Logic Programming

Vitaly Shmatikov

Reading Assignment

- Mitchell, Chapter 15

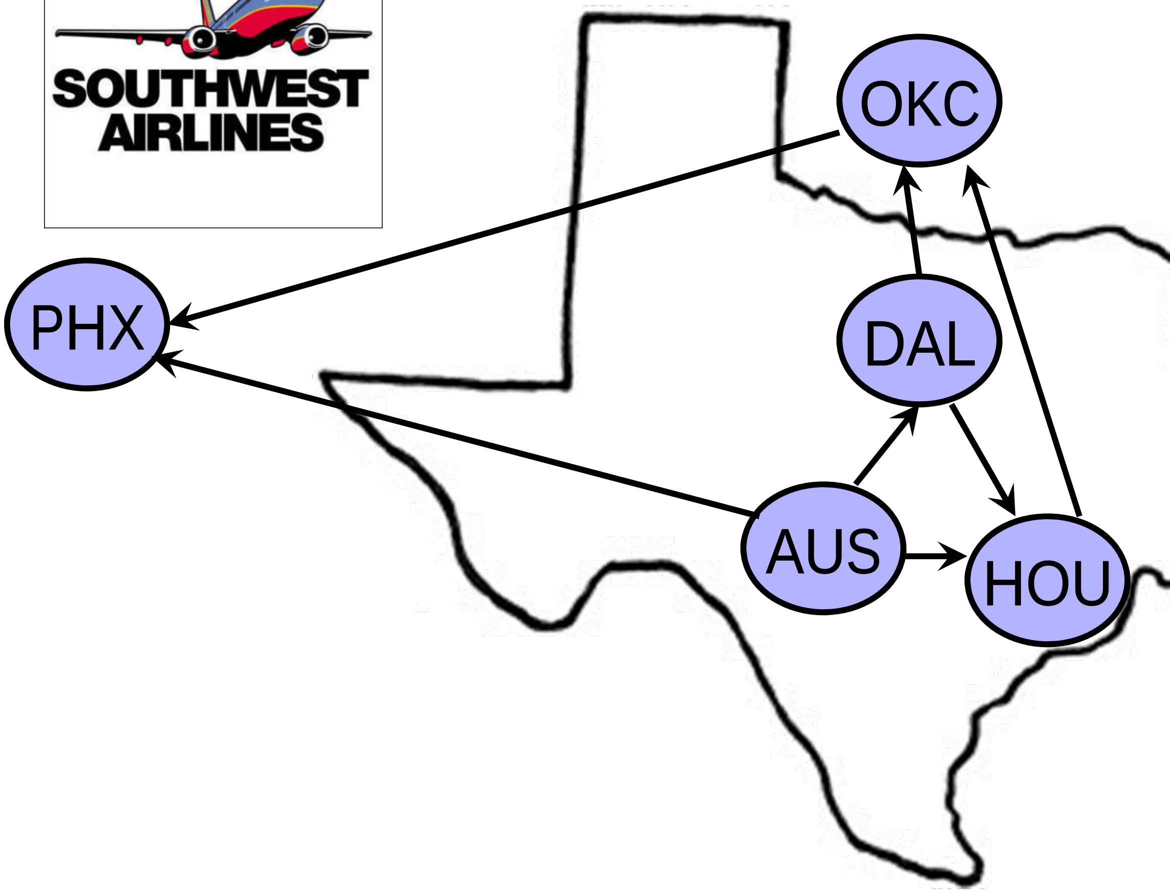
Logic Programming

- Function (method) is the basic primitive in all languages we have seen so far
  - F(x)=y – function F takes x and return y
- Relation (predicate) is the basic primitive in logic programming
  - R(x,y) – relationship R holds between x and y

Prolog

- Short for Programmation en logique
  - Alain Colmeraurer (1972)
- Basic idea: the program declares the goals of the computation, not the method for achieving them
- Applications in AI, databases, even systems
  - Originally developed for natural language processing
  - Automated reasoning, theorem proving
  - Database searching, as in SQL
  - Expert systems
  - Recent work at Berkeley on declarative programming

Example: Logical Database



In Prolog:

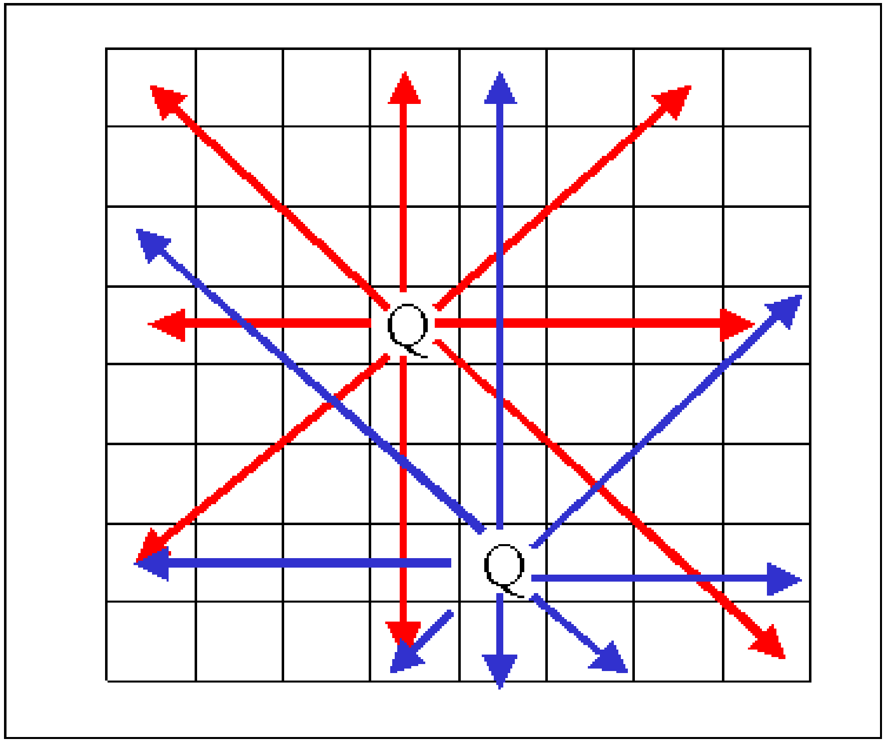
```
nonstop(aus, dal).
nonstop(aus, hou).
nonstop(aus, phx).
nonstop(dal, okc).
nonstop(dal, hou).
nonstop(hou, okc).
nonstop(okc, phx).
```

Logical Database Queries

- Where can we fly from Austin?
- SQL
  - SELECT dest FROM nonstop WHERE source="aus";
- Prolog
  - ?- nonstop(aus, X).
  - More powerful than SQL because can use recursion

N-Queens Problem

- Place N non-attacking queens on the chessboard
  - Example of a search problem (why?)

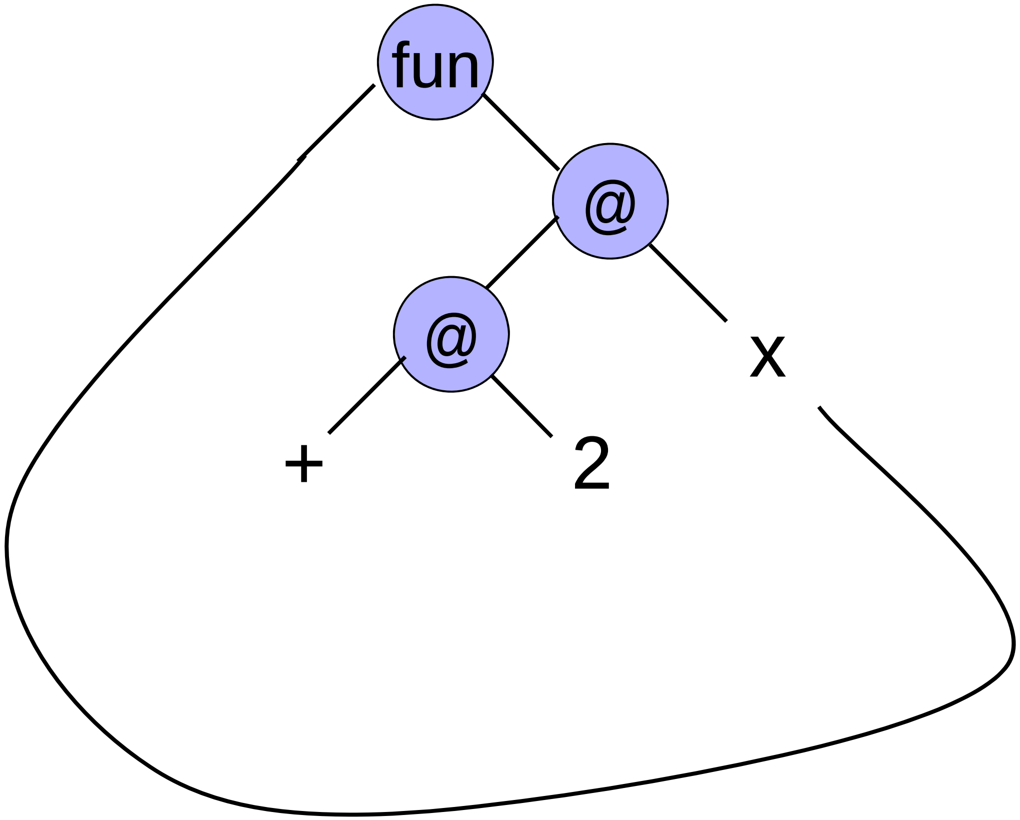


N-Queens in Prolog

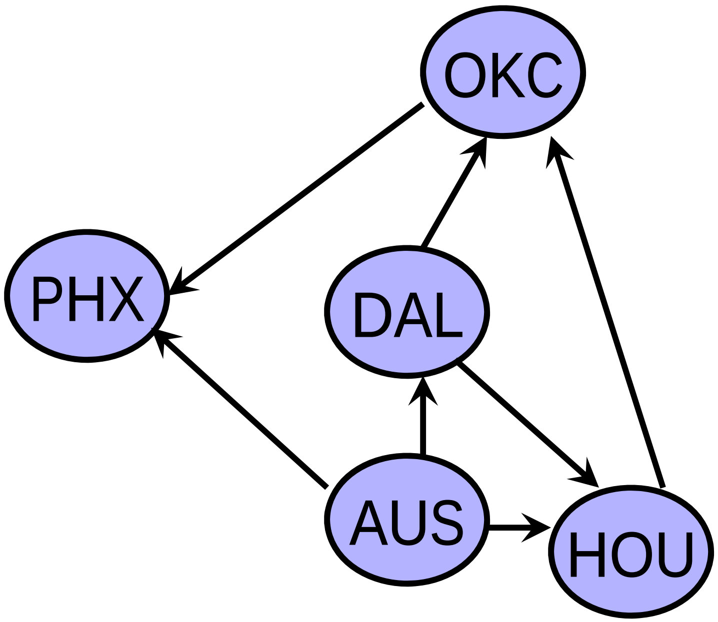
```
diagsafe(_, _, []).
diagsafe(Row, ColDist, [QR|QRs]) :-
    RowHit1 is Row + ColDist, QR =n= RowHit1,
    RowHit2 is Row - ColDist, QR =n= RowHit2,
    ColDist1 is ColDist + 1,
    diagsafe(Row, ColDist1, QRs).
safe_position([_]).
safe_position([QR|QRs]) :-
    diagsafe(QR, 1, QRs),
    safe_position(QRs).
nqueens(N, Y) :-
    sequence(N, X), permute(X, Y), safe_position(Y).
```

Type Inference in ML

- Given an ML term, find its type



Flight Planning Example



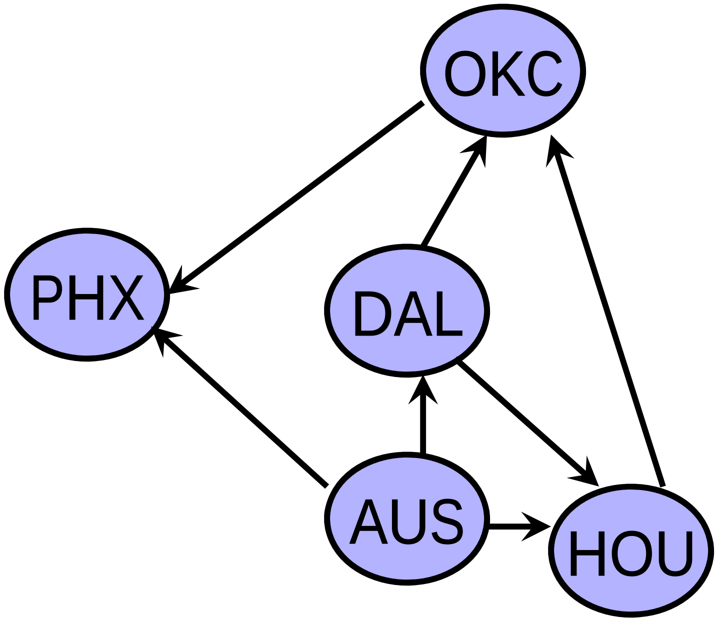

```
nonstop(aus, dal).
nonstop(aus, hou).
nonstop(aus, phx).
nonstop(dal, okc).
nonstop(dal, hou).
nonstop(hou, okc).
nonstop(okc, phx).
```

Each line is called a clause and represents a known fact

A fact is true if and only if we can prove it true using some clause

Relation: nonstop(X, Y) – there is a flight from X to Y

Queries in Prolog



?- nonstop(aus, dal).

Yes

?- nonstop(dal, okc).

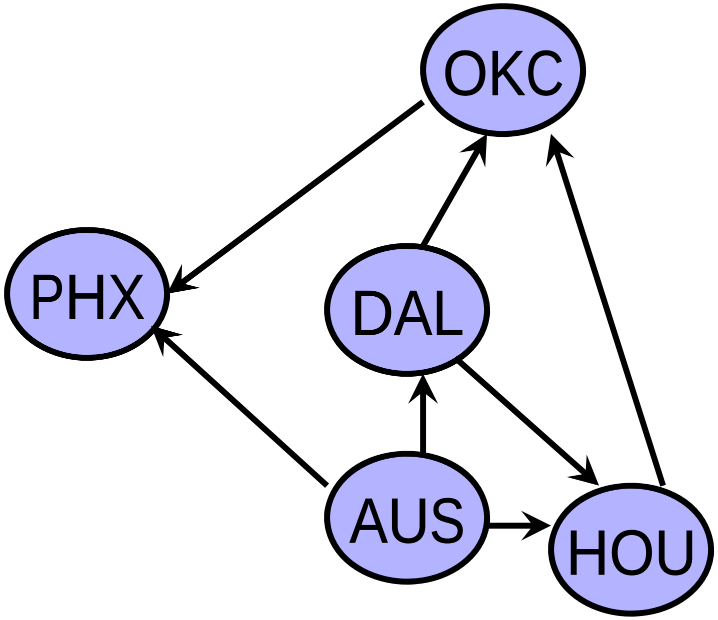
Yes

?- nonstop(aus, okc).

No

?-

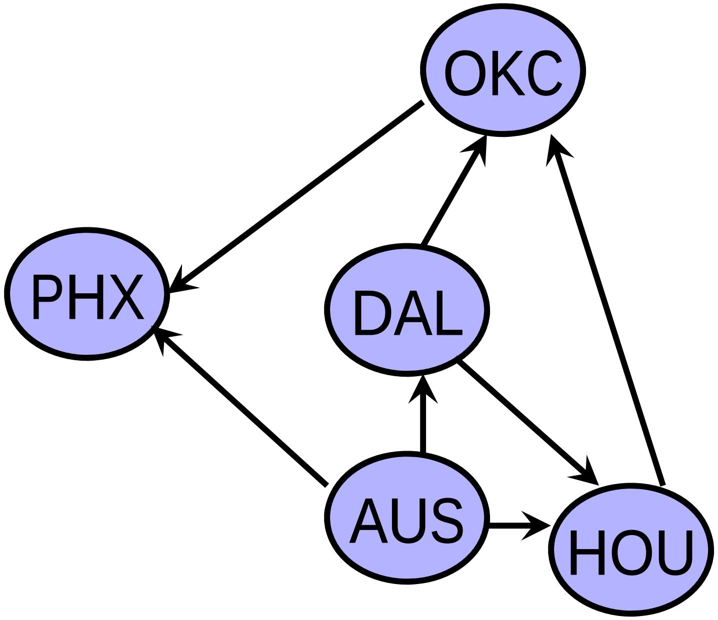
Logical Variables in Prolog



Is there an X such that nonstop(okc, X) holds?

```
?- nonstop(okc, X).
X=phx ;
No
?- nonstop(Y, dal).
Y=aus ;
No
?-
```

Non-Determinism



?- nonstop(dal, X).

X=hou ;

X=okc ;

No

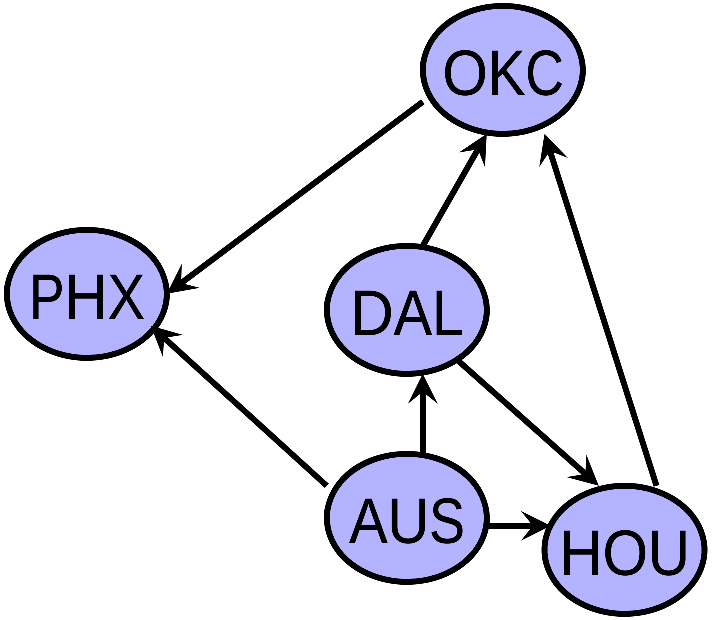
?- nonstop(phx, X).

No

?-

Predicates may return multiple answers or no answers

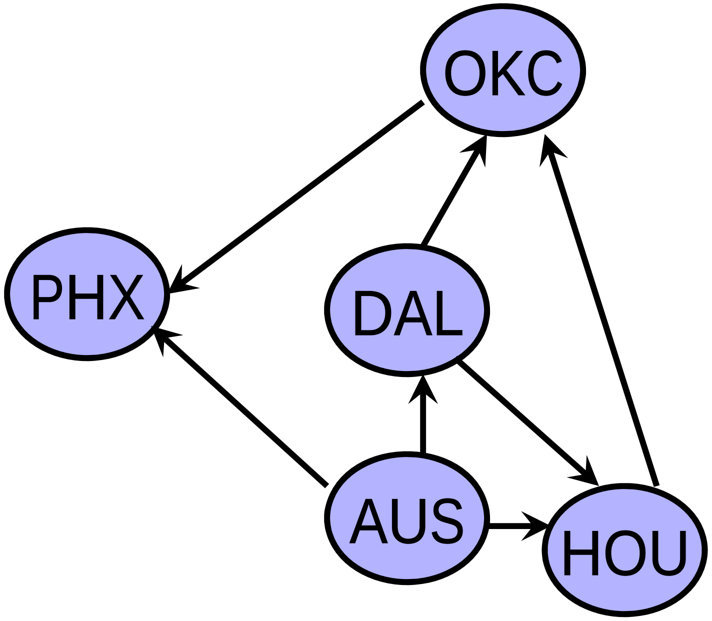
Logical Conjunction




```
?- nonstop(aus, X), nonstop(X, okc).
X=dal ;
X=hou ;
No
?-
```

Combine multiple conditions into one query

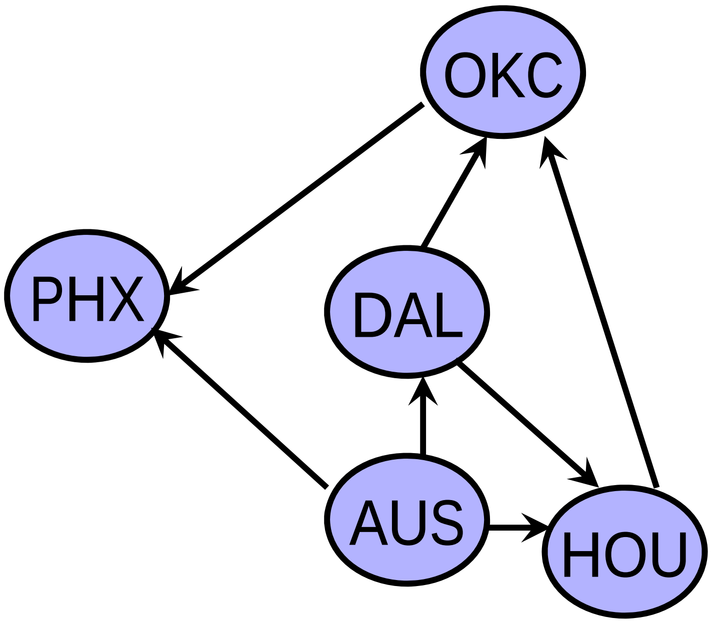
Derived Predicates



- Define new predicates using rules
- conclusion :- premises.
  - conclusion is true if premises are true

```
flyvia(From, To, Via) :-
    nonstop(From, Via),
    nonstop(Via, To).
?- flyvia(aus, okc, Via).
Via=dal ;
Via=hou ;
No
?-
```

Recursion



- Predicates can be defined recursively

```
reach(X, X).
reach(X,Z) :-
    nonstop(X, Y), reach(Y, Z).
?- reach(X, phx).
X=aus ;
X=dal ;
...
?-
```

Prolog Program Elements

- Prolog programs are made from terms
  - Variables, constants, structures
- Variables begin with a capital letter
  - Bob
- Constants are either integers, or atoms
  - 24, zebra, ‘Bob’, ‘.’
- Structures are predicates with arguments
  - n(zebra), speaks(Y, English)

Horn Clauses

- A Horn clause has a head h, which is a predicate, and a body, which is a list of predicates p1, p2, ..., pn
  - It is written as h $\leftarrow$ p1, p2, ..., pn
  - This means, "h is true if p1, p2, ..., and pn are simultaneously true"
- Example
  - snowing(C) $\leftarrow$ precipitation(C), freezing(C)
  - This says, "it is snowing in city C if there is precipitation in city C and it is freezing in city C"

Facts, Rules, and Programs

- A Prolog fact is a Horn clause without a right-hand side
  - Term.
    - The terminating period is mandatory
- A Prolog rule is a Horn clause with a right-hand side (:- represents $\leftarrow$)
  - term :- term1, term2, ... termn.
  - LHS is called the head of the rule
- Prolog program = a collection of facts and rules

Horn Clauses and Predicates

- Any Horn clause $h \leftarrow p1, p2, \ldots, pn$ can be written as a predicate $p1 \wedge p2 \wedge \ldots \wedge pn \supset h$, or, equivalently, $\neg(p1 \wedge p2 \wedge \ldots \wedge pn) \vee h$
- Not every predicate can be written as a Horn clause (why?)
  - Example: $literate(x) \supset reads(x) \vee writes(x)$

Lists

- A list is a series of terms separated by commas and enclosed in brackets
  - The empty list is written []
  - A "don't care" entry is signified by _, as in [_, X, Y]
  - A list can also be written in the form [Head | Tail]

Appending a List

```
append([], X, X).
append([Head | Tail], Y, [Head | Z]) :-
        append(Tail, Y, Z).
```

- The last parameter designates the result of the function, so a variable must be passed as an argument

- This definition says:
  - Appending X to the empty list returns X
  - If Y is appended to Tail to get Z, then Y can be appended to a list one element larger [Head | Tail] to get [Head | Z]

List Membership

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

- The test for membership succeeds if either:
  - X is the head of the list [X | _]
  - X is not the head of the list [_ | Y] , but X is a member of the remaining list Y
- Pattern matching governs tests for equality
- "Don't care" entries (_) mark parts of a list that aren't important to the rule

More List Functions

- X is a prefix of Z if there is a list Y that can be appended to X to make Z
  - prefix(X, Z) :- append(X, Y, Z).
  - Suffix is similar: suffix(Y, Z) :- append(X, Y, Z).
- Finding all the prefixes (suffixes) of a list

```
?- prefix(X, [my, dog, has, fleas]).
X = [];
X = [my];
X = [my, dog];
...
```

Answering Prolog Queries

- Computation in Prolog (answering a query) is essentially searching for a logical proof
- Goal-directed, backtracking, depth-first search
  - Resolution strategy:

    if h is the head of a Horn clause

    $h \leftarrow$ terms

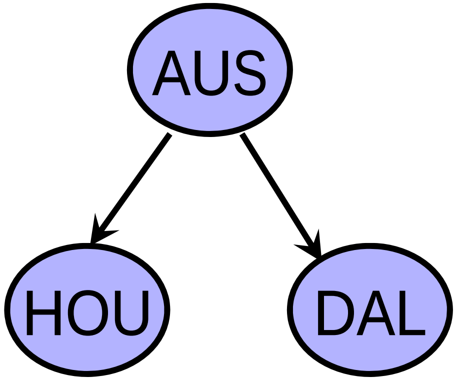
    and it matches one of the terms of another Horn clause

    $t \leftarrow t1, h, t2$

    then that term can be replaced by h's terms to form

    $t \leftarrow t1$, terms, $t2$
  - What about variables in terms?

Flight Planning Example



?- n(aus, hou).

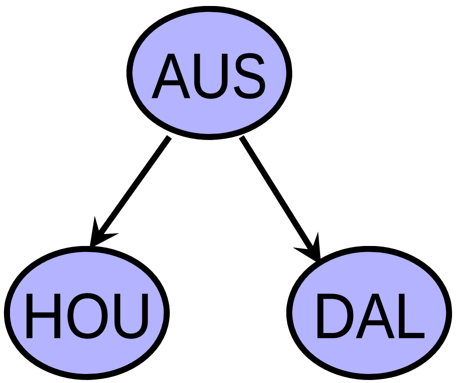
?- n(aus, dal).

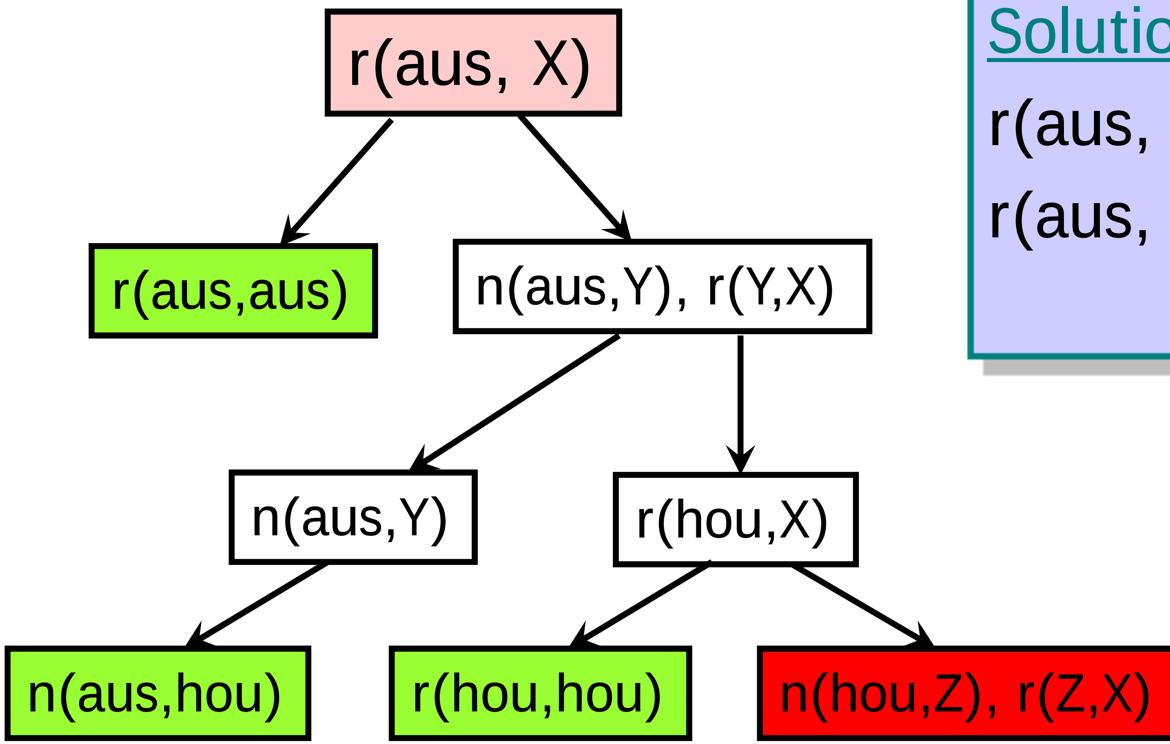
?- r(X, X).

?- r(X, Z) :- n(X, Y), r(Y, Z).

?- r(aus, X)

Flight Planning: Proof Search





<u>Solution</u>

r(aus, aus)

r(aus, hou)

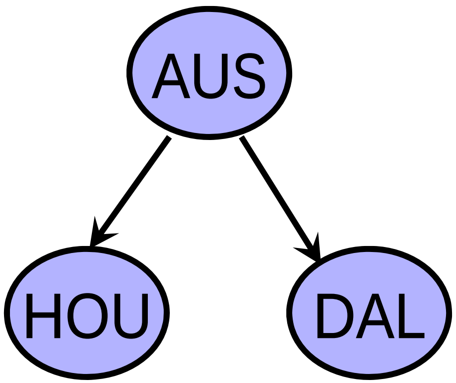
Rule 1:

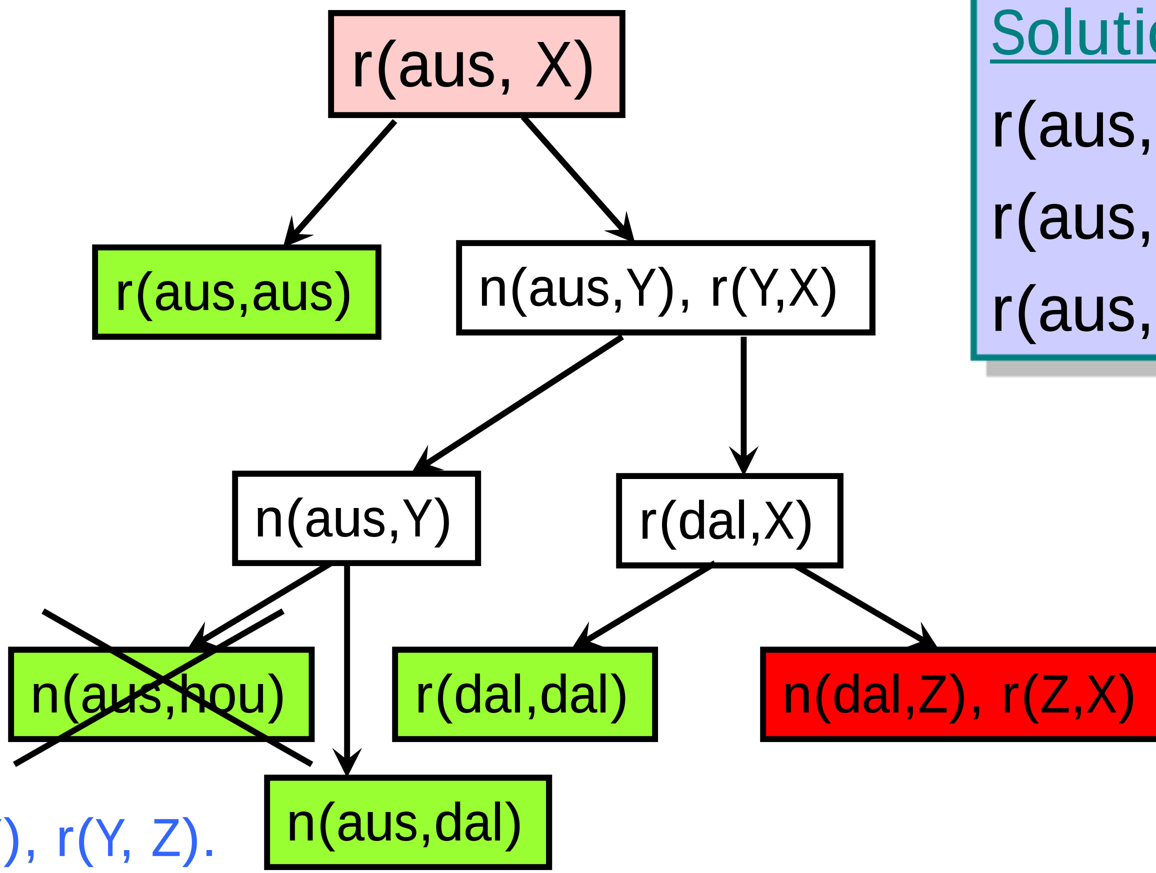
r(X, X).

Rule 2:

r(X, Z) :- n(X, Y), r(Y, Z).

Flight Planning: Backtracking





Solution

r(aus, aus)

r(aus, hou)

r(aus, dal)

Rule 1:

r(X, X).

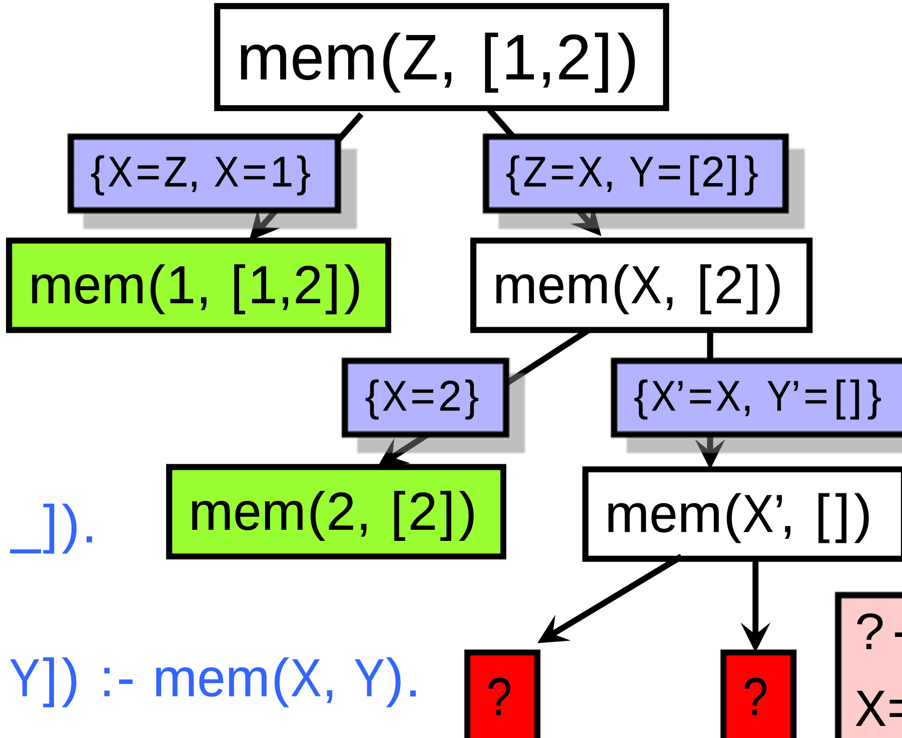
Rule 2:

r(X, Z) :- n(X, Y), r(Y, Z).

Unification

- Two terms are unifiable if there is a variable substitution such that they become the same
  - For example, f(X) and f(3) are unified by [X=3]
  - f(f(Y)) and f(X) are unified by [X=f(Y)]
  - How about g(X,Y) and f(3)?
- Assignment of values to variables during resolution is called instantiation
- Unification is a pattern-matching process that determines what instantiations can be made to variables during a series of resolutions

Example: List Membership



Rule 1:

mem(X, [X | _]).

Rule 2:

mem(X, [_ | Y]) :- mem(X, Y).

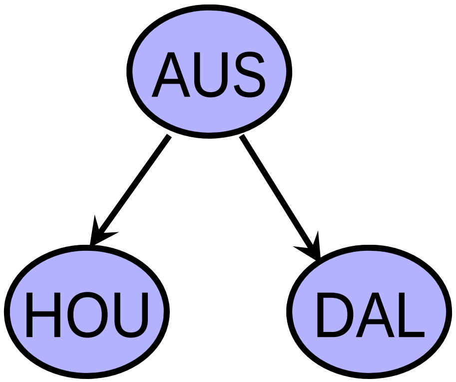
Prolog

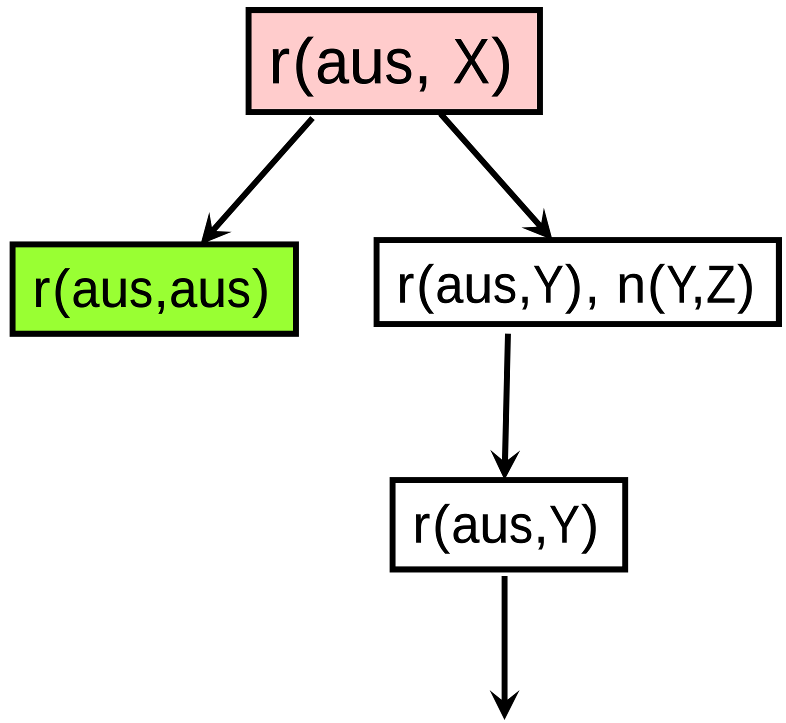
```
?- mem(X, [1,2]).
X=1 ;
X=2 ;
No
?-
```

Soundness and Completeness

- Soundness
  - If we can prove something, then it is logically true
- Completeness
  - We can prove everything that is logically true
- Prolog search procedure is sound, but incomplete

Flight Planning: Small Change





Infinite loop!

Solution

r(aus, aus)

Rule 1:

r(X, X).

Rule 2:

r(X, Z) :- r(X, Y), n(Y, Z).

instead of n(X,Y), r(Y,Z)

“Is” Operator

- is instantiates a temporary variable
  - Similar to a local variable in Algol-like languages
- Example: defining a factorial function

```
?- factorial(0, 1).
?- factorial(N, Result) :-
        N > 0, M is N - 1,
        factorial(M, SubRes), Result is N * SubRes.
```

Tracing

- Tracing helps programmer see the dynamics of a proof search
- Example: tracing a factorial call

```
?- factorial(0, 1).
?- factorial(N, Result) :-
        N > 0, M is N - 1,
        factorial(M, SubRes), Result is N * SubRes.
?- trace(factorial/2).
```

  – Argument to “trace” must include function’s arity

```
?- factorial(4, X).
```

Tracing Factorial

- ?- factorial(4, X).
- Call: (7) factorial(4, _G173)
- Call: (8) factorial(3, _L131)
- Call: (9) factorial(2, _L144)
- Call: (10) factorial(1, _L157)
- Call: (11) factorial(0, _L170)
- Exit: (11) factorial(0, 1)
- Exit: (10) factorial(1, 1)
- Exit: (9) factorial(2, 2)
- Exit: (8) factorial(3, 6)
- Exit: (7) factorial(4, 24)

- X = 24

These are temporary variables

These are levels in the search tree

The Cut

- When inserted on the right-hand side of the rule, the cut operator ! operator forces subgoals not to be re-tried if r.h.s. succeeds once
- Example: bubble sort
  - bsort(L, S) :- append(U, [A, B | V], L),
    B < A, !,
    append(U, [B, A | V], M),
    bsort(M, S).
  - bsort(L, L).

Gives one answer rather than many

Tracing Bubble Sort

- ?- bsort([5,2,3,1], Ans).
- Call: (7) bsort([5, 2, 3, 1], _G221)
- Call: (8) bsort([2, 5, 3, 1], _G221)
- ...
- Call: (12) bsort([1, 2, 3, 5], _G221)
- Redo: (12) bsort([1, 2, 3, 5], _G221)
- ...
- Exit: (7) bsort([5, 2, 3, 1], [1, 2, 3, 5])
- Ans = [1, 2, 3, 5] ;
- No

Without the cut, this would have given some wrong answers

Negation in Prolog

- not operator is implemented as goal failure
  - not(G) :- G, !, fail
    - “fail” is a special goal that always fail
  - What does this mean?
- Example: factorial
  - factorial(N, 1) :- N < 1.
  - factorial(N, Result) :- not(N < 1), M is N - 1,

    factorial(M, P),

    Result is N * P.