CEng 242 Homework 6

Due 30^{th} May 2004

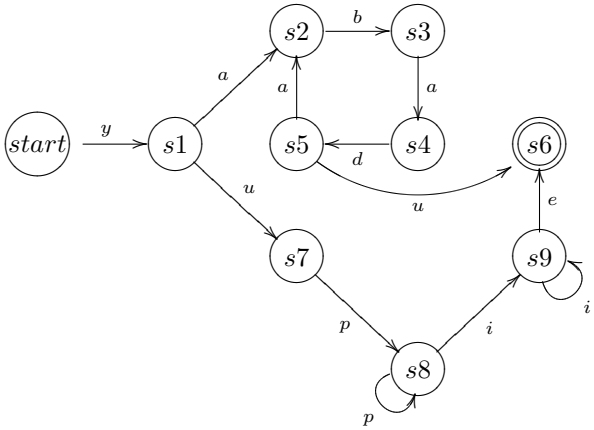
In this homework, you will use Prolog to implement a simple DFA parser. In the lectures, you are given the following code to parse a DFA:

```
parse(S,[]) :- final(S).
parse(S,[X|Rest]) :- transition(S,X,SNext), parse(SNext,Rest).

accept(L) :- parse(start,L).
```

You can define the transition relations by:

```
transition(start,y,s1).
transition(s1,a,s2).
transition(s2,b,s3).
transition(s3,a,s4).
transition(s4,d,s5).
transition(s5,a,s2).
transition(s5,u,s6).
transition(s1,u,s7).
transition(s7,p,s8).
transition(s8,p,s8).
transition(s8,i,s9).
transition(s9,i,s9).
transition(s9,e,s10).

final(s6).
final(s10).
```

for the following automata. It accepts the language $y(abad(abad)^*u|up(p)^*i(i)^*e)$



For spell checking, an indexed search of a given word in a dictionary is sufficient. However when you need to suggest correct spellings of a word, you have to find all possible words that are close to the written string. The user may:

- Replace: stroke a wrong key on the keyboard ("csr instead of car")
- Insert: skip a letter ("cr instead of car")
- Delete: stroke an extra letter ("casr instead of car")

One approach in spelling suggestion is to form a finite automata from the dictionary and make an error tolerant search on the automata to find the suggestions. In languages like Turkish

where there may be huge number of suffixed combinations of words, the finite automata seems to be the only solution.
Following this motivation, you will write a Prolog program to traverse a DFA with a given edit distance maximum. User is allowed to make the given number of errors and the correct string is computed as the result. The errors can be handled as:

- Replace: input symbol is consumed for each transition available from the current state.
- Insert: no input symbol is consumed but each transition available from the current state is made.
- Delete: input symbol is consumed but no transition is made.

So, in addition to normal "consume input symbol, make the corresponding transition" operation, those three error handling operations will be done during the parsing. When an error is being handled, the number of errors allowed is decremented. When this counter is 0, no more error handling operation is made.
You will write the predicate `accept(`*`Inp`*`,`*`Correct`*`, `*`N`*`)` where *`Inp`* is the list giving the input string, *`Correct`* is the corrected list (this will parse with 0 error in our DFA), and *`N`* is the maximum number of errors.
You may end up with repeated suggestions because of the mixing order of errors. One strategy is to use the `setof/3` built-in predicate of Prolog. It gets a solution template, the goal to solve and finds a list of solutions where the duplicates are removed as:
`?- setof(p(X),member(X,[1,2,3,2,1]),L)`
will instantiate `L=[p(1),p(2),p(3)]`. So you can write the predicate:
`suggest(Inp,Result,N) :- setof(X,accept(Inp,X,N),Result).`
which will remove the multiple occurrences of same solutions and return a list of solutions.

Example:

```
?- suggest([y,u,p,p,i,i,e],L,0).
L = [[y, u, p, p, i, i, e]] ;
No

?- suggest([y,u,p,e],L,0).
No

?- suggest([y,u,p,e],L,1).
L = [[y, u, p, i, e]] ;
No

?- suggest([y,a,b,a,a,b,e],L,1).
No

?- suggest([y,a,b,a,a,b,u],L,1).
No

?- suggest([y,a,b,a,a,b,u],L,2).
L = [[y, a, b, a, d, u]] ;
No

?- suggest([y,a,b,a,d,a,b,a,d,u],L,2).
L = [[y, a, b, a, d, a, b, a, d, u]] ;
No

?- suggest([y,a,b,a,d,a,b,a,d,u],L,4).
```

```
L = [[y, a, b, a, d, a, b, a, d, a, b, a, d, u],
     [y, a, b, a, d, a, b, a, d, u],
     [y, a, b, a, d, u]] ;
No

?- suggest([y,a,m,a,d,a,m,a,d,u],L,2).
L = [[y, a, b, a, d, a, b, a, d, u]] ;
No

?- suggest([y,a,d,u],L,2).
L = [[y, a, b, a, d, u]] ;
No

?- suggest([y,a,d,a,b,u],L,2).
L = [[y, a, b, a, d, u]] ;
No

?- suggest([y,a,b,a,d,u],L,2).
L = [[y, a, b, a, d, u]] ;
No

?- suggest([y,u,m,m,i,e],L,2).
L = [[y, u, p, i, e], [y, u, p, i, i, e], [y, u, p, p, i, e]] ;
No

?- suggest([y,u],L,6).
L = [[y, a, b, a, d, u], [y, u, p, i, e], [y, u, p, i, i, e],
     [y, u, p, i, i, i, e], [y, u, p, i, i, i, i, e], [y, u, p, p, i, e],
     [y, u, p, p, i, i, e], [y, u, p, p, i, i, i, e], [y, u, p, p, p, i, e],
     [y, u, p, p, p, i, i, e], [y, u, p, p, p, p, i, e]] ;
No
```

Submit only parsing predicates and auxiliary predicates related to parsing in your submission file '`hw6.pl`'. Assume your dfa will be given separately in '`dfa.pl`'. Only submit '`hw6.pl`' This homework has a very simple solution, do not underestimate the size of your solution.