26 Sept PL Amrigar

10:50 am

* Prolog contd.
* Ancestor rules

parent → mother
→ father

ancestor → parent
X Y  X Y

ancestor → parent, ancestor
X2  X1 Y  Y2

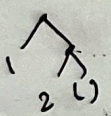
* Ancestor Log
* Prolog Data - numbers, atoms, variables, anonymous variables

primitive data

* Non primitive data - structures
* Syntactic sugar 1+2 → +(1,2)
operator overloading → C++
defining your own f's with it's

- Lists (1 2) in scheme

in prolog, the same just not the dot.
it's '[1]'




'[1]'(a,b) is denoted as [a|b] ≡ (a b)

* Pattern matching, it's symmetric

LHS f.{0:X, 1:a}
RHS f:{0:b, 1:Y}

[X|Y] → pair with head X & tail Y

pattern matching eg. contd.

[x|Y] → [a, b]

x → a

Y → [b]

[[a,b], x|z] = [a, [x|z], a, b]

? - [x, [a,b], x|z] = [a, [x|z], a, b]

x = a

z = [b]

* prolog Program

Predicate → which has a boolean value

12 12

recursively append the head of the 2nd to the tail of the 1st

```
(define (my append ls1 ls2)
  (if (null? ls1)
      ls2
      (cons (car ls1)
            (my append (cdr ls1) ls2))))
```

for scheme writing a fn to return a result

→ but we don't have fn's in Prolog

it's just relation b/w inputs & outputs

```
my append([], x, z):-
    z = x.
```

we use pattern matching

```
my_append([], x, x).
my_append([x|xs], ys, [x|zs]):-
    my_append(xs, ys, zs).
```