Prolog -2

- Negation by failure
- Lists
  - Unifying lists
- Functions on lists
  - Member_of()
  - Don't care variables
- Prolog search trees + traces

Review of Prolog

- Language constructs
  - facts, rules, queries
- Logic programming — Prolog
  Non-deterministic — rule order/subgoal order
- Separation of logic (what) from control (how)
- Horn clauses
  - How variables are bound to values
- Goal-oriented semantics
- Recursive rules
- Negation as failure

Negation by Failure, revisited

```
not(X) :- X, !, fail.
not( _ ) .
```

if X succeeds in first rule, then the goal fails because of the last term.

if we type ";" the cut (!) will prevent us from backtracking over it or trying the second rule so there is no way to undue the fail.

if X fails in the first rule, then the goal fails because subgoal X fails. the system tries the second rule which succeeds, since "_" unifies with anything.

Negation by Failure

- **Not equivalent to logical *not* in Prolog**
  - **Prolog can only assert that something is true**
  - **Prolog cannot assert that something is false, but only that it cannot be proven with the given rules**

Prolog Syntax

- **Names come from first order logic**

  a(X,Y) :- b (c(Y)), integer(X).

  – Predicates are evaluated

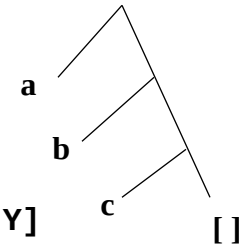
  – Functors with terms are unified

  – Call this a clause or rule

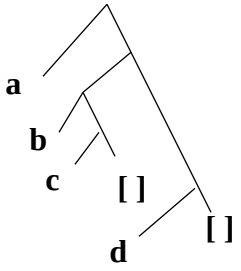
  cf. "Computing with Logic", D. Warren and D. Meyers

Lists

list	head	tail
`[a,b,c]`	`a`	`[b,c]`
`[X,[cat],Y]`	`X`	`[[cat],Y]`
`[a,[b, c],d]`	`a`	`[[b,c],d]`
`[X | Y]`	`X`	`Y`

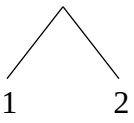


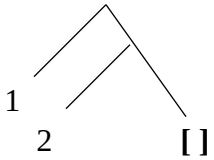
a list consists of a sequence of terms



Unifying Lists

```
[X,Y,Z] = [john, likes, fish]
  X = john, Y = likes, Z = fish
[cat] = [X | Y]
  X = cat, Y = [ ]
[ [the, Y] | Z] = [ [X, hare] | [ is here] ]
  X = the, Y = hare, Z = [ is here]
[1 | 2]          versus [1, 2]
```

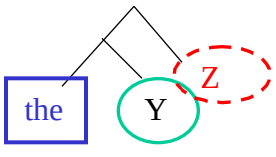


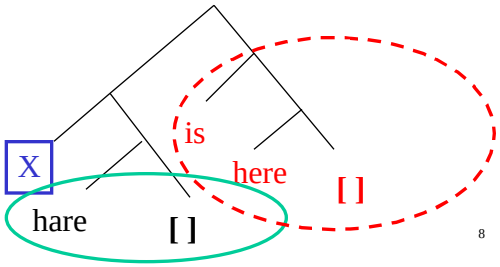


Lists

- **Sequence of elements separated by commas, or**
- **[first element | rest_of_list]**
  - **sort of "[car(list) | cdr(list)]" notation**

```
[ [the | Y] |  Z] = [ [X, hare] | [ is, here] ]
```

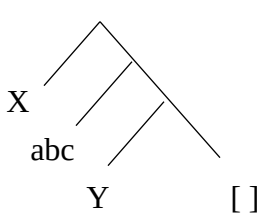


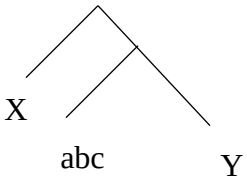


Lists

`[X, abc, Y] =? [ X, abc | Y]`

there is no value binding for Y, to make these two trees isomorphic.





Lists

don't care variable unifies with anything

`[a,b | Z] =? [ X | Y]`

`X = a, Y = [b | Z], Z = _`

look at the trees to see why this works!

`[ a, b, c] = [ X | Y]`

`X = a, Y = [b,c]  ;`

`no`

It's better to stick to head and tail form of Prolog lists -- it's less confusing!

Member_of Function

```
member(A, [A|B]).
member(A, [B|C]) :- member (A,C).
```

goal-oriented semantics: **can get value assignment for goal member(A,[B|C]) by showing truth of subgoal member(A,C) and retaining value bindings of the variables**

procedural semantics: **think of head of clause as procedure entry, terms as parameters. then body consists of calls within this procedure to do the calculation. variables bindings are like "returned values".**

Example

```
?- member(a,[a,b]).
   yes
?- member(a,[b,c]).
   no
?- member(X,[a,b,c]).
   X = a  ;
   X = b ;
   X = c ;
   no
```

Invertability of Prolog arguments

1. member(A, [A | B]).
2. member(A, [B | C]) :- member (A, C).

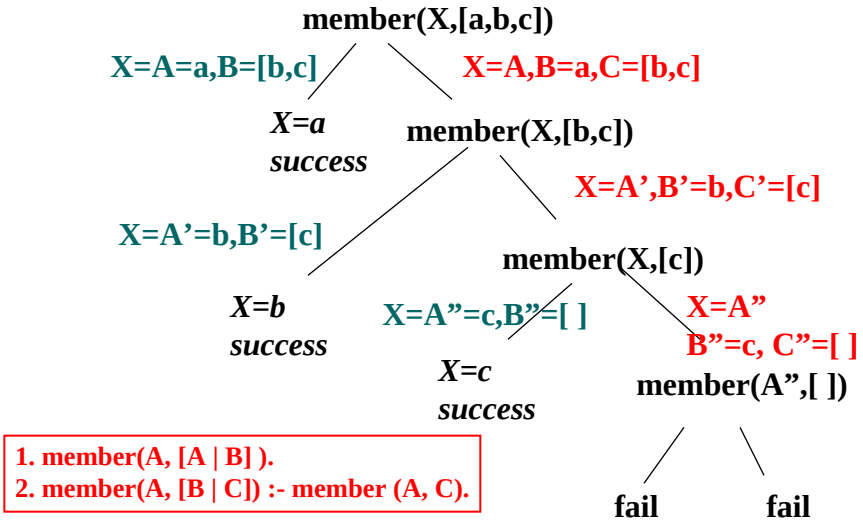
Example

```
?- member(a,[b, c, X]).
   X= a  ;
   no
?- member(X,Y).
   X = _123
   Y = [ X | _124] ) ;
   X = _123
   Y = [_125, X | _126]  ;
   X = _123
   Y =  [ _127, _128, X |_129]
```

1. member(A, [A | B]).
2. member(A, [B | C]) :- member (A, C).

Lazy evaluation of unbounded list structure. Unbound X as first element, second element, third element, etc.

Prolog Search Tree

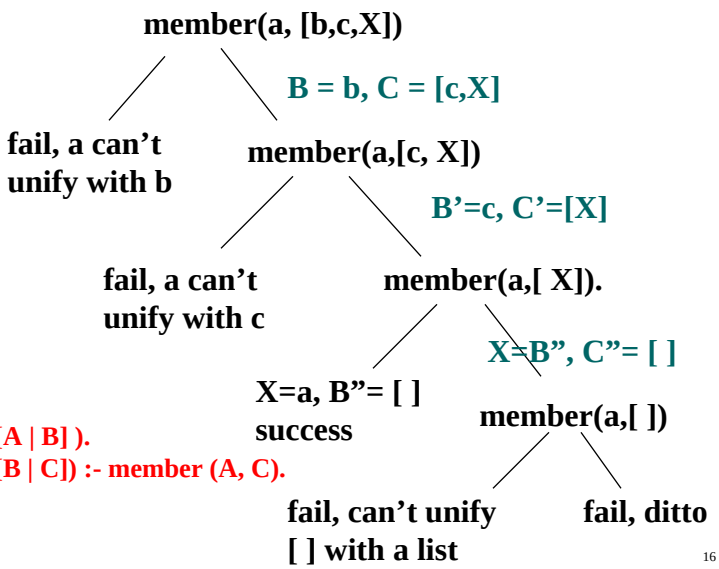


1. member(A, [A | B]).
2. member(A, [B | C]) :- member (A, C).

1. member(A, [A | B]).
2. member(A, [B | C]) :- member (A, C).

?- member(X, [a,b,c]).
match rule 1. member(A, [A | B]) so X = A = a, B = [b,c]
X = a ;
match rule 2. member(A, [B | C]) so X = A, B = a, C = [b,c]
then evaluate subgoal member(X, [b,c])
match rule 1. member(A',[A' | B']) so X = b, B' = [c]
X = b ;
match rule 2. member(A',[B' | C']) so X = A', B' = b, C' = [c]
then evaluate subgoal member(X, [c])
match rule 1. member(A",[A" | B"]) so X=A"= c, B"=[ ]
X = c ;
match rule 2. member(A",[B" | C"]) so X=A", B"=c, C"=[ ], but member(X, [ ]) is unsatisfiable, *no*

Another Search Tree



1. member(A, [A | B]).
2. member(A, [B | C]) :- member (A, C).

Prolog Search Trees

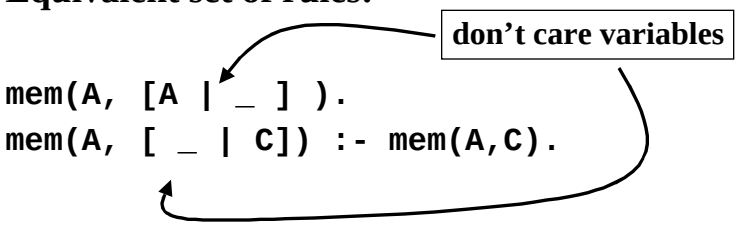
- **Really have built an evaluation tree for the query `member(X,[a,b,c]).`**
- **Search trees provide a formalism to consider all possible computation paths**
- **Leaves are success nodes or failures where computation can proceed no further**
- **By convention, to model Prolog, leftmost subgoal is tried first**

Prolog Search Trees, cont.

- **Label edges with variable bindings that occur by unification**
- **There can be more than one success node**
- **There can be infinite branches in the tree, representing non-terminating computations (performed lazily by Prolog); lazy evaluation implies only generate a node when needed.**

Another Member_of Function

Equivalent set of rules:



Can examine search tree and see the variables which have been excised were auxiliary variables in the clauses.

Tracing in Quintus Prolog

?-[mem]. %loads file mem.pl
?-trace. %turns on tracing facility
[trace]
| ?- memberof(X,[a,b,c]).
(1) 0 Call: memberof(_6783,[a,b,c]) ?
(1) 1 Head [1->2]: memberof(_6783,[a,b,c]) ?
(1) 0 Exit: memberof(a,[a,b,c]) ?
X = a ;
(1) 0 Redo: memberof(a,[a,b,c]) ?
(1) 1 Head [2]: memberof(_6783,[a,b,c]) ?
(2) 1 Call: memberof(_6783,[b,c]) ?
(2) 2 Head [1->2]: memberof(_6783,[b,c]) ?
(2) 1 Exit: memberof(b,[b,c]) ?
(1) 0 Exit: memberof(b,[a,b,c]) ?
X = b ;

1. memberof(A, [A | B]).
2. memberof(A, [B | C]) :- memberof (A, C).

Matches memberof(a,[a | b,c]) with rule #1.

Asked for another answer, tries using rule #2. Tries memberof(X,[b,c]) and matches memberof(b,[b | c]) with rule #1.

Tracing in Quintus Prolog

(1) 0 Redo: memberof(b,[a,b,c]) ?
(2) 1 Redo: memberof(b,[b,c]) ?
(2) 2 Head [2]: memberof(_6783,[b,c]) ?
(3) 2 Call: memberof(_6783,[c]) ?
(3) 3 Head [1->2]: memberof(_6783,[c]) ?
(3) 2 Exit: memberof(c,[c]) ?
(2) 1 Exit: memberof(c,[b,c]) ?
(1) 0 Exit: memberof(c,[a,b,c]) ?
X = c ;
(1) 0 Redo: memberof(c,[a,b,c]) ?
(2) 1 Redo: memberof(c,[b,c]) ?
(3) 2 Redo: memberof(c,[c]) ?
(3) 3 Head [2]: memberof(_6783,[c]) ?
(4) 3 Call: memberof(_6783,[]) ?
(4) 4 Head [1->2]: memberof(_6783,[]) ?

Asked for another answer, tries memberof(X,[b,c]) using rule #2. Tries memberof(X,[c]) and matches memberof(c,[c]) by rule #1.

(4) 4 Head [2]: memberof(_6783,[]) ?
(4) 3 Fail: memberof(_6783,[]) ?
(3) 2 Fail: memberof(_6783,[c]) ?
(2) 1 Fail: memberof(_6783,[b,c]) ?
(1) 0 Fail: memberof(_6783,[a,b,c]) ?

no

Finds no more choices to try

1. member(A, [A | B]).
2. member(A, [B | C]) :- member (A, C).

Tracing in Quintus Prolog