Prolog

the idea of logic programming

Prolog and Logic

Prolog and Databases

how it works

a limitation

data structures in Prolog

The Essence of Prolog

Prolog is a logic programming language
(PROgramming in LOGic)

It is also *declarative* and *interactive*

Declare *facts* and *rules*
Ask questions; Prolog answers.

Deductive Inference

Prolog and Logic

Fact examples:

```
parent(mary,john).
female(mary).
parent(ann,mary).
```

Rule examples:

```
mother(X,Y) :- parent(X,Y), female(X).
grandparent(X,Z) :-    parent(X,Y),
                       parent(Y,Z).
```

Compare to Logic:

$\forall X: \forall Y: \forall Z:$
$parent(X,Y) \wedge parent(Y,Z) \rightarrow grandparent(X,Z)$

How to Read a Rule

```
grandparent(X,Z) :-    parent(X,Y),
                       parent(Y,Z).
```

To prove that X is the grandparent of Z (:-)

prove that X is *a* parent of some Y and (,)

(also) that *the same* Y is a parent of Z (.)

One fact relates to many questions.

```
parent(mary,john).
```

"Is Mary a parent of John?"

```
|?- parent(mary,john).
yes.
```

"Who is Mary a parent of?"

```
|?- parent(mary,X).
X = john.
```

"Is Mary a parent (of anyone)?

```
|?- parent(mary,_).
yes.
```

One fact relates to many questions (cont.)

```
parent(mary,john).
```

“Who is a parent of John?”

“Does John have (anyone as) a parent?

“Who is a parent of whom?”

```
| ?- parent(X,Y).
X = mary, Y = john.
```

“Is anyone a parent of anyone?”

Other Forms of Interaction

Facts:

```
parent(mary,john).
female(mary).
parent(ann,mary).
```

Rules:

```
mother(X,Y) :- parent(X,Y), female(X).

grandparent(X,Z) :- parent(X,Y), parent(Y,Z).
```

Queries:

```
|?- parent(X,Y),  parent(Y,Z).
      X=ann, Y=mary, Z = john.

|?- parent(X,_), parent(_,Z).
      X=ann, Z=john.

|?- grandparent(X,Y).
      X=ann, Y=john ;
      no

|?- parent(john,ann).
      no
```

Databases and Prolog

a database relation is a relation.

select:

```
empinfo(X,Y,unit23).
empinfo(X,Y,Z), Y<1995.
```

project:

```
year(Y) :- empinfo(_,Y,_).
year_unit(Y,Z) :- empinfo(_,Y,Z).
```

join:

```
empplus(W,X,Y,Z) :-
          empinfo(W,X,Y),
          manages(Y,Z).
```

Two-Clause Rules

```
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- parent(X,Z),  ancestor(Z,Y).


different(X,X) :- !,fail.
different(_,_).
```

The General Form of Prolog Rules

A rule's body can have any number of propositions.

P :- Q_1 , Q_2 , ... , Q_n .
head body

The comma is the $\wedge$ of logic.

The logic version of this is called a Horn clause:

$$Q_1 \wedge Q_2 \wedge \ldots \wedge Q_n \rightarrow P$$

Prolog also allows semicolons, meaning $\vee$

P :- Q_1 ; Q_2 ; ... ; Q_n .

... which is equivalent to the n-clause rule

P :- Q_1 .
P :- Q_2 .
...
P :- Q_n .

What about AND & OR on the Left?

To get the effect of $\wedge$ on the left, use

$$
\begin{array}{lll}
P_1 & :- & Q. \\
P_2 & :- & Q. \\
\ldots & & \\
P_n & :- & Q.
\end{array}
$$

But **the effect of $\vee$ on the left is *impossible*.**

That is, there are *no rules* corresponding to

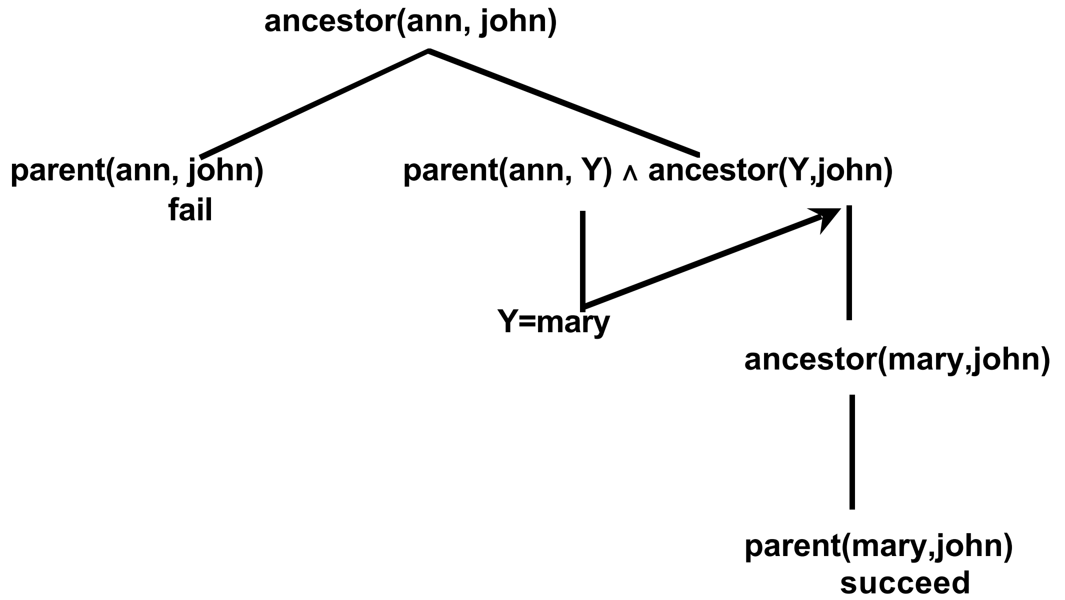
$$Q_1 \wedge Q_2 \wedge \ldots \wedge Q_n \rightarrow P_1 \vee P_2$$

and *no facts* of the form $P_1 \vee P_2$.

This is a *limitation* on Prolog and was a deliberate choice - for the sake of efficiency.

How Prolog Works

scope
unification
recursion




```
ancestor(X,Z) :- parent(X,Z).
ancestor(X,Z) :- parent(X,Y), ancestor(Y,Z).
```

Structured Objects in Prolog resemble structures or records.

Fact:

```
person(nm(john,smith),date(december,3,1995)).
```

Questions, Queries and Prolog responses:

English: "Who was born in 1995?"

Query:
```
|?- person(P,date(_,_,1995)).
```
Response:
```
     P = nm(john,smith)
```

English: "Who was born in what month of 1970?"

Query:
```
|?- person(nm(_,L),date(M,_,1970)).
```
Response:
```
     L = Jones, M = April
```

LISTS in Prolog

Notation

The *empty* list	`[ ]`
3-element list	`[a,b,c]`
List of lists	`[[],[a],[b,c],[d,e,f,g]]`
Head & Tail	`[H|T] = [a,b,c].`
	gives `H = a,` `T = [b,c]`

Membership (a 2-clause rule)

```
member(X,[X|Tail]).
member(X,[Head|Tail]) :- member(X,Tail).

|?- member(c,[a,b,c,d]).
      yes
```

Concatenation

```
append([],L,L).
append([X|L1],L2,[X|L3])    :- append(L1,L2,L3).


|?- append([a,b],[c,d],Result).

      Result = [a,b,c,d].


|?- append(First, [g,h], [d,e,f,g,h]).

      First = [d,e,f].
```

More Rules for Lists

Push	`push(X,L,[X|L]).`
Last element	`last(E,L) :- append(_,[E],L).`
Delete	`delete(X,[X|Tail],Tail).`

```
delete(X,[X|Tail],Tail).

delete(X,[Y|Tail],[Y|Tail1]):-
     delete(X,Tail,Tail1).
```

Even and Odd

```
evenlength([]).
evenlength([_|X]) :- oddlength(X).
oddlength([_|X]) :- evenlength(X).
```

Reverse and Palindrome

```
reverse( [ ] , [ ] ) .

reverse( [H | T ] , Answer)  :-      reverse(T , R),
                                     append( R, [H], Answer).


palindrome(X) :- reverse(X , X).
```