Prolog - A Logic Programming language

Prolog lets us describe properties about things
eg whether a sentence conforms to a grammar
whether an element is a member of a list
whether a list is a sublist of another
whether a placing of pieces in a game is valid

We can use a sentence checking program to check
a given sentence or to derive correct sentences

Haskell is functional, whereas Prolog is relational
We might have some *facts* about a graph

eg node(a). node(b). node(c). node(d).
arc(a,b). arc(a,c). arc(b,d). arc(d,c).

We call this a *theory*, or a *program* and we can ask it a *query*

Which nodes are connected by an arc to c?

? arc(W,c). "find W s.t. arc(W,c) is true"
We may also have *rules* describing what a path is

e.g. path(X,Y) :- arc(X,Y).
"path from X to Y is true if arc from X to Y is true"

    path(X,Y) :- arc(X,Z), path(Z,Y).
"for any X,Y and Z, path(X,Y) holds if arc(X,Z) holds and path(Z,Y) holds"
Is there a path from a to c?
? path(a,c).

For which nodes is there not a path to b?
? node(W), \+path(W,b).

"find nodes W s.t. path(W,b) is not true"
How Prolog deduces the answers (1)

```
?- path(a,c).
?- arc(a,c).
YES!

?- path(b,c).
?- arc(b,c).
NO!

?- path(c,c).
?

?- arc(a,Z), path(Z,c).
Z = b
?- path(b,c).
?- arc(b,U), path(U,c).
U = d
?

?- arc(d,c).
YES!
```
to find \( W \): \( \text{node}(W), \neg \text{path}(W, b) \)

Prolog assumes that \( \text{path}(a, b) \) is false if it cannot be deduced - *negation by failure*

node(a) is true - is \( \text{path}(a, b) \) false?
\( \text{path}(a, b) \) can be deduced, so it is *not* false.

node(c) is true - is \( \text{path}(c, b) \) false?
\( \text{path}(c, b) \) cannot be deduced:
- to do so requires \( \text{arc}(c, b) \) or \( \text{arc}(c, Z) \) for some \( Z \);
- neither is true - so \( \text{path}(c, b) \) is false and \( c \) is an answer.

node(b) is true - is \( \text{path}(b, b) \) false?
\( \text{path}(b, b) \) cannot be deduced ......
Prolog reduces a query \( Q \) to sub-queries .... until it reaches known facts.

From the known facts ASP finds all consequences at once ....
ASP - another Logic Programming Language

ASP - finds all facts implied by the program at once

```prolog
node(a). node(b). node(c). node(d).
arc(a,b). arc(a,c). arc(b,d). arc(d,c).
source(X) :- node(X), not arcTo(X).
arcTo(X) :- arc(Y,X).
```

```
node(a), node(b), node(c), node(d),
arca,b), arc(a,c), arc(b,d), arc(d,c),
arcto(b), arcTo(c), arcTo(d),
source(a)
```
ASP - another Logic Programming Language

node(a).  node(b).  node(c).
arc(a,c).  arc(c,b).
\{arc(a,b), arc(b,a)\}1.
path(X,Y) :- arc(X,Y).
path(X,Y) :- path(Z,Y), arc(X,Z).
:-path(X,X).

node(a), node(b), node(c),
node(c),
ar(a,b),  arc(c,b),
ar(a,c),
path(a,b), path(a,c),
path(c,b)

ASP - Answer Set programming
Otter uses *clausal form* (disjunctions of literals)

1. Write data in logic and convert to clauses
2. Write conclusion in logic
3. Negate conclusion and convert to clauses
4. Derive a contradiction

Otter uses the deduction rule of *resolution*

\[ c(d) \lor c(e) \quad \text{and} \quad \neg c(e) \lor h(e) \implies c(d) \lor h(e) \]

\[ c(d) \lor c(e) \quad \text{and} \quad \neg c(X) \lor h(X) \implies c(d) \lor h(e) \quad \text{both by resolution} \]
An Example: Two naughty children

Either Dolly and Ellen was the culprit but only one.
The culprit was in the house.

Dolly: "It wasn't me, I wasn't in the house; Ellen did it."
Ellen: "I didn't do it; Dolly was in the house."

Neither told the truth.

Who did it?
**Two naughty children in Logic**

1. \( c(d) \lor c(e) \)  
   At least one of 2 girls did it

2. \( \neg c(d) \lor \neg c(e) \)  
   Exactly one of them was the culprit

3. \( \neg c(X) \lor h(X) \)  
   The culprit was in the house

4. \( c(d) \lor h(d) \lor \neg c(e) \)  
   Negation of dolly’s testimony

5. \( c(e) \lor \neg h(d) \)  
   Negation of Ellen's testimony

6. \( \neg c(e) \lor c(d) \)  
   Negation of goal
Two naughty children in Logic

1. \( c(d) \lor c(e) \)
2. \( \neg c(d) \lor \neg c(e) \)
3. \( \neg c(X) \lor h(X) \)
4. \( c(d) \lor h(d) \lor \neg c(e) \)
5. \( c(e) \lor \neg h(d) \)
6. \( \neg c(e) \lor c(d) \)

7. \( (1+6) \ c(d) \)
8. \( (7+3) \ h(d) \)
9. \( (8+5) \ c(e) \)
10. \( (7+2) \ \neg c(e) \)
11. \( (10+9) \ \bot \)
Using Automated Reasoning in our Course

Reasoning about Programs – year 1
Prolog language – year 2
Artificial Intelligence – year 2
Software verification – year 3
Machine Learning – year 3
Automated Reasoning – year 4
Probabilistic Inference and Data Mining – year 4
Multi-Agent Systems – year 4

plus bits here and there in other courses
and in projects