XSB

Tim Finin
University of Maryland
Baltimore County

2/19/02

Overview

- What XSB offers
- Tabling
- Higher order logic programming
  - sort of
- Negation

UMBC
an Honors University in Maryland

What XSB Offers

- LP languages have been based on SLDNF (SLD resolution with negation as failure) and implemented with the WAM (Warren Abstract Machine) model.
- These suffer from some drawbacks: they address either positive or negative loops.
- **Positive loops** result from recursion without negation.
  - It has been addressed with magic sets and tabling, which assign failing values to derivation paths which contain positive loops.
- **Negative loops** result from recursion through negation.
  - derivations are assigned the value undefined by well founded semantics.
- XSB attempts to handle both and do so with efficiency approaching that of standard SLDNF as implemented, for example, by Prolog.

UMBC
an Honors University in Maryland

SLD-resolution rule

```
<- A1,..,A(i-1),Ai,A(I+1),...,Am   B0 <- B1,...,Bn
---------------------------------------------------
<- (A1,...,A(i-1),B1,...,Bn,A(i+1),...,Am)σ
```

where

- `A1,...,Am` are atomic formulas (goals)
- `B0 <- B1,…,Bn` is a (renamed) definite clause in *P*
- `mgu(Ai,B0)=σ`

UMBC
an Honors University in Maryland

Goal and clause selection

A *goal selection function* specifies which goal `Ai` is selected by the SLD-rule.

☐Prolog goes left-to-right

The order in which clauses are chosen is determined with a *clause selection rule*.

- Prolog selects clauses in the order in which they are added to the database

UMBC
an Honors University in Maryland

SLD-resolution is Sound and Complete for Horn Clauses

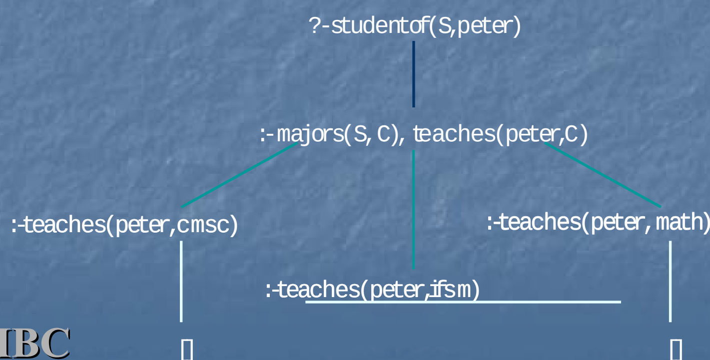
- Any query (goal) that is provable with SLD-resolution is a logical consequence of the program.
- Any query (goal) that is (true) in the least Herbrand model is provable with SLD-resolution.

  In the case of an infinite SLD-tree, the selection function has to be fair (as in breadth first search). For finite SLD-trees left-first-with-backtracking as used in Prolog gives a complete method.

UMBC
an Honors University in Maryland

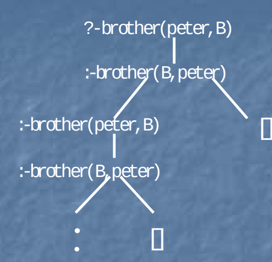
SLD-tree

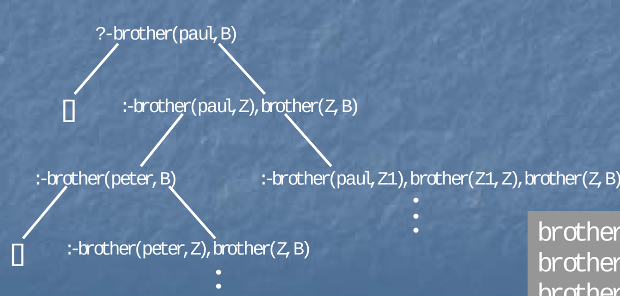
```
studentof(X,T):- majors(X,C), teaches(T,C).
majors(paul, cmsc).
majors(paul, ifsm).
majors( maria, math).
teaches(adrian, ifsm).
teaches(peter, math).
teaches(peter, cmsc).
```



Infinite SLD-trees

```
brother(X,Y) :- brother(Y,X).
brother(paul, peter).
```






```
brother(paul,peter).
brother(peter,adrian).
brother(X,Y):-
  brother(X,Z), brother(Z,Y).
```

Positive Loops in Prolog

We might like to use rules with loops:

```
% Logically: parent(X,Y) ⇔ child(X,Y).
parent(X,Y) :- child(X,Y).
child(X,Y) :- parent(X,Y).
parent(adam,able).
child(cain,eve).
% Logically: spouse(X,Y) ⇔ spouse(Y,X).
spouse(X,Y) :- spouse(Y,X).
spouse(adam,eve)
```


Positive Loops in Prolog

Sometimes we are forced to…

```
% Loops because there are cycles in the owes
  graph
avoids(Source,Target) :- owes(Source,Target).
avoids(Source,Target) :-
        owes(Source,Intermediate),
        avoids(Intermediate,Target).
owes(andy,bill).
owes(bill,carl).
owes(carl,bill).
```

Non-looping Prolog version

```
avoids(X,Y) :- avoids(X,Y,[ ]).
avoids(X,Y,L) :- owes(X,Y), \+ member(Y,L).
avoids(X,Y,L) :-
  owes(X,Z),
  \+ member(Z,L),
  avoids(Z,Y,[Z|L]).
owes(andy,bill).
owes(bill,carl).
owes(carl,bill).
```


More problems

Even if we prevent loping or the graph contains no cycles, its structure may lead to exponential computations.

Non-looping XSB version

```
:- table avoids/2.
avoids(Source,Target) :- owes(Source,Target).
avoids(Source,Target) :-
    owes(Source,Intermediate),
    avoids(Intermediate,Target).
owes(andy,bill).
owes(bill,carl).
owes(carl,bill).
```

UMBC
an Honors University in Maryland

SLD resolution: Program Clause Resolution

Given a tree with a node labeled

A:-A1,A2…An

and a rule in the program of the form

H :- B1,B2…Bk

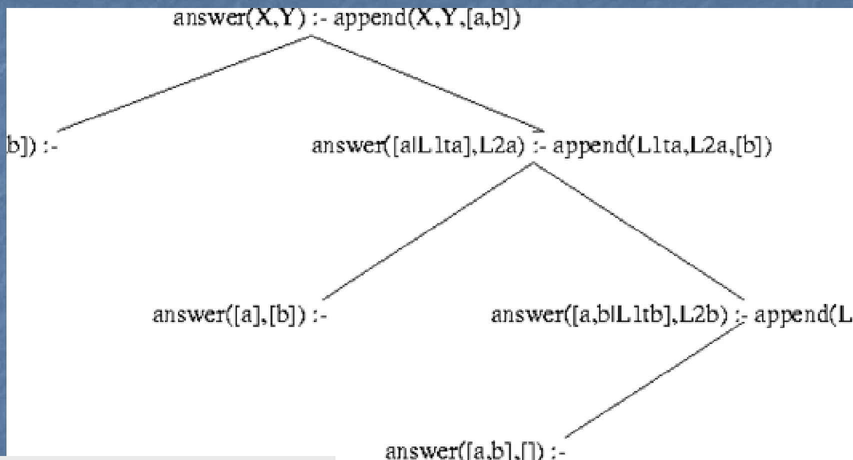
and given that H and B1 match with matching variable assignment Theta, then add a new node as a child of this one and label it with

(A :- B1,…,Bk,A2…An)Theta

if it does not already have a child so labeled. Note that the matching variable assignment is applied to all the goals in the new label.

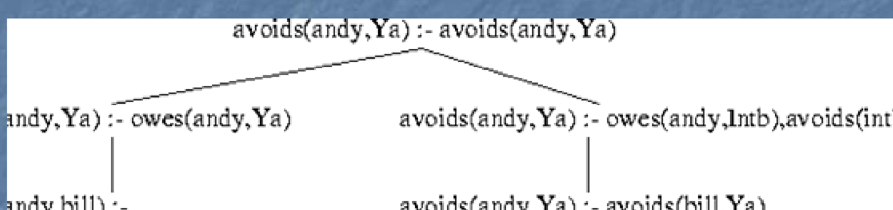
UMBC
an Honors University in Maryland

SLD tree for the query: append(X,Y,[a,b])

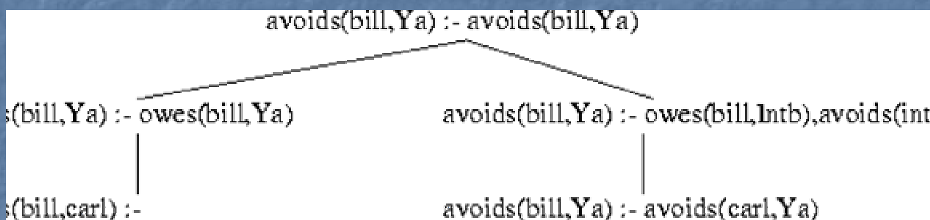



```
append([],L2,L2).
  append([X|L1t],L2,[X|L3t]) :-
     append(L1t,L2,L3t).
```


Tree for avoid(andy,Ya) goal server



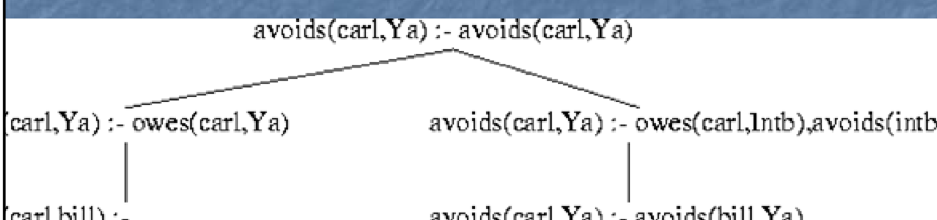
Tree for avoid(bill,Ya) goal server



UMBC

an Honors University in Maryland

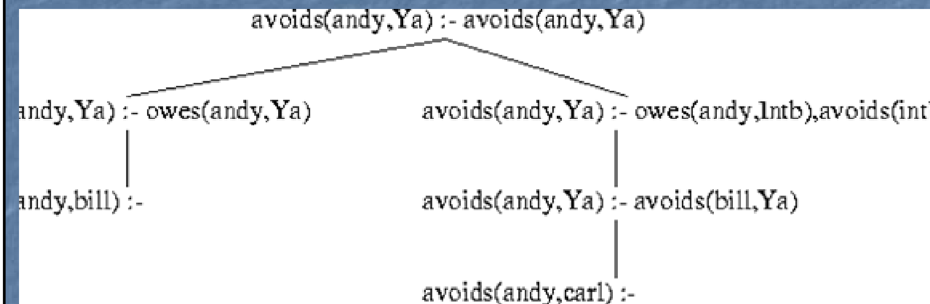
Tree for avoids(carl,Ya) goal server



UMBC

an Honors University in Maryland

Updated tree for avoid(andy,Ya) goal server



Updated tree for avoid(andy,Ya) goal server

SLG resolution rules

- **Program Clause Resolution:** Given (1) a tree with a node labeled A:-A1,A2…An, which is either a server tree root node or A1 is not tabled and (2) a rule H:-B1,B2…Bk where H and A1 match with substitution Theta, then add a new child node with label (A:- B1,…,Bk,A2…An)Theta, if it does not already have a child so labeled.
- **Subgoal Call:** Given a nonroot node with label, where A1 is indicated as tabled, and there is no tree with root A1 :- A1, create a new tree with root A1 :- A1.
- **Answer Clause Resolution** Given a non-root node with label A:-A1, A2…An, and an answer of the form B :- in the tree for A1, then add a new child node labeled by (A :- A2…An) Theta , where Theta is the substitution obtained from matching B and A1 (if there is not already a child with that label.)

Left recursive version

The left-recursive version is, in fact, more efficient in XSB.

```
:- table avoids/2.
avoids(Source,Target) :- owes(Source,Target).
avoids(Source,Target) :-
  avoids(Source,Intermediate),
  owes(Intermediate,Target).
```

Only one table is generated by the query avoids(andy,Ya) instead of three.

Hilog terms

- XSB has a simple extension that provides some "higher-order" logic programming capability.
- A term in XSB is:
  - an atomic symbol, or
  - a variable, or
  - of the form: t0(t1,t2,...,tn) where the ti are terms.
- Transitive closure example:

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


Some examples

```
map(F)([],[]).
map(F)([X|Xs],[Y|Ys]) :-
   F(X,Y),
   map(F)(Xs,Ys).
twice(F)(X,R) :-
   F(X,U),
   F(U,R).

:- hilog successor,double,square.

successor(X,Y) :- Y is X+1.
double(X,Y) :- Y is X+X.
square(X,Y) :- Y is X*X.
```

```
| ?- map(successor)([2,4,6,8],L).
L = [3,5,7,9]
| ?- map(double)([2,4,6,8,10],L).
L = [4,8,12,16,20];
| ?- twice(successor)(1,X).
X = 3
| ?- twice(twice(successor))(1,X).
X = 5
| ?- twice(twice(square))(2,X).
X = 65536
| ?- twice(twice(twice(double)))(1,X).
X = 256
```

How it's done

If T is a variable or compound term or a hilog term, rewrite:

T(A1…An) → apply(T,A1,…An)

So the clauses:

```
closure(R)(X,Y) :- R(X,Y).
closure(R)(X,Y) :- R(X,Z), closure(R)(Z,Y).
ancestor(X,Y) :- closure(parent)(X,Y).
parent(adam,cain). parent(cain,enoch)
```

Become:

```
apply(closure(R),X,Y) :- apply(R,X,Y).
apply(closure(R),X,Y) :- apply(R,X,Z), apply(closure(R),Z,Y).
ancestor(X,Y) :- apply(parent,X,Y).
apply(parent,adam,cain). apply(parent,cain,enoch).
```


Negation in XSB

- \+ +P, fail_if(+P), not(+P) are all like Prolog's \+ +P :

  not(P) :- call(P), !, fail

  not(_).
- tnot(P) is xsb's negation operator and allows for the correct execution of programs with well founded semantics.
- **P must be a tabled predicate.**

Negation

```
bachelor(X) :-
   male(X),
   \+ married(X).
male(bill).
male(jim).
married(bill).
married(mary).
```

```
?- bachelor(bill).
no
| ?- bachelor(jim).
yes
| ?- bachelor(mary).
no
| ?- bachelor(X).
X = jim;
No
```

UMBC
an Honors University in Maryland

Floundering goals

- Prolog treats \+ as *negation as failure* which is sound if we make the *closed world assumption*
- To guarantee reasonable answers, the negation operator should be applied only to **ground literals**.
- If it is applied to a nonground literal, the program is said to **flounder**.
- Some LP languages define not like

  not(P) :- ground(P) -> (\+ P) | error("…").

UMBC
an Honors University in Maryland

Floundering goal

```
bachelor(X) :-
   \+ married(X),
  male(X),
male(bill).
male(jim).
married(bill).
married(mary).
```

```
?- bachelor(bill).
no
| ?- bachelor(jim).
yes
| ?- bachelor(mary).
no
| ?- bachelor(X).
no
```


Stratified Negation

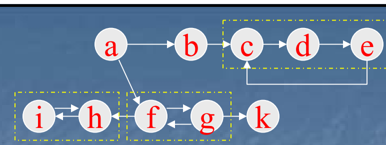
- A program is **stratified** if there are no cycles in the call graph which contain a call to P and to not(P).

  i.e., **no recursion through negation**
- XSB allows non-stratified programs to be evaluated.

```
% A set is normal if it doesn't
% contain itself
normal(S) :- \+ in(S,S)
% mySet is the set of all sets
% that are normal.
in(S,mySet) :- normal(S).
% is mySet normal?
?- normal(mySet)
% this leads to Russell's paradox
% and will cause Prolog  to loop
```

normal $\xleftarrow{-}$ in

Stratified Negation



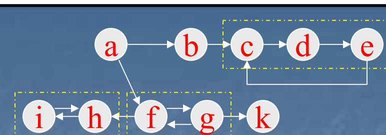
- Suppose we have some kind of reduction operator and want to apply it to an object and want to repeatedly apply it until it can't be reduced any further.
- If there are cycles, treat the objects in the cycle (the strongly connected components or SCCs) as equivalently reduced.

```
:- table reachable/2.
reachable(X,Y) :- reduce(X,Y).
reachable(X,Y) :-
  reachable(X,Z),
  reduce(Z,Y).
```

```
reduce(a,b).
reduce(b,c).
reduce(c,d).
reduce(d,e).
reduce(e,c).
reduce(a,f).
reduce(f,g).
reduce(g,f).
reduce(g,k).
reduce(f,h).
reduce(h,i).
reduce(i,h).
```

UMBC
an Honors University in Maryland

Stratified Negation



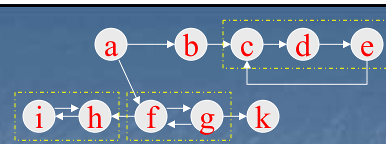
- A node is reducible if it can be further reduced.
- A node X can be fully reduced to another Y if X can be reduced to Y and Y is not further reducible.

```
:- table reducible/1.
reducible(X) :-
  reachable(X,Y),
  tnot(reachable(Y,X)).
fullyReduce(X,Y) :-
  reachable(X,Y),
  tnot(reducible(Y)).
```

```
reduce(a,b).
reduce(b,c).
reduce(c,d).
reduce(d,e).
reduce(e,c).
reduce(a,f).
reduce(f,g).
reduce(g,f).
reduce(g,k).
reduce(f,h).
reduce(h,i).
reduce(i,h).
```

UMBC
an Honors University in Maryland

Stratified Negation



- We might want to return a representative object from a SCC.
- We'll pick (arbitrarily) the smallest.

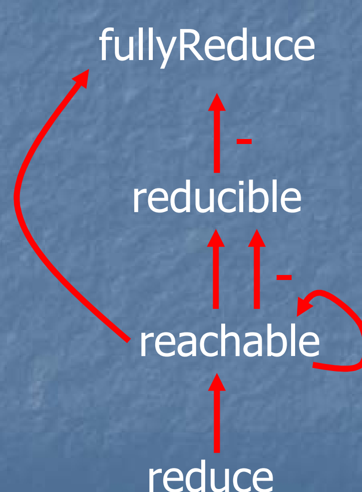
```
fullyReduceRep(X,Y) :-
   fullyReduce(X,Y),
   tnot(smallerequiv(Y)).
smallerequiv(X) :-
   reachable(X,Y),
   Y@<X,
   reachable(Y,X).
```

```
| ?- fullyReduce(a,X).
X = c;
X = h;
X = d;
X = k;
X = i;
X = e;
No
| ?- fullyReduceRep(a,X).
X = c;
X = h;
X = k;
no
```

UMBC
an Honors University in Maryland

This is a Stratified Program

- There is no recursion thru negation.
- We can safely do the computation if we work from the bottom up.



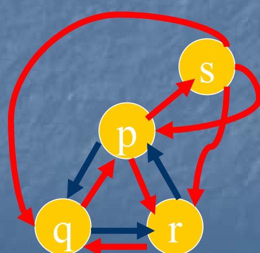
UMBC
an Honors University in Maryland

Stratified or not?

•This simple program, when executed, does not recurse thru negation.

•S can be proven (by XSB) even tho Prolog would loop.

•If we change the order of the literals in the clauses for p, q and r?




```
:- auto_table.
p:- q, tnot(r), tnot(s).
q:- r, tnot(p).
r:- p, tnot(q).
s :-
  tnot(p),
  tnot(q),
  tnot(r).
```

UMBC
an Honors University in Maryland

Black arcs are + and red -

Well-founded semantics of non-stratified negation.

- XSB handles non-stratified programs computing answers using “well founded semantics”
- In WFS there are three truth values: true, false and unknown
- Atoms that depend on themselves negatively are assigned the value unknown.
- Example:
  - The barber shaves everyone who does not shave himself.
  - shaves(barber,X) :- tnot(shaves(X,X))

UMBC
an Honors University in Maryland

example

```
person(john).
person(bill).
person(mark).
person(harry).
person(barber).

:- table shave/2.

shave(john,john).
shave(bill,bill).

shave(barber,Y) :-
  person(Y),
  tnot(shave(Y,Y)).
```

```
[5:10pm] linuxserver1 103(02)=>xsb
XSB Version 2.2 (Tsingtao) of April 20, 2000
[i686-pc-linux-gnu; mode: optimal; engine: chat; scheduling: batched]
| ?- [shave].
[shave loaded]
Yes

| ?- shave(X,Y).
X = john
Y = john;
X = bill
Y = bill;
X = barber
Y = mark;
X = barber
Y = harry;
X = barber
Y = barber;
No

| ?- tnot(shave(barber,barber)).
Yes

| ?- shave(barber,barber).
yes
```

UMBC

an Honors University in Maryland