Automata Theory and Computability

Assignment 1

(Due on Tue 6th Sep 2016)

1. Let L be a regular language. Prove that the language

$$\textit{mid-thirds}(L) = \{v \mid \exists u, w : |u| = |v| = |w| \text{ and } uvw \in L\}$$

   is also regular. Argue for yourself that your construction is correct, but *don't* write the proof of correctness.

2. Consider the language L over the alphabet $\{a, b\}$ defined by the following MSO sentence:

$$\forall x \forall y (((Q_a(x) \wedge Q_b(y)) \implies x < y) \wedge (Q_a(x) \implies \exists z Q_b(z))).$$

   Give a regular expression describing the language L.

3. Give Monadic Second Order (MSO) logic sentence over the alphabet $\{a, b\}$ which defines the following languages:

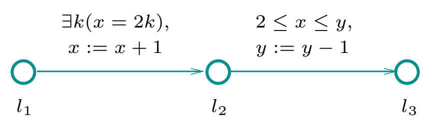
   (a) $(ab)^*$.

   (b) All strings over $\{a, b\}$ satisfying the condition that "between any two consecutive a's there are an even number of b's."

4. Construct an automaton that accepts all the satisfying assignments of the Presburger formula $\exists y(x = 2y + 1)$, using the procedure described in class. What should be the output of the resulting automaton on the strings "00000" and "10011" respectively?

5. A (straight-line) Presburger program is a sequence of `if`-statements, and uses two variables x and y. The guard of each `if`-statement is a Presburger logic formula with free variables in $\{x, y\}$, and the body is an assignment statement of the form $x := e$ where e is a term (over the variables x and y) in Presburger logic. More precisely, such a program can be modelled as sequence of control locations $l_1, \ldots, l_{n+1}$ ($n \geq 1$), and transitions t_i from l_i to l_{i+1} being labelled with a Presburger guard $g(t_i)$ and an update statement $u(t_i)$. The program executes in the expected manner, beginning in the initial location l_1, in an initial state s where x and y take arbitrary values in $\mathbb{N}$, checking whether the state satisfies the guard of transition t_1, and if so, applying the update $u(t_1)$ to s and going to location l_2. A similar step is then performed from l_2, and so on. If the guard of a transition is not satisfied by the current state, or if the update assigns a negative value to a variable, the program gets "stuck."

   For example the figure below shows a Presburger program. When started in a state $(x \mapsto 2, y \mapsto 5)$, it goes to l_2 in the state $(x \mapsto 3, y \mapsto 5)$, and finally to l_3 in the state $(x \mapsto 3, y \mapsto 4)$.



Given a precondition *pre* on the initial states, and a post-condition *post* on the final states, we say a Presburger program P satisfies the conditions $(pre, post)$ iff every execution of P that begins in a state satisfying *pre* either never reaches l_{n+1}, or reaches there in a state satisfying *post*. For example, because of the given execution, the program above does *not* satisfy the pre/post-condition $(x < y, x > y)$.

Give a procedure to check whether a given Presburger program P, with Presburger conditions *pre* and *post*, satisfies the pair $(pre, post)$.

6. McNaughton and Papert showed that the class of languages definable by the first-order fragment of MSO(Σ) (where we disallow quantification over set variables) coincides with the class of languages defined by *counter-free* automata over Σ. A DFA $\mathcal{A} = (Q, s, \delta, F)$ is said to have a *counter* if there exist distinct states $q_0, q_1, \ldots, q_n$ in $\mathcal{A}$, with $n \geq 1$, and a string $w \in \Sigma^*$, such that $\delta(q_i, w) = q_{i+1}$ for each $i \in \{0, \ldots, n-1\}$ and $\delta(q_n, w) = q_0$. A DFA is said to *counter-free* if it has no such counter, and a regular language is said to be counter-free if the minimal automaton accepting it is counter-free. Prove the following characterisation of counter-free languages: Let $L \subseteq \Sigma^*$ be regular. Then L is counter-free iff there do not exist words u, v, and w in Σ^* such that $uv^iw \in L$ for infinitely many i and $uv^iw \notin L$ for infinitely many i.