Logic and Constraint Programming

PROLOG

Prof. Fabrizio Fornari

May 13, 2022

Extend the “family” program

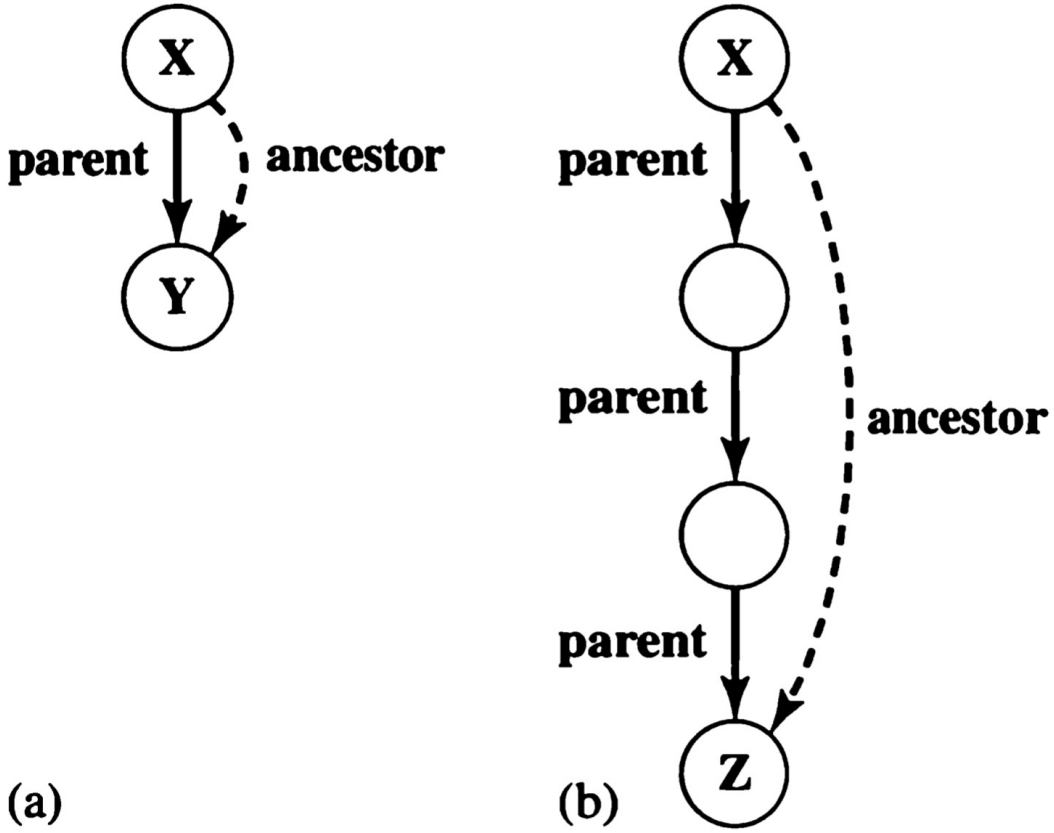
Let us add one more relation to our family program, the **ancestor** relation.

1st rule:

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

2nd rule:

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



Extend the “family” program

Let us add one more relation to our family program, the **ancestor** relation.

1st rule:

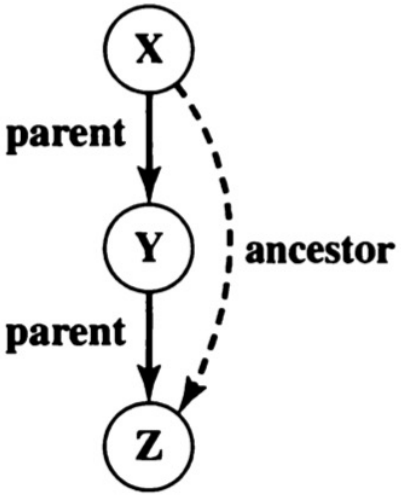
```
ancestor( X, Z) :-
    parent( X, Z).
```

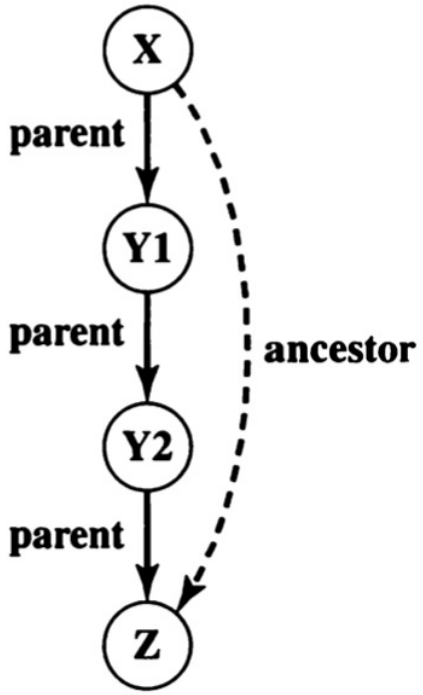
2nd rule:

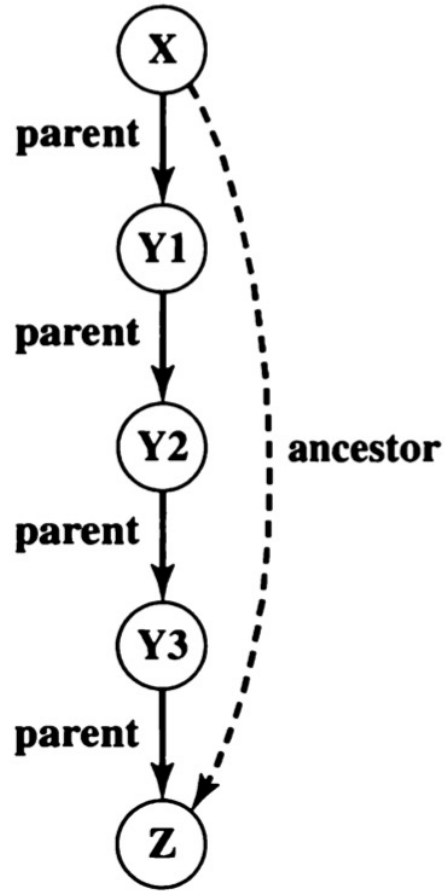
```
ancestor( X, Z) :-
    parent( X, Y),
    parent( Y, Z).
```

3rd rule:

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







Extend the “family” program

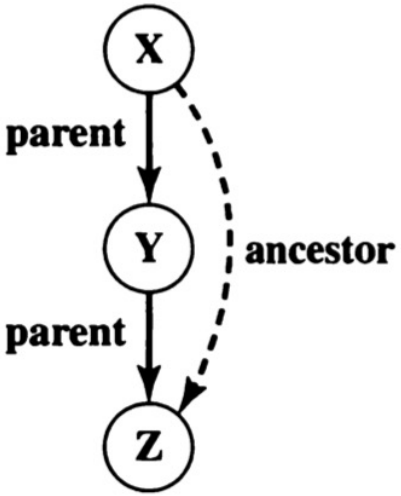
1st rule:
ancestor(X, Z) :-
parent(X, Z).

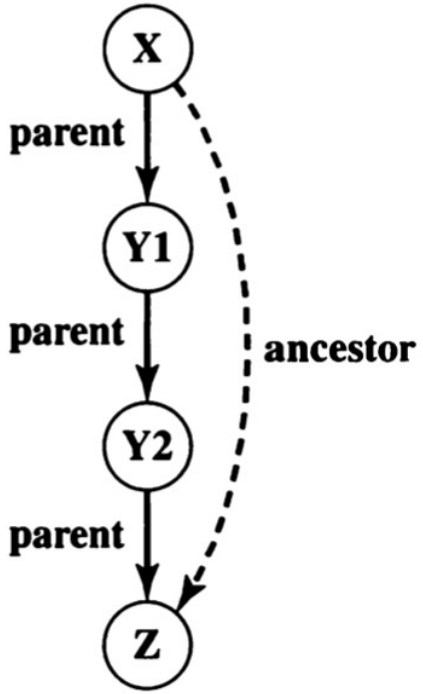
2nd rule:
ancestor(X, Z) :-
parent(X, Y),
parent(Y, Z).

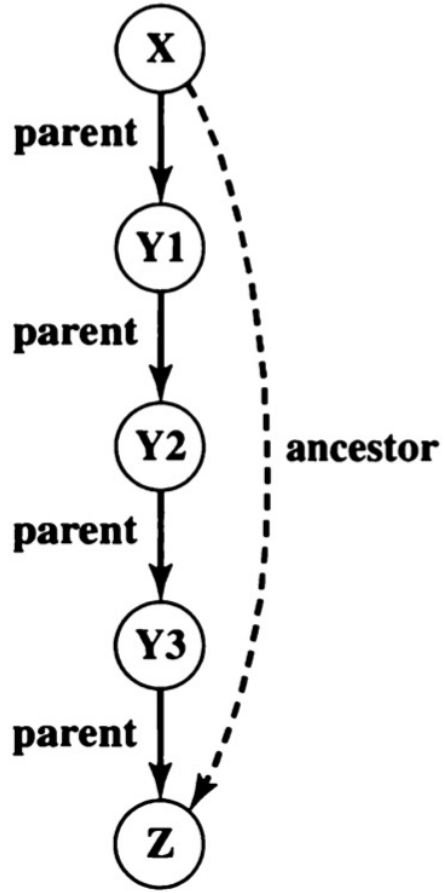
3rd rule:
ancestor(X, Z) :-
parent(X, Y1),
parent(Y1, Y2),
parent(Y2, Z).

4th rule:
ancestor(X, Z) :-
parent(X, Y1),
parent(Y1, Y2),
parent(Y2, Y3),
parent(Y3, Z).

Nth rule:
…







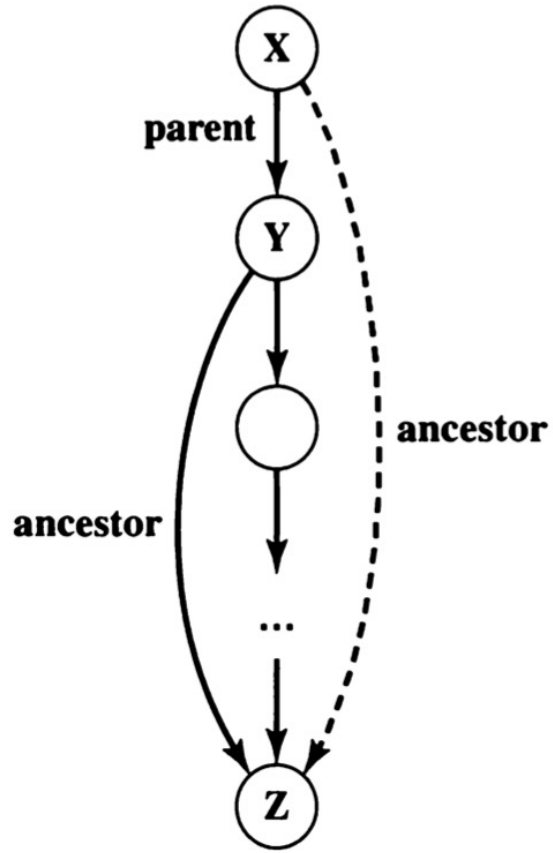
This program is lengthy and it only works to some extent.

Recursive rules

Formulation of the **ancestor** relation.

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

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



Recursive rules

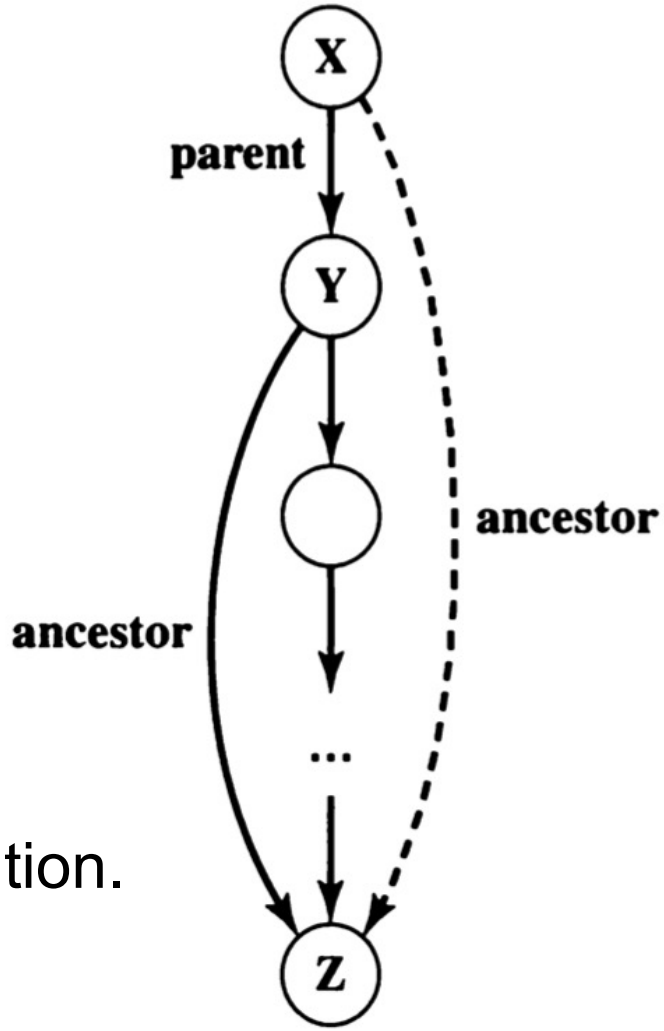
Ancestor relation program

1st rule:

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

2nd rule:

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



The key is the use of **ancestor** itself in its definition.
Such a definition is called ***recursive*** definition.

Recursive rules

Are logically correct and understandable.

Prolog can easily use recursive definitions.

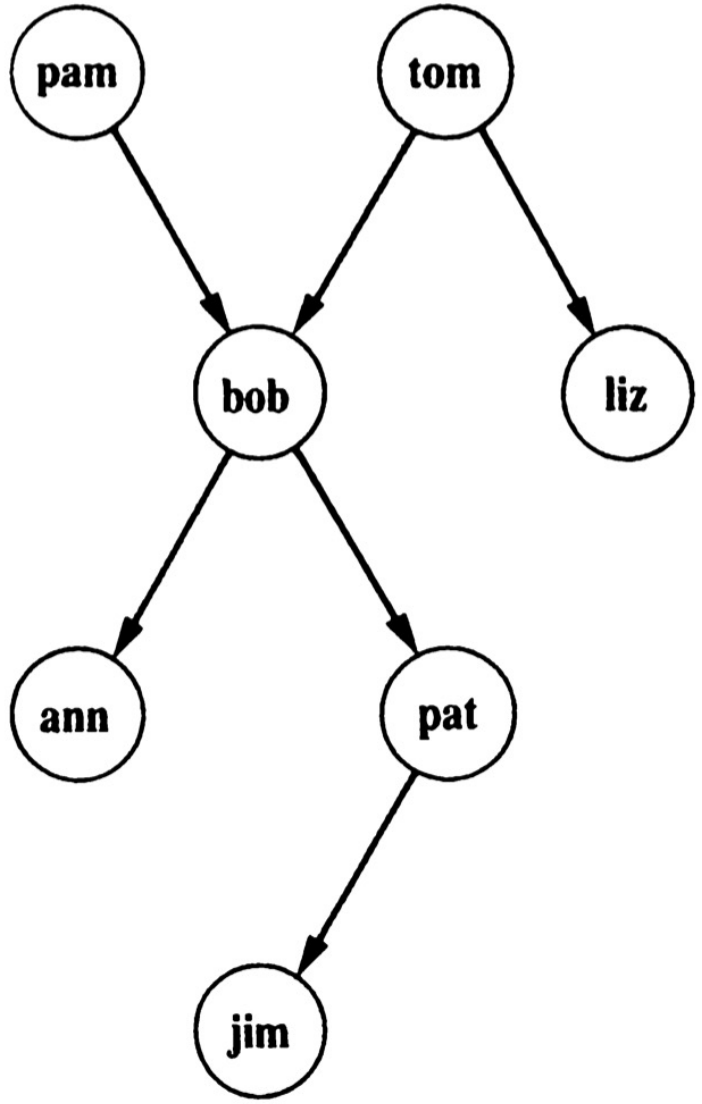
Recursive programming is, one of the fundamental principles of programming in Prolog.

It is necessary for solving task of significant complexity.

Ancestor program

Let us ask Prolog: Who are Pam's successors?

How can we formulate such a question?



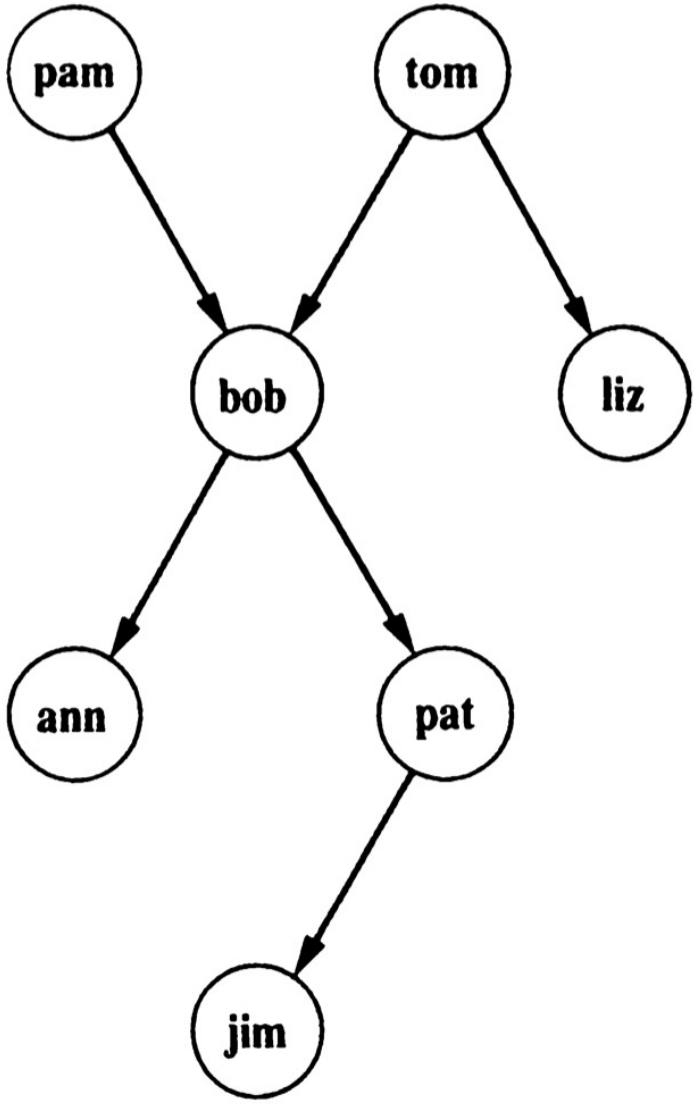
Ancestor program

Let us ask Prolog: Who are Pam's successors?

?- ancestor(pam, X).

X = bob;
X = ann;
X = jim

Prolog's answers are correct and they logically follow from our definition of the **ancestor** and the **parent** relation.



Ancestor program

```
parent(pam,bob).        % Pam is a parent of Bob
parent(tom,bob).
parent(tom, liz).
parent(bob, ann).
parent(bob, pat).
parent(pat, jim).

female(pam).            % Pam is female
female(liz).
female(pat).
female(ann).

male(tom).              % Tom is male
male(bob).
male(jim).

mother(X, Y) :-         % X is the mother of Y if
   parent(X,Y),         % X is a parent of Y and
   female(X).           % X is female
```

The whole set of clauses about the same relation is called a ***procedure***.

The *ancestor* procedure

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

sister( X, Y):-
   parent( Z, X),
   parent( Z, Y),
   female( X),
   X \= Y.

ancestor( X, Z):-
   parent( X, Z).

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

Comments

```
% X is grandparent of Z if
%X is parent of Y and
% Y is parent of Z

% X is sister of Y if

% X and Y have the same parent and
% X is female and
% X and Y are different

% Rule a1: X is ancestor of Z

% Rule a2: X is ancestor of Z
```

Comments in Prolog:
/* This is a comment */
% This is also a comment

How Prolog answers questions

To answer a question, Prolog tries to satisfy all the goals.

To satisfy a goal means to to demonstrate that the goal is true, assuming that the relations in the program are true.

In other words, it means to demonstrate that the goal *logically follows* from the facts and rules in the program.

If the question contains variables, Prolog also has to find what are the particular objects (in place of the variables) for which the goals are satisfied.

If Prolog cannot demonstrate for any instantiation of variables that the goals logically follow from the program, then Prolog's answer to the question will be 'false'.

How Prolog answers questions

In mathematical terms:

- facts and rules as a set of *axioms*,

- user's question as a *conjectured theorem*;

Prolog tries to prove this theorem – that is, to demonstrate that it can be logically derived from the axioms.

How Prolog answers questions

A classical example about Socrates and fallible men.

The axioms are:
All men are fallible
Socrates is a man.

A theorem is:
Socrates is fallible.

The first axiom can be rewritten as:
For all X, if X is a man then X is fallibe.

```
fallible( X) :- man( X).          % All men are fallible
man( sorates).                    % Socrates is a man
```

```
?- fallible( socrates).           % Socrates is fallible?
```

How Prolog answers questions

A more complicated example from the family program:
?-ancestor(tom, pat).

X = tom, Z = pat

1st rule:

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

2nd rule:

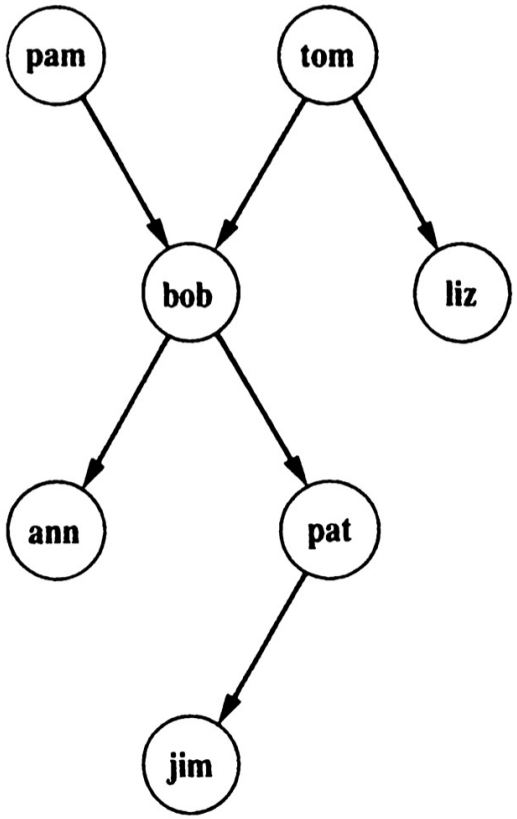
```
ancestor( X, Z) :-
    parent( X, Y),
    ancestor( Y, Z).
```

How Prolog answers questions

A more complicated example from the family program:
?-ancestor(tom, pat).

X = tom, Z = pat

1st rule:
ancestor(X, Z) :-
    parent(X, Z).



The goal ancestor is replaced with parent(tom, pat)

No clause in the program matches the goal parent(tom,pat)

How Prolog answers questions

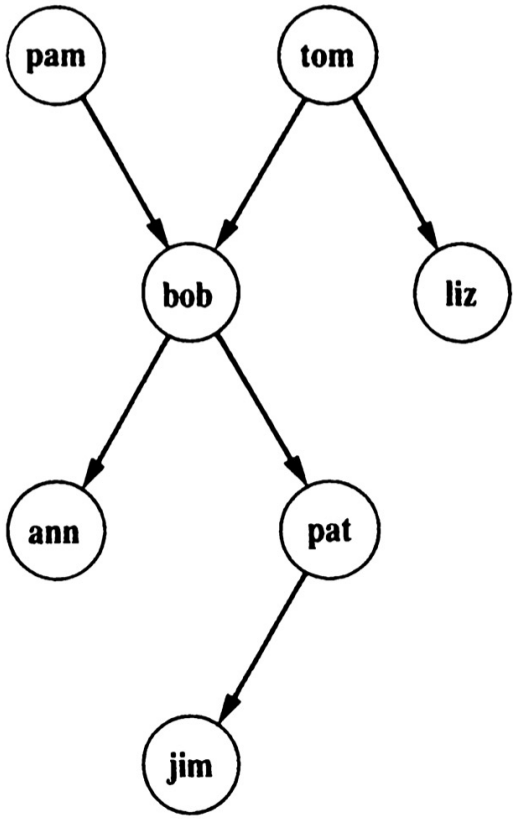
A more complicated example from the family program:
?-ancestor(tom, pat).

X = tom, Z = pat

2nd rule:
ancestor(X, Z) :-
    parent(X, Y),
    ancestor(Y, Z).

Prolog tries the 2nd rule.

parent(tom, Y),
ancestor(Y, pat).



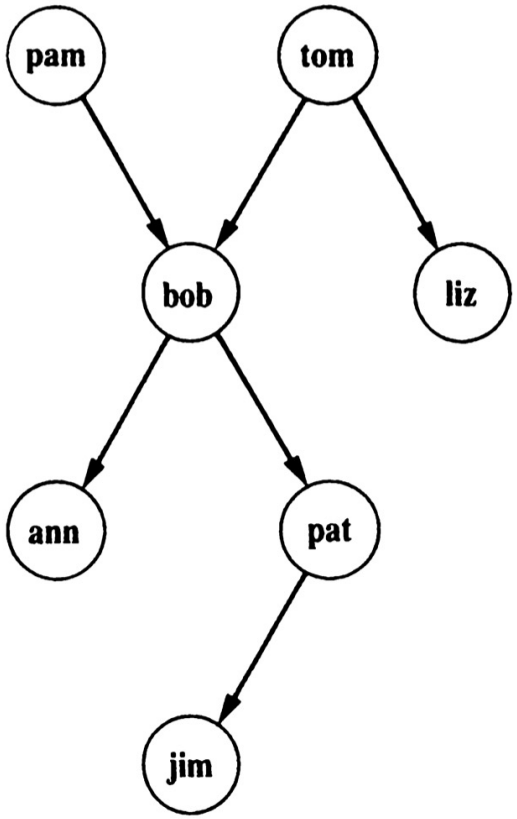
How Prolog answers questions

A more complicated example from the family program:
?-ancestor(tom, pat).

Prolog tries the 2nd rule.

parent(tom, Y),
ancestor(Y, pat).

The first goal matches two facts:
parent(tom,bob) and ***parent(tom,liz)***.



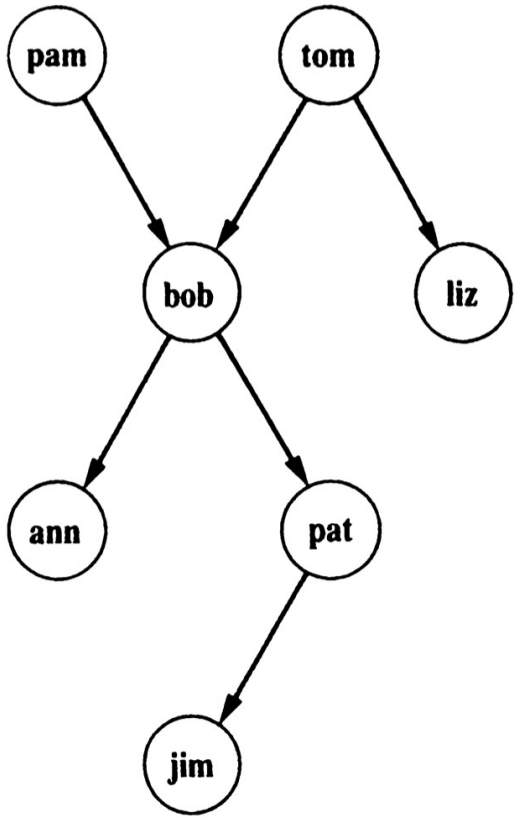
How Prolog answers questions

A more complicated example from the family program:
?-ancestor(tom, pat).

Prolog tries the 2nd rule.

parent(tom, Y),
ancestor(Y, pat).

The first goal matches two facts:
parent(tom,bob) and ***parent(tom,liz)***.



NOTE: Prolog first tries to use the fact that appears first in the program.

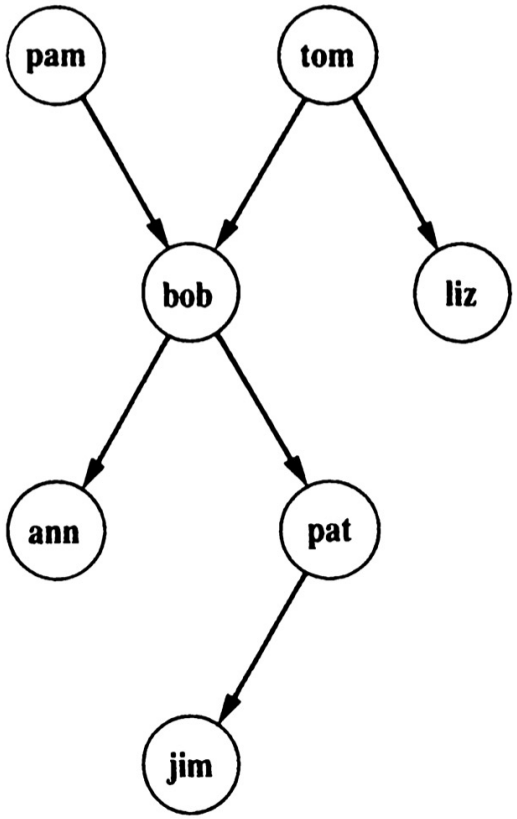
This forces Y to become instantiated to **bob**. The second goal becomes:
ancestor(bob, pat).

How Prolog answers questions

A more complicated example from the family program:
?-ancestor(tom, pat).

1st rule:
ancestor(X', Z') :-
parent(X', Z').

Note the '



Prolog tries to apply the first rule over:
ancestor(bob, pat).

X' = bob,
Z' = pat

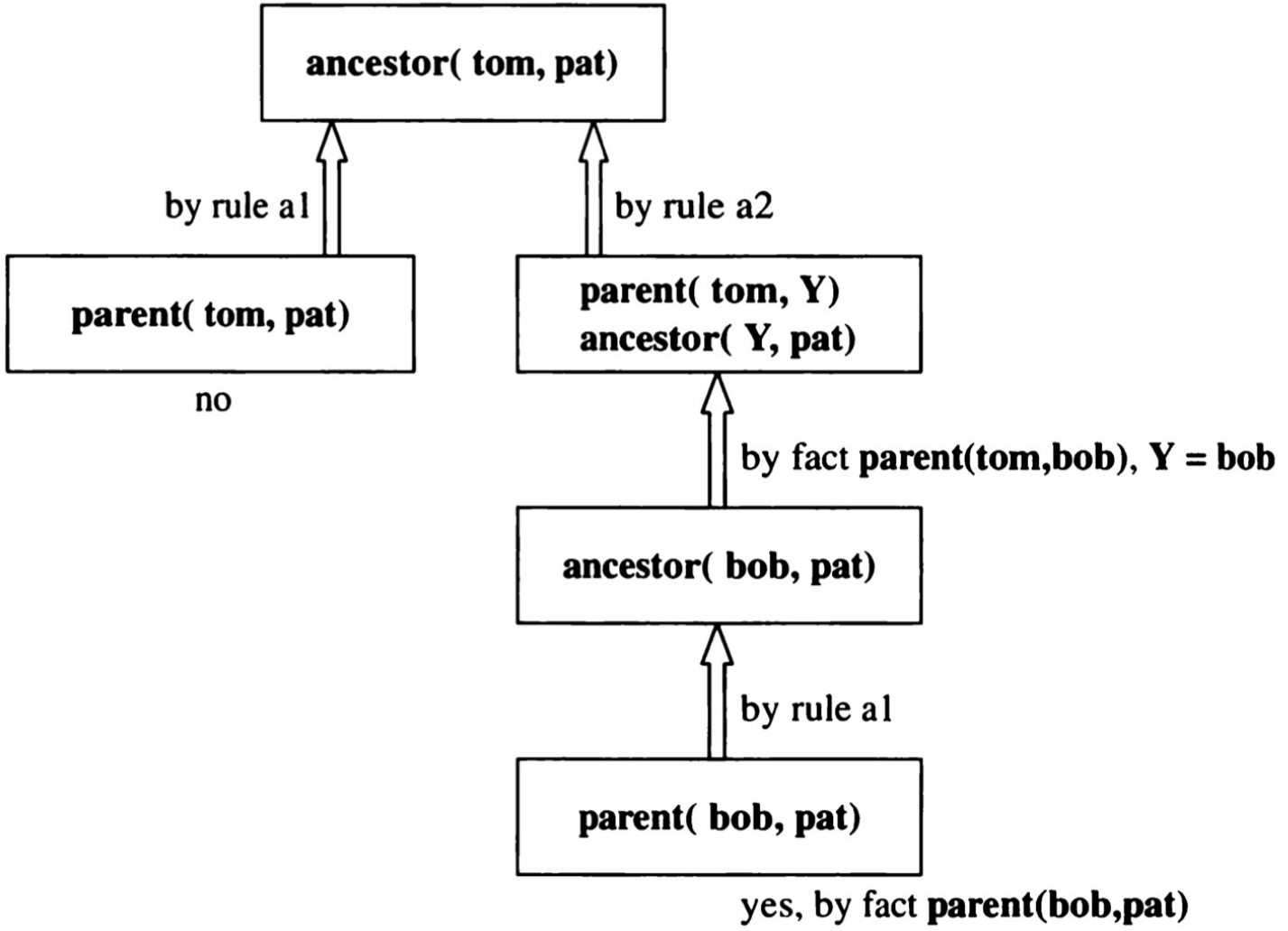
The current goal is replaced by:
parent(bob, pat).

How Prolog answers questions

An execution trace
has the form of a tree.

The top goal is satisfied
when a path is found from
the root node (top goal) to
a leaf node labelled ‘yes’
(a goal that is satisfied).

The execution of Prolog
programs is the
searching for such paths.

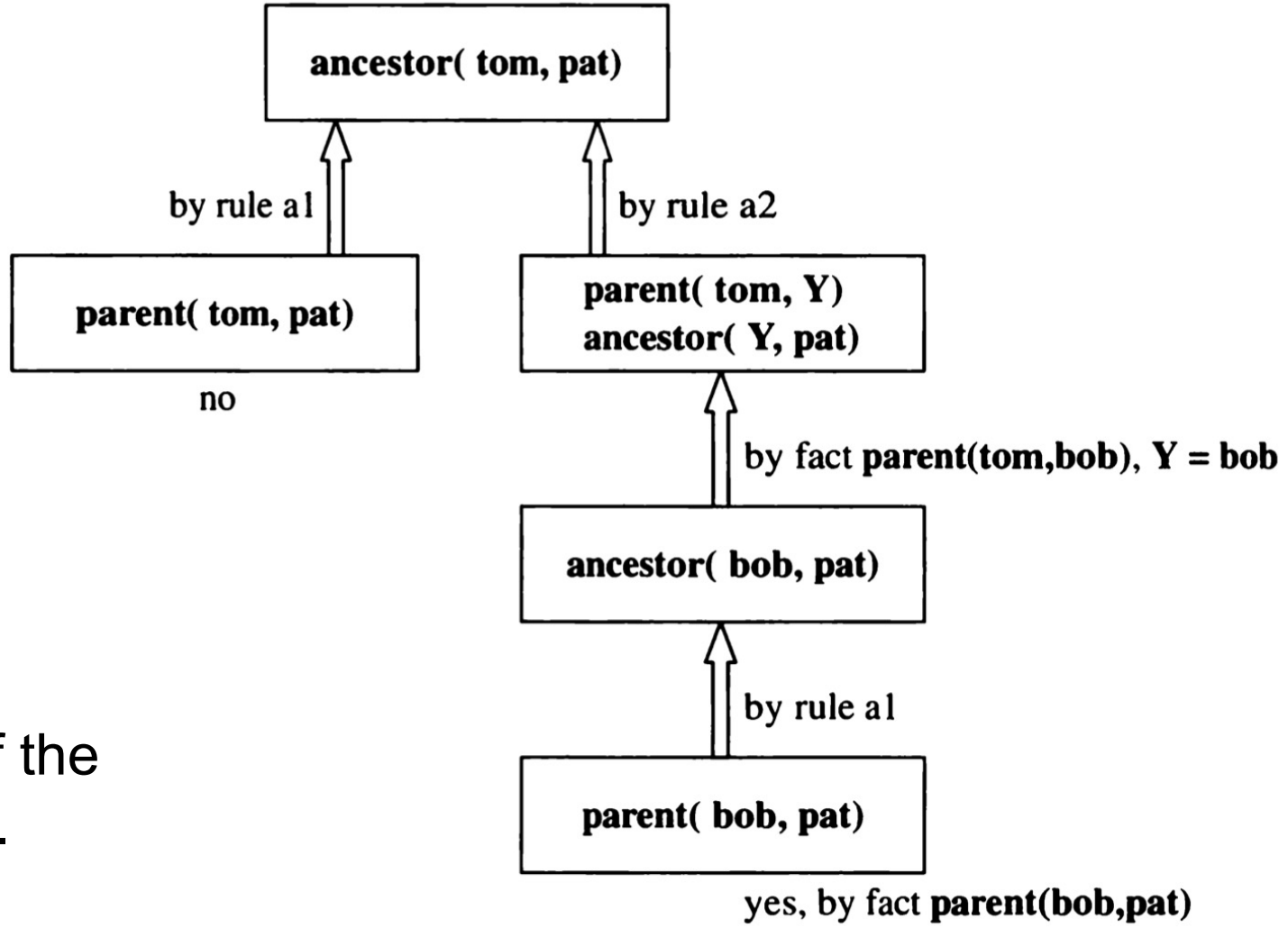


How Prolog answers questions

During the search Prolog may enter an unsuccessful branch.

When Prolog discovers that a branch fails it automatically *backtracks* to previous node and tries to apply an alternative clause at that node.

Automatic backtracking is one of the distinguishing features of Prolog.

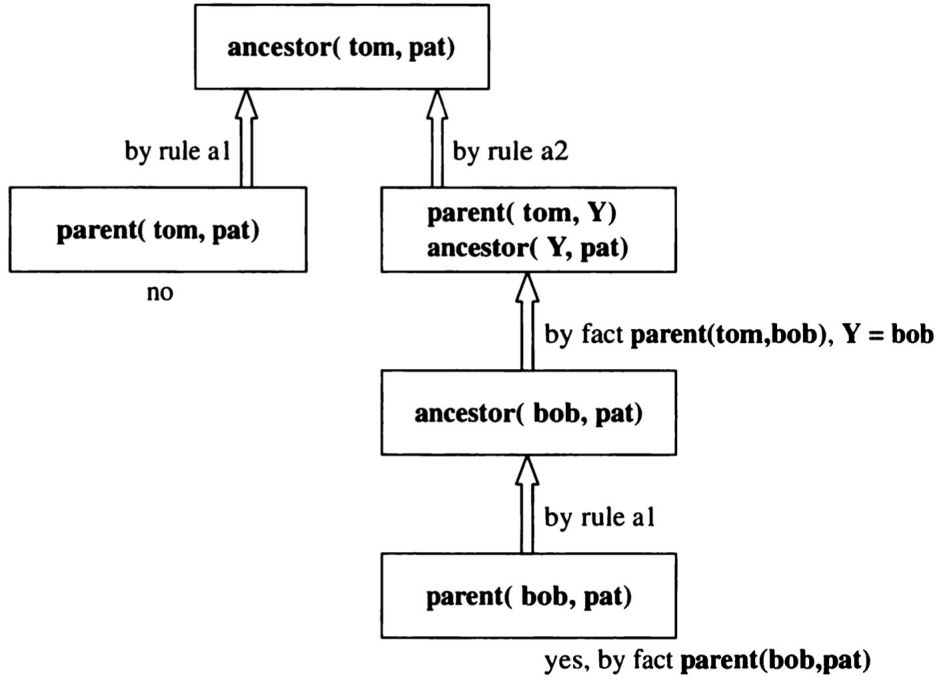


How Prolog answers questions

Try to understand how Prolog derives to the following questions, using our “family” program.

Try to draw the corresponding derivation diagram of:

1. ?- parent(pam, bob).
2. ?- mother(pam, bob).
3. ?- grandparent(pam, ann).
4. ?- grandparent(bob, jim).

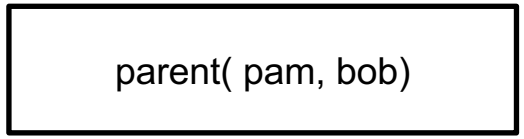


How Prolog answers questions

Try to understand how Prolog derives to the following questions, using our “family” program.

Try to draw the corresponding derivation diagram of:

1. ?- parent(pam, bob).



Yes, by fact parent(pam, bob)

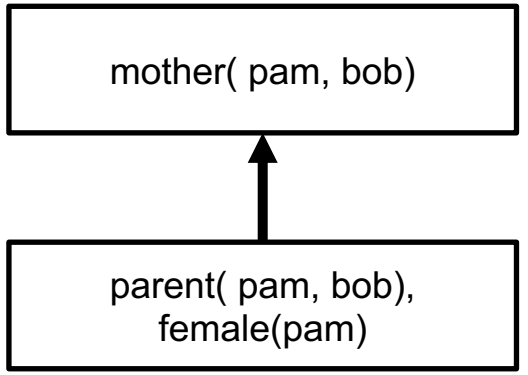
How Prolog answers questions

Try to understand how Prolog derives to the following questions, using our “family” program.

Try to draw the corresponding derivation diagram of:

1. ?- parent(pam, bob).

2. ?- mother(pam, bob).



by facts:
parent(pam, bob) is true
and
female(pam) is true

How Prolog answers questions

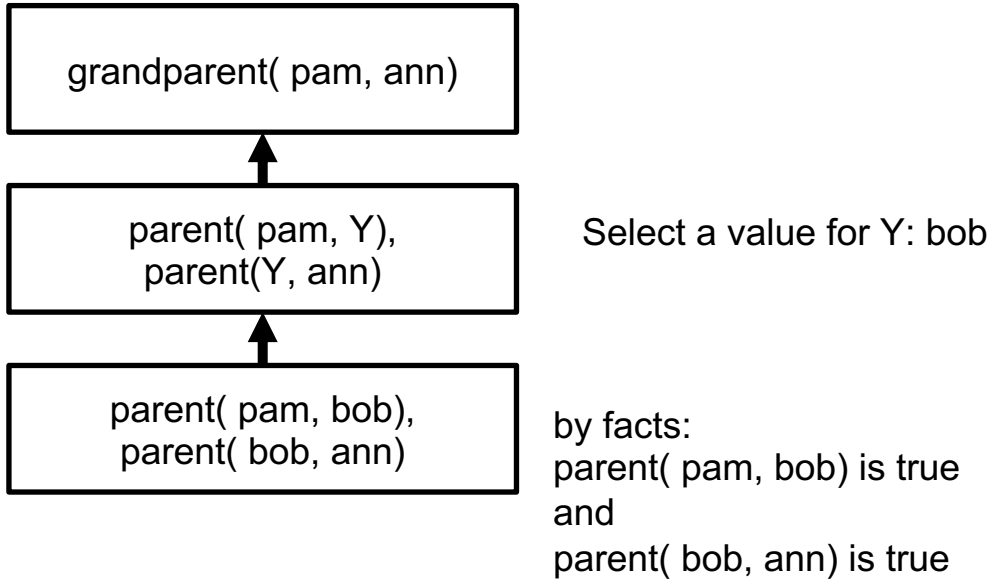
Try to understand how Prolog derives to the following questions, using our “family” program.

Try to draw the corresponding derivation diagram of:

1. ?- parent(pam, bob).

2. ?- mother(pam, bob).

3. ?- grandparent(pam, ann).

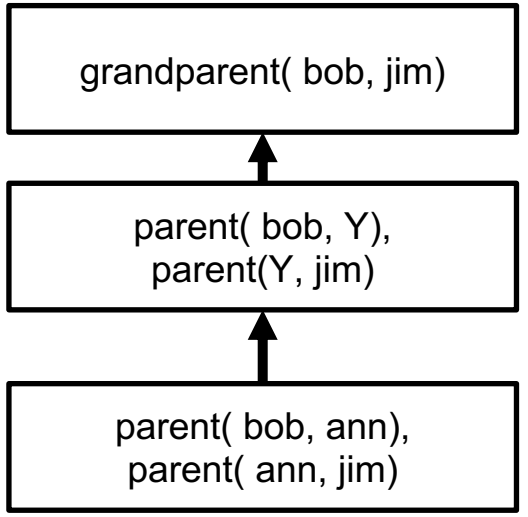


How Prolog answers questions

Try to understand how Prolog derives to the following questions, using our “family” program.

Try to draw the corresponding derivation diagram of:

1. ?- parent(pam, bob).
2. ?- mother(pam, bob).
3. ?- grandparent(pam, ann).
4. ?- grandparent(bob, jim).



Select a value for Y: ann
(the first one to appear)

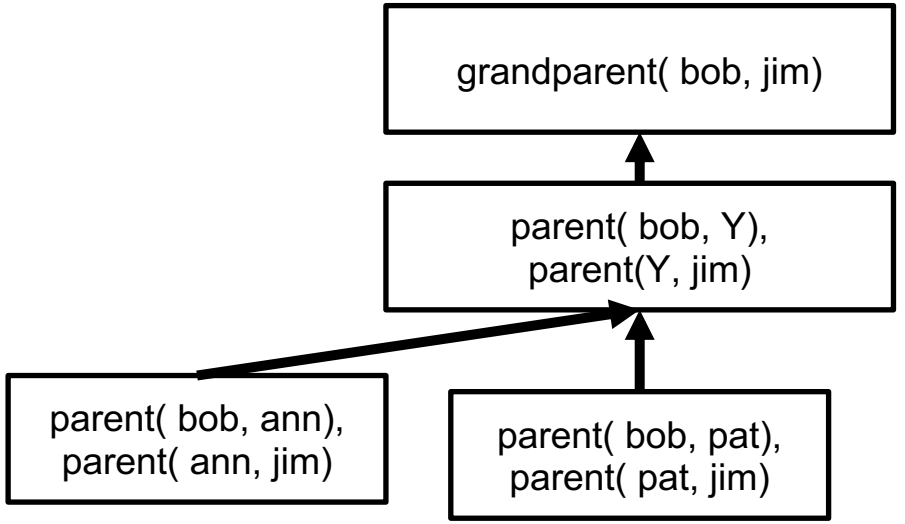
by facts:
parent(bob, ann) is true
and
no facts about
parent(ann, jim)
so it is false

How Prolog answers questions

Try to understand how Prolog derives to the following questions, using our “family” program.

Try to draw the corresponding derivation diagram of:

1. ?- parent(pam, bob).
2. ?- mother(pam, bob).
3. ?- grandparent(pam, ann).
4. ?- grandparent(bob, jim).

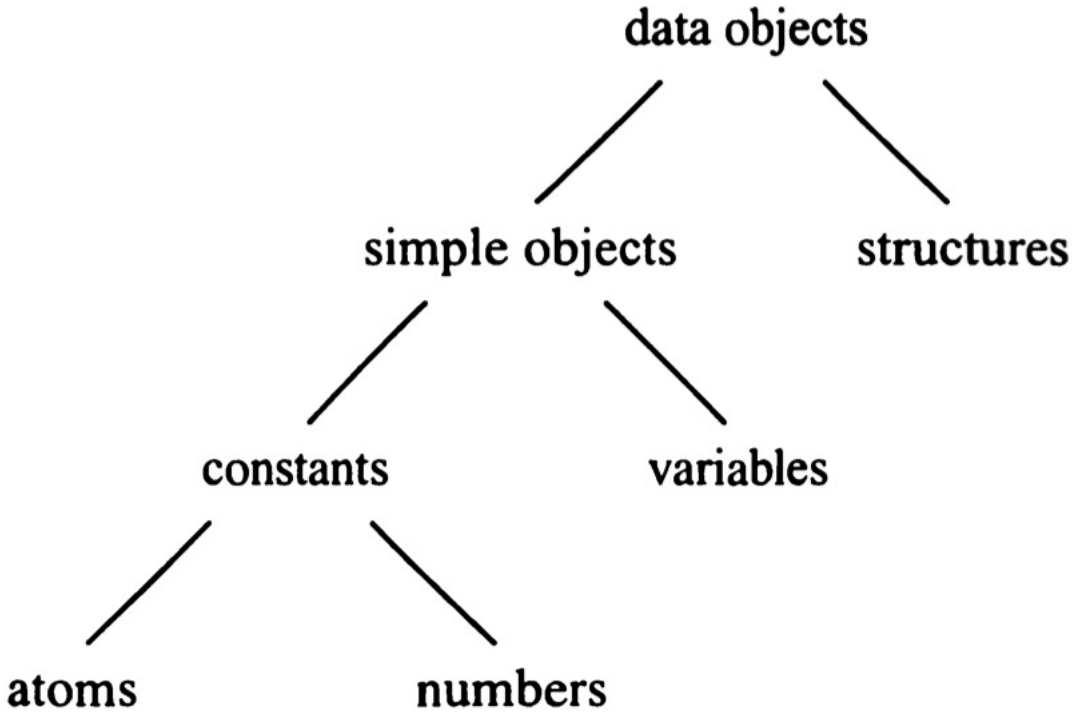


Select another value for Y: pat (the secon one to appear)

by facts:
parent(bob, pat) is true

and
parent(pat, jim) is true

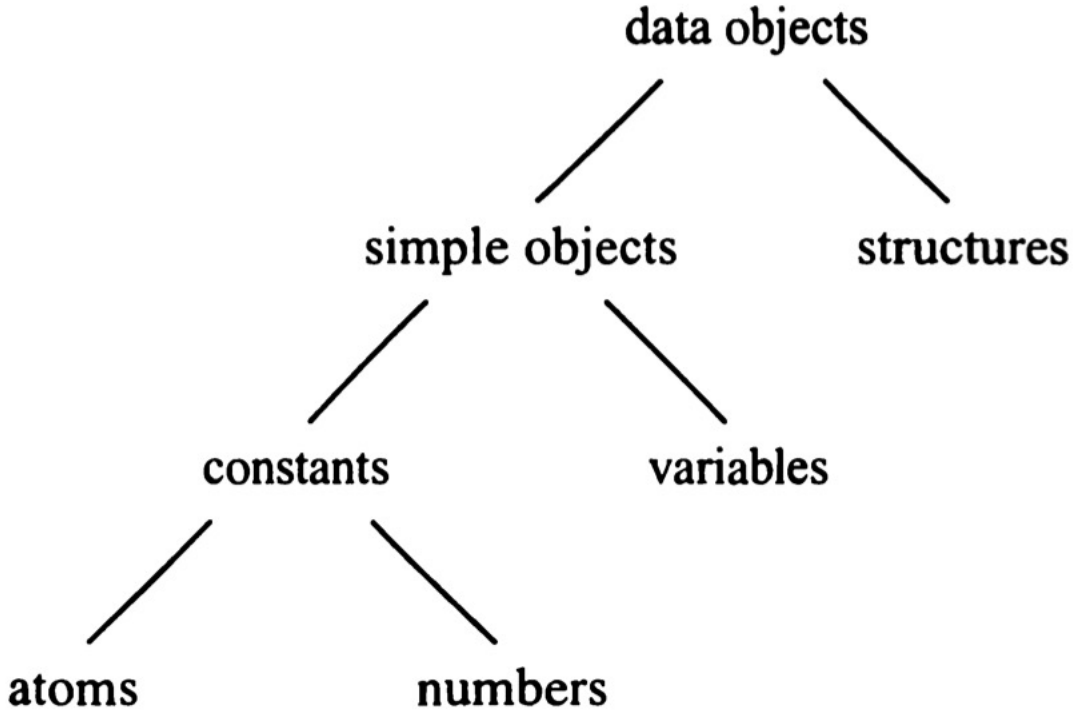
Syntax and Semantics of Prolog's concepts



Atoms

1. Strings of letters, digits, and the underscore character, '_', starting with lower-case letter:

anna	x_
nil	x___y
X25	alpha_beta_procedure
X_25	miss_Jones
X_25AB	

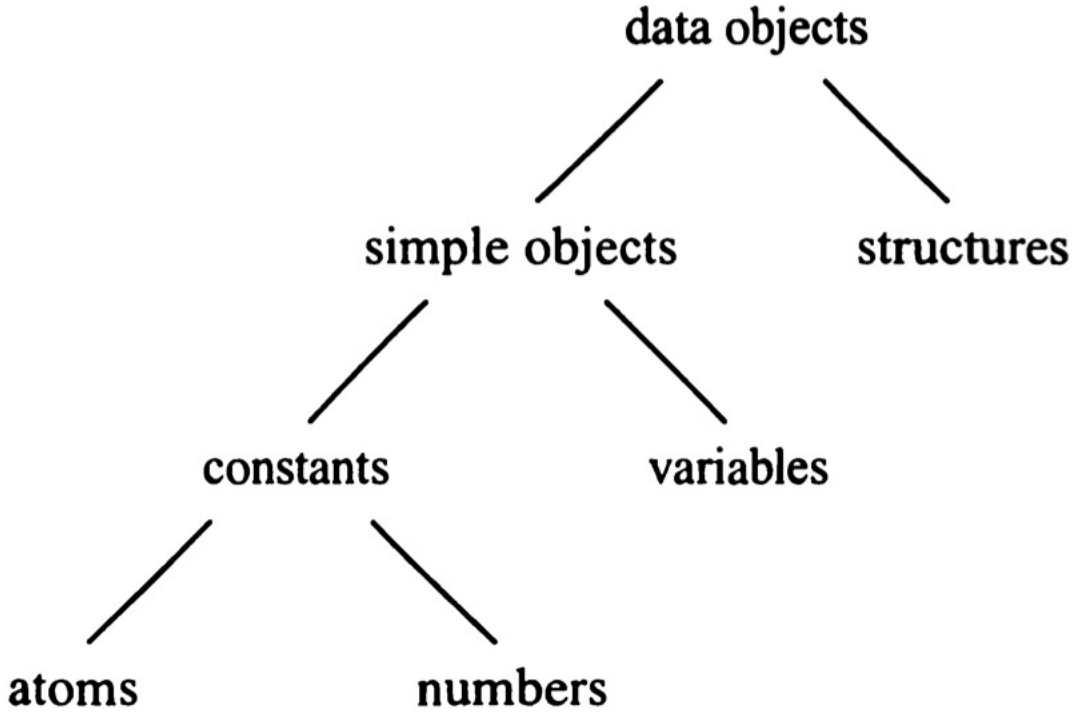


Atoms

2. Strings of special characters:

<--->
=======>
+
…
.:.
::=

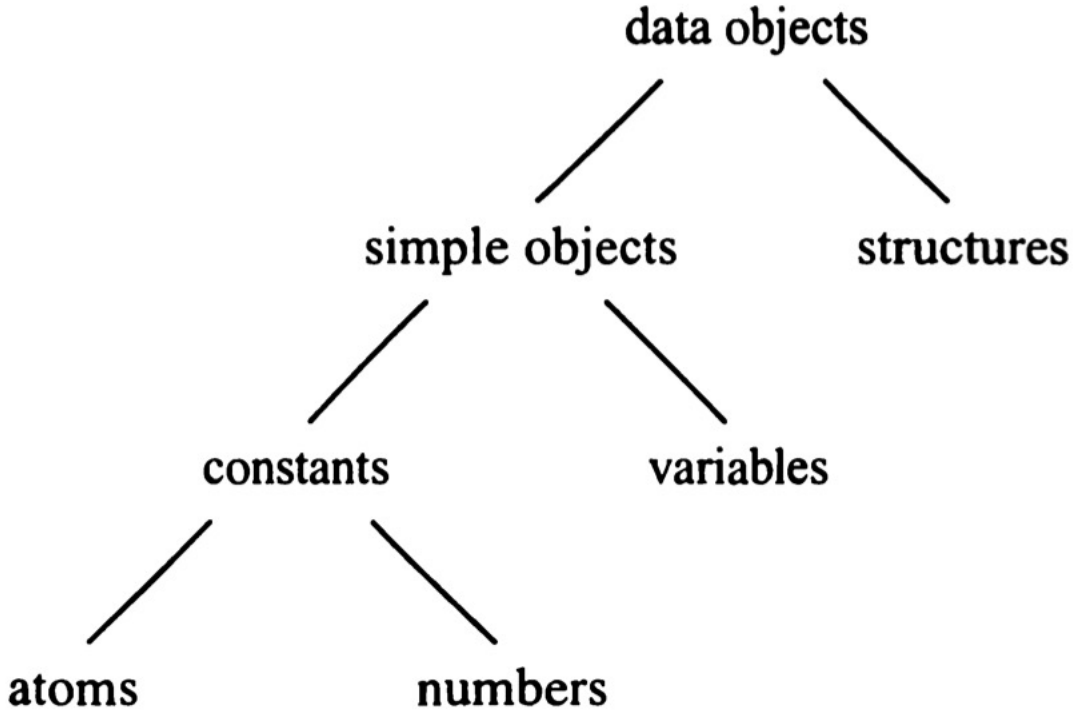
But some strings of special characters already have a predefined meaning. E.b., ‘:-’



Atoms

3. Strings of characters enclosed in single quotes:

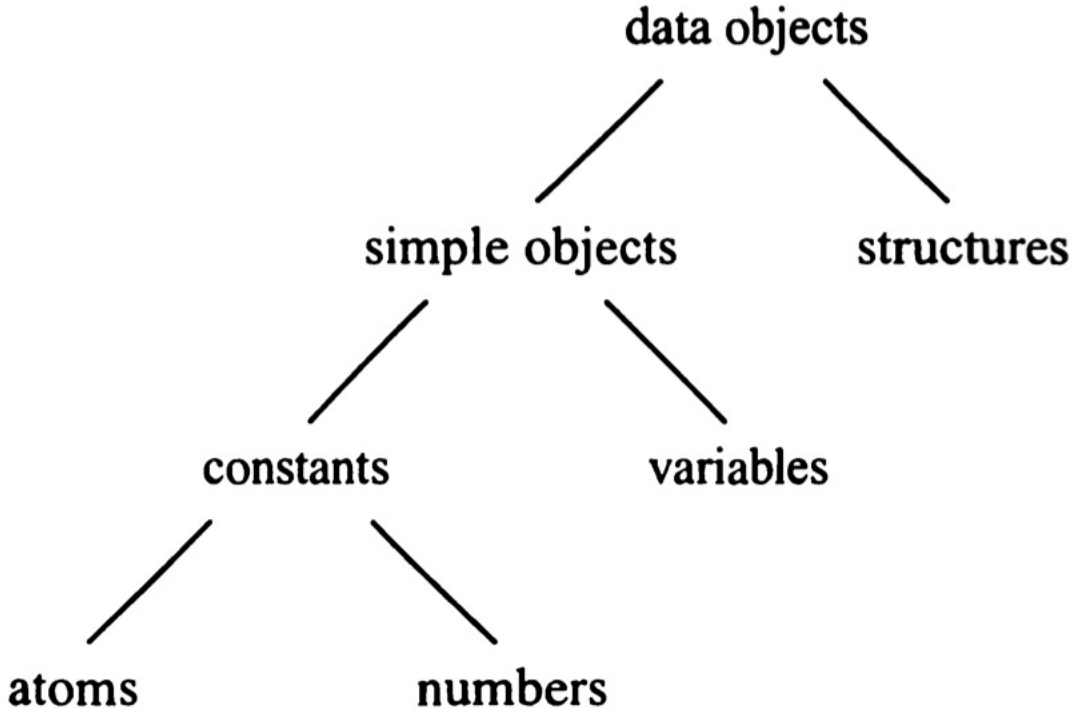
‘Tom’
‘South_America’
‘Sarah Jones’



Numbers

Integer numbers:
1 1313 0 -97

Real numbers:
3.14 -0.0035 100.2. 7.15E-9

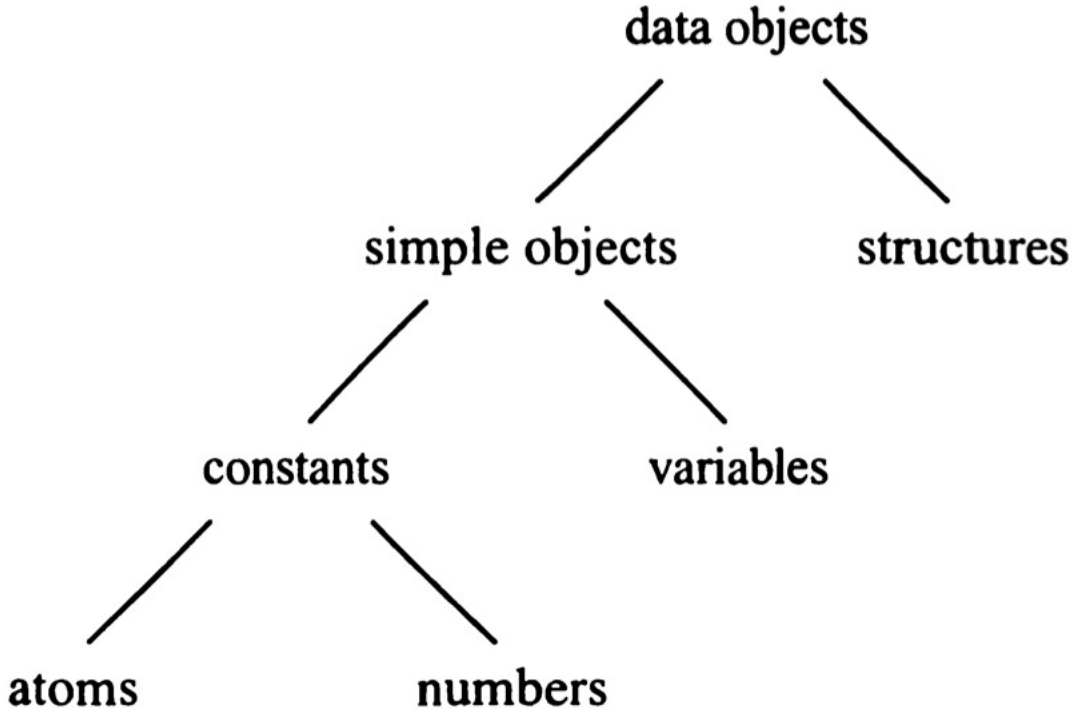


Variables

Variables are strings of letters, digits, and underscore characters.

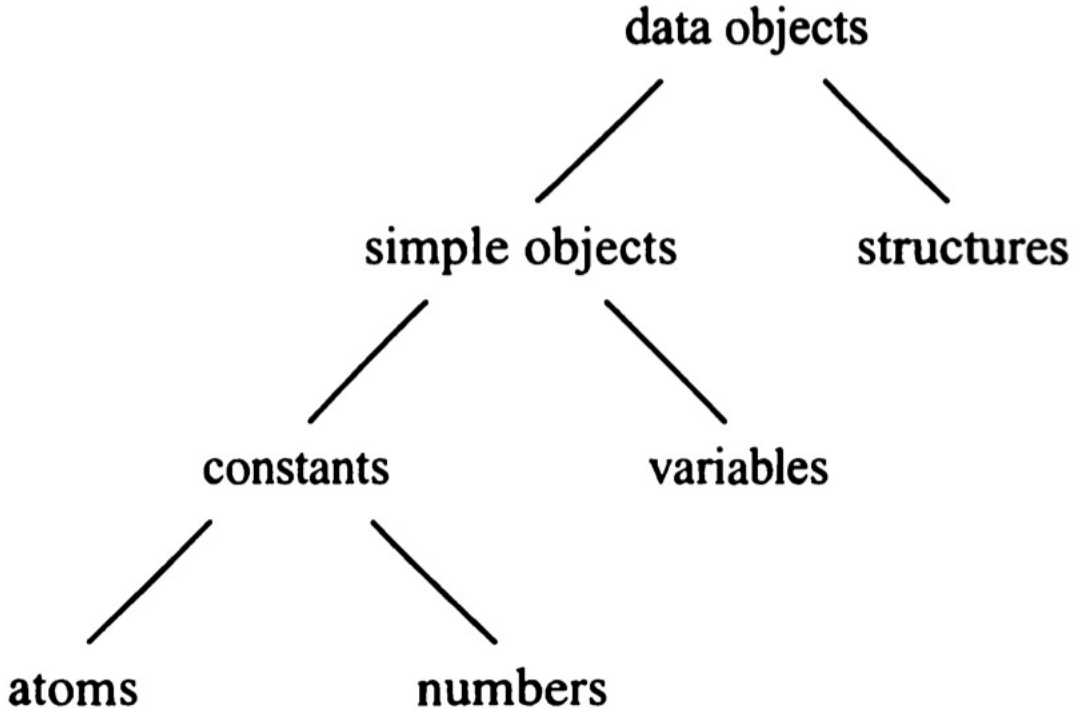
They start with an upper-case letter or an underscore character:

X
Result
Object2
Participant_list
ShoppingList
_A
_x23
_23



Variables

Variables are strings of letters, digits, and underscore characters.



Define the relation **has_a_child()**

has_a_child(X) :- parent(X, Y).

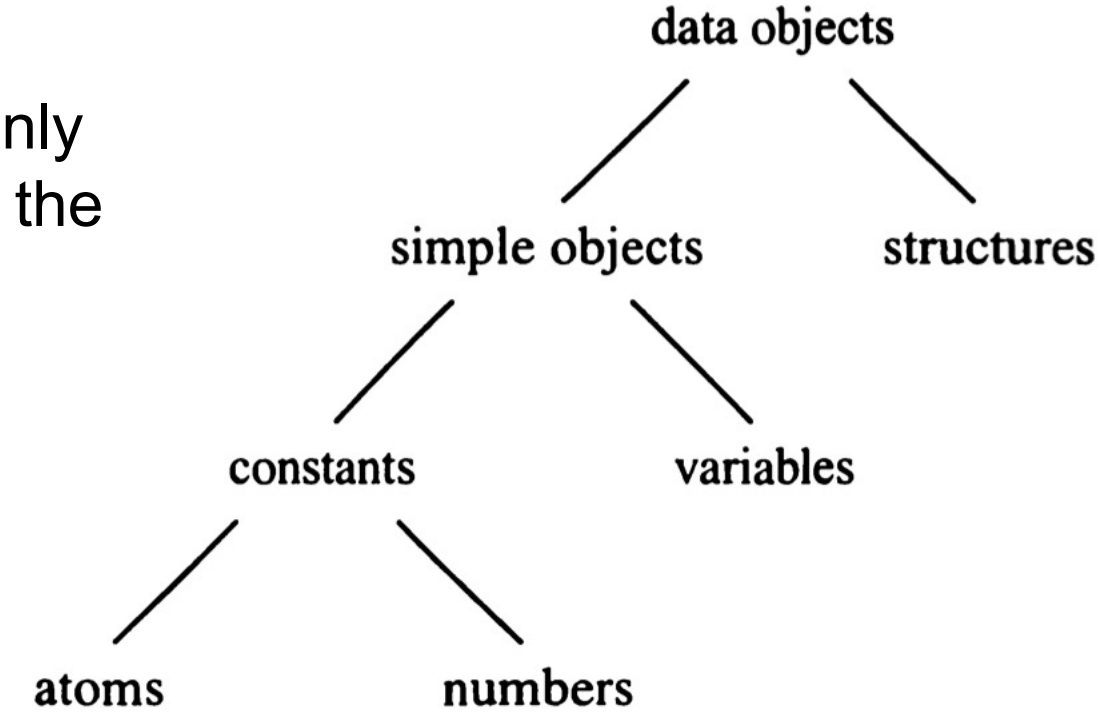
Variables

Variables are strings of letters, digits, and underscore characters.

When a variable appears in a clause once only we don't neet to give it a name. We can use the 'anonymous' variable written as a single underscore '_'

Define the relation **has_a_child()**

has_a_child(X) :- parent(X, _).



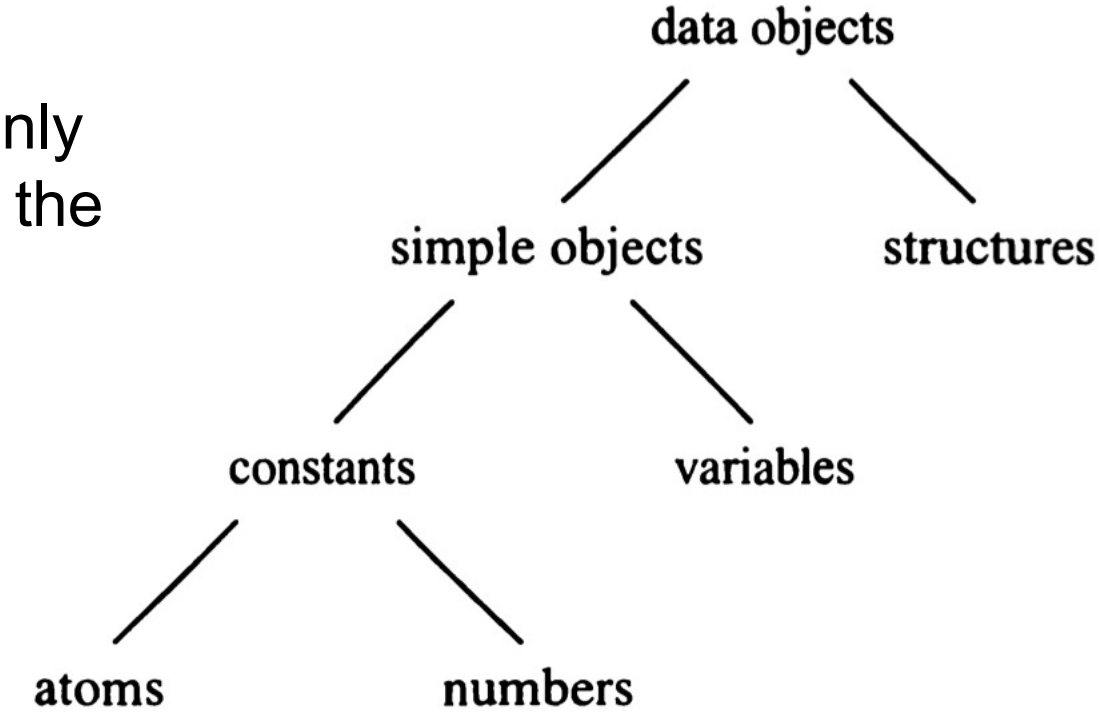
Variables

Variables are strings of letters, digits, and underscore characters.

When a variable appears in a clause once only we don't neet to give it a name. We can use the 'anonymous' variable written as a single underscore '_'

If an anonymous variable appears in a question its value is not output when Prolog answers the question.

?- parent(X, _).

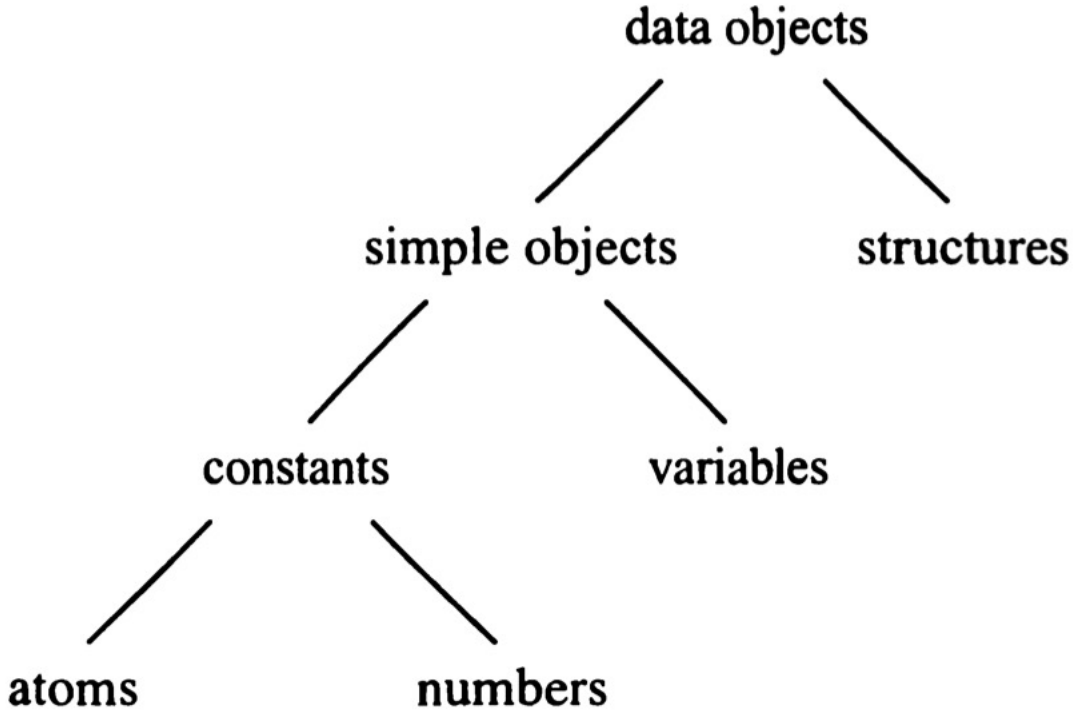


Variables

Variables are strings of letters, digits, and underscore characters.

The *lexical scope* of variable names is one clause.

If X12 occurs in two clauses, it signifies two different variables. But each occurrence of X12 within the same caluse means the same variable.

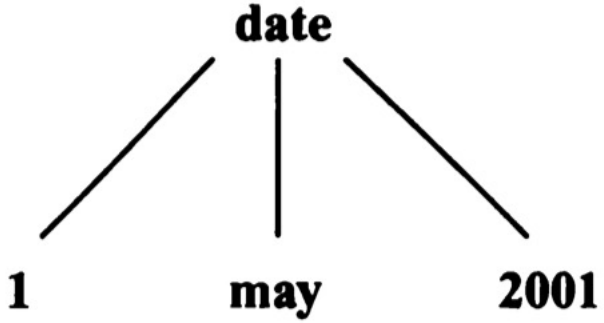


Structures

Structures are objects that have several components.

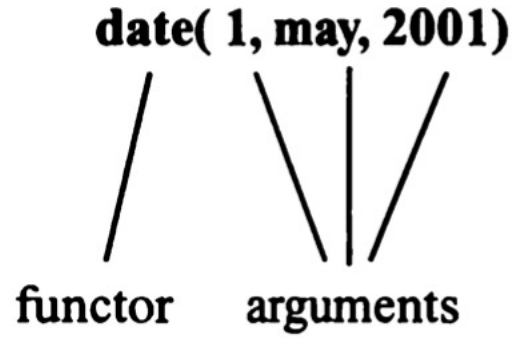
date(1, may, 2001)

All the components in this example are constants (two integers and one atom).



Components can also be variables or other structures. Any day in May can be represented by the structure:

date(Day, may, 2001)



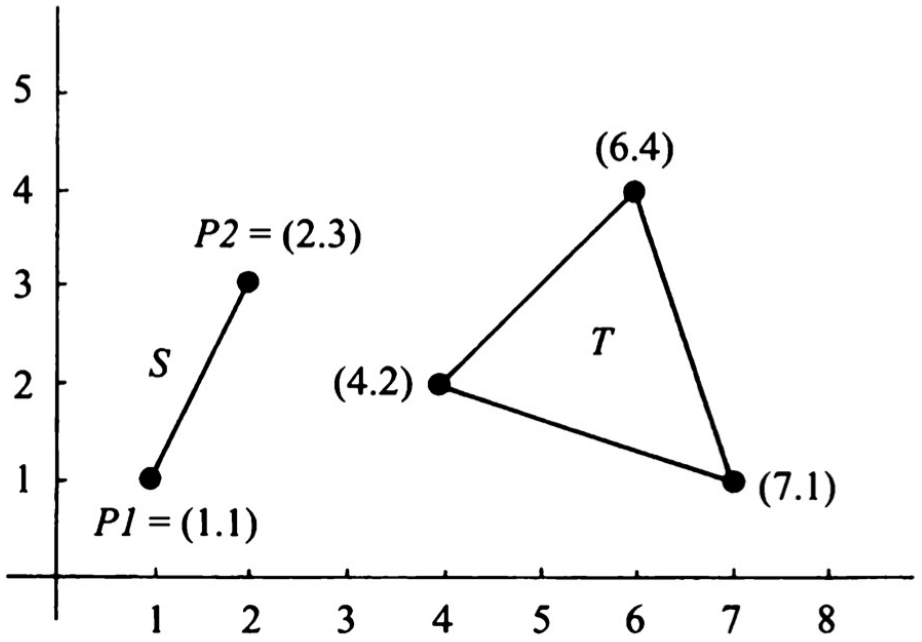
Structures

Structures can be used to represent geometric objects

What defines a point?

Two coordinates

P1 = point(1,1)
P2 = point(2,3)



What defines a segment?

Two points

S = seg(P1, P2) = seg(point(1,1), point(2,3))

A triangle can be defined by three points

T = triangle(point(4,2), point(6,4), point(7,1))

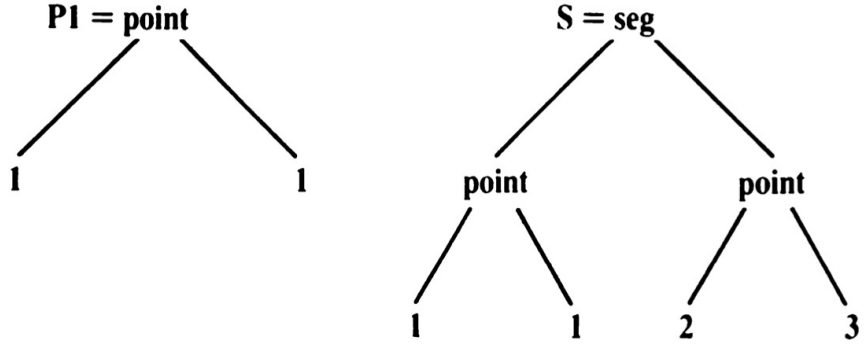
Structures

Structures can be used to represent geometric objects

What defines a point?

Two coordinates

P1 = point(1,1)
P2 = point(2,3)



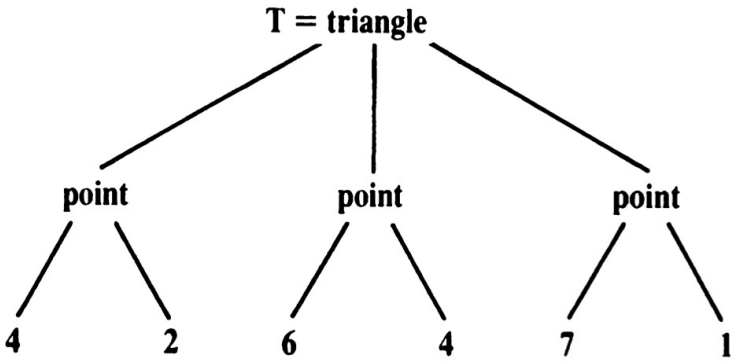
What defines a segment?

Two points

S = seg(P1, P2) = seg(point(1,1), point(2,3))

A triangle can be defined by three points

T = triangle(point(4,2), point(6,4), point(7,1))



Structures

If we add points in 3D space we can use another functor:

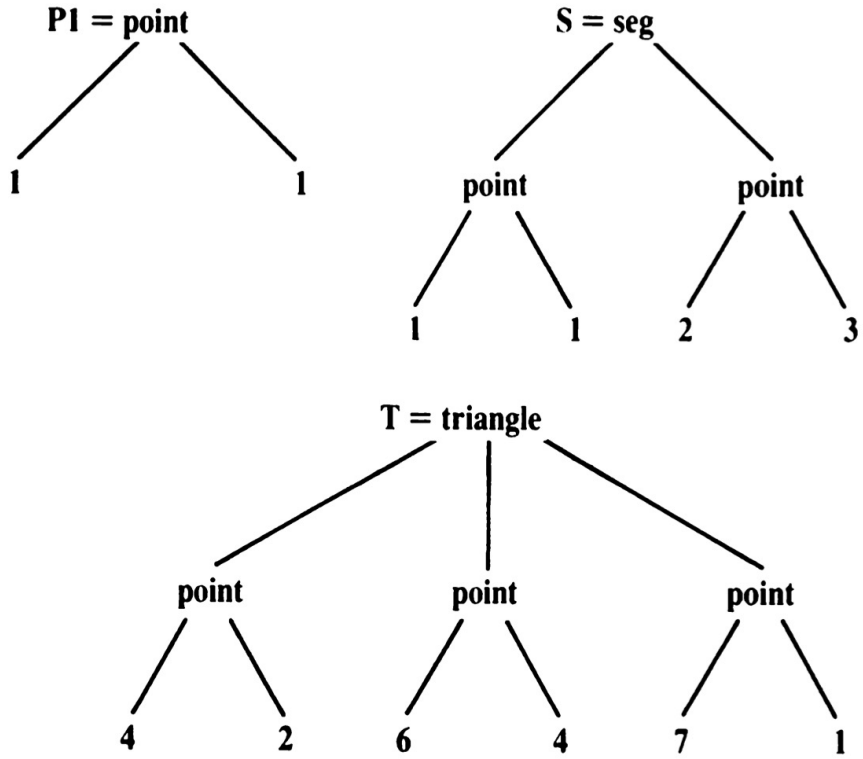
point3(X, Y, Z)

Or we can use the same name, **point**, for both 2D and 3D

point(X1, Y1) and **point(X, Y, Z)**

Prolog will recognize the difference by the number of arguments. Each functor is defined by two things:

1. The name, whose syntax is that of atoms;
2. The *arity* – that is, the number of arguments.



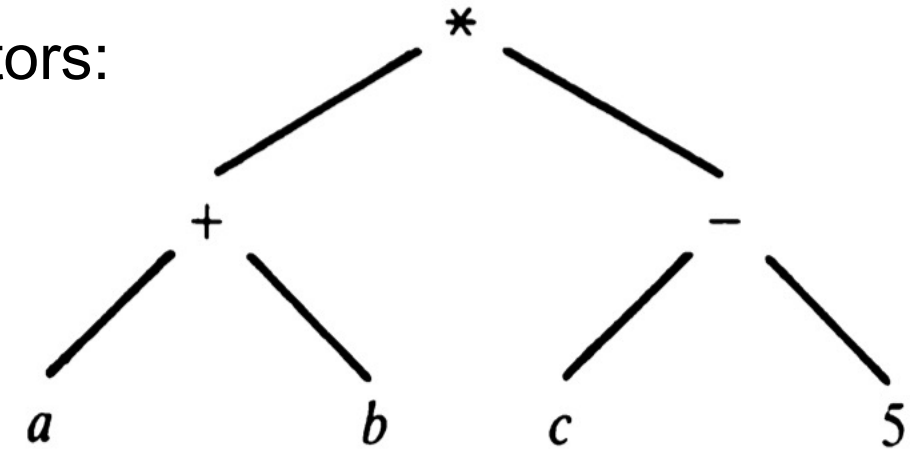
Structures

All structured objects in Prolog are trees, represented in the program by terms.

The tree structure of the arithmetic expression: (a + b) * (c – 5)

It can be writte using symbols ‘*’, ‘+’ and ‘-’ as functors:
*(+(a, b),-(c,5))

Prolog also allows to use the prefiz, infix and postfix notation



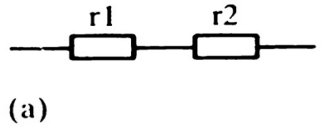
Prolog and Electric Circuits

Simple electric circuits

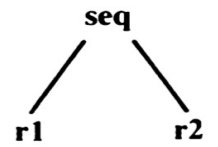
The atoms **r1**, **r2**, **r3** and **r4** are the names of the resistors.

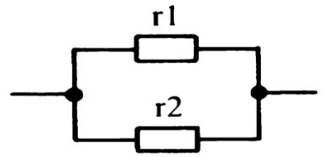
The functor **par** and **seq** denote the parallel and the sequential compositions of resistors respectively. The corresponding Prolog terms are:

seq(r1, r2)
par(r1, r2)
par(r1, par(r2, r3))
par(r1, seq(par(r2, r3), r4))

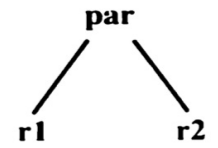


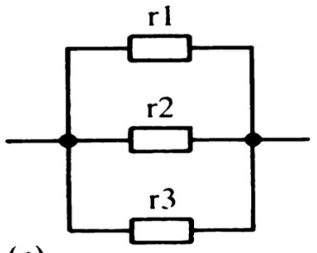
(a)



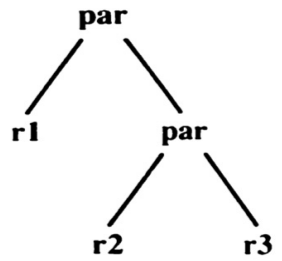


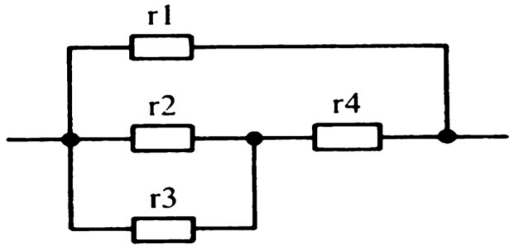
(b)



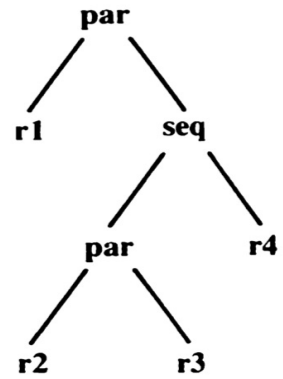


(c)





(d)



Prolog and Electric Circuits

We can construct, a circuit of any complexity.
E.g., a sequential circuit of 10 resistors:

seq(r1, seq(r2, seq(r3, seq(r3, seq(r5, seq(r6, seq(r7, seq(r8, seq(r9, r10)))))))))

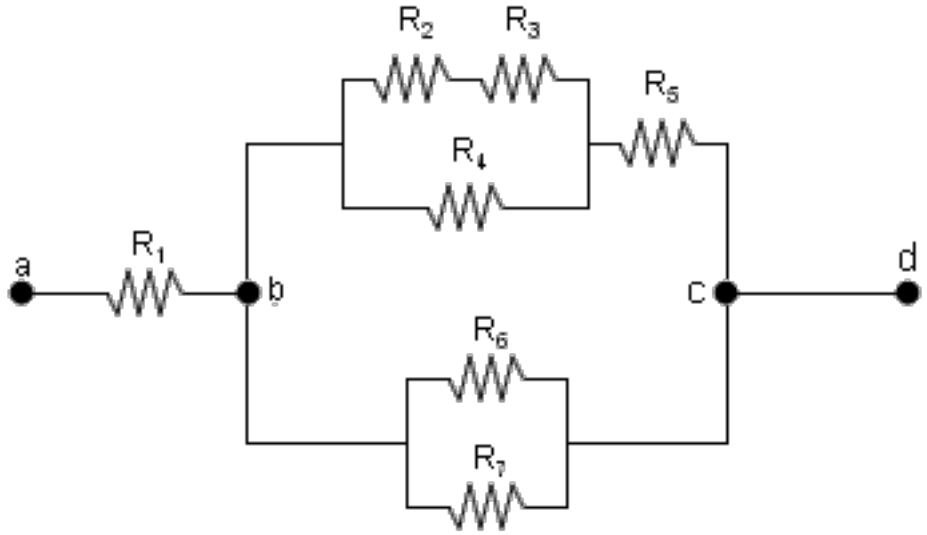
We view circuits as sentences, and their elements as words. The electrical behavior and functions are the meaning of the sentences. Circuit structures are defined by a logic grammar. A set of grammar rules, when converted into Prolog clauses, forms a logic program which perform top-down parsing. When an unknown circuit is given, this logic program will analyze the circuit and derive a parse tree for the circuit.

...

After a circuit is parsed, not only its syntactic structure, but also its electrical behavior and functions can be derived as the meaning of the circuit structure.

ANALYZING CIRCUIT STRUCTURES AS LANGUAGE http://www.fit.ac.jp/~tanaka/paper/nova-chapter5.pdf

Prolog and Electric Circuits



$R_1 = 7\ \Omega$ $R_2 = 5\ \Omega$ $R_3 = 11\ \Omega$ $R_4 = 8.67\ \Omega$ $R_5 = 5\ \Omega$ $R_6 = 7\ \Omega$ $R_7 = 35\ \Omega$

In parallel e.g., $\frac{R6 * R7}{R6 + R7}$

In series e.g., R2+R3

Finding Equivalent Resistance of a Resistive Circuit