POPL2 class (2020-04-16)

Srijith P K

Prolog

- Logic programming
- A single data structure as the foundation of the language
- Simple syntax. p(X) :- q(X).
- Program{data equivalence.
- Weak typing : type of a variable in Prolog only becomes relevant when particular operations are performed
  - finding program bugs more difficult.
- Incremental program development.
- Extensibility. A Prolog system can be extended and modied.

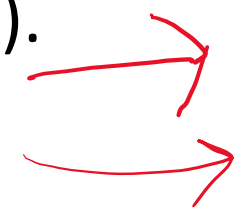
Some Prolog examples

- Father(X) = X
- ‘mia’ = mia vs. ‘2’ = 2
- vertical(line(point(X,Y),point(X,Z))).
- horizontal(line(point(X,Y),point(Z,Y))).

Recursion

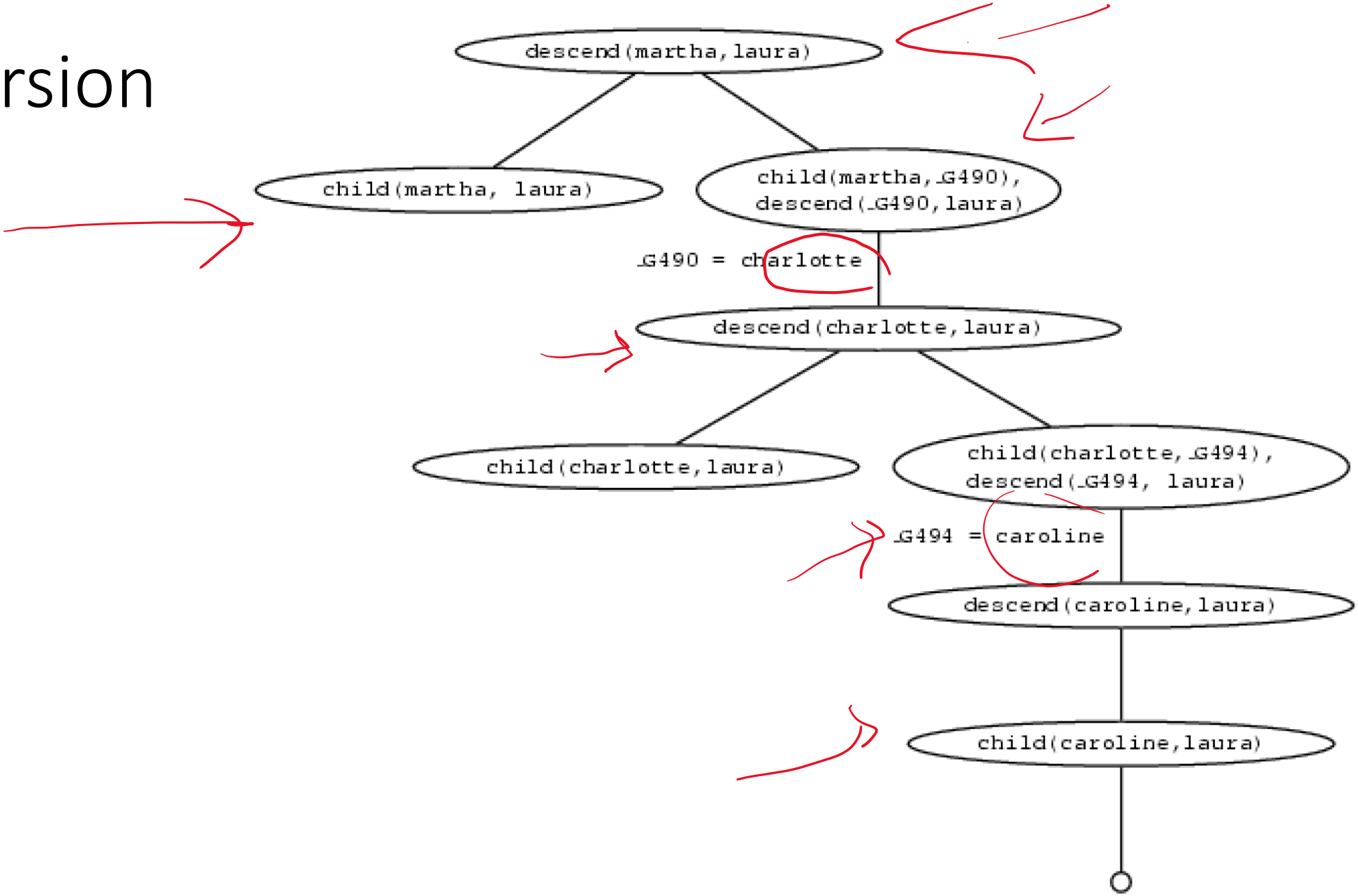
- child(charlotte,caroline).
- child(caroline,laura).
- descend(X,Y) :- child(X,Y).
- descend(X,Y) :- child(X,Z), child(Z,Y).

```
child(martha,charlotte).
child(charlotte,caroline).
child(caroline,laura).
child(laura,rose).
```



```
descend(X,Y) :- child(X,Y).
descend(X,Y) :- child(X,Z),
descend(Z,Y).
```

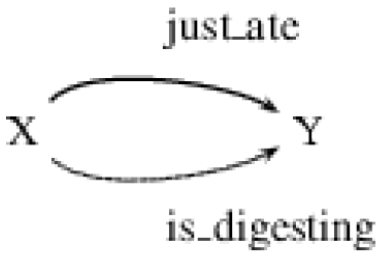
Recursion

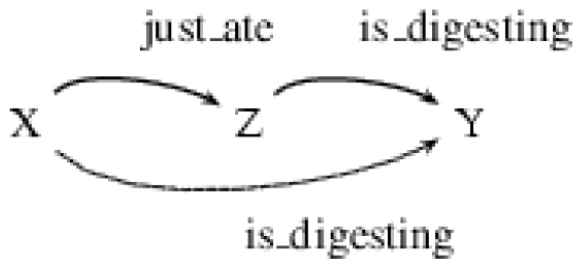

descend(martha,laura)
child(martha, laura)
child(martha,_G490),
descend(_G490,laura)
_G490 = charlotte
descend(charlotte,laura)
child(charlotte,laura)
child(charlotte,_G494),
descend(_G494, laura)
_G494 = caroline
descend(caroline,laura)
child(caroline,laura)

Example

```
is_digesting(X,Y) :- just_ate(X,Y).
is_digesting(X,Y) :-
        just_ate(X,Z),
        is_digesting(Z,Y).

just_ate(mosquito,blood(john)).
just_ate(frog,mosquito).
just_ate(stork,frog).
```

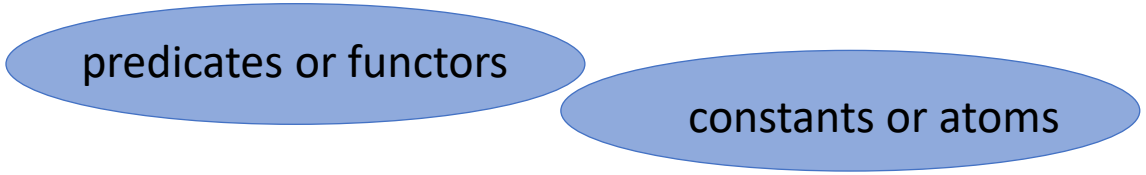




```
mother(wilhelmina,juliana).
mother(juliana,beatrix).
mother(juliana,christina).
mother(juliana,irene).
mother(juliana,margriet).
mother(beatrix,friso).
mother(beatrix,alexander).
mother(beatrix,constantijn).
mother(emma,wilhelmina).

father(hendrik,juliana).
father(bernard,beatrix).
father(bernard,christina).
father(bernard,irene).
father(bernard,margriet).
father(claus,friso).
father(claus,alexander).
father(claus,constantijn).
father(willem,wilhelmina).

queen(beatrix).
queen(juliana).
queen(wilhelmina).
queen(emma).
king(willem).
```



Variables has to be upper-case letter or underscore.

Prolog rules

```
parent(X, Y) :- mother(X,Y).
parent(X, Y) :- father(X,Y).
ruler(X) :- queen(X).
ruler(X) :- king(X).
```

```
predecessor(X,Y) :-
parent(X,Y),
ruler(X),
ruler(Y).
predecessor(X,Y) :-
ruler(X),
parent(X,Z),
predecessor(Z,Y).
```

Prolog queries

```
?- parent(beatrix,alexander).
?- parent(beatrix,X).
?- parent(beatrix,emma).
?- parent(X,Y), ruler(Y).
?- predecessor(X,beatrix).
```

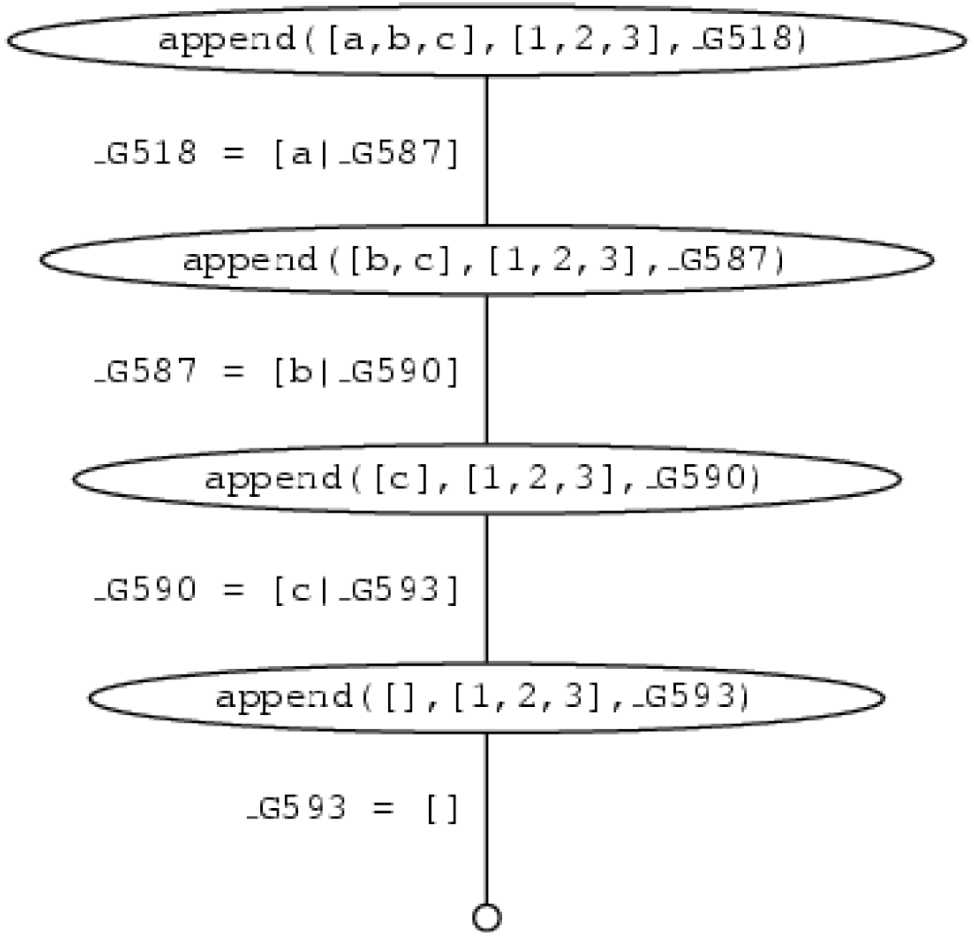
Append

```
append([],L,L).
append([H|T],L2,[H|L3]) :-
append(T,L2,L3).
```

```
append([a, b, c], [1, 2, 3], _G518)
append([b, c], [1, 2, 3], _G587)
append([c], [1, 2, 3], _G590)
append([], [1, 2, 3], _G593)
append([], [1, 2, 3], [1, 2, 3])
append([c], [1, 2, 3], [c, 1, 2, 3])
append([b, c], [1, 2, 3], [b, c, 1, 2, 3])
append([a, b, c], [1, 2, 3], [a, b, c, 1, 2,
3])
```

X = [a, b, c, 1, 2, 3]

yes



Reverse

- Reverse using append

```
naiverev([],[]).
naiverev([H|T],R) :- naiverev(T,RevT),append(RevT,[H],R).
```

- Reverse using accumulators

```
accRev([H|T],A,R) :- accRev(T,[H|A],R).
accRev([],A,A).


rev(L,R) :- accRev(L,[],R).
```