As usual, hand this in on Moodle as FirstnameLastnameHw5.pdf or FirstnameLastnameHw5.txt.

1. [15 pts.] Recall the vocabulary \( \Sigma_{abc} = (\leq^2, A^1, B^1, C^1; \text{min}, \text{max}) \) of strings from alphabet \( \{a, b, c\} \). The associated axiom, \( \Gamma_{abc} \), says that \( \leq \) is a total ordering of positions, with \( \text{min}, \text{max} \) the first and last elements respectively, and each position holds exactly one letter – \( A(p) \) means that \( a \) occurs at position \( p \).

\[
\Gamma_{abc} \overset{\text{def}}{=} \forall x \, x \leq x \land \\
\forall xy \,(x \leq y \land y \leq x \rightarrow x = y) \land \\
\forall yz \,(x \leq y \land y \leq z \rightarrow x \leq z) \land \\
\forall xy \,(x \leq y \lor y \leq x) \land \\
\forall x \,(\text{min} \leq x \land x \leq \text{max}) \land \\
\forall x \,((A(x) \lor B(x) \lor C(x)) \land (A(x) \rightarrow (\neg B(x) \land \neg C(x))) \land (B(x) \rightarrow \neg C(x)))
\]

Each of the following formulas, \( \sigma_i \), defines a set of non-empty strings over the alphabet \( \{a, b, c\} \). Give an English statement \( E_i \) characterizing this set and write a regular expression, \( r_i \), which matches exactly this set. Note, since we don’t allow a world with no elements, we exclude the empty string, so your regular expressions, \( r_i \), should not match the empty string.

(a) \( \sigma_1 \overset{\text{def}}{=} A(\text{min}) \)
(b) \( \sigma_2 \overset{\text{def}}{=} \neg A(\text{max}) \)
(c) \( \sigma_3 \overset{\text{def}}{=} \forall xy \,(x \leq y \land A(x) \rightarrow \neg B(y)) \)

2. [20 pts.] For any string \( w \in \Sigma^* \), let \( w^R \) be the reverse of \( w \), e.g., \( \text{hello}^R = \text{olleh} \). For a language \( S \subseteq \Sigma^* \), let \( S^R \) be the reverse of every string in \( S \), i.e.,

\[
S^R = \{ w^R \mid w \in S \}.
\]

Prove that if \( S \) is regular, then so is \( S^R \).

[Hint: define a simple recursive algorithm that transforms any regular expression \( e \) to an expression \( e' \) such that \( L(e') = L(e^R) \) and prove by induction that your algorithm is correct.]

3. [30 pts.] Prove that if we do a dfs on a directed acyclic graph (DAG) \( G \), then the reverse of the finish times is a topological order for \( G \).

4. [35 pts.] Let \( D \) be a directed graph. Do a dfs on \( D \). Let \( D^R \) be the reverse graph of \( D \), i.e., the edge \( (a, b) \in E^D \) iff \( (b, a) \in E^{D^R} \). Do a dfs on \( D^R \) in which the roots of dfs trees are chosen in DFSmain in the order of the reverse finish times from the first dfs.

Try this on a bunch of small examples.

Prove that the dfs trees in the second run of dfs are exactly the strongly connected components of \( D \). This thus gives an \( O(n + m) \) algorithm to compute SCC’s.