X and Y that make it true
but we can never ask "who is what?"

But a simple change of style opens up things like that:

```
isa(tom, cat).
isa(spike, dog).
isa(jerry, mouse).
isa(jerrys-nephew, mouse).
would-eat(X, Y) :- isa(X, cat), isa(Y, mouse).
hates(X, Y) :- would-eat(Y, X).
hates(X, Y) :- isa(X, cat), isa(Y, dog).
```

Or even more flexibly:

```
fact(tom, isa, cat).
```

fact(spike, isa, dog).
fact(jerry, isa, mouse).
fact(jerrys-nephew, isa, mouse).
fact(X, would-eat, Y) :- fact(X, isa, cat), fact(Y, isa, mouse).
fact(X, hates, Y) :- fact(Y, would-eat, X).
fact(X, hates, Y) :- fact(X, isa, cat), fact(Y, isa, dog).

NOT doesn't work at all well

- solutions consist of valuations of variables (maybe empty)
- a false predicate fails, it does not produce a valuation
  Predicates fail exactly because no valuation works
- changing a success into a failure is trivial
- changing a failure into a success doesn't really work
  Successes are supposed to produce valuations
  but a failure give us nothing to work with.

To get in, type the command "`prolog`" or "`prolog` *filename*`.pl`"
To get out, type "`halt.`" Remember the dot at the end.
To enter definitions manually (not very good)
type `consult(user).`
and control-D when you're done
There is no way to save the definitions you type.
To read definitions from a file (much better)
type `consult(`*filename*`)`. Do not put the `.pl` at the end of the filename.
ignore warnings about "Singleton variables"
To ask a query, just type it.
If there are multiple solutions, it will print the first then wait.
press `;` to see the next solution
or press ENTER to stop and get back to the command prompt

Backtracking

- As before, Prolog maintains a list of predicates that must all be true
  and the current substitution
- It also maintains a stack of (predicate-list, substitution) pairs
  that could also lead to an answer
- When there is a choice (two or more rules match the same predicate)
  one is chosen to go into the current predicate list
  the others go onto the stack
- If the current predicate list turns out to be false
  just backtrack to the alternative on top of the stack
- Or if the user asks for another solution after one has been delivered