1 Nested-word automata (12=4+8 points)

1.1 Problem 1 (4 points)
Construct a deterministic NWA (over finite words) over the alphabet $\Sigma = \{a, b\}$ that only accepts well-matched nested-words (i.e., upon termination the stack should be empty, and there should be no unmatched calls or returns).

1.2 Problem 1 (8 points)
Show that NWAs (over finite words) are closed under concatenation.

Careful: The first nested-word might have unmatched calls and similarly the second NWA might have unmatched returns. Make sure they match properly.

Important: Specify in your construction what are the input and output NWAs. I don’t require a proof of correctness, but clearly state the invariants maintained by your construction (e.g., after reading the nested word $a_1\ldots$ the NWA C is in state $q$ with stack $p_1\ldots p_n$ iff $\ldots$).

2 Probabilistic model checking (13=8+5 points)

2.1 Problem 1: 8 points
Exercise 10.1 on page 899 of Principles of model checking.

Remark: least fixed point is on page 762, and Transient state probabilities are on page 768 (we saw both in class). Carefully define the matrix $A$ and the vector $b$. You can use Matlab, Octave, any solver, or solve by hand the induced sets of equations. (You can use Octave without installing it here http://octave-online.net/) For point c), you only need to use one method. But show the works.
2.2 Problem 2: 5 points

Exercise 10.3 on page 899 of Principles of model checking.

3 Symbolic transducers (17=7+3+7 points)

We refer to the paper “Symbolic Finite State Transducers: algorithms and applications” by Veanes et al.

3.1 Problem 1: 7 points

Provide the code. Use Bek (http://rise4fun.com/bek) to model the following two functions:

- \textit{deletealpha}, that deletes all the alphabetic characters in \([a - z]\) from a string. E.g., \(\text{deletealpha}(ab1a33) = 133\).

- \textit{stutter}, that duplicates every character of a string. E.g., \(\text{stutter}(abcabc) = aabbccaabbcc\).

Now use Bek to prove that

- \textit{deletealpha} is idempotent: running it twice in a row is equivalent to running it twice;

- \textit{deletealpha} and \textit{stutter} commute.

- the output of \textit{stutter} always has even length.

3.2 Problem 2: 3 points

Consider the algorithm for checking equivalence described in Section 3.2 of the paper. The algorithm relies on Proposition 2.

Is the proposition still true if \(A\) and \(B\) are not single-valued? What if only one of \(A\) and \(B\) is single-valued. If yes, argue why, if not provide a concrete counterexample.

3.3 Problem 3: 7 points

In class we argued that the equivalence problem for finite state transducers is undecidable. We reduced the Post Correspondence Problem (PCP) to it. In this problem you are required to “complete” the proof.
An instance of PCP over the alphabet Σ is given by two sequences of strings \((α_1, \ldots, α_k)\) and \((β_1, \ldots, β_k)\) such that \(α_i, β_i \in Σ^*\) for every \(i\). A solution to the PCP is a sequence \(i_1 \ldots i_n\) such that every \(i_j \in [1..k]\) and
\[
α_{i_1} \ldots α_{i_n} = β_{i_1} \ldots β_{i_n}
\]
Let \(M = \max\{|α_1|, \ldots, |α_n|, |β_1|, \ldots, |β_n|\}\). In class we argued that one can construct an FST that accepts the following transduction.
\[
\{(i_1 \ldots i_n, t) \mid |t| \leq nM \land (t \neq α_{i_1} \ldots α_{i_n} \lor t \neq β_{i_1} \ldots β_{i_n})\}
\]
Show how to construct such a finite state transducers.

**Important:** provide the intuition first and then the construction! Describe what each part of the FST is supposed to do. Provide the set of transitions symbolically, don’t draw any diagram.

### 4 \(L^*\) (6 points)

#### 4.1 Problem 1: 6 points

We briefly mentioned in class that the following problem is NP-Complete.

Given two DFAs \(A\) and \(B\), such that \(L(A) \cap L(B) = ∅\), find the minimal DFA \(C\) that separates them. That is

- \(L(A) \subseteq L(C)\);
- \(L(B) \cap L(C) = \emptyset\);
- every \(C'\) with the two properties above has at least \(n\) states, where \(n\) is the number of states in \(C\).

On the other hand we saw that \(L^*\) can learn a DFA accepting a given regular language \(R\) in polynomial time. Let’s say we adapt \(L^*\) in the following way:

- Membership queries: given a string \(w\), mark it as positive example if \(w \in L(A)\) and negative if \(w \in L(B)\)
- Equivalence queries: is the conjectured automaton \(M\) such that \(L(A) \subseteq L(M)\) and \(L(B) \cap L(M) = \emptyset\)? If not provide a counterexample.

Why doesn’t this algorithm work?
5 Reactive Synthesis: 6 points

Solve the following reactive synthesis problem. A depiction of the situation is given in Figure 1.

You have to synthesize a controller for a traffic light that has the following inputs, outputs, and requirements.

**Input signals**
- C: car is waiting on farm road
- P: pedestrians want to cross the highway
- Eh: emergency vehicle on highway
- Ef: emergency vehicle on farm road

**Outputs signals**
- h: highway light is green
- f: farm road light is green

**Requirements**
- Lights should not be both green
- Lights should both turn green every now and then
- If pedestrians approach, farm light should go green (i.e. on input, the system should immediately output f).

You have to:
• Formalize the requirements in LTL;

• Is the system realizable? If so provide a controller otherwise provide a winning strategy for the environment.