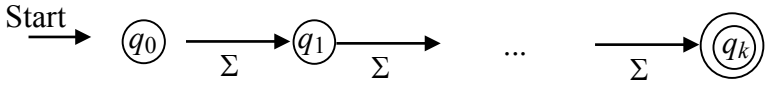
Exercises for FLL, Fall 2017, sheet 4 – Solutions

Return Thursday October 26, in class

Exercise 1. Let L be a regular language specified by a regexp E. Sketch a procedure for deciding whether $L = \Sigma^k$ for some $k > 0$.

Solution. There are many ways of how this can be decided. One way is to first transform E into an equivalent DFA, then from this DFA construct the minimal DFA A for L. Then obviously $L = \Sigma^k$ for some k iff A has the form



Note. The use of the word "obviously" in mathematical proofs is a delicate affair. One never knows what the reader is ready to accept as obvious. Here I think we have a borderline case, and one might feel the need to prove that if $L = \Sigma^k$ then the minimal automaton actually has the given form (it is really obviously obvious that this kind of DFA accepts $L = \Sigma^k$, so the only possibly questionable claim is its minimality). Minimality of DFAs of the shown form could be proven by going through the table-filling algorithm and showing that all the shown states are distinguishable. This would be a case for extra grading points.

Exercise 2. Give a CFG for all words over the terminal alphabet $T = \{\boldsymbol{a}, \boldsymbol{b}, +, *, (,), \boldsymbol{\varepsilon}, \varnothing\}$ that are regular expressions over $\Sigma = \{a, b\}$.

Solution. Put $V = \{E\}$ (which automatically makes E the start variable). Then simply replicate the inductive definition of regexps:

$E \to \boldsymbol{a} \mid \boldsymbol{b} \mid \boldsymbol{\varepsilon} \mid \varnothing \mid (EE) \mid (E+E) \mid (E^*)$

Exercise 3. Describe a generic method by which a CFG for the language $L(E)$ of any regular expression E can be constructed from E. *Hint*: use induction on the structure of E.

Solution. A method to come up with a CFG for any language described by a regular expression E over an alphabet Σ (or T, if we think of it from the angle of CFGs) is to combine CFG's for subexpressions E' of E, by induction over the structure of E:

Basis:
For $E = \varnothing$, $\boldsymbol{\varepsilon}$, or $\mathbf{a}$ (where $a \in \Sigma$) it is (almost) trivial to find CFGs $G_\varnothing = (V_\varnothing, \Sigma, P_\varnothing, S_\varnothing)$, $G_\varepsilon = (V_\varepsilon, \Sigma, P_\varepsilon, S_\varepsilon)$, $G_a = (V_a, \Sigma, P_a, S_a)$, that generate the corresponding languages, so I will skip this here.

Induction:

1. (union) Let E, F be regexps over Σ, with grammars $G_E = (V_E, \Sigma, P_E, S_E)$, $G_F = (V_F, \Sigma, P_F, S_F)$ for the languages of E and F. Without loss of generality, we may assume that V_E and V_E are disjoint. Then we get an obvious grammar for the language of $(E + F)$ by $G_{E+F} = (V_E \cup V_E \cup \{S^*\}, \Sigma, P_E \cup P_F \cup \{ S^* \to S_E , S^* \to S_F\}, S^*)$, that is, we create a new

start symbol from which either the old start symbol of the CFG for the language of E or the old start symbol of the CFG for the language of F can be produced.

2. (concatenation) Similar to union, except the new ruleset is $P_E \cup P_F \cup \{S^* \rightarrow S_E\, S_F\}$.
3. (Kleene star) A grammar for the language of E^*, given $G_E = (V_E, \Sigma, P_E, S_E)$, is $G_{E^*} = (V_E, \Sigma, P_E \cup \{ S_E \rightarrow S_E\, S_E\, , S_E \rightarrow \varepsilon \}, S_E)$.