maria.dascalu@upb.ro

Write a program that builds deterministic finite-state automaton (DFA) from regular expressions (RE), using a table of λ-transitions. You can use any programming language.

- (2p) Implement a representation (e.g. class) of the deterministic finite-state automaton.
- (1p) Build the prefix form of the initial regular expression.
- (1p) Build the transition table.
- (1p) Test the DFA.

Solution:

Part 1. RE to transitional system (TS) (= finite automation with λ-transitions)

Part 2. TS to DFA

Part 1. RE to TS

For representing the RE as a TS, we will use a transition table, with p lines and 3 columns:

- p is the numbers of the states (1…p);
- the first column, called *symbol*, contains the symbols used to make the transitions;
- the second and the third columns (*next1* and *next2*) contains the states after the transitions.

The algorithm:

Input: RE

Output: the ST, represented through the following vectors: symbol, next1 and next.

Method:

Step1. The tree attached to the expression is built. The brackets are not represented in the tree.

Step2. The tree is traversed in pre-order and each visited node receives a number k: 1,2,…. The nodes which contain "." operator (e.g., in the RE "ab" there is a "." operator between the lexicons) do not receive any number=> prefixed form of the initial RE

Step3. The tree is traversed in post-order and each node N receives a pair of numbers (i,f), which represents (the initial state, the final state) of the automaton corresponding to the sub-tree with the root node in n:

- if the node N has the number k, then N.i=2k-1, N.f=2k

- if the node is labeled with ".", then N.i = S.i, N.f = D.f, where S is the left child of the node N and D is the right child of the node N

Step4. The vector of transitional symbols and intermediary states (next1, next2) is initialized (with " ", respectively with 0).

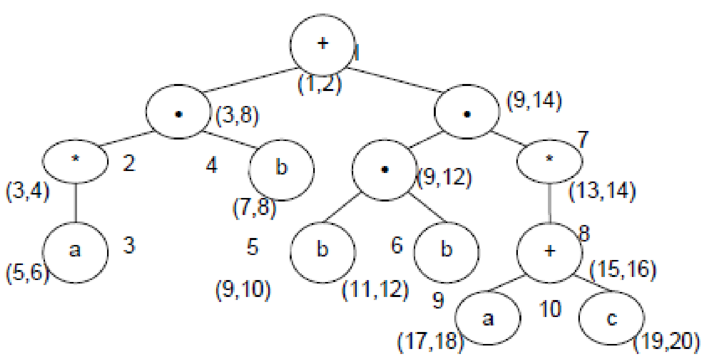
Step5. The tree is traversed again. Let's consider N to be the current node and S and D its children. Taking into account the label of node N, we do the following:

- if N is labeled with "|" (or "+", depending of the notation):
  next1[N.i] = S.i, next2[N.i] = D.i,
  next1[S.f] = N.f, next1[D.f] = N.f
- if N is labeled with ".":
  next1[S.f] = D.i
- if N is labeled with "*":
  next1[N.i] = S.i, next2[N.i] = N.f,
  next1[S.f] = S.i, next2[S.f] = N.f
- if N is a lexicon, let's say "a" (a leaf):
  simbol[N.i] = 'a', next1[N.i] = N.f

<u>Example:</u>

Input: E = a*b + bb(a + c)*

The tree:



maria.dascalu@upb.ro

The output: the transition table

States	Symbol	next1	next2
1		3	9
2		0	0
3		5	4
4		7	0
5	a	6	0
6		5	4
7	b	8	0
8		2	0
9	b	10	0
10		11	0
11	b	12	0
12		13	0
13		15	14
14		1	0
15		17	19
16		15	14
17	a	18	0
18		16	0
19	c	20	0
20		6	0

Part 2.TS to DFA

Input: $ST=(Q,\Sigma,\delta,q_0,F)$, $\delta:Qx(\Sigma\cup\{\lambda\}) \rightarrow \wp(Q)$

Output: DFA= $A'=(Q',\Sigma',\delta',q_0',F')$, $\delta':Q'x\Sigma' \rightarrow Q$

Method:

q_0' = Lambda(q_0); $Q' = \{q_0'\}$; marcat(q_0') = false; $F' = \emptyset$;

if ($q_0' \cap F \neq \emptyset$) **then** $F' = F' \cup \{q_0'\}$;

while ($\exists q' \in Q'$ **and** marcat(q')==false) **do begin**

  foreach ($a \in \Sigma$) **do begin**

    s=Lambda(Delta(q',a));

    if ($s \neq \emptyset$) **then begin**

      if($s \notin Q'$) **then begin**

        $Q' = Q' \cup \{s\}$; marcat(s)=false;

        if ($s \cap F \neq \emptyset$) **then** $F' = F' \cup \{s\}$;

      end

```
            δ'(q',a)=s;
        end
    end
    marcat(q')=true;
end
```

In the algorithm, 2 functions appear:

Lambda (state) – the closure of δ to λ-transitions (let's called it R)

Delta (state, symboll) = { δ(s,symbol) | ∀s∈state }

Solution for implementing the Lambda function:

Input: transitional table

Output: R = the closure of δ to λ-transitions

Method: using a stack

```
R = S; STACK= S;
while (STACK ≠ Φ) {
    q = STACK.pop(); // q is extracted from the stack
    T = δ(q, λ) ;
    if(T ≠ Φ) {
        for ( p  ∈ T ) {
            if ( p ∉ R) {
                R = R ∪{p};
                STACK.push(p);//p is added to the stack
            }
        }
    }
}
```