```prolog
%  Trie to record non-empty sequences of symbols

%  A trie represented as a list of representations of its subtrees.
%  Each tree as   tr( Symbol, B, Trie ),
%  where  B  is  c  or  n  (a complete sequence or not)

% in_trie( Word, Trie ) - list Word of symbols is represented in Trie

in_trie( [Symbol|Ss], Trie ) :- member( tr(Symbol,B,Trie1), Trie ),
        in_tree( Ss, tr(Symbol,B,Trie1) ).

% in_tree( Word, Tree ) - list Word of symbols is represented
%                         in the trie within the tree

in_tree( [], tr(_,c,_) ).
in_tree( Ss, tr(_,_,Trie1) ) :- in_trie( Ss, Trie1 ).

%   ?- in_trie( W, [ tr(a,n,[tr(b,n,[tr(c,c,[])])]), tr( b,c,[]) ] ).


% new_tree( Word, Tree ) - Tree is the tree containing only Word

new_tree( [Symbol], tr(Symbol,c,[]) ).
new_tree( [Symbol|Ss], tr(Symbol,n,[Tree]) ) :- new_tree( Ss, Tree ).

% replace( E, L, Enew, Lnew ) - E is an element of list L and
%         Lnew is list L with element E replaced by Enew.
%         (One occurence of E is replaced.)

replace( E, [E|T], EE, [EE|T] ).
replace( E, [H|T], EE, [H|Tnew] ) :- replace( E, T, EE, Tnew ).

% into_trie( Word, Trie, TrieNew ) - TrieNew is Trie with Word added.
%       They are the same if Word is in Trie

into_trie( [Symbol], Trie, TrieNew ) :-
%       member( tr(Symbol,B,Trie1), Trie ),
        replace( tr(Symbol,_,Trie1), Trie,
                 tr(Symbol,c,Trie1), TrieNew ).
into_trie( [Symbol|Ss], Trie, TrieNew ) :-
        Ss=[_|_],
%        member( tr(Symbol,B,Trie1), Trie ),
         replace( tr(Symbol,B,Trie1), Trie,
                  tr(Symbol,B,Trie2), TrieNew ),
         into_trie( Ss, Trie1, Trie2 ).
into_trie( [Symbol|Ss], Trie, [Tree|Trie] ) :-
        nonmember( tr(Symbol,_,_), Trie ),
        new_tree( [Symbol|Ss], Tree ).

% nonmember( E, L ) - E is not unifiable with any element of list L
%                      (SICSTus built-it)
```