Übung 2

November 5, 1999

The object of today's exercise is to write a predicate that will remove ϵ-transitions from a finite state automaton.

Representing Automata An automaton is normally defined as a 5-tuple $\langle \Sigma, Q, q_0, F, \delta \rangle$, with Σ a finite set (the vocabulary), Q a finite set (the states), $q_0 \in Q$ (the start state), $F \subseteq Q$ (the final states) and $\delta \subseteq Q \times \Sigma \cup \{\epsilon\} \times Q$ (the transition relation).

We will represent automata via clauses asserted directly to the Prolog database. We will leave Σ implicit. Q will always be a set of integers from 0 to n, also implicit. Every automaton will have a name, which may be provided by the user (for automata consulted from a file) or else, for automata constructed automatically, the name will be an integer. The start state will be given via a unit clause of the form `start(AutName,StartState)`. The final states will be given via a unit clause of the form `finals(AutName,ListOfStates)`. It will also be useful to record the highest numbered state of each automaton, which we will do via a unit clause of the form `maxstate(AutName,State)`.

The transition relation will be given via unit clauses of the form

```
AutName::(State0,Symbol,State).
```

Note that `::/2` is a user defined operator.

First Step In order to test your predicate you will need some automata with ϵ-transitions in them. The constructions implementing the regular operations (concatenation, alternation and iteration) typically produce automata with lots of ϵ-transitions.

My own implementation of these three operations, plus some simple automata that make useful building-blocks for constructions, can be found in the file

```
~cornell/lab.pl
```

So the first thing you should do is to copy this file into your own working directory:

```
unix prompt> cp ~cornell/lab.pl .
```

Note that `lab.pl` contains the definition of the '`::/2`' operator. So (a) you don't have to define it yourself but (b) you will have to consult `lab.pl` before you consult any other file containing this operator.

Second Step Now you need to construct some automata with ϵ-transitions in them. You might try the following.

```
?- [lab].
...
?- star(symb(a),Aut1), union(Aut1,symb(b),Aut).
```

Then remember what Prolog returns as the value of the variable `Aut`. That is the name of an automaton now in the Prolog database which recognizes the language $(a^*)|b$. You should try fooling around with your own constructions. The bigger they are, the more ϵ-transitions they will have, and the more useful your solution to the main part of the Übung will be!

An Example Here is an idea of what you can expect if your code works. Suppose we start with the automaton for the single string a, i.e., the automaton named `symb(a)`, an instance of the automaton template `symb(A)` from `lab.pl`. This is pictured in Fig. 1. Then the result of calling "`?- star(symb(a),Aut1).`" will be as in Fig. 2.

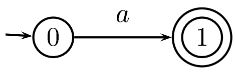

Figure 1: Graphic presentation of the automaton `symb(a)`.

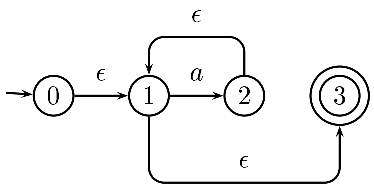

Figure 2: Result of `?- star(symb(a),Aut1).`

On the assumption that the previous call returned `Aut1=0`, a subsequent call "`?- union(0,symb(b),Aut2).`" should construct the automaton in Fig. 3. On the assumption that now `Aut2=1`, a call to "`?- eremove(1,Aut3).`" should

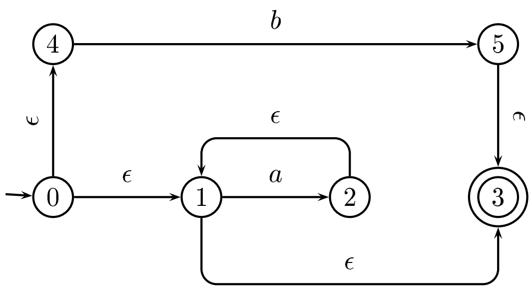

Figure 3: Result of `?- union(0,symb(b),Aut2).`

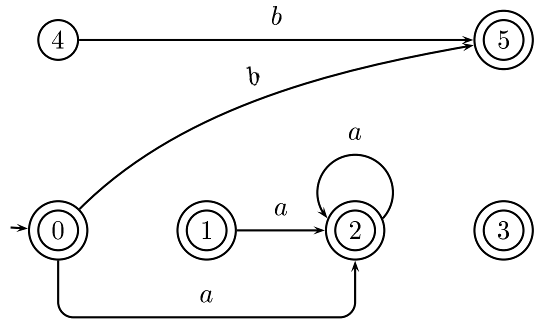

Figure 4: Result of `?- eremove(1,Aut3).`

result in the automaton pictured in Fig. 4. Observe that states 1, 3 and 4 are no longer reachable, but that's okay for now!

The Main Event Here is a formal description of the ϵ-removal construction. (Based on Sippu & Soisalon-Soininen 1988.) Given an input automaton $\mathcal{A} = \langle \Sigma, Q, q_0, F, \delta \rangle$. The transformed automaton will be $\mathcal{B} = \langle \Sigma, Q, q_0, F', \delta' \rangle$. (So, as you can see, the only thing you really have to construct is the new transition relation and the new set of final states.)

The new transition relation δ' is as follows.

$$\delta'(q_i, a, q_j) \iff (\exists q \in Q)[\delta^*(q_i, \epsilon, q), \text{ and } \delta(q, a, q_j)]$$

Note that in the definition that's δ^* and δ, that is, the transition relation (and its transitive closure) from $\mathcal{A}$. The idea is that, whenever you can get from q_i to q by a series of free jumps, and then from q to q_j by reading an a, then make a new arc in the new automaton $\mathcal{B}$ going directly from q_i to q_j, reading a.

Important! The reflexive transitive closure of δ is defined as follows. Here w is a *string*, not a single symbol. So it can be the empty string. On the other hand a here denotes a single symbol. So wa denotes an arbitrary string whose last character is a. If w is empty, then $wa = \epsilon a = a$.

$$\begin{aligned} &\delta^*(q, \epsilon, q) && \text{for all } q \in Q\ . \\ &\delta^*(q_1, wa, q_2) \iff && \delta^*(q_1, w, q) \text{ for some } q \in Q \text{ and } \delta(q, a, q_2)\ . \end{aligned}$$

This is just the automata theoretic version of the path-relation in a graph. In a graph, there is always an empty path from any node to itself. So also here.

This means that $\delta'(q_i, a, q_j)$ should hold in the special case that $q = q_i$. You shouldn't have to do anything special to ensure that this will happen. But it means that all the non-ϵ-transitions in the input automaton get to be part of the output automaton. However, as you can see, no ϵ-transitions will make it into the output automaton.

Final States The new final states F' will be

$$\{q \in Q \mid (\exists q' \in Q)[\delta^*(q, \epsilon, q') \wedge q' \in F]\}$$

Again, in case $q = q'$, this entails that all the final states in F will be placed in F'. In addition, we add all the states from which there is a free jump to a final state in the underlying automaton.

Hints This problem is rather easy if you break it down into the right parts. So here are some hints as to how you might break the problem down into easier parts.

In both parts of the construction (constructing the transition relation and constructing the set of final states) you need to be able to identify all of those node pairs q_1 and q_2 which are connected by a sequence (possibly empty!) of ϵ-transitions. So you should write a predicate which identifies such pairs of states. Then you can use `setof/3` to translate these formal definitions almost word-for-word into Prolog code. So you will be greatly helped by defining a predicate

```
%%% e_reachable_s_trans( +Aut_name, ?State0, ?Symbol, ?State )
%%% ==========================================================
%%% This holds if there is a series of e-transitions in
%%% automaton Aut_name from State0 to some state State1,
%%% and from State1 there is a transition on Symbol to
%%% State.
%%%
```

Observe that this only expects `Aut_name` to be bound on entry, so it can be used as a goal in a single call to `setof/3` to build your entire transition table for you!

This in turn will be made much easier if you define a predicate `etrans_star` as follows.

```
%%% etrans_star( +AutName, ?State0, ?State )
%%% ========================================
%%% This predicate holds if there is a sequence of N
%%% e-transitions that get you from State0 to State.
%%% If N is 0, then State0 = State.
%%%
```

This predicate, `etrans_star/3`, can also be used to build a predicate `e_reachable_final` that holds in an automaton between all states `Q0` and `Q` just in case there is an ϵ-path from `Q0` to `Q`, and `Q` is final.