Logic Programming: A Declarative Paradigm

Rushikesh K Joshi
IIT Bombay

The Declarative Style

Formula Substitution

we declare multiple formulae

and then when a problem is given, the solving engine can find substitutions

The Functional Declarative Style

We have seen how declarative style programs can be written in the functional style

Functions define **input-output relationship**, mapping the inputs to an output

The style needs the capability of **lazy evaluation** (lazy expansion on need basis)

The Logic Declarative Style

We view the formulae not as functions with input output relationship, but as **symbolic axioms**

They can be used to automatically derive solutions for an unknown problem

Facts and Rules

facts are known relations

and

rules provide ways to infer new relations from the known ones

Facts and rules- an example

```
child(sally,tom).
child(uma,sally).
child(rama,dasharath).
child(lava,rama).
child(kusha,rama).
child(arjuna,pandu).
child(pandu,vichitravirya).
child(vichitravirya,shantanu).
child(abhimanyu,arjuna).
child(parikshit,abhimanyu).
child(janmejay,parikshit).

wife(subhadra,arjuna).
wife(utttara,abhimanyu).
wife(kunti,pandu).
wife(ambalika,vichitravirya).
wife(satyavati,shantanu).
wife(iravati,parikshit).
grandchild(X,Y):-
child(X,Z),child(Z,Y).
grandparent(X,Y):-grandchild(Y,X).
grandmother(X,Y):-
grandparent(Z,Y),wife(X,Z).
granddaughter(X,Y):-
grandparent(Y,Z),wife(X,Z).
```

Check if the rules of inferencing are correct?!

Try List Reversal

last ([A | [ ]], A).

last ([A | L], E) :- last (L, E).

reverse ([A|L], [E|U]) :- last(L,E), reverse(L,U)

We dont have return values, but we can include unknown variables and provide rules to infer them in terms of known facts

[E|U] is the unknown list, which is inferred

Is there something wrong in the above solution?

Find K-th Element

```
kthfind([A|L],K,A,I):-K is I.

kthfind([A|L],K,E,I):-J is I+1, kthfind(L,K,E,J).

kth([A|L],K,E):-kthfind([A|L],K,E,0).
```

path connections

```
connected(delhi,mumbai).
connected(mumbai,bangalore).
connected(bangalore,chennai).
connected(chennai,kolkata).
connected(kolkata,patna).
path(A,B):-connected(A,B).
path(A,C):-connected(A,B),path(B,C).
```

graph with connections

```
connected(delhi,mumbai).
connected(mumbai,bangalore).
connected(bangalore,chennai).
connected(chennai,kolkata).
connected(kolkata,patna).
connected(A,B):-connected(B,A).
```

The Reward Criterion

```
btech(tikku).
btech(ronu).
btech(pinki).
chessplayer(dillis).
footballplayer(ronu).
athelete(pinki).
sportsperson(X):-chessplayer(X).
sportsperson(X):-footballplayer(X).
sportsperson(X):-athelete(X).
gatesreward(X):-btech(X),sportsperson(X).
```

Multiple solutions

```
btech(tikku).
btech(ronu).
btech(pinki).
chessplayer(dillis).
chessplayer(pinki).
footballplayer(ronu).
athelete(pinki).
sportsperson(X):-chessplayer(X).
sportsperson(X):-footballplayer(X).
sportsperson(X):-athelete(X).
gatesreward(X):-btech(X),sportsperson(X).
```

The cut operator: backtracking is pruned

```
btech(tikku).
btech(ronu).
btech(pinki).
chessplayer(dillis).
chessplayer(pinki).
footballplayer(ronu).
athelete(pinki).
sportsperson(X):-chessplayer(X),!.
sportsperson(X):-footballplayer(X),!.
sportsperson(X):-athelete(X),!.
gatesreward(X):-btech(X),sportsperson(X).
```

Where to place the cut?

```
btech(tikku).
btech(ronu).
btech(pinki).
chessplayer(dillis).
chessplayer(pinki).
footballplayer(ronu).
athelete(pinki).
sportsperson(X):-chessplayer(X).
sportsperson(X):-footballplayer(X).
sportsperson(X):-athelete(X).
gatesreward(X):-btech(X),sportsperson(X),!.
```

The cut operator – try this program

```
connected(delhi,mumbai).
connected(mumbai,bangalore).
connected(bangalore,chennai).
connected(chennai,kolkata).
connected(kolkata,patna).
connected(A,B):-connected(B,A),!.
```

Once the goal is satisfied, further search for another solution is aborted

So how does Prolog Engine find the solutions?

The method is called **Unification**

We have clauses ordered in a sequence

some of them are facts with known bound values

some of them are rules with unknown (free) variables given goal as sibling(X,Y)

Start matching the clauses from top to bottom in sequence

you come across a direct fact, solution is found

you come across a rule, expand it with right hand side

We just expanded a goal to be satisfied

now we have more subgoals to satisfy

again try to satisfy each one (left to right order) from top to bottom

but there is one problem...

If I find a match for a subgoal, I am substituting a free variable by a bound value

I should now substitute all occurrences of that variable in the remaining subgoals by this value just found

What if a substitution does not work?

We go back in the subgoals list and try another binding for the free variables

this retry step is called **backtracking**

```
parent(harry,tom).
parent(sally,tom).
parent(harry,june).
parent(sally,june).
parent(harry,john).
parent(jenny,john).
sibling(X,Y):-parent(X,Z),parent(Y,Z).
```

Try goal sibling(X,Y) with this program

```
parent(harry,tom).
parent(sally,tom).
parent(harry,june).
parent(sally,june).
parent(harry,john).
parent(jenny,john).
sibling(X,Y):-
        parent(X,Z),parent(Y,Z),X\=Y.
```

Try goal sibling(X,Y) with this program