

Recursion in Prolog

Module of Logics and Artificial Intelligence course

Recursion

Recursion is usually used when:

- Follow **relationships chains** among objects
- Efficiently manipulate **lists**

Let us take a look at the former case, a classical example of this kind is the **transitive use of a relation**:

- If Arya is a sibling of Sansa, and Sansa is a sibling of Bran, hence Arya is a sibling of Bran.
- If point A is connected to point B and point B is connected to point C hence point A is connected to point C.
- If the key is in the drawer and the drawer is in the bedside hence the key is in the bedside.

In order to prove these assertions we need to use recursion to explore the relations, keeping in mind we want to represent the transitivity of the relations.

Example 1 – The office

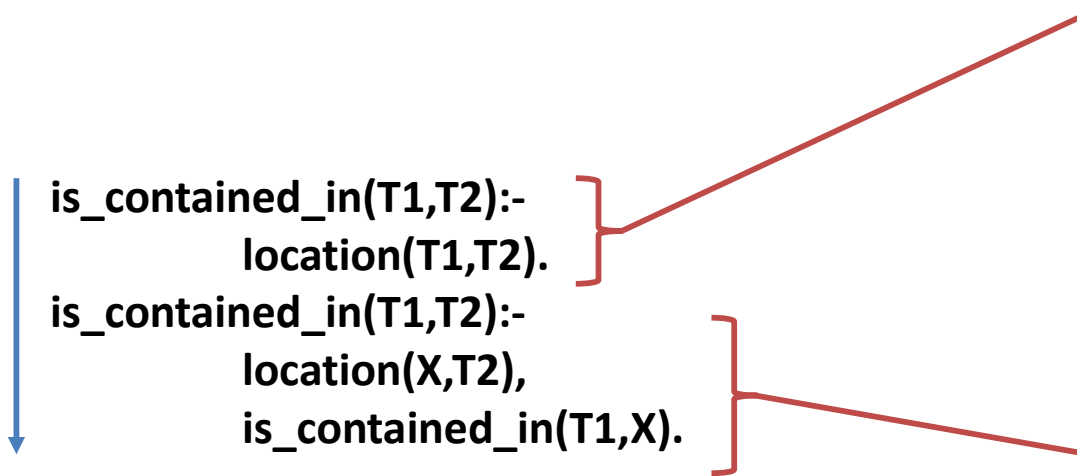
In this knowledge base the position of some objects within an office is modelled:

```
location(desk,office).  
location(computer,office).  
location	flashlight,desk).  
location(envelope,desk).  
location(stamp, envelope).  
location(key, envelope).
```

Our goal is to check if the **key** is in the office

Example 1 – The office

In this knowledge base the position of some objects within an office is modelled:



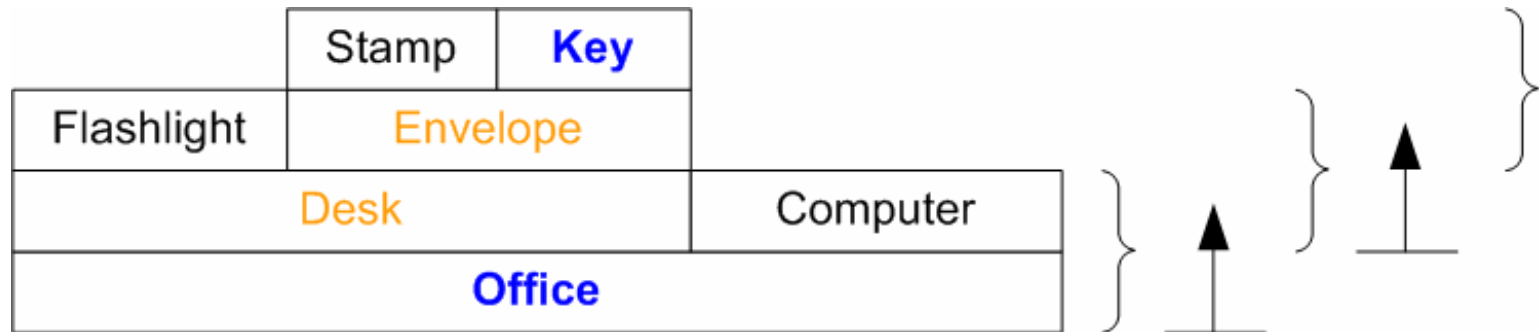
```
is_contained_in(T1,T2):-  
    location(T1,T2).  
is_contained_in(T1,T2):-  
    location(X,T2),  
    is_contained_in(T1,X).
```

boundary condition: it determines the stopping condition (IF) for the recursion. **It is true when the direct relation is satisfied.** In other cases (ELSE) Prolog will go over to the second definition of `is_contained_in`, the recursive one.

It finds an object **X** contained in **T2** and calls `is_contained_in` on **T1** and **X**, transitively looking for an object that directly contains **T1**.

Example 1 – The office

The behavior of the query `is_contained_in(key,office)` that will return **true** is shown below:



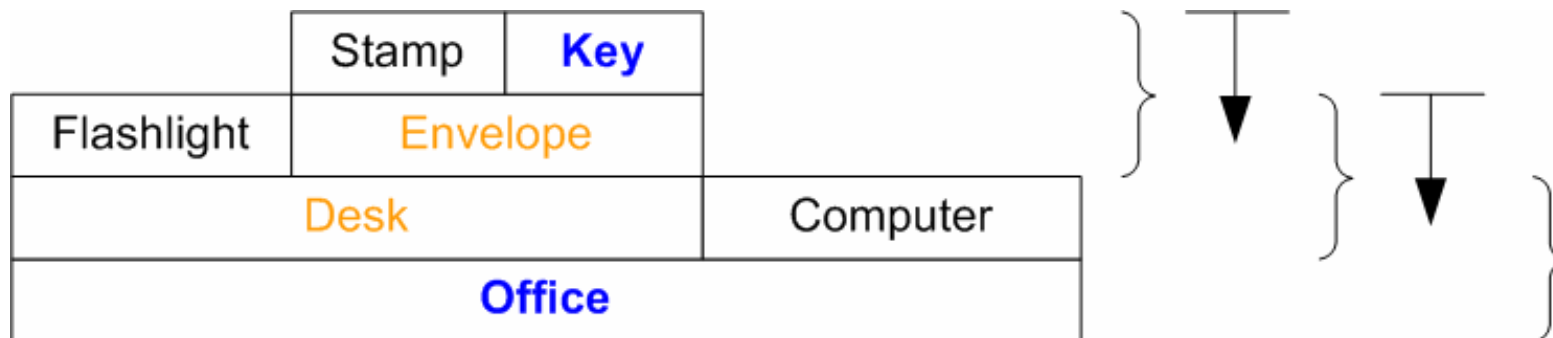
It finds the objects contained in the office and the query is repeated on these. The search moves, step by step, from **T2** towards **T1**. These rules are efficient for queries like `is_contained_in(X,office)`.

Example 1 – The office

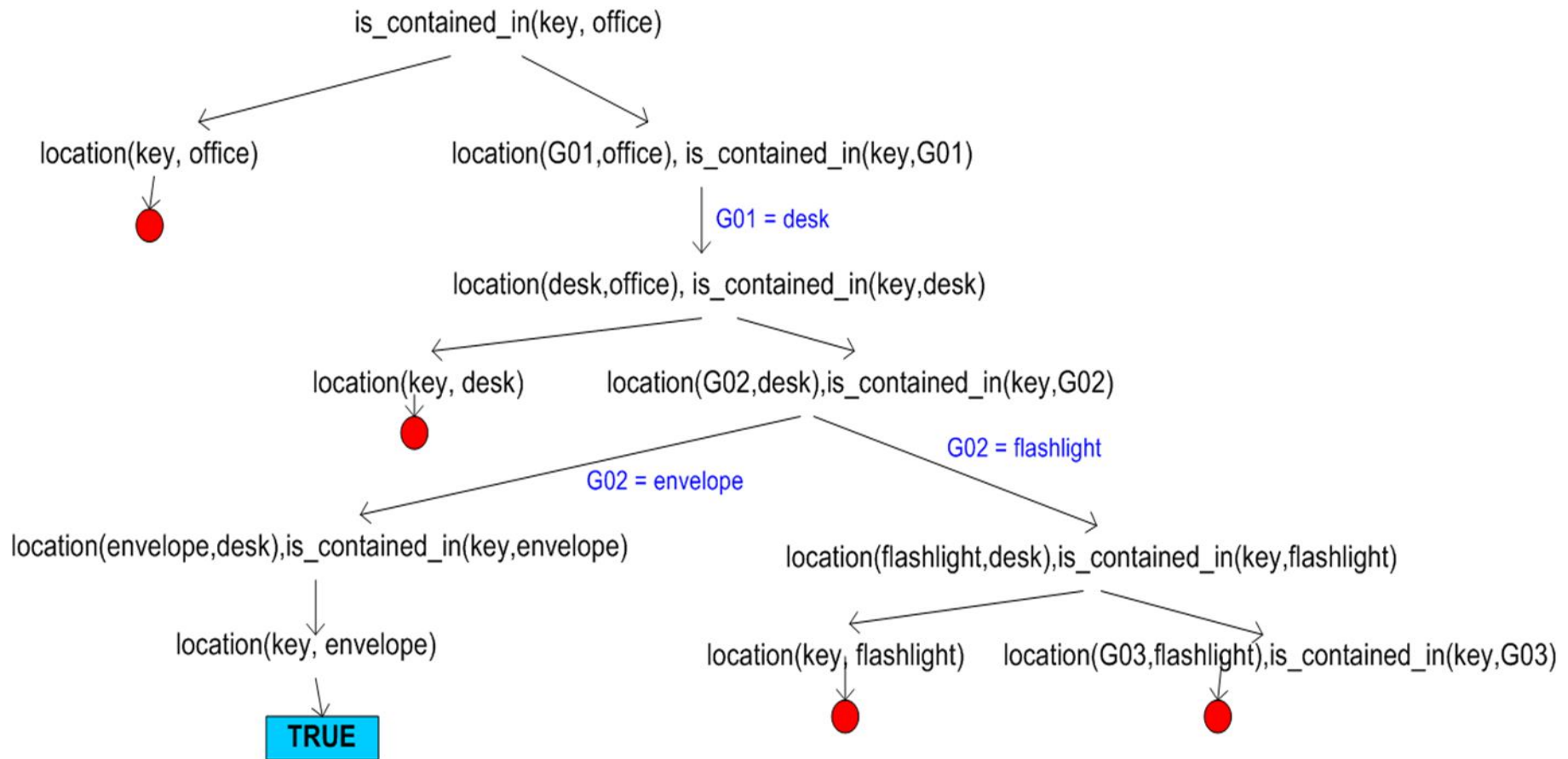
We can write down rules that can efficiently face queries like **is_contained_in(key,X)**, designed to retrieve the objects that contains the **key** (envelope, desk and office).

is_contained_in(T1,T2):- location(T1,T2).

is_contained_in(T1,T2):- location(T1,X), is_contained_in(X,T2).



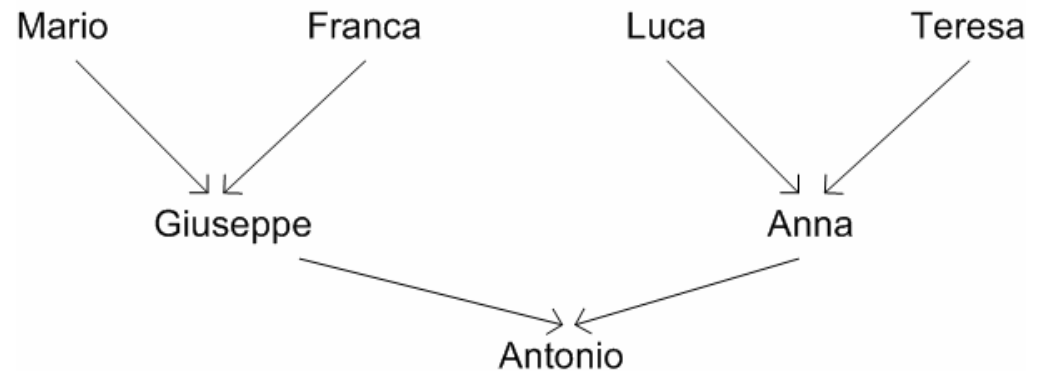
Example 1 – The office



Example 2 – The descendants

The knowledge base contains facts about the family tree shown in figure:

parent(giuseppe,antonio).
parent(anna,antonio).
parent(mario,giuseppe).
parent(franca,giuseppe).
parent(luca,anna).
parent(teresa,anna).



Create two recursive rules, the first one to check if **X** is a descendant of **Y** and the second one to check if **X** is an ancestor of **Y**.

descendant(X,Y).
ancestor(X,Y).

Example 2 – The descendants

**descendant(X,Y) :-
 parent(Y,X).**

**descendant(X,Y) :-
 parent(Y,F),
 descendant(X,F).**

OR

**descendant(X,Y) :-
 parent(Y,X).**

**descendant(X,Y) :-
 parent(F,X),
 descendant(F,Y).**

**ancestor(X,Y) :-
 parent(X,Y).**

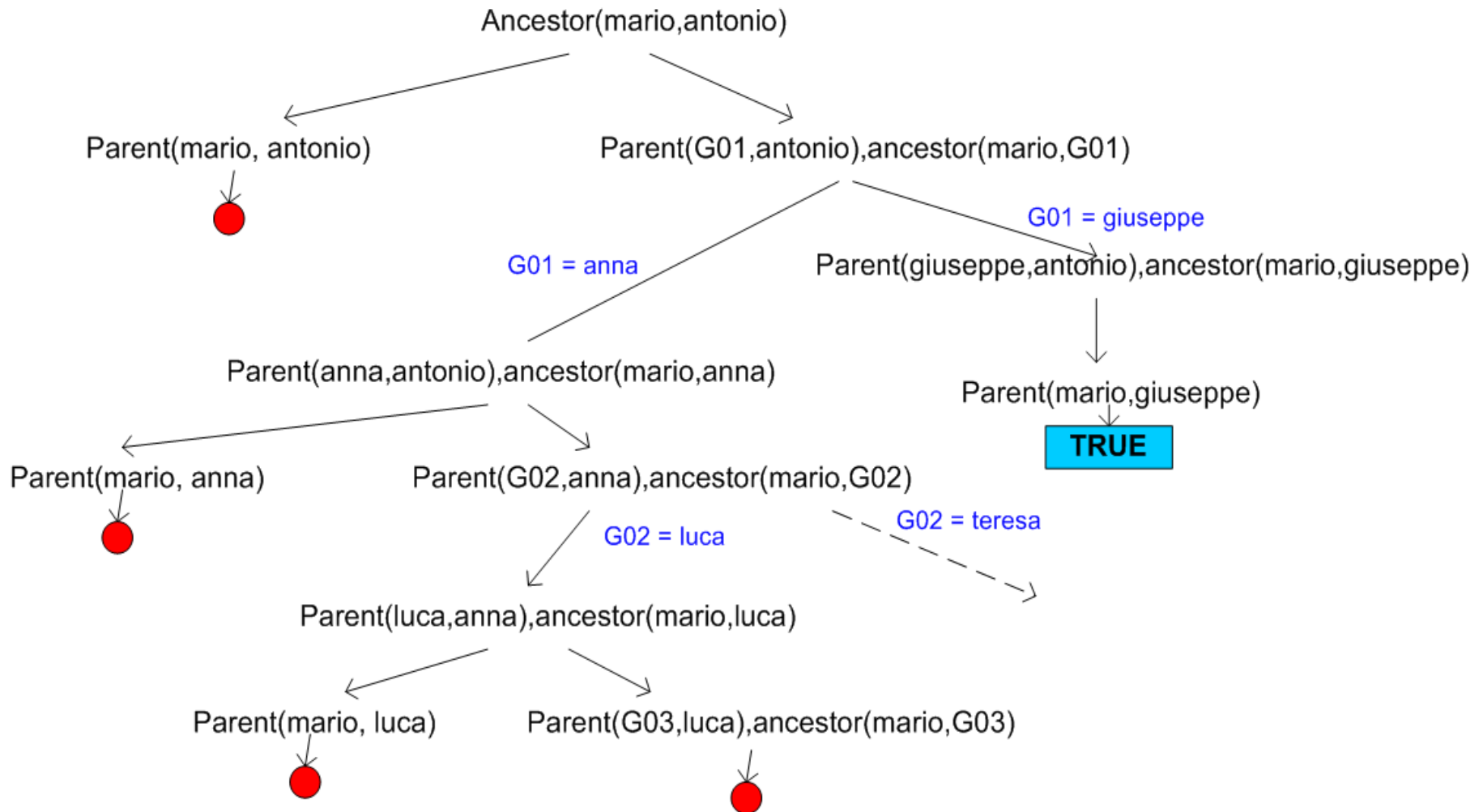
**ancestor(X,Y) :-
 parent(G,Y),
 ancestor(X,G).**

OR

**ancestor(X,Y) :-
 parent(X,Y).**

**ancestor(X,Y) :-
 parent(X,F),
 ancestor(F,Y).**

Example 2 – The descendants



Example 3 – The path

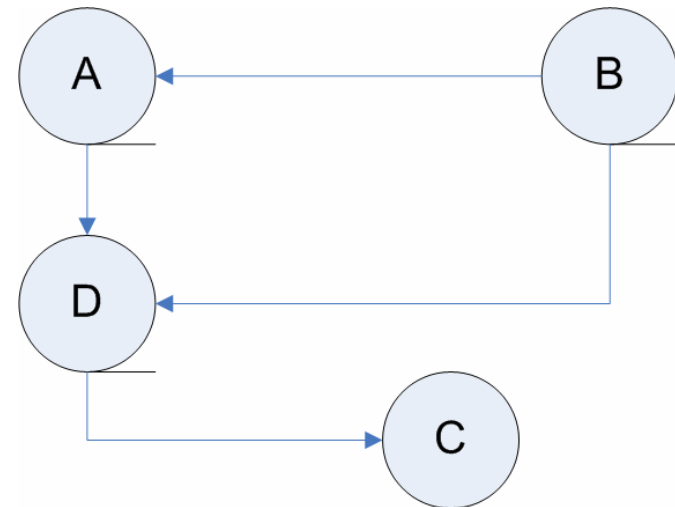
The knowledge base models the connections among four points (A,B,C,D):

arc(d,c).

arc(a,d).

arc(b,a).

arc(b,d).



Write a rule that checks if a path exists that connects via one or more arc a point to another. If one of the arguments is a variable, the program must return all the connected points:

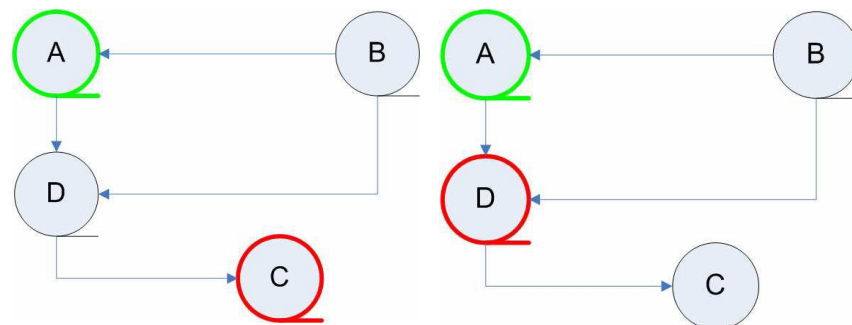
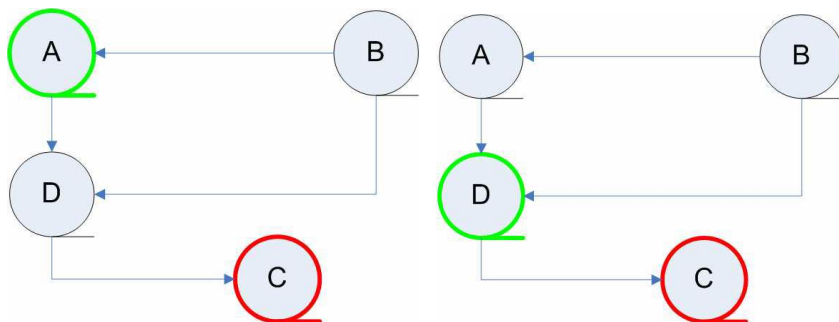
path(X,Y).

Example 3 – The path

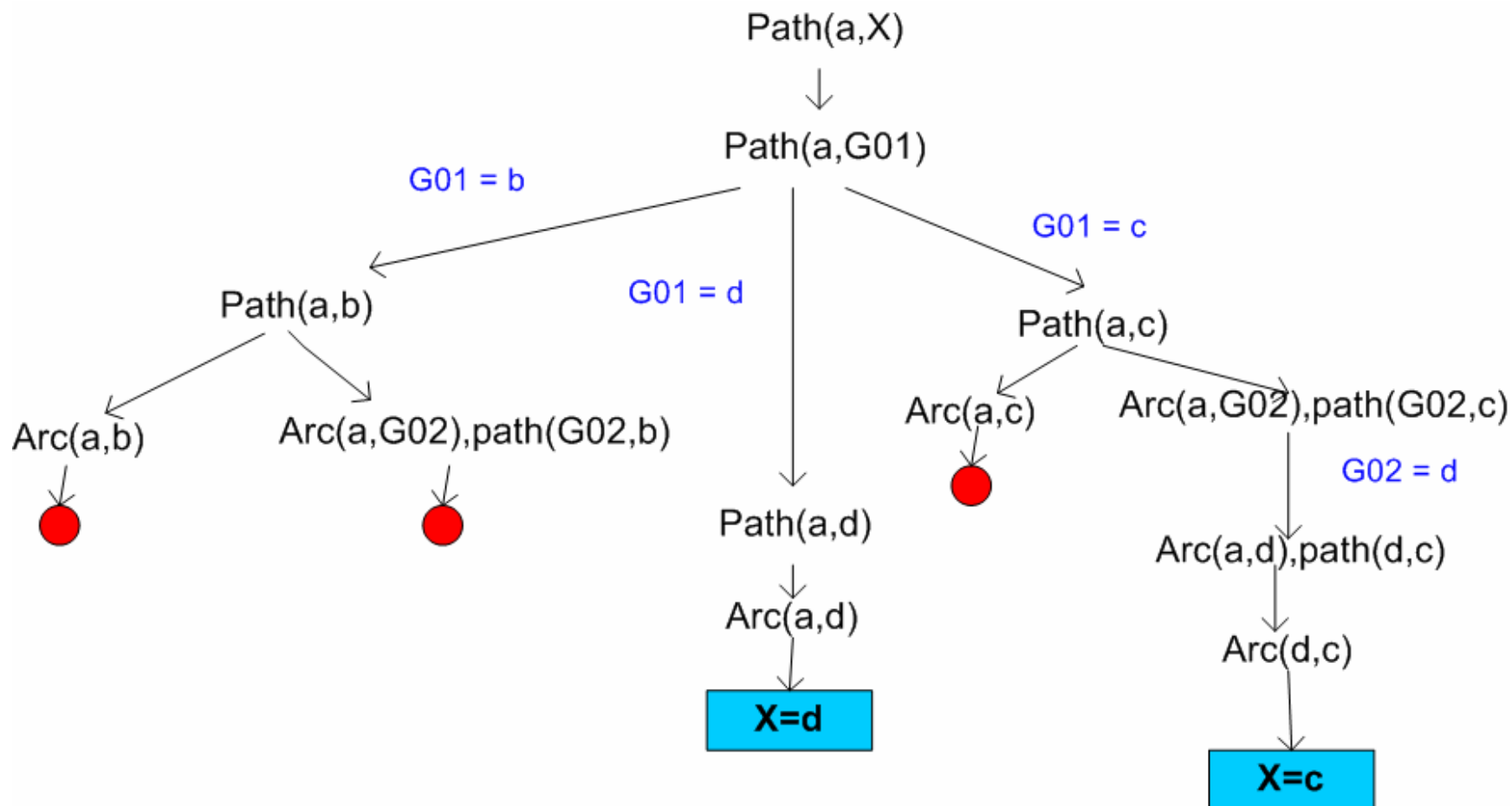
`path(X,Y) :-`
`arc(X,Y).`
`path(X,Y) :-`
`arc(X,Z),`
`path(Z,Y).`

OR

`path(X,Y) :-`
`arc(X,Y).`
`path(X,Y) :-`
`arc(Z,Y),`
`path(X,Z).`



Example 3 – The path



Example 4 – The factorial

Write a rule that computes the factorial function of a number:

Example 4 – The factorial

Write a rule that computes the factorial function of a number:

$$n! = \begin{cases} 1 & \text{if } n = 0, \\ (n - 1)! \times n & \text{if } n > 0 \end{cases}.$$

Example 4 – The factorial

Write a rule that computes the factorial function of a number:

$$n! = \begin{cases} 1 & \text{if } n = 0, \\ (n - 1)! \times n & \text{if } n > 0 \end{cases}.$$

fact(0,1).

The boundary condition set the factorial of **0 = 1**

fact(N,Ans):-

N1 is N - 1,

It computes before the factorial of **N - 1**, then it is multiplied by **N**. In this approach we go to 0 and coming back we compute the factorial of 1,2..N

fact(N1,A1),

Ans is N * A1.

Roughly we can say that we compute the factorial of 1 knowing the factorial of 0, hence we compute the factorial of 2 knowing the factorial of 1 and so on.

Example 4 – The factorial

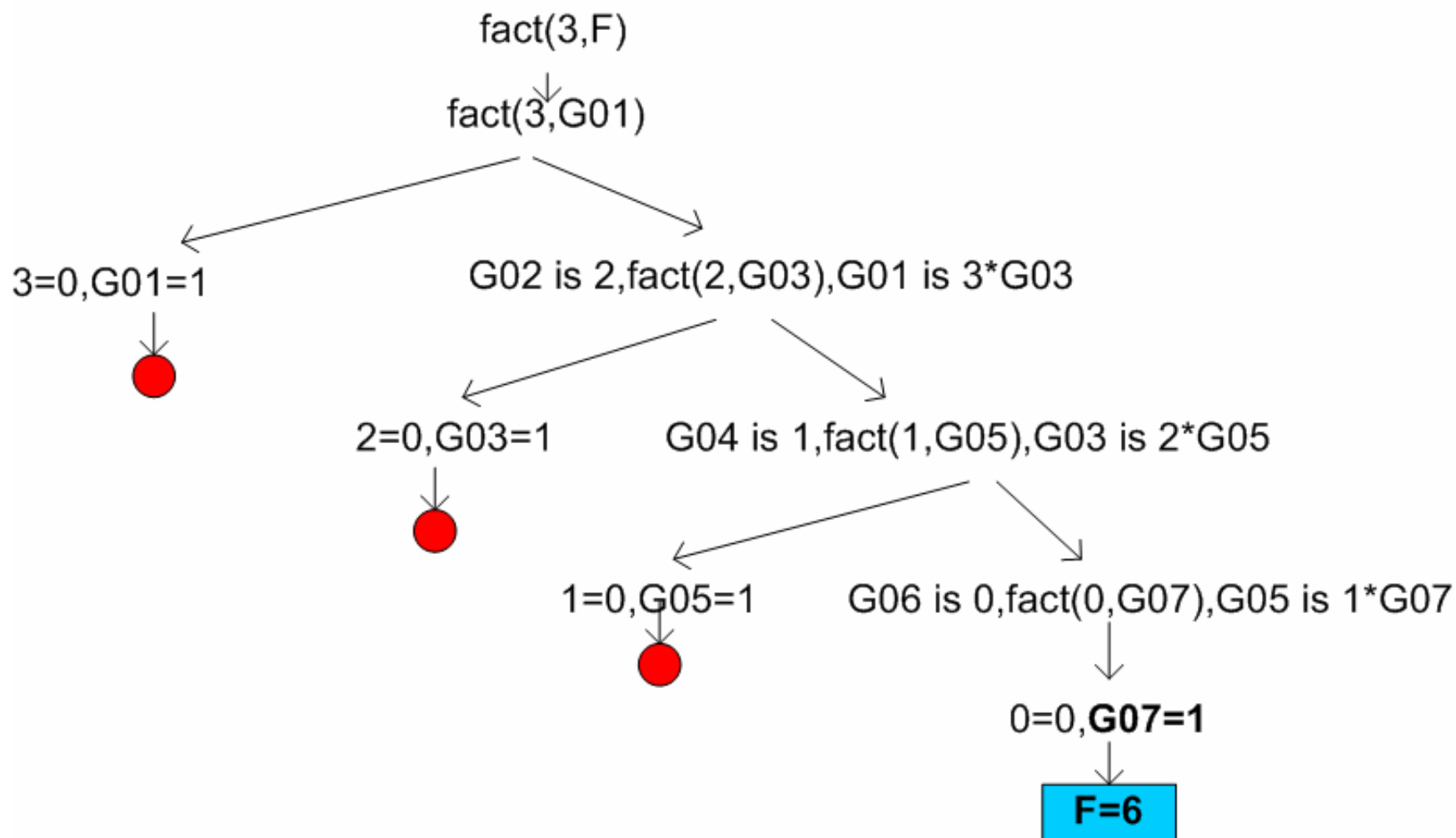
We could think to write down a (**wrong**) alternative to the previous code:

```
fact(0,1).  
fact(N,F):-  
    N1 is N - 1,  
    F is F * N,  
    fact(N1,F).
```

This is **wrong**! When the program tries to compute **F** that is not instantiated yet! So the Prolog Interpreter answer:

ERROR: is/2: Arguments are not sufficiently instantiated

Example 4 – The factorial



Example 5 – Even and Odd numbers

Write two rules, one to check if a number is even and another one to check if the number is odd:

Example 5 – Even and Odd numbers

Write two rules, one to check if a number is even and another one to check if the number is odd:

even(0).

even(N):-

N1 is N-2,
N1 >= 0,
even (N1).

odd(1).

odd (N):-

N1 is N-2,
N1 >= 1,
odd (N1).

Examples:

even(4) → 4-2=2 → 2-2=0 → *true*

odd(4) → 4-2=2 → 2-2=0 → 0-2=-2 → *false*

Example 6 – The power function

Write a Prolog program that computes $Z = X^Y$:

Example 6 – The power function

Write a Prolog program that computes $Z = X^Y$:

```
power(X,1,X).  
power(X,Y,Z):-  
    I is Y-1,  
    power(X,I,Z1),  
    Z is Z1*X.
```

The rule is applied over descending values until it gets to 1. In that moment the boundary condition set $Z = X^1 = Z$.

This is then multiplied by X, Y-1 times.

Example 6 – The power function

