CHAPTER 11 : LOGIC LANGUAGES

CMPSC 460 – Principles of Programming Languages

Logic Programming

- *No need to view computation as functions*
  - *Data*
  - *Rules between the data*
  - *Computing: searching among rules and data*

The Origins of Prolog

- University of Edinburgh
  - Automated theorem proving
    - Kowalski, R.A. Predicate logic as a programming language. In Information Processing 74, North-Holland, New York, 1974, pp. 569-574.
- University of Aix-Marseille
  - Natural language processing
    - Roussel, P. Prolog: Manuel Reference et d'Utilisation. Technical Report, Groupe d'Intelligence Artificielle, Marseille-Luminy, Sept. 1975.

The Origins of Prolog

- Fifth Generation computer systems
  - Japan's Fifth Generation Computer Systems
  - THE FIFTH GENERATION PROJECT-A TRIP REPORT

Representing Knowledge

- Two main kinds

1. Facts
2. Rules

Propositions

- *Simple terms*:
  - *Constant*: a symbol that represents an object
  - *Variable*: a symbol that can represent different objects at different times

Propositions

- Propositions:
  - Atomic proposition
  - Compound proposition

- Propositions can be stated in two forms:
  - *Fact*: proposition is assumed to be true
  - *Query*: truth of proposition is to be determined

Propositions

- *Atomic propositions* consist of:
  - A relation (functor)
  - A list of terms of a relation

Propositions

- Compound proposition:
  - Have two or more atomic propositions
  - Propositions are connected by operators

Facts


```
predicate_name(arg1, arg2, …).
```

Prolog - Facts

- `man(jake).`
- `woman(jane).`
- `name(john, doe).`
- `ship(kennedy).`
- `color(kennedy, gray).`

Prolog - Facts


```
like(jake, steak).

like(steak, jake).
```

Prolog - Queries

- *Means of retrieving information from a logic program*

Prolog


```
abraham     mona       clancy          jackie
 |            |        |                  |
 +---------o--+        +-o------o-------o-+
           |             |      |       |
          homer         selma  marge   patty
            |                   |
            +-o-----o-------o---+
              |     |       |
            bart  lisa   maggie
```

Querying Facts

1. Is Homer a male?
2. Is Lisa a male?
3. Who is the child of Abraham?
4. List all the females?

Multi-condition Query

1. Who is the wife of Abraham?
2. Who is the son of Homer?
3. List all the males or fathers.

Prolog


```
orbits(mercury, sun).
orbits(venus, sun).
orbits(earth, sun).
orbits(mars, sun).
orbits(moon, earth).
orbits(phobos, mars).
orbits(deimos, mars).
```

```
?orbits(X, Y), orbits(Y, Z).
```

Prolog - Rules

- *Enables the programmer to define new relationships in terms of existing relationships*

$$\texttt{A :- B}_1\texttt{, B}_2\texttt{, ..., B}_n\texttt{.}$$

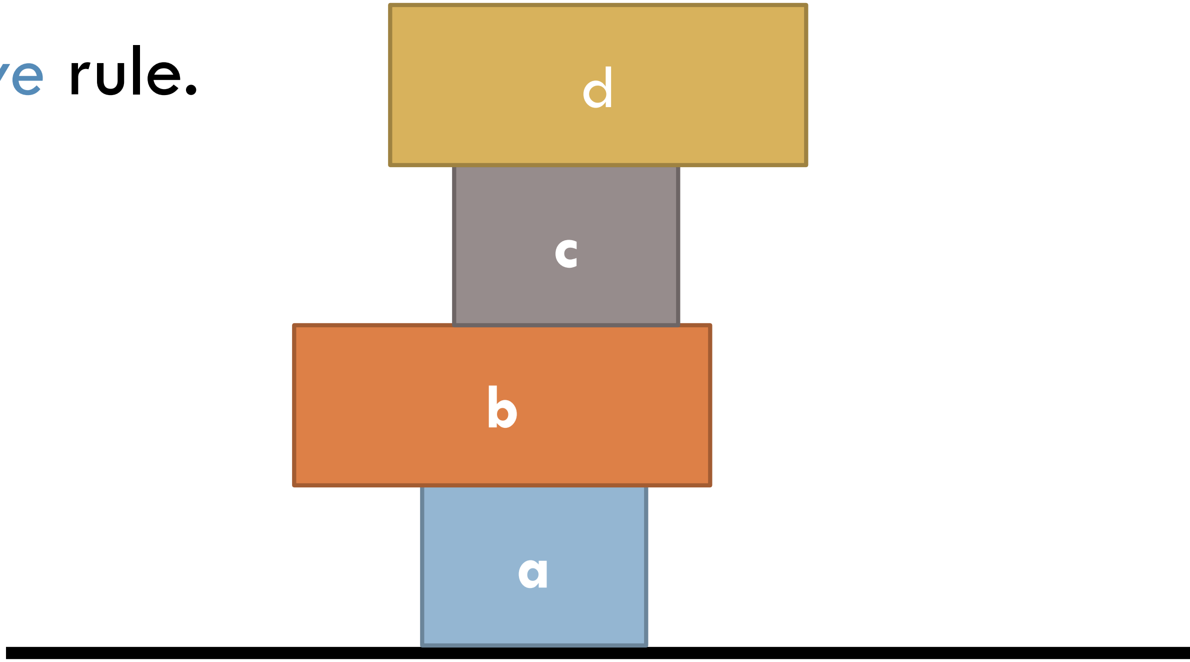
Prolog - Rules


```
planet(P)       :- orbits(P, sun).
satellite(S)    :- orbits(S, P), planet(P).
```

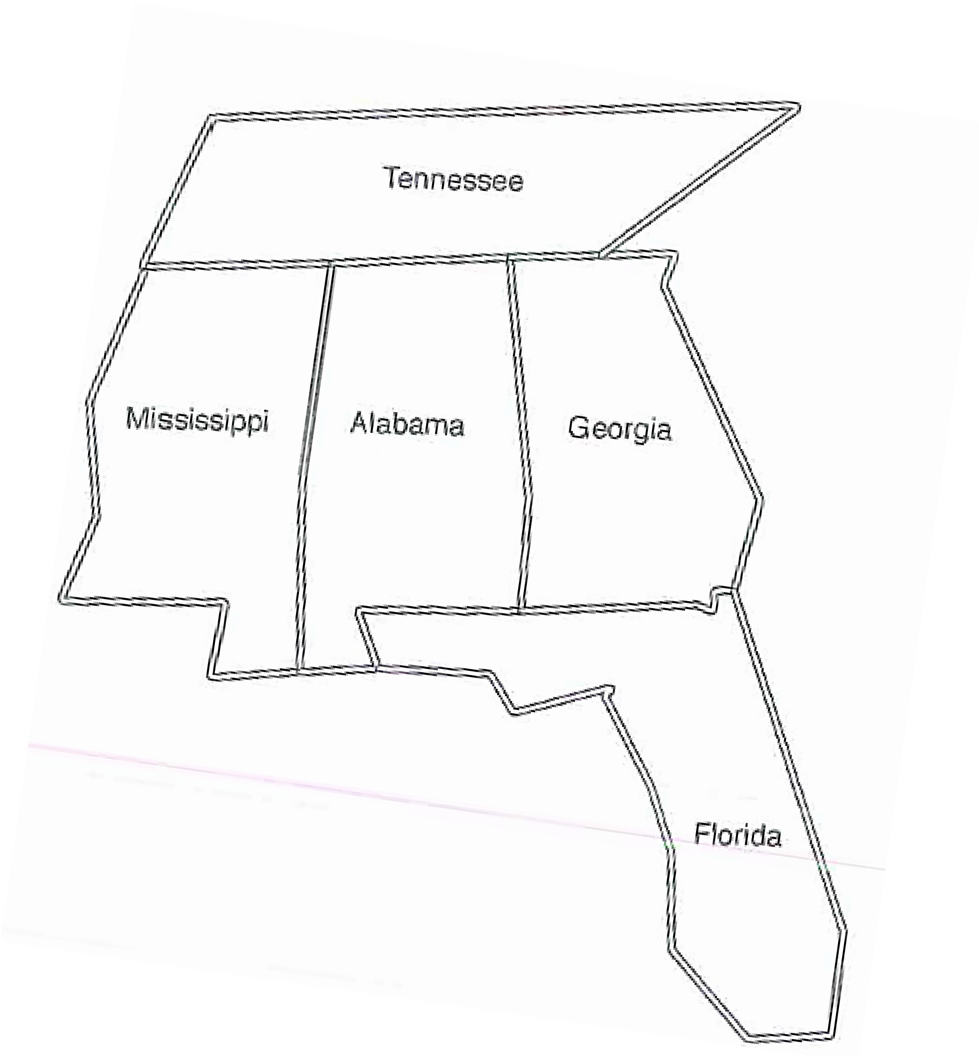
```
? satellite(S).
```

The Block World

1. Represent the block word using the *on* relationship.
2. Define the *above* rule.



Map Coloring



Map Coloring


```
different(red, green).
different(red, blue).
different(red, yellow).
different(green, red).
different(green, blue).
different(green, yellow).
different(blue, red).
different(blue, green).
different(blue, yellow).
different(yellow, blue).
different(yellow, red).
different(yellow, green).
```

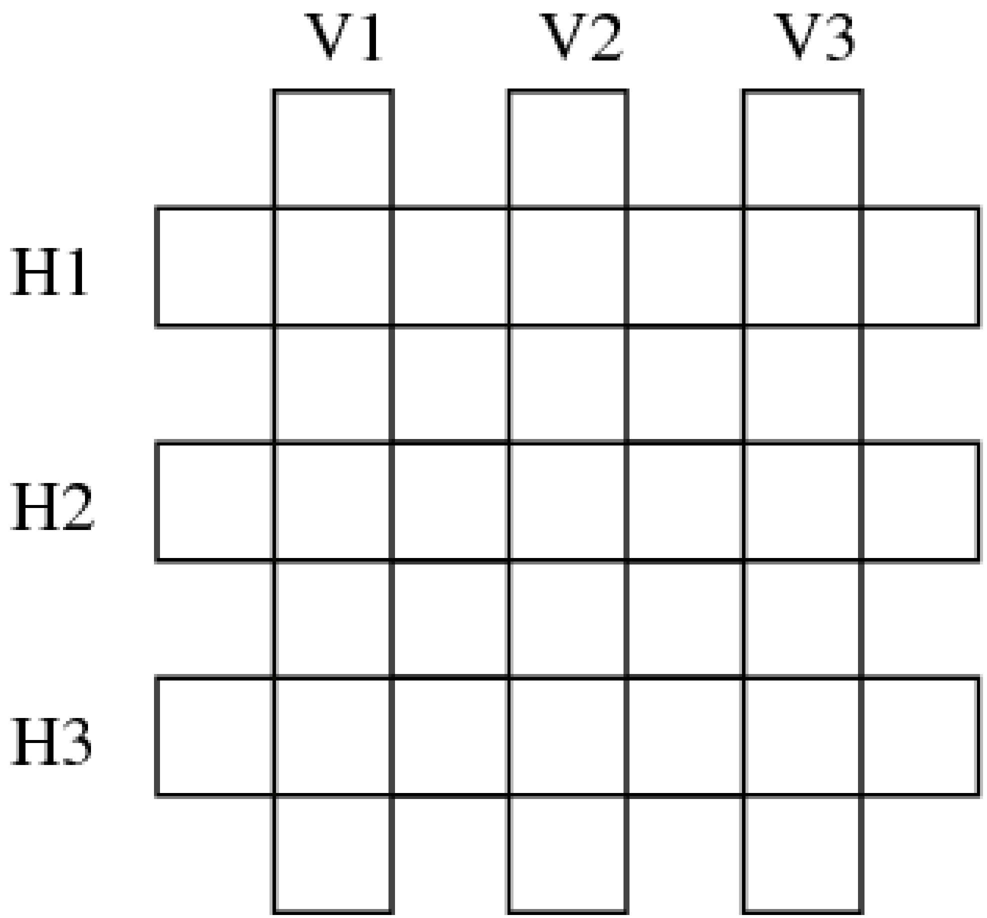
Map Coloring


```
coloring(A, F, G, M, T) :-
                different(A, M),
                different(A, G),
                different(A, T),
                different(A, F),
                different(F, G),
                different(G, T),
                different(M, T).
```

Crossword Puzzle

Words:

- *abalone*
- *abandon*
- *anagram*
- *connect*
- *elegant*
- *enhance*



Crossword Puzzle

		V1		V2		V3	
		a		a		c	
H1	a	b	a	n	d	o	n
		a		a		n	
H2	e	l	e	g	a	n	t
		o		r		e	
H3	e	n	h	a	n	c	e
		e		m		t	

Prolog - Arithmetic

- `is` **operator: takes an arithmetic expression as right operand and variable as left operand**

```
?- X is 1+2.
   X = 3

?-  C= 1+B.
    C= 1+B

?- B=3, C is 1+B, A is B / 17 + C.
```

Prolog – Numeric Comparison

Notation
X =< Y.
X =:= Y.
X =\= Y.
X >= Y
X > Y
X < Y.

Example - Factorial


```
factorial(0, 1).
factorial(N, Fact) :-
    N > 0,
    N1 is N - 1,
    factorial(N1, Fact1),
    Fact is N * Fact1.
```

Prolog – Arithmetic Example


```
speed(ford,100).
speed(chevy,105).
speed(dodge,95).
speed(volvo,80).
time(ford,20).
time(chevy,21).
time(dodge,24).
time(volvo,24).
distance(X,Y) :-  speed(X,Speed),time(X,Time),
                                 Y is Speed * Time.
```

Prolog – Example


```
male(henry).
male(tom).
female(kate).
female(betty).

married(tom, betty).
```

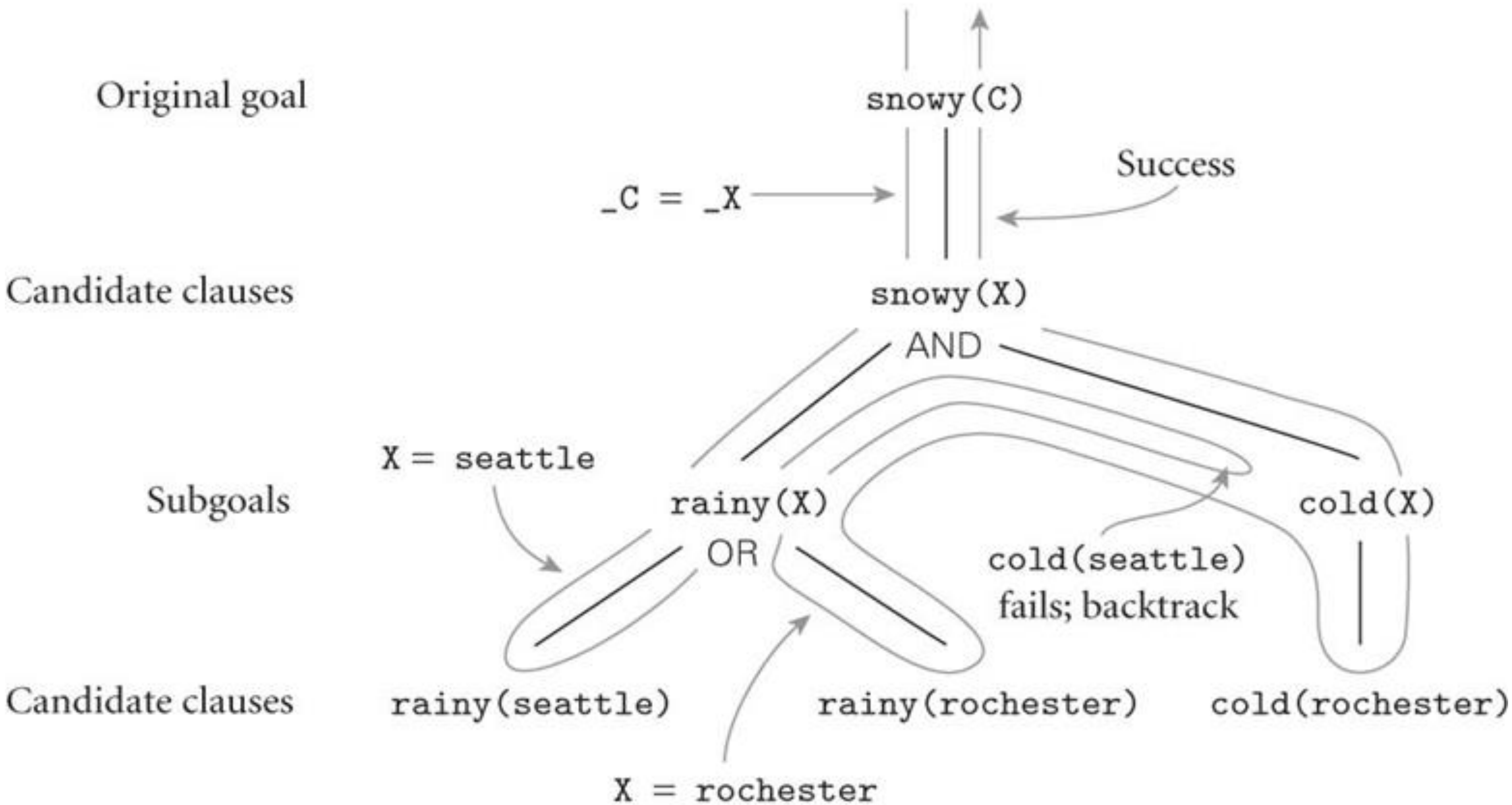
Prolog - Inferencing Process

- Queries are called goals
- If a goal is a compound proposition, each of the facts is a subgoal

- Approaches:
  - *Forward chaining*
  - *Backward chaining*

Prolog


```
rainy(seattle).
rainy(rochester).
cold(rochester).
snowy(X) :- rainy(X), cold(X).
```



Prolog - Inferencing Process

- *Backward chaining (Top-down) resolution*
  - Begin with goal and attempt to find sequence that leads to set of facts in database
    - looks for facts or rules with which the goal can be UNIFIED
  - Uses depth search

Prolog - Unification

- *D*etermining useful values for variables in propositions

- Unification Rules:
  1. A constant unifies only with itself.
  2. Two facts unify if they have the same fact name, the same arity and the attributes unify recursively.
  3. Uninstantiated variable unifies with anything (a variable or a constant).

Prolog - Trace

- Built-in structure that displays instantiations at each step
- *Tracing model* of execution - four events:
  - *Call*: beginning of attempt to satisfy goal
  - *Exit*: when a goal has been satisfied
  - *Redo*: when backtrack occurs
  - *Fail*: when goal fails

Prolog – Trace Example


```
loves(hank,kristen).
loves(phillip,kristen).

jealous(A,B):-  loves(A,C),  loves(B,C).

?- trace.
?- jealous(X,Y).
```

Prolog – Example


```
male(henry).
male(tom).
male(jim).
female(kate).
female(betty).
parent(jim, betty).
married(tom, betty).
% Is this rule correct?
bachelor(P) :- male(P), not(married(P,_)); not(parent(P,_)).
```

Prolog

Are the following two queries the same:

A. `male(X), parent(X, charles).`

B. `parent(X, charles), male(X).`

```
male(james).
male(charles).
male(jack).
male(george).
male(kevin).

female(catherine).
female(elizabeth).
female(sophia).

parent(charles, james).
parent(elizabeth, james).
parent(jack, charles).
parent(catherine, charles).
parent(sophia, elizabeth).
parent(george1, sophia).
```

Prolog


```
| ?- male(X), parent(X, charles).
   Call: (1) male(_328) ?
?  Exit: (1) male(james) ?
   Call: (2) parent(james,charles) ?
   Fail: (2) parent(james,charles) ?
   Redo: (1) male(james) ?
?  Exit: (1) male(charles) ?
   Call: (3) parent(charles,charles) ?
   Fail: (3) parent(charles,charles) ?
   Redo: (1) male(charles) ?
?  Exit: (1) male(jack) ?
   Call: (4) parent(jack,charles) ?
?  Exit: (4) parent(jack,charles) ?
X = jack ?
yes
```

Prolog


```
?- parent(X, charles), male(X).
   Call: (1) parent(_328,charles) ?
?  Exit: (1) parent(jack,charles) ?
   Call: (2) male(jack) ?
?  Exit: (2) male(jack) ?
X = jack ?
yes
```

Prolog


```
edge(a, b).  edge(b, c).  edge(c, d).
edge(d, e).  edge(b, e).  edge(d, f).

path(X, Y) :- path(X, Z), edge(Z, Y).
path(X, X).
```

Exercise

Write the following Prolog programs:

- abs: that determines the absolute value of a number
- max: that determines the maximum of two numbers

Exercise

Is this program correct:

```
max(X,Y,X) :- X >= Y.
max(X,Y,Y).
```

Prolog !

- a goal that always succeeds
- commits Prolog to any choices that were previously made

Prolog - Lists

1. `[a, b, c] = [a, b, c].`
2. `[a, b, c] = [A, B, C].`
3. `[a, a, c] = [B, B, C].`
4. `[a, b, c] = [B, B, C].`
5. `[ ] = [ ].`

Prolog - Lists


```
1. [a, b, c] = [H|T].
2. [a] = [H|T].
3. [] = [H|T].
4. [1, [2, 3]] = [H|T].
5. [a, b, c] = [H1|[H2|T]].
6. [a, b, c, d, e, f, g] = [_, _ |[H|T]].
```

Prolog - Lists

- ***length***: determines the length of a list

```
length([], 0).
length([_|T], L) :- length(T, LT), L is LT + 1.
```

Prolog - Lists

- **append: concatenates two lists together**

```
append([], Ys, Ys).

append([X|Xs], Ys, [X|Zs]) :- append (Xs, Ys, Zs).
```

Prolog – Lists Exercise

- Write the following Prolog programs:

1. ***average***: determines the average of a list of numbers
2. ***member***: determines whether an atom is part of a list

Prolog – Multiple Use

- `append ([], [a, b, c], Z).`
- `append (X, Y, [a, b, c]).`
- `append ([b, c], Y, [a, b, c]).`

Prolog – Negation Problem


```
not(member(X, [mary, fred, barb])).
```

Prolog – Closed World Assumption

- The closed-world assumption
  - Prolog has no knowledge of the world other than its database.

References

- Michael L. Scott, Programming Language Pragmatics, Morgan Kaufmann, 3rd edition, 2009.
- Robert W. Sebesta, Concepts of Programming Languages, Addison Wesley, 10th edition, 2012.
- John C. Mitchell, Concepts in Programming Languages, Cambridge University Press, 2002.
- Leon Sterling and Ehud Shapiro, The Art of Prolog, MIT press, 1986.