Sintaxis y Semántica del Lenguaje

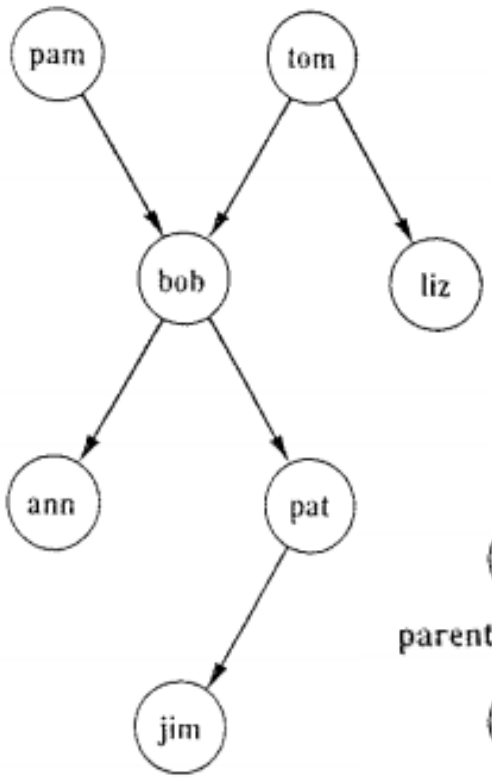
Programación con PROLOG

Prolog

- Mecanismos básicos
  - Pattern Matching
  - Estructuración de datos basada en árboles
  - Backtracking automático

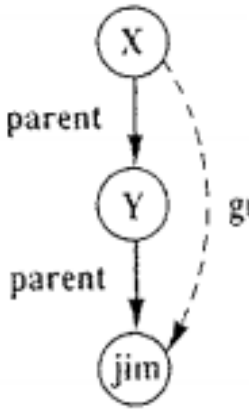
Definición de Relaciones mediante Hechos

```
parent( pam, bob).
parent( tom, bob).
parent( tom, liz).
parent( bob, ann).
parent( bob, pat).
parent( pat, jim).
```



- Seis cláusulas

```
?- parent( bob, pat).
?- parent( X, liz).
?- parent( X, Y).
```




```
parent( Y, jim)   and   parent( X, Y)
```

```
parent( X, Y)   and   parent( Y, jim)
```

Definición de Relaciones mediante Reglas

```
female( pam).
male( tom).
male( bob).
female( liz).
female( pat).
female( ann).
male( jim).
```

For all X and Y,
Y is an offspring of X if
X is a parent of Y.

```
offspring( Y, X) :- parent( X, Y).
```

head body

For all X and Y,
X is the mother of Y if
X is a parent of Y and
X is a female.

```
mother( X, Y) :- parent( X, Y), female( X).
```

```
sister( X, Y) :-
    parent( Z, X),
    parent( Z, Y),
    female( X),
    different( X, Y).
```

Reglas Recursivas

For all X and Z,
X is a predecessor of Z if
X is a parent of Z.

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

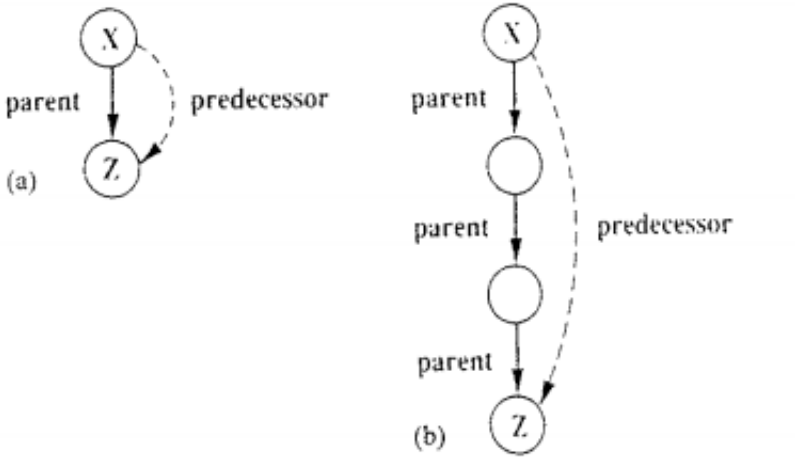
```
predecessor( X, Z) :-
  parent( X, Z).

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

```
predecessor( X, Z) :-
  parent( X, Y1),
  parent( Y1, Y2),
  parent( Y2, Z).

predecessor( X, Z) :-
  parent( X, Y1),
  parent( Y1, Y2),
  parent( Y2, Y3),
  parent( Y3, Z).
```

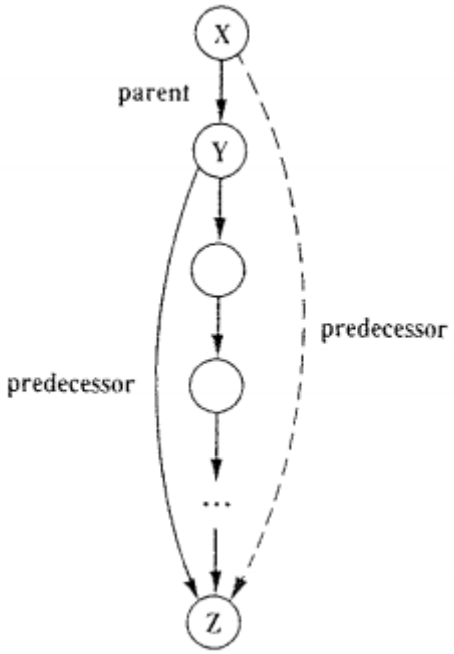
Problema!!!!



Reglas Recursivas

For all X and Z,
X is a predecessor of Z if
there is a Y such that
(1) X is a parent of Y and
(2) Y is a predecessor of Z.

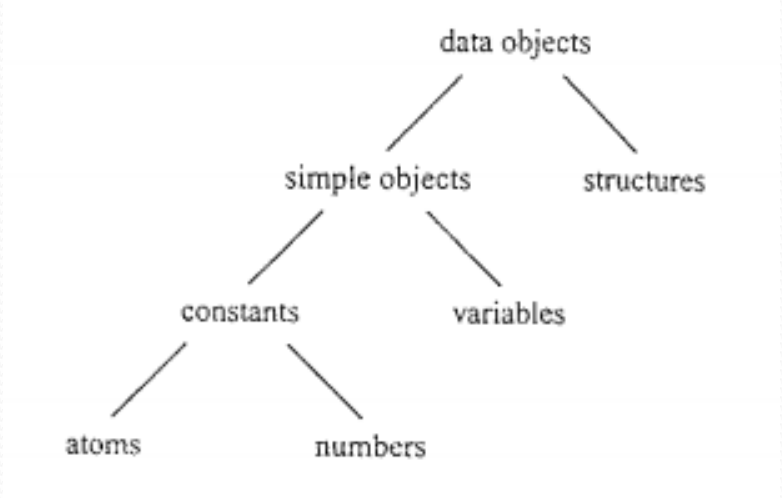
```
predecessor( X, Z)  :-
   parent( X, Y),
   predecessor( Y, Z).
```




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

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

Data Objects



Scope: UNA CLAUSULA

- Atomos

anna
nil
x25
x_25
x_25AB
x_
x___y
alpha_beta_procedure
miss_Jones
sarah_jones

'Tom'
'South_America'
'Sarah Jones'

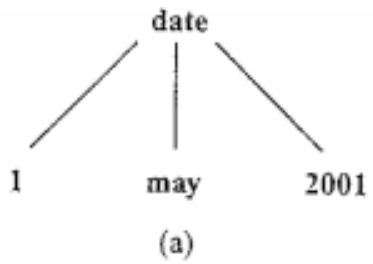
Variables

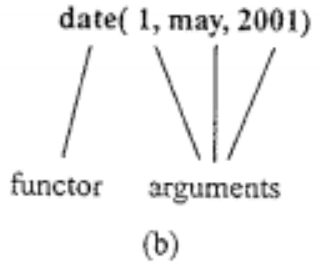
X
Result
Object2
Participant_list
ShoppingList
_x23
_23

```
hasachild( X) :- parent( X, _).
somebody_has_child :- parent( _, _).
```

Estructuras

date(1, may, 2001)





- Matching

(1) If S and T are constants then S and T match only if they are the same object.

(2) If S is a variable and T is anything, then they match, and S is instantiated to T. Conversely, if T is a variable then T is instantiated to S.

(3) If S and T are structures then they match only if

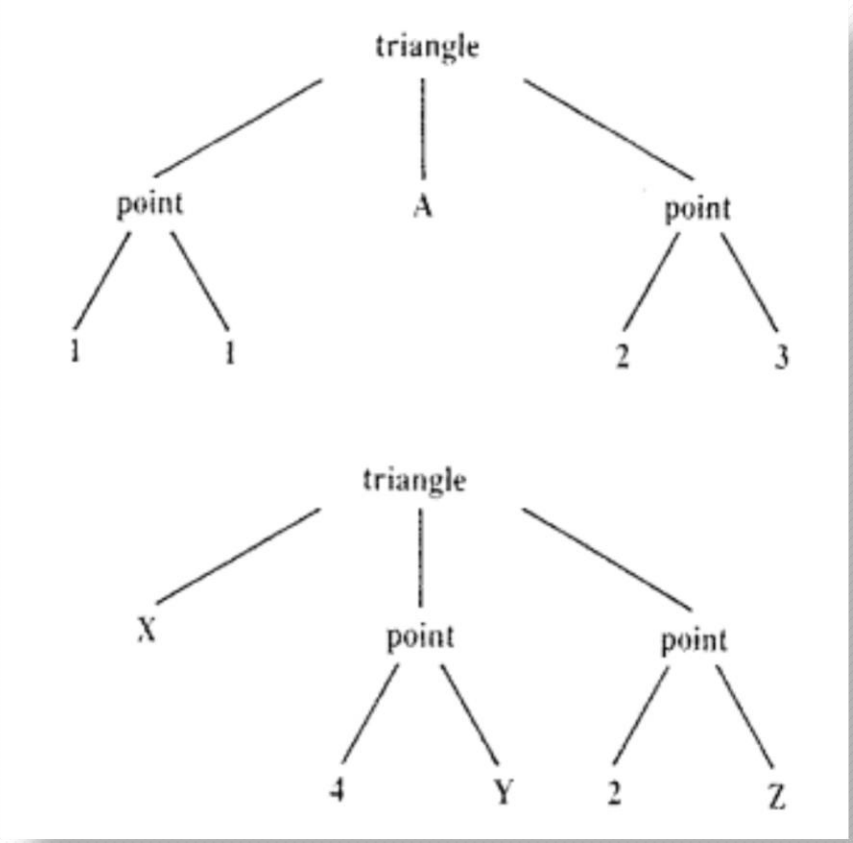
(a) S and T have the same principal functor, and

(b) all their corresponding components match.

The resulting instantiation is determined by the matching of the components.

date(D, M, 2001) and date(D1, may, Y1)

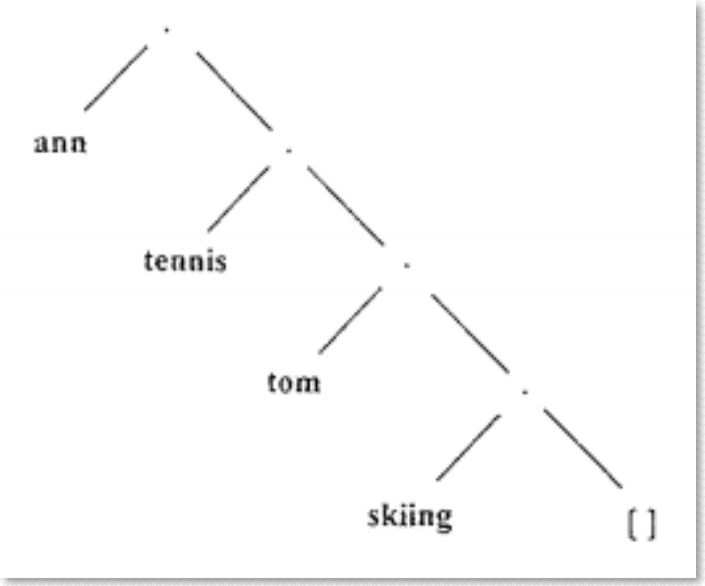
Matching



Listas

- Secuencias de items: [ann, tennis, tom, skiing]
- ***Primer item:*** Head ***Resto de items:*** Tail

.(Head, Tail) → .(ann, .(tennis, .(tom, .(skiing, [ ]))))



[a,b,c] = [a | [b,c]] = [a,b | [c]] = [a,b,c | [ ]]

Operaciones en listas

```
member( X, L)

member( b, [a,b,c] )
```

```
member( X, [X | Tail] ).

member( X, [Head | Tail] ) :-
  member( X, Tail).
```

```
conc( L1, L2, L3)

conc( [a,b], [c,d], [a,b,c,d] )
```

```
conc( [ ], L, L).

conc( [X | L1], L2, [X | L3] ) :-
  conc( L1, L2, L3).
```

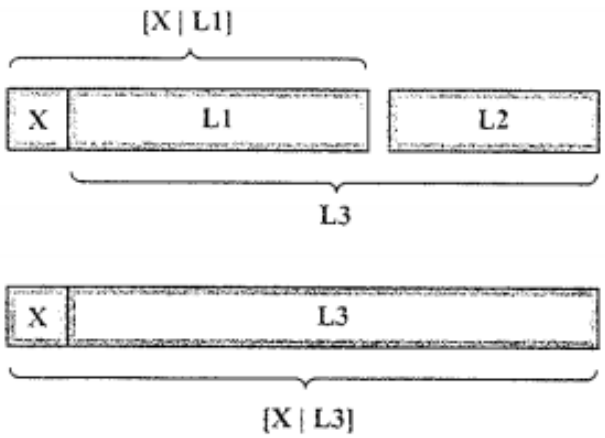
```
?- conc( Before, [may | After],
         [jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec] ).

Before = [jan,feb,mar,apr]
After = [jun,jul,aug,sep,oct,nov,dec].
```

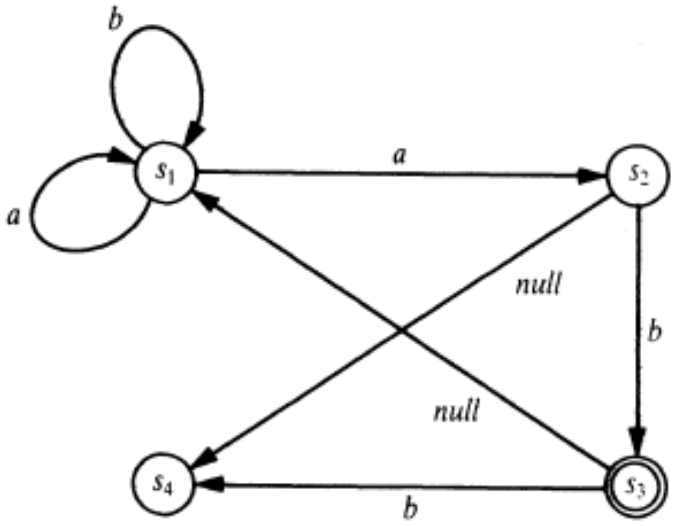
```
?- conc( _, [Month1,may,Month2 | _],
         [jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec] ).

Month1 = apr
Month2 = jun
```

```
member1( X, L) :-
  conc( _, [X | _], L).
```



Ejemplo AFN en Prolog (PL)




```
final( s3).

trans( s1, a, s1).
trans( s1, a, s2).
trans( s1, b, s1).
trans( s2, b, s3).
trans( s3, b, s4).

silent( s2, s4).
silent( s3, s1).
```

```
?- accepts( s1, [a,a,a,b] ).

yes
```

```
?- accepts( S, [a,b] ).

S = s1;

S = s3
```

```
?- String = [_, _, _], accepts( s1, String).

String = [a,a,b];

String = [b,a,b];

no
```

```
accepts( S, [] ) :-
  final( S).

accepts( S, [X | Rest] ) :-
  trans( S, X, S1),
  accepts( S1, Rest).

accepts( S, String) :-
  silent( S, S1),
  accepts( S1, String).
```