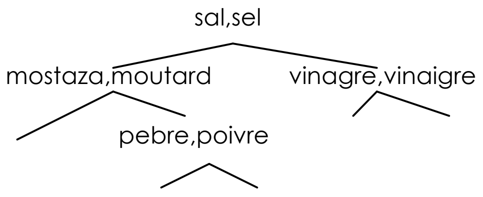
# Práctica 2

**Ex 1.** In Práctica 1 you represented the finite automata bellow in Prolog and implemented a predicate recognize which recognized the strings accepted by the automata. A parser, in addition to answer "yes" or "no" when presented with a string, should also output the required transitions in the automata for recognizing the string. For instance, if we give the string [j,a,j,a,!] as input, the parser should output [1,j,2,a,1,j,2,a,3,!,4]. Extend your práctica 1 program to implement a parser. Test your parser with different inputs.

**Ex 2.** Similarly, in Práctica 1 you implemented a simple parser (output: yes/no) for a grammar. Extend you program to a full parser, which when presented with a string, outputs the sequence of rules in the grammar used to derive the input string (in case it belongs to the language generated by the grammar).

**Ex 3.** Consider a data type "dictionary" intended to provide an efficient representation of a set of pairs of **names** with **values**. Thus, the dictionary **dic(Name,Value,Dic1,Dic2)** pairs **Name** with **Value**, together with all the pairings provided by subdictionaries **Dic1** and **Dic2**. We assume that the dictionary is ordered so that the names in **Dic1** are before **Name**, and all in **Dic2** are after, and both **Dic1** and **Dic2** are themselves ordered. Thus no names can be repeated in an ordered dictionary. Suppose the ordering is alphabetical order. Ordering relationships may be expressed using the symbol '**<**' for the two-place predicate 'is before' (for comparing strings you may use '**@<**') . As an example the following is an alphabetically ordered dictionary pairing Catalan and French words:

```
dic(sal,sel,
        dic(mostaza,moutard,
                void,
                dic(pebre,poivre,void,void)),
        dic(vinagre,vinaigre,void,void)).
```



Since the dictionary is ordered, it is possible to find values quickly without searching the whole tree. Write a Prolog predicate to 'look-up' a name in the dictionary and find its paired value. The predicate to be defined is

**lookup(Name,Dic,Value)**

meaning **Name** is paired with **Value** in dictionay **Dic,** which may be called by look(Name,Value) defined as look(Name,Value):-diction(D), lookup(Name,D,Value). i.e. ?- look(mostaza,V) should return V = moutard

In Práctica 3, 4 and 5 we will use the predicate lookup to allocate memory addresses to variables in the compilation process. This will be done by calling lookup (X,Dic,Address) with an un-instantiated dictionary Dic. Try:

?- lookup(y,D,Ay), lookup(x,D,21), lookup(z,D,23).

**Submitting your answer**
The práctica can be solved in teams of two people (1 submission for team). Submission is by email (to rafael.ramirez@upf.edu), the subject of the email must be *'P2- nombre1 apellido1- nombre2 apellido2*. and the message body should contain the answer for Ex.1, Ex.2 and Ex.3 as well as some tests to your programs. Deadline is the beginning of your next Práctica. Late submissions will be penalized.