Logic Programing

• Goal:
  – Tell computer the facts
  – Ask a question
  – Computer uses facts as needed to answer the question.
  – “logic + control = algorithm”

• Reality:
  – This is very hard
    – Sorted (x, y)<= permutation(x, y) ^ inorder(y)
Prolog

• Fallback goal
  – Separate logic from control
    – Control implicit in way logic formulated
      - Order of clauses
      - Order of subgoals in clause
      - Overall formulation
  – Additional explicit notation for control
    - We’ll see an example later this lecture

Order of clauses

\[
\text{path0}(A, B) :- \text{path0}(A, C), \text{arc}(C, B).
\]
\[
\text{path0}(A, B) :- \text{arc}(A, B).
\]

versus

\[
\text{path1}(A, B) :- \text{arc}(A, B).
\]
\[
\text{path1}(A, B) :- \text{path1}(A, C), \text{arc}(C, B).
\]

\[
\text{path0}(x_1, x_6).
\]

Graph showing the connection between nodes x1, x2, x3, x4, x5, and x6.
Order of Subgoals

\[
\text{path1}(A, B) :- \ \text{arc}(A, B) . \\
\text{path1}(A, B) :- \ \text{path1}(A, C), \ \text{arc}(C, B) . \\
\text{versus} \\
\text{path2}(A, B) :- \ \text{arc}(A, B) . \\
\text{path2}(A, B) :- \ \text{arc}(C, B), \ \text{path2}(A, C) . \\
\]

Path1(X, x6).

Overall formulation

\[
\text{path2}(A, B) :- \ \text{arc}(A, B) . \\
\text{path2}(A, B) :- \ \text{arc}(C, B), \ \text{path2}(A, C) . \\
\text{versus} \\
\text{path3}(A, B) :- \ \text{arc}(A, B) . \\
\text{path3}(A, B) :- \ \text{arc}(A, C), \ \text{path3}(C, B) \\
\]
Additional explicit control

- Cut can be used to suppress redundant solutions

\[
a(X) :- b(X), !, c(X).
\]

! Always succeeds, but if you fail back to it, then goal that matched head of clause fails
- Ignore other ways for \( b(X) \) to succeed
- Ignore other clauses for \( a(X) \).

Cut

\[
\begin{align*}
  \text{foo} & :- a. \\
  \text{foo} & :- b. \\
  a & :- c1, !, c3. \\
  a & :- d. \\
  b & :- write('b'), nl. \\
  c1 & :- write('c1 first'), nl. \\
  c1 & :- write('c1 second'), nl. \\
  c3 & :- write('c2 fail'), nl, fail. \\
  d & :- write('d'), nl.
\end{align*}
\]

\[
\begin{align*}
  \text{foon} & :- a. \\
  \text{foon} & :- b. \\
  \text{an} & :- c1, c3. \\
  \text{an} & :- d. \\
  \text{fie} & :- a. \\
  \text{fie} & :- b, a.
\end{align*}
\]
Pruning redundant solutions

```
isa-mother(X) :- female(X),parent(X,_),!.
isa-father(X) :- male(X),parent(X,_),!.
parent(fred,sue).
parent(janet,sue).
parent(fred,tim).
parent(janet,tim).
male(fred).female(janet).
?- isa-mother(X).X = janet; no
```

Negation by failure

- The goal \(+\)(G) succeeds whenever the goal G fails.
- | ?- member(b,[a,b,c]).
  - yes
  - | ?- \(+\)(member(b,[a,b,c])).
  - no
Disjoint sets

overlap(S1,S2) :- member(X,S1),member(X,S2).
disjoint(S1,S2) :- \+(overlap(S1,S2)).

| ?- overlap([a,b,c],[c,d,e]).
| yes
| ?- overlap([a,b,c],[d,e,f]).
| no
| ?- disjoint([a,b,c],[c,d,e]).
| no
| ?- disjoint([a,b,c],[d,e,f]).
| yes
| ?- disjoint([a,b,c],X).
| no %<---------Not what we wanted??????

Negation by failure

• Negation by failure \+(G) works properly only in the following cases:
  – When G is fully instantiated at the timePROLOG processes the goal \+(G).
    – In this case, we interpret \+(G) to mean “goal G does not succeed”.
  – When all variables in G are unique to G, i.e., they don't appear elsewhere in the same clause.
    – In this case, we interpret \+(G(X)) to mean “There is no value of X that will make G(X) succeed”.

Iterative deepening

idpath(A, B, Bound):- pathb(A, B, Bound).

idpath(A, B, Bound):-
    Higher is Bound + 1,
    print('trying bound '),
    print(Higher),
    nl,
    idpath(A, B, Higher).

idpath(A, B):- idpath(A, B, 0).