Prolog II

- Prolog
- Facts (only head)
  - mammal(human) <-
- Query
  - <- mammal(x), legs(x,y)
  - Horn clause without a head

Resolution and Unification (how queries are expressed)

- a <- a_1 ..... a_n
- b <- b_1 ..... b_m
- If bi matches a then we can infer the clause:
- b <- b_1, ..., b_{i-1}, a_1, ... , a_n, b_{i+1} .... , b_m.
- Another view: combine left hand /right hand cancel
- b <- a. and c<-b. b,c <- a,b gives c<-a

Resolution

- Goal or list of goals is a Horn clause without a head
- Match one of the goals with the head of known clause
- Simplest case
  - mammal(human). <- (fact)
  - <-mammal(human). (query)
  - mammal(human) <- mammal(human)
  - <- (query is proved)

Unification

- Making two terms “the same”
- me = me
  - yes
- me = you
  - no
- me = X.
  - X = me
- f(a,X) = f(Y,b).
  - X =b Y=a

Computation

- Goal: is a a list of goal as a Horn clause without head
- Attempt to apply resolution by matching one of the goal with head of known clause
- Then replace with body, new list of goals
- Repeat until elimination of all goals (proved)

An example

```
Facts and rules:
legs(x,2) <- mammal(x), arms(x,2).
legs(x,4) <- mammal(x), arms(x,0).
mammal(horse)<-.
arms(horse,0)<-.
```

```
Query:
    <- legs(horse,4).
```

```
Resolution:
   legs(x,4) <- mammal(x), arms(x,0), legs(horse,4).
Unification:
   legs(horse,4) <- mammal(horse), arms(horse,0), legs(horse,4)
                 <- mammal(horse), arms(horse,0).
Resolution
   mammal(horse) <- mammal(horse), arms(horse,0).
                 <- arms(horse,0).
   arms(horse,0)   <-  arms(horse,0).
                 <-                         Initial query is true
```

Arithmetic

- write(3+). evaluates to 3+5
- X is 3+5, write(X) X = 8

Gcd in Prolog:

```
gcd(U,O,U).
gcd(U,V,W) :- R is U mod V, gcd(V,R,W).
```

Lists

- [a, b, c]
- [a,b,c] can also be written [a,b,c | []] or [a, b | [c]] or [a | [b, c]]
- [H|T] = [a,b,c]
  - H = a, T = [b,c]
- [a|T] = [H,b,c]
  - T = [b,c], H = a
- [H,T] is syntactic sugar for .(H,T) (. is cons)

Actual code examples

- ancestor
- links
- append

Important

Queries are yes/fail rather than yes/no
No means the system can not prove it, not that is necessarily false

Prolog:
Order of clauses top-to-bottom
Order of goals left-to-right

Always depth-first search

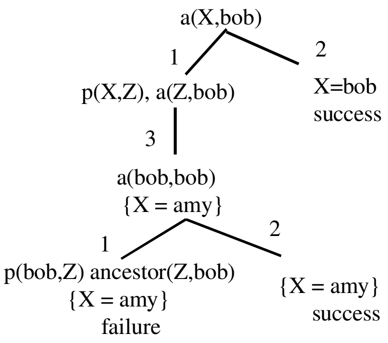
Prolog Search Tree

1 a(X,Y) :- p(X,Z), a(Z,Y).
2 a(X,X).
3 p(amy,bob).

Depth-first search strategy

Problem:
a(X,Y) :- a(Z,Y), p(X,Z)
goes into an infinite loop

ORDER MATTERS

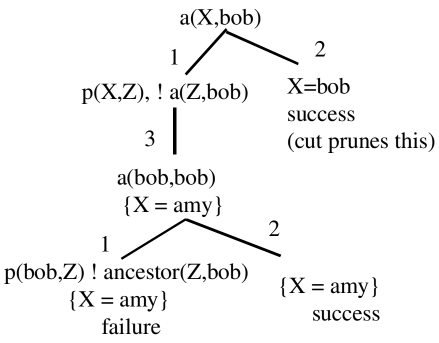


Cuts

1 a(X,Y) :- p(X,Z), !, a(Z,Y).
2 a(X,X).
3 p(amy,bob).

Cut "freezes" the choice made, if it is reached on backtracking, the subtrees of parent node are not examined. Cut "prunes" the search tree.

Can be used for efficiency


a(X,bob)
1
2
p(X,Z), ! a(Z,bob)
X=bob
success
(cut prunes this)
3
a(bob,bob)
{X = amy}
1
2
p(bob,Z) ! ancestor(Z,bob)
{X = amy}
failure
{X = amy}
success