

# Homework #2

Due: Wednesday, January 21th, 2015, 11:59 PM

**Problem 1 (Automata Tutor)** Complete the problems that constitute CSE 105's HW2 on Automata Tutor. (Problems will be posted soon.)

**Problem 2 (Closure properties / Haskell)** Theorem 1.25 in the textbook proves that regular languages are closed under union. The proof is constructive, i.e., it gives an algorithm that on input two DFAs for languages  $L_1$  and  $L_2$ , produces a DFA for  $L_1 \cup L_2$ . In the class notes on Haskell, you have also seen that the proof is immediately translated into a working computer program that implements the transformation. In this problem you are asked to prove some similar closure properties of regular languages.

- a Prove that regular languages are closed under intersection by giving a transformation that on input two DFAs for languages  $L_1$  and  $L_2$ , produces a DFA for  $L_1 \cap L_2$ .
- b For any language  $L \subseteq \Sigma^*$  and symbol  $a \in \Sigma$ , let  $a \setminus L = \{w \in \Sigma^* \mid aw \in L\}$ , i.e., the set of strings in  $L$  that begin in  $a$ , but with the first  $a$  removed. Prove that for any such  $L$  and  $a$ , if  $L$  is regular then also  $a \setminus L$  is regular by giving a transformation  $T_a(M)$  that on input a DFA  $M$  with alphabet  $\Sigma$  and a symbol  $a \in \Sigma$ , outputs a DFA for the language  $a \setminus L(M)$ .

For each part, your solution should consist of a mathematical description of the transformation, and a haskell program implementing it. The mathematical proof should be typeset. For the haskell part, start from the template files `HW22a.hs` and `HW22b.hs` provided on the course webpage, and replace the `...` as directed by the comments.

Your solution to this problem should consist of 4 files (`HW22a.pdf`, `HW22a.hs`, `HW22b.pdf`, `HW22b.hs`) and submitted using Bundle from the class accounts.

**Problem 3 (Modeling computation)** In this problem you are asked to come up with a mathematical definition for a new computational model, along the lines of the notes on DFAs posted on the class webpage.

We consider a generalization of DFAs called second-order deterministic finite automata, or DFA<sup>(2)</sup>s for short. The transition function of a DFA<sup>(2)</sup> depends not only on the current symbol and the current state, but also the *previous* state. That is, the transition function is a function  $\delta^{(2)}: Q \times Q \times \Sigma \rightarrow Q$ , that maps every pair of states ( $q_{\text{prev}}, q_{\text{curr}}$ ) and every input symbol  $a$  to a next state  $q_{\text{next}} = \delta^{(2)}(q_{\text{prev}}, q_{\text{curr}}, a)$ . There is a special case for processing the very first input symbol, when there is no previous state; we

define the next state to be  $\delta^{(2)}(q_0, q_0, a)$ , where  $q_0$  is the start state and  $a$  is the first input symbol.

**a** Define the computations performed by the automaton, along the same lines as in the lecture notes. Specifically, your solution should include the definition of:

- A set of possible configurations  $C_M$  that can be used to define the computations of the automaton
- An input function  $I_M: \Sigma^* \rightarrow C$  mapping the input string to the initial configuration
- The set of halting configurations  $H_M \subseteq C_M$ .
- An output function  $O_M: H_M \rightarrow \{T, F\}$  mapping the halting configurations to either (T) rue or (F) alse.
- A transition relation  $R_M \subseteq C_M \times C_M$  describing valid steps in a computation. The automaton should be deterministic, i.e., for any configuration  $c_1 \in C_M$  the number of next possible configurations  $c_2 \in C_M$  (such that  $(c_1, c_2) \in R_M$ ) should be 0 if  $c_1 \in H_M$  is halting, and 1 if  $c_1 \notin H_M$  is not halting.

Given the above definitions, the result of running an automaton on an input string is defined exactly the same way as we did for DFAs: the result of running  $M$  on input  $w$  is  $O_M(c_n)$  where  $c_1 = I_M(w)$ ,  $(c_i, c_{i+1}) \in R_M$  for  $i = 1, \dots, c_{n-1}$  and  $c_n \in H_M$ . (You don't need to repeat this part of the definition.)

Then, implement the definitions in haskell starting from the template file `HW23a.hs` on the course webpage, and replacing the `...` as directed in the comments. Refer to the `DFA.hs` file for an example of how all functions are defined for the standard DFAs. The definition of the `runDFA2` and `execDFA2` functions is identical as for DFAs, and the functions have already been provided in the template file. If your definitions are correct, executing `execDFA2 m w` should output `True` if the second order DFA  $m$  accepts  $w$ , and `False` otherwise.

Your answer to this problem should consist of a document describing your mathematical definitions `HW23a.pdf`, and a file containing the haskell implementation `HW23a.hs`.

**b** Prove that  $\text{DFA}^{(2)}$ s are equivalent in power to DFAs, i.e., prove that any DFA  $M = (Q, \Sigma, \delta, s, q)$  can be transformed into an equivalent  $\text{DFA}^{(2)}$   $M' = (Q', \Sigma, \delta', s', q')$  such that  $\mathcal{L}(M) = \mathcal{L}(M')$ , and, similarly, any  $\text{DFA}^{(2)}$   $M = (Q, \Sigma, \delta, s, q)$  can be transformed into a DFA  $M' = (Q', \Sigma, \delta', s', q')$  such that  $\mathcal{L}(M) = \mathcal{L}(M')$ .

Your solution to this part should consist of a document `HW23b.pdf` containing the mathematical definition of the above transformations, and a file `HW23b.hs` containing the haskell implementation of these functions. As usual, start from the template file `HW23b.hs` on the course webpage, and complete the definitions as directed.

Your solution to this problem should consist of 4 files (`HW23a.pdf`, `HW23a.hs`, `HW23b.pdf`, `HW23b.hs`) and submitted using `Bundle` from the class accounts.