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

Length

length (nillist) = 0

length (l) = 1 + length (tail(l))

Member

member (e,nillist) = false

member (e, head(l)==e) = true

member (e, l) =
member (e, tail(l))

Push back

pushback(nillist,e) = cons(e,nillist)

pushback(l,e) = cons
(head(l),pushback(tail(l),e))

Append

append(l,nillist) = l

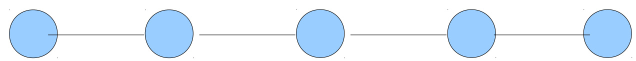
append (l1,l2) =
append(push_back(l1,head(l2)),
tail(l2))

Map

map (f,nillist)=nillist

map (f,l) = cons (f(head(l)),
map (f,tail(l)))

Reversed Map



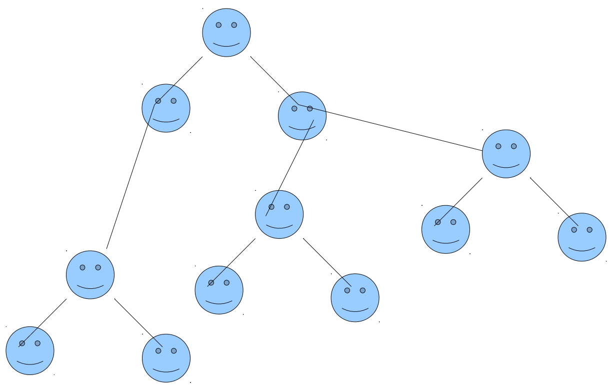
(a, (b , (v , (d , e))))

Map f to l in reverse order

rmap (f, l) = ??

f can be any single argument function such as print, square, isprime etc

Binary Tree



Tree = (value, (left, right))
left = nillist | Tree
right = nillist | Tree

dftmap (f, tree) = ??
bftmap (f,tree) = ??

Execution of Programs written in Declarative Style

Compare it with formula substitution

We have a set of formulae

And a problem to be solved

We solve the problem by picking up a formula, substitute it for a pattern in the problem, and continue this till we reduce the problem to a solution

Imperative vs Declarative

Declarative Programming focus on simply declaring.

Imperative Program works out a solution using control constructs and variables

Logic Programming
The Horn Clause Logic

Example

$$S\ (A,B,..)\ :\text{-}\ u\ (A,..)\ \Lambda\ v\ (A,B,..)\ \Lambda\ w\ (B,..)\ \Lambda\ .....$$

Membership in Prolog

```
member(P,[P|[ ]]).
member(P,[P|L]).
member(P,[Q|L]):-member(P,L).
```

Lists in Prolog

use square brackets to indicate a Head and a Tail

Example

[H | T]

H is a head element
T is the tail

Example

[A | B]

A is the head element
B is the tail

Nil list in Prolog

[ ]

A list with 1 element

[A | [ ]]

A Declarative Fact

member (P, [P | [ ]]).

P is a member in a list which has P in its head, with tail as nillist

A Declarative Fact

Member (P, [P | L]).

P is a member in a list which has P in its head

The Recursive Declarative Fact

member(P, [Q | L]) :-
member (P, L).

Compare the LP style with Declarative Style in Functional Programming

```
member(P,[P|[ ]]).
member(P,[P|L]).
member(P,[Q|L]):-member(P,L).
```

```
member (e,nillist) = false
member (e, head(l)==e) = true
member (e, l) =  member (e, tail(l))
```