Logic Programming For Networking

Lecture 2

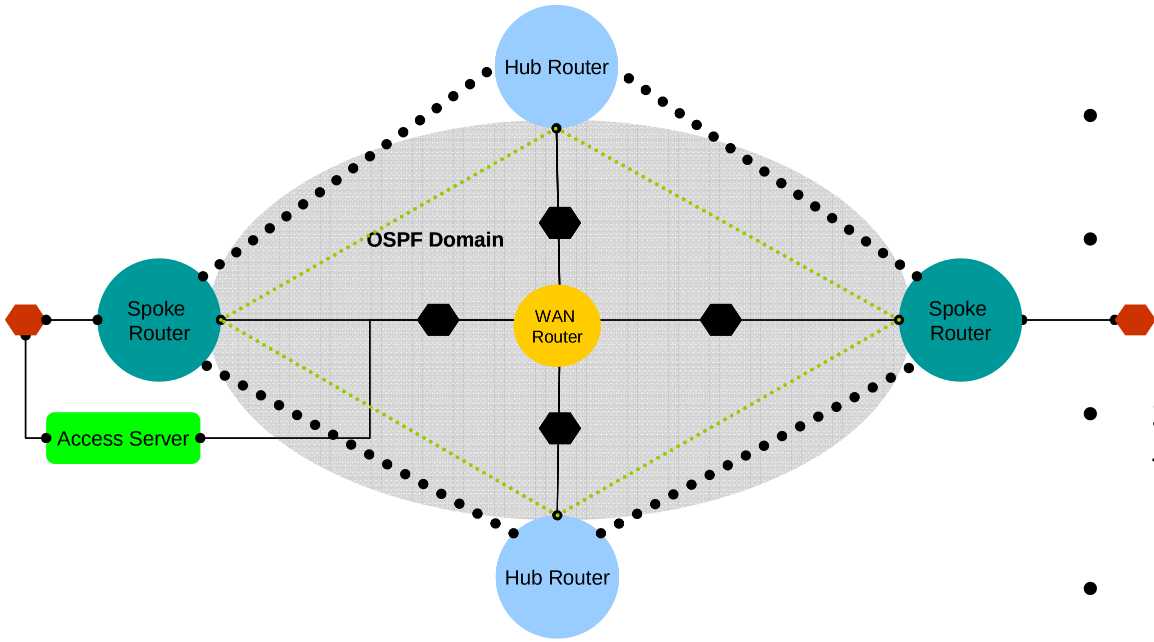
CS 598D, Spring 2010
Princeton University

Sanjai Narain
narain@research.telcordia.com
908 337 3636

Outline

- Testbed outline
- Main logic programming ideas
- Networking problems for which Prolog is appropriate
- Problems for which Prolog is inadequate and needs assistance of constraint solvers
- Logic programming theory sketch

Testbed: A Fault-Tolerant VPN



- Obtain good appreciation of configuration complexity for a real network solution
- How to do automatic routing over IPSec?
- Use GRE tunnels underneath and run routing protocol over it
- Solve different configuration problems for testbed
- Useful shared experience

Main Logic Programming Ideas

- Logic programming underlies ConfigAssure, MulVAL and RapidNet
- A logic program is a set of "definite" clauses of the form $A \leftarrow B_0,..,B_k,\ k \geq 0$
- Database facts and recursive query rules are special cases of definite clauses
- SLD-resolution is inference procedure. It is top-down
- Definite clauses have a procedural interpretation, so one can write efficient specifications
- Prolog is an implementation of logic programming
  - Programming language + pattern matching + relational database
- Datalog is Prolog without data structures. It also has a bottom-up inference procedure
- Applications to networking:
  - Requirement specification
  - Analyzing ad hoc configuration languages
  - Evaluating requirements against configuration
  - Routing protocol design
  - Vulnerability analysis
  - Control language for driving constraint solvers or visualizers

Simple Prolog Programs

- List membership

```
member(X, [X|Y]).
member(X, [U|V]):-member(X, V).
```

- Running programs means querying these

```
?-member(X, [a,b]).
X=a;
X=b
```

- Data structures are represented by terms
- Fields are extracted by unification pattern matching

- List concatenation

```
append([], X, X).
append([U|V], X, [U|Z]):-append(V, X, Z).

?-append([1,2], [3, 4], X).
X=[1,2,3,4]
```

- Inputs can be computed from output

```
?-append(X, Y, [1,2]).
X=[], Y=[1,2]
X=[1], Y=[2]
X=[1,2], Y=[]
```

- All solutions can be computed with findall

```
?-findall(X-Y, append(X, Y, [1,2]), S)
S=[ []-[1,2],  [1]-[2],  [1,2]-[] ]
```

Simple Database And Recursive Query Rule

```
parent(bill,mary).
parent(mary,john).

ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- parent(X,Z),ancestor(Z,Y).

?- ancestor(bill, X).
X=mary;
X=john
```

Ad Hoc Configuration File Analysis Problem

Sample Cisco IOS Configuration Commands

```
hostname router1
!
interface Ethernet0
    ip address 1.1.1.1 255.255.255.0
    crypto map mapx
!
crypto map mapx 6 ipsec-isakmp
    set peer 3.3.3.3
    set transform-set transx
    match address aclx
!
crypto ipsec transform-set transx esp-3des hmac
!
ip access-list extended aclx
    permit gre host 3.3.3.3 host 4.4.4.4
```

- Challenges
  - Configuration language documentation can run into thousand+ pages
  - How to extract information from configuration files without having to know the entire configuration language?
  - How to assemble information from different parts of file?
  - How to making algorithms robust to inevitable changes in the configuration language?

- Grammar approach is inappropriate

- Query-based approach
  - Express the configuration commands as a database
  - Query it to take what you need
  - No need to predict what part of the command language is relevant

Ad-hoc Configuration File Analysis in Prolog

- IOS Configuration File ios_file_1

```
hostname router1
interface Ethernet0
     ip address 1.1.1.1 255.255.255.0
Interface Ethernet1
     ip address 2.2.2.2 255.255.255.0
```

- Prolog database of IOS commands

```
ios_cmd(ios_file_1, [0, hostname, router1], []).
ios_cmd(ios_file_1, [0, interface, 'Ethernet0'], [ [1, ip, address, '1.1.1.1', '255.255.255.0']]).
ios_cmd(ios_file_1, [0, interface, 'Ethernet1'], [ [1, ip, address, ‘2.2.2.2', '255.255.255.0‘]]).
```

- IP address information extraction

```
ipAddress(Host, IF, Address, Mask):-
     ios_cmd(File, [0, hostname, Host|_], _),
     ios_cmd(File, [0, interface, IF|_], Args),
     member([_, ip, address, Address, Mask], Args).

?-ipAddress(H, I, A, M).
        H=router1, I=’Ethernet0’, A=’1.1.1.1’, M=’255.255.255.0’;
        H=router1, I=‘Ethernet1’, A=‘2.2.2.2’, M=‘255.255.255.0’
```

Prolog As Metalevel Language: Generating Graphviz

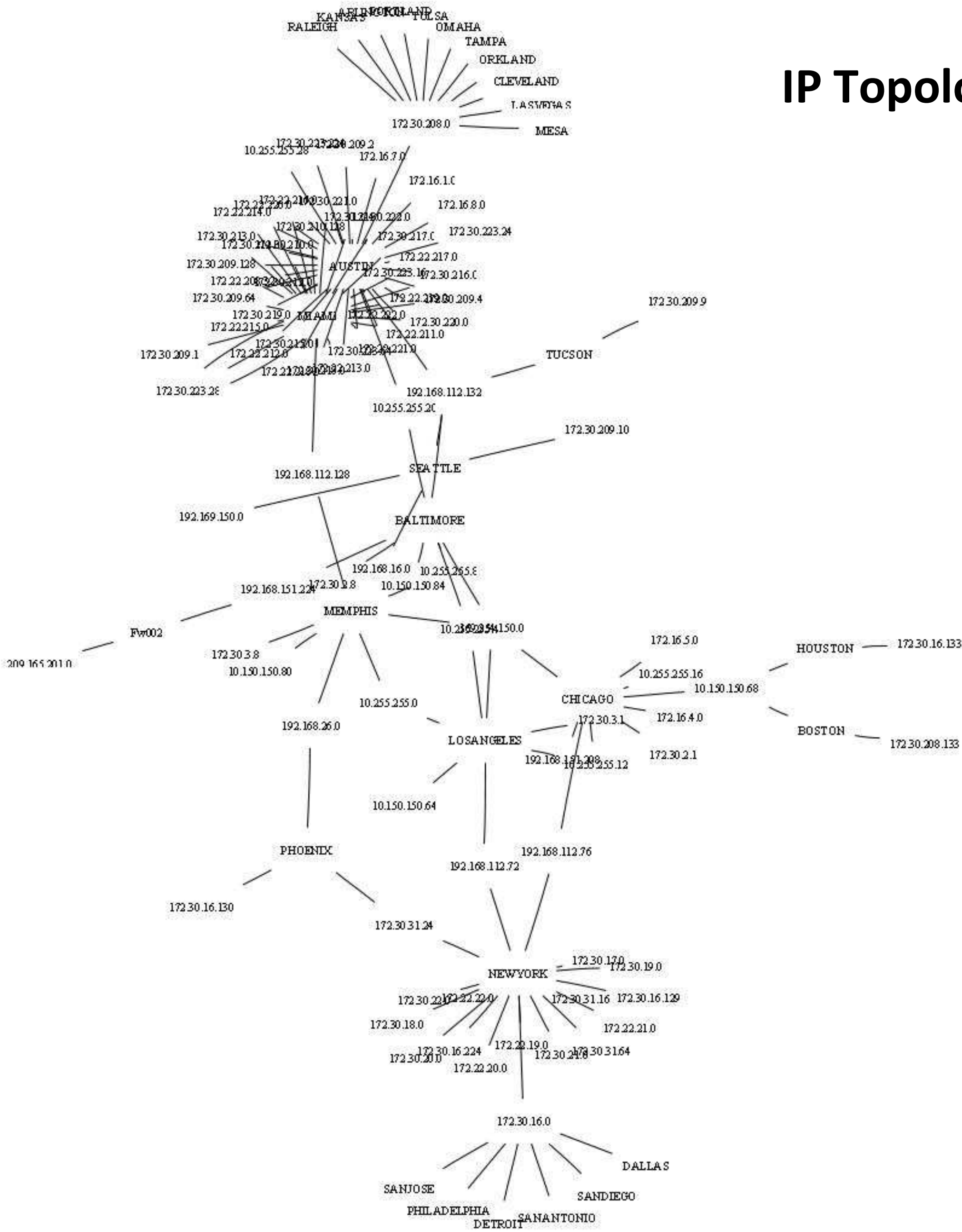
- Use extracted IP address table to visualize IP topology. Make use of findall feature

```
makeRouterSubnetGraph:-
   findall([H-N], (ipAddress(H, I, A, M), subnet(A, M, N)), S),
   tell(‘ipnet.txt’),
   makeGraphViz(S),
   told.

makeGraphViz(Edges):-
   write('digraph G {size=“8.5,11"; ratio=fill;
      node[fontsize=10,shape=plaintext];edge[dir=none,style="setlinewidth(1.0)"];'),nl,
   printGraphEdges(Edges),
   write('}').

printGraphEdges([]).
printGraphEdges([[U-V|Attributes]|Z]):-
   write('"'),write(U),write('"->"'),write(V),write('"'),
   write(Attributes),write(';'),nl,
   printGraphEdges(Z).
```

Demo



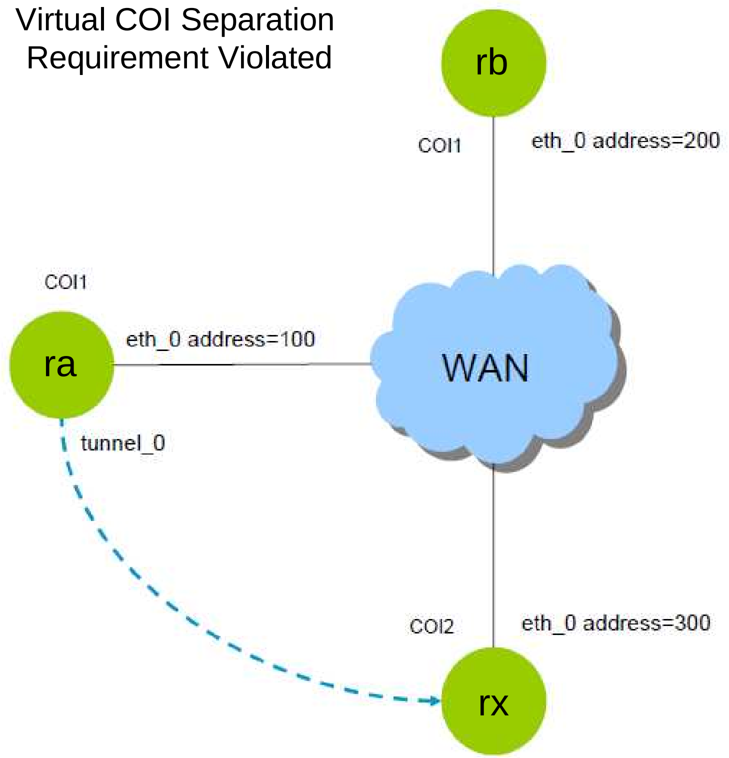
IP Topology

- Parse 34 configuration files with about 50,000 commands:

ios_cmd('.//telcordia//ARLINGTON.txt', [0], []).
ios_cmd('.//telcordia//ARLINGTON.txt', [0, 'ARLINGTON#'], []).
ios_cmd('.//telcordia//ARLINGTON.txt', [0, show, run], []).
ios_cmd('.//telcordia//ARLINGTON.txt', [0, 'Building', 'configuration...'], []).
ios_cmd('.//telcordia//ARLINGTON.txt', [0], []).
ios_cmd('.//telcordia//ARLINGTON.txt', [0, 'Current', configuration, :, 13748, bytes], []).
ios_cmd('.//telcordia//ARLINGTON.txt', [0, !], []).
ios_cmd('.//telcordia//ARLINGTON.txt', [0, !, 'Last', configuration, change, at, '11:24:33', 'EDT', 'Thu', 'Mar', 20, 2008, by, removed], []).
ios_cmd('.//telcordia//ARLINGTON.txt', [0, !, 'NVRAM', config, last, updated, at, '11:24:34', 'EDT', 'Thu', 'Mar', 20, 2008, by, removed], []).
ios_cmd('.//telcordia//ARLINGTON.txt', [0, !], []).
ios_cmd('.//telcordia//ARLINGTON.txt', [0, version, 12.2], []).
ios_cmd('.//telcordia//ARLINGTON.txt', [0, no, service, pad], []).
ios_cmd('.//telcordia//ARLINGTON.txt', [0, service, timestamps, debug, datetime, msec, localtime, 'show-timezone'], []).
ios_cmd('.//telcordia//ARLINGTON.txt', [0, service, timestamps, log, datetime, msec, localtime, 'show-timezone'], []).

Prolog As Specification Language

Virtual COI Separation Requirement Violated



static_route(ra, 0, 32, 400).
gre(ra, tunnel_0, 100, 300).
ipAddress(ra, eth_0, 100, 0).
ipAddress(rb, eth_0, 200, 0).
ipAddress(rx, eth_0, 300, 0).
coi([ra-coi1, rb-coi1, rx-coi2]).

Configuration database

Specification

```
good:-gre_connectivity(ra, rb).
bad:-gre_tunnel(ra, rx).
bad:-route_available(ra, rx).

gre_connectivity(RX, RY):-
        gre_tunnel(RX, RY),
        route_available(RX, RY).

gre_tunnel(RX, RY):-
        gre(RX, _, _, RemoteAddr),
        ipAddress(RY, _, RemoteAddr, _).

route_available(RX, RY):-
        static_route(RX, Dest, Mask, _),
        ipAddress(RY, _, RemotePhysical, 0),
        contained(Dest, Mask, RemotePhysical, 0).

contained(Dest, Mask, Addr, M):-
        Mask>=M,
        N is ((2^32-1)<< Mask)/\Dest,
        N is ((2^32-1)<< Mask)/\Addr.
```

Evaluating Requirements

```
?- good.
no
?- bad.
yes
```

Problems For Which Prolog Is Inadequate

- Repair: Change configurations so that good holds and bad does not; at minimum cost
- Synthesis: Generate correct configurations so good holds and bad does not
- Reconfiguration planning: Sequence configuration change without violating invariants
- Firewall policy equivalence evaluation

Prolog needs assistance of constraint solvers to solve these

Projects and Next Class

Projects

- Adapt the IOS configuration file analyzer to Xorp
- Adapt the IP topology visualization program to other protocols
- Adapt the requirement evaluation program to other requirements. Read the paper “Network Configuration Validation” to see how English requirements are specified in Prolog

Next class

- Some more Prolog features: cut and negation as failure
- Evaluating firewall policies
- Using Prolog as a metalevel language to solve theory of configuration problems with constraint solvers

Logic Programming Theory Sketch

Clausal Form of First-Order Logic

- Every variable is a term
- If f is a k-argument function symbol and $t_1,..,t_k$ are terms then $f(t_1,..,t_k)$ is a term
- If p is a k-ary predicate symbol and $t_1,..,t_k$ are terms then $p(t_1,..,t_k)$ is a literal
- A clause is of the form $B_1,..,B_k \leftarrow A_1,..,A_m$, $k\geq 0$, $m \geq 0$, each A_i, B_j a literal
- It means that for all variables in the clause, the conjunction of $A_1,..,A_m$ implies the disjunction of $B_1,..,B_k$

Horn Clauses and Their Procedural Interpretation

- The clause $B_1,..,B_m \leftarrow A_1,..,A_n$, $m\geq 0$, $n \geq 0$ is called a Horn clause if m=0 or m=1
- In the procedural interpretation of Horn clauses, there are four kinds of clauses:

1. $B \leftarrow A_1,..,A_n$, n>0 is a procedure. Also known as a definite clause (no disjunction).
2. $B \leftarrow$ is a fact. It is unconditionally true.
3. $\leftarrow A_1,...,A_n$, n>0, is a goal statement
4. $\leftarrow$ is the halt statement

Rule of Inference: SLD-Resolution

- Given
  - A goal statement $\leftarrow A_1,..,A_{i-1}, \mathbf{A_i}, A_{i+1},..,A_n$ and
  - A procedure $B \leftarrow B_1,..,B_m$ where B *unifies* with A_i with most general unifier σ

- Derive the new goal
  - $\leftarrow(A_1,..,A_{i-1}, \mathbf{B_1,..,B_m} ,..,A_n)\sigma$

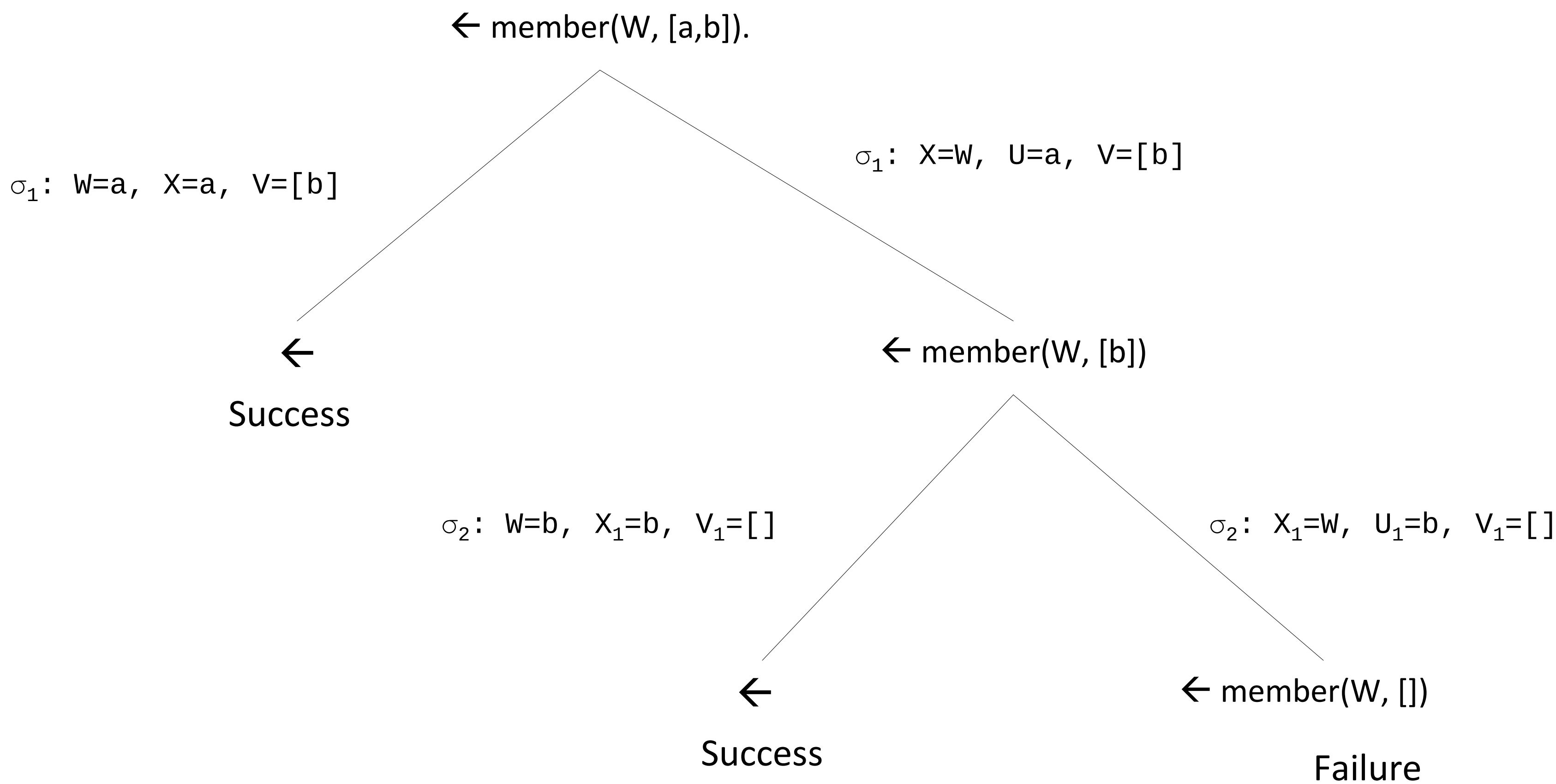
- If the new goal is empty, then halt with success and return the composition of unifiers accumulated along the branch from the goal

- This rule is sound and complete

SLD-resolution Search Tree

member(X, [X|V]).

member(X, [U|V]) ← member(X, V).



References

- Applications discussed in this presentation
  - S. Narain, G. Levin, R. Talpade. Network Configuration Validation. Chapter in Guide to Reliable Internet Services and Applications, edited by Chuck Kalmanek, Richard Yang, and Sudip Misra. Springer, 2010
- Theory of logic programming
  - R. Kowalski. Predicate logic as a programming language
  - M.H. van Emden and R. A. Kowalski. Semantics of predicate logic as a programming language
- Unification algorithm
  - J.A. Robinson. Logic: Form and Function. Elsevier, North Holland, 1979
- SWI-Prolog. http://www.swi-prolog.org/
- Prolog tutorial http://www.csupomona.edu/~jrfisher/www/prolog_tutorial/contents.html

Two Equivalent Specifications of Sort With Different Performance

INSERTION SORT

```
insert(X,[],[X]).
insert(X, [Y|Sorted], [Y|Sorted1]) :-
        X > Y,
        insert(X, Sorted, Sorted1).
insert(X, [Y|Sorted], [X,Y|Sorted]) :-
        X=<Y.

insertsort([],[]).
insertsort([X|Tail],Sorted) :-
        insertsort(Tail, SortedTail),
        insert(X, SortedTail, Sorted).

?- insertsort([3,2,1], X).
X=[1,2,3]
```

QUICK SORT

```
quicksort([],[]).
quicksort([X|Tail], Sorted) :-
        split(X, Tail, Small, Big),
        quicksort(Small, SortedSmall),
        quicksort(Big, SortedBig),
        append(SortedSmall, [X|SortedBig], Sorted).

split(_, [], [], []).
split(X, [Y|Tail], [Y|Small], Big) :- X > Y, split(X, Tail,
        Small, Big).
split(X, [Y|Tail], Small, [Y|Big]) :- X =< Y, split(X, Tail,
        Small, Big).

?- quicksort([3,2,1], X).
X=[1,2,3]
```

Possible due to the procedural interpretation of definite clauses