```
/* Words in PLOC */
/* an ndfa representation */

/* A syllable in PLOC is made up of:
a) a single vowel
b) a single vowel followed by a 'u'
c) a  consonant followed by a single vowel
d) a 'consonant group' followed by a single vowel
[rules a, b, c and d define a core syllable]
e) a core syllable followed by a single consonant

The vowels in PLOC are : a, e, i, o, u
The consonants in PLOC are : b,c,d,f,g,k,l,m,n,p,r,s,t,v,z.
A consonant group is made up of a first_consonant followed by a
second_consonant
The group of first_consonants is represented by the list
[b,d,f,g,k,p,r,s,t]
The group of second_consonants is represented by the list [l,r,s,t]

A word is made up of a succession of n syllables; it cannot contain
a double consonant (a repeated consonant, e.g. dd or tt)
*/

% PROLOG PROGRAM

go :- nl,
   write('Word recognizer - in PLOC, an imaginary language'), nl,
   write('--------------------------'),nl,nl,
   write('callable goal is: wploc(String_to_be_examined)'),nl,
   write('where String_to_be_examined is entered as a letter list'),nl,
   write('e.g. [a,t,o,u,p,l,o,c]'),nl,
   write('the output is an analysis in terms of syllable
constituents'),nl,
   write('or the message : No'),nl,nl.

 wploc(String) :-  checkstring(String),
                   ndfa(syploc,String,Analysis),
                   write(Analysis).
```

```
 ndfa(Graph_Name,Symbol_List,[Graph_Name,initial(Initial_State)|
Analysis]) :-
        initial(Graph_Name,Initial_State),
        traverse(Graph_Name,Initial_State,Symbol_List,Analysis).

traverse(Graph_Name,
          State,
          [Symbol|Symbol_List],
          [transition(from(State),
                    to(Next_State),
                    reading(Symbol),
                    of_category(Category))|Category_List]) :-
         transition(Graph_Name,State,Next_State,Category),
         link(Symbol,Category),
traverse(Graph_Name,Next_State,Symbol_List,Category_List).

% silent move - used here to move from one syllable to the next
traverse(Graph_Name,
          State,
          Symbol_List,
          [transition(from(State),
                    to(Next_State),
                    syllable_border)|Category_List]) :-
         transition(Graph_Name,State,Next_State,silent_move),
       traverse(Graph_Name,Next_State,Symbol_List,Category_List).

% nothing more to read - making sure a final state has been
reached
traverse(Graph_Name,Final_State,[],[final(Final_State)]) :-
         final(Graph_Name,Final_State).
/* the transitions */
 % reading a vowel, moving from q0 (initial) to q1 (final)
   transition(syploc,q0,q1,vowel).
 % reading a consonant, moving from q0 to qc
   transition(syploc,q0,qc,cons).
 %  reading a first_consonant, moving from q0 to qfc
    transition(syploc,q0,qfc,first_cons).
 %  reading a second_consonant, moving from qfc to qc
    transition(syploc,qfc,qc,second_cons).
```

```
 % reading a vowel, moving from qc to q1
    transition(syploc,qc,q1,vowel).
 % reading a vowel, moving from q0 to qv
   transition(syploc,q0,qv,vowel).
 % reading a u, moving from qv to q1
   transition(syploc,qv,q1,u).
 % reading a consonant, moving from q1 to qcf (final)
   transition(syploc,q1,qcf,cons).
 % back to the beginning, start of a new syllable
   transition(syploc,qcf,q0,silent_move).

/* linking symbols and categories */
  link(Symbol,Category) :- listof(Category,Category_List),
                           inlist(Symbol,Category_List).

 /* initial and final states */
 initial(syploc,q0).
 final(syploc,q1).
 final(syploc,qcf).

 /* the category lists */
 listof(vowel,[a,e,i,o,u]).
 listof(u,[u]).
 listof(cons,[b,c,d,f,g,k,l,m,n,p,r,s,t,v,z]).
 listof(first_cons, [b,d,f,g,k,p,r,s,t]).
 listof(second_cons, [l,r,s,t]).

/* membership of a list  */
 inlist(First,[First|_]).
 inlist(Element,[_|Tail_List]):- inlist(Element,Tail_List).

/* checkstring : reject double consonants */
    checkstring(Symbol_List) :-
        listof(cons,Cons),
        inlist(C,Cons),
        append([_,[C,C],_], Symbol_List),
        !,fail.

    checkstring(_).
```