PROLOG (Programming in Logic)

Alain Colmerauer and Phillippe Roussel (Univ. of Aix-Marseille)
- natural language processing (1972)

Robert Kowalski (University of Edinburgh)
- automated theorem proving

Prolog system tries to determine whether the goal clause is satisfiable (true under the rule of logic), given the logic program (facts about relationship)

Prolog program consists of

- Fact Statements
- Rule Statements
- Goal Statements

Closed-world assumption

- true is true
- false means either false or 'don't know'

Applications

- Expert system – deduction
- Natural language processing
- Relational database

Note. Still grand experiment

Logic Programming Languages

Declarative language (Cf. procedural(imperative), functional)
based on symbolic logic, predicate calculus

Predicate Calculus

- Proposition
  logical statement that is true or false
  composed of objects and relationship among them

  Ex. man(jake) — 1 tuple
  like(jake,fishing) — 2 tuple

- Modes
  fact mode — man(jake) // constant
  query mode — man(X) // variable

- Compound proposition

		precedence
negation	$\neg a$	3
conjunction	$a \wedge b$	2
disjunction	$a \vee b$	2
equivalence	$a \equiv b$	2
implication	$a \supset b$	1
	$a \subset b$	1

Ex. $a \wedge \neg b \supset d \rightarrow (a \wedge (\neg b)) \supset d$

- Qualifiers

  Universal $\forall x.\ P$
  Existential $\exists x.\ P$

  Ex.
  $\forall X.$ (woman(X) $\supset$ human(X))
  $\exists X.$ (mother(mary,X) $\wedge$ male(X)) // Mary has a son

Clausal Form

All propositions can be expressed in clausal form

Ex. If all the A's are true, then at least one B is true.

$$\underbrace{(B_1 \cup B_2 \cup \cdots \cup B_n)}_{\text{consequent}} \subset \underbrace{(A_1 \cap A_2 \cap \cdots \cap A_m)}_{\text{antecedent}}$$

Predicate Calculus and Proving Theorems

Resolution

Alan Robinson (1965)

Inference rule that allows inferred propositions to be computed from given propositions

Ex

older(joanne, jake) $\subset$ mother(joanne, jake)
wiser(joanne, jake) $\subset$ older(joanne, jake)

---

wiser(joanne, jake) $\subset$ mother(joanne, jake)

Steps:

1. AND the left sides
2. AND the right sides
3. Remove common terms from both sidea

- Unification: determining useful values for variables
- Instantiation: temporary assigning of values to variables (to allow unification).

Note. If fail, need backtracking.

How to prove theorem using resolution?

Proof by contradiction.

1. hypothesis (original proposition)
2. goal (negated hypothesis)
3. Find inconsistency using resolution.

Horn Clauses

clause that contain at most one conclusion (left-side)

1. headed Horn clause: relationship

   human(jake) $\subset$ man(jake)

2. headless Horn clause: fact

   father(bob, jake)

Logic Program

set of Horn clause

Execution of a logic program begins when
it is given a goal clause.

Example 1.

fact statement (hypothesis)

```
male(david).
male(edward).
male(thomas).
male(stuart).
female(mary).
female(valerie).
female(loretta).
female(kathleen).
parent(david,thomas).
parent(david,stuart).
parent(edward,mary).
parent(edward,valerie).
parent(thomas,loretta).
parent(valerie,kathleen).
married(stuart,mary).
married(mary,stuart).
```

rule statement (condition)

```
sibling(X,Y) :- parent(Z,X), parent(Z,Y), (X\==Y).
in_law(X,Y) :- married(X,Z), sibling(Z,Y).
aunt(X,Y) :- female(X), sibling(X,Z), parent(Z,Y).
aunt(X,Y) :- female(X), in_law(X,Z),parent(Z,Y).
```

and

GNU Prolog System on Esus

Assume you saved your prolog program as *test.pl*

```
$ gprolog
 ------------------------------------------
 GNU Prolog 1.2.19
 By Daniel Diaz
 Copyright © 1979-2005 Daniel Diaz
 ------------------------------------------

?- [test].                    // Compile

 ------------------------------------------
  ---test.pl compiles, 21 lines read
         - 3367 bytes written, 10 ms
  yes
 ------------------------------------------

?- male(david).
 yes

?- parent(david, thomas).
 true

?- parent(david, X).
 X = thomas   ?              // ;   for next solution
                             // a   for all solutions
                             // RET to stop

?- a
 X = stuart
 yes

? halt.

$
```

```
?- in_law(stuart,valerie).
   true

?- in_law(thomas,mary).
   no.
```

We need another rule for in_law.

```
in_law(X,Y)  :-  married(Y,Z),sibling(Z,X).
```

Exercise.

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

father(X,Y) :-
mother(X,Y) :-

son(X,Y) :-
daughter(X,Y) :-

uncle(X,Y) :-

grandchild(X,Y) :-

grandson(X,Y) :-
                    or
```

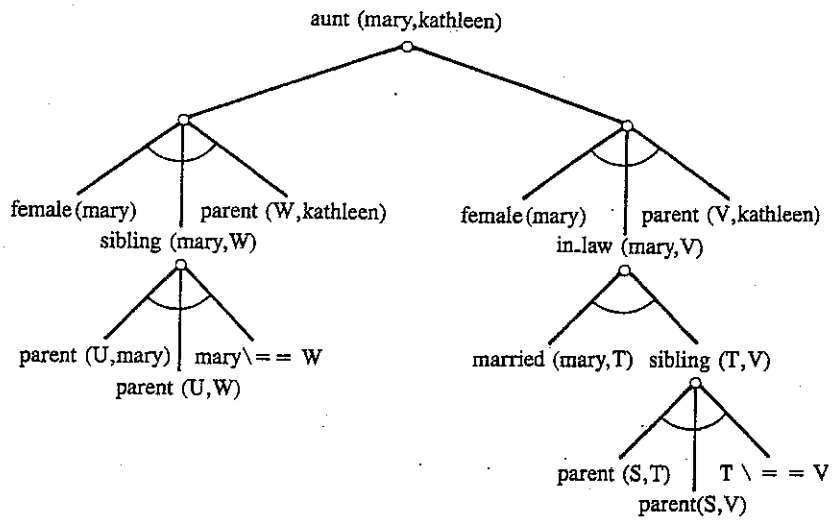

Figure A search tree corresponding to the goal aunt(mary,kathleen).

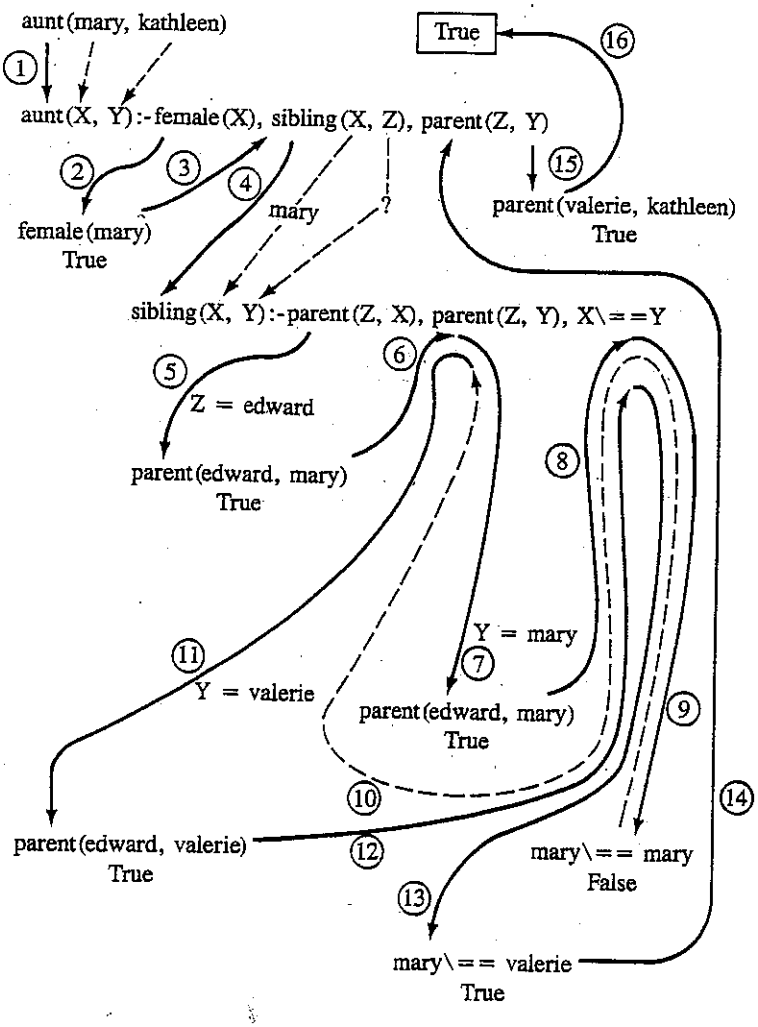

Figure Prolog's attempt to prove aunt(mary,kathleen).

Example 2. Relational Database

Supplier

S#	Name	Status	City	
S1	Smith	20	London	supplier(s1, smith, 20, london).
S2	Jones	10	Paris	supplier(s2, jones, 10, paris).
S3	Blake	30	Paris	supplier(s3, blake, 30, paris).
S4	Clark	20	London	supplier(s4, clarke, 20, london).
S5	Adams	30	Athens	supplier(s5, adams, 30, athenes).

- **Suppliers in London**

```
?- supplier(A,B,C,D), D = london.

   A = s1
   B = smith
   C = 20
   D = london ?          ;

   A = s4
   B = clark
   C = 20
   D = london            ;

   no
```

- **Suppliers in Paris whose status is greater than 20**

```
?- supplier(A,B,C,D), D = paris, C > 20.

   A = s3
   B = blake
   C = 30
   D = paris
```

- **Suppliers in London or Paris, whose state >= 20**

```
?- supplier(A,B,C,D), (D = london; D = paris), C >= 20.

A = s1
B = smith
C = 20
D = london          ;

A = s3
B = blake
C = 30
D = paris           ;

A = s4
B = adams
C = 20
D = london
```

- **Suppliers who does not live in London,** and **status > 20**

```
?- supplier(A,B,C,D), D \= london, C > 20).

A = s3
B = blake
C = 30
D = paris           ;

A = s5
B = adams
C = 30
D = Athens
```

Example 3. Hunter's Word Problem

The following is a word problem that Hunter gave as an exercise in CS 355 in the past.

Hunter, Laura, Jim, Sally, and Jack work in the same building with five adjacent offices. Hunter does not work in the 5th office and Laura does not work in the first office. Jim does not work in the first or last office. Jack's office is adjacent to Laura's. Sally works in the 5th office. Laura works in some office lower than Hunter's. Who works in what offices?

Try solving this manually yourself first. The Prolog solution is:

```
adjacent(X,Y) :- X =:= Y + 1.
adjacent(X,Y) :- X =:= Y - 1.

offices([office(_, 1), office(_, 2), office(_, 3),
         office(_, 4), office(_, 5)]).

layout(Floorplan) :- offices(Floorplan),
       member(office(hunter, H), Floorplan),
       member(office(laura,  L), Floorplan),
       member(office(jim,    J), Floorplan),
       member(office(sally,  S), Floorplan),
       member(office(jack,  JA), Floorplan),
       H =\= 5,
       L =\= 1,
       J =\= 1, J =\= 5,
       adjacent(JA, L),
       S =:= 5,
       L < H.
```

When we state layout(X), Prolog finds a solution for the variable X which satisfies all of these conditions and it responds:

X = [office(jack,1),office(laura,2),office(hunter,3),office(jim,4),office(sally,5)] ;

X = [office(jack,1),office(laura,2),office(jim,3),office(hunter,4),office(sally,5)] ;

no

EX 4. **Fibonacci Number**

F0 = 1, F1 = 1, Fn = F n-1 + F n-2, n>1

```
fib(0,1).
fib(1,1).
fib(N,F) :- N = M + 1, M = K + 1, fib(M,G), Fib(K,H),
            F = G + H, N > 1.
```

Top-down control (recursion)

```
:- fib(2,F)
   N = 2
:- 2 = M + 1, M = K + 1, fib(M,G), fib(K,H), F=G + H, 2 > 1.
   M = 1
:- 1 = K + 1, fib(1,G), fib(K,H), F = G + H.
   K = 0
:- fib(1,G), fib(0,H), F = G + H.
   G = 1 and H = 1
:- F = 1 + 1.
   F = 2
```

Note.

(1)
```
:- fib(N, 3).
   N = 3
```

(2)
```
:- fib(N, F).
   N = 0, F = 1 ;
   N = 1, F = 1 ;
   N = 2, F = 2 ;
   N = 3, F = 3 ;
        :
```

input-output parameters are not distinguished

(3) **for N + Fn = 13,**

```
:- fib (N, F), N + F = 13.
   N =5, F = 8
```