CSCI 3130: Formal Languages and Automata Theory — Homework 5
The Chinese University of Hong Kong, Fall 2018 — due 11:59pm Thursday November 22

Collaborating on homework is encouraged, but you must write your own solutions in your own words and list your collaborators. Copying someone else's solution will be considered plagiarism and may result in failing the whole course.

Please answer clearly and concisely. Explain your answers. Unexplained answers will get lower scores or even no credits.

(1) (30 points) Prove that the following languages are decidable.

(a) $L_1 = \{\langle R\rangle \mid R \text{ generates at least one string of even length}\}$
Here R is a regular expression over alphabet $\{\texttt{a}, \texttt{b}\}$

(b) $L_2 = \{\langle D\rangle \mid D \text{ is a DFA that accepts only a finite number of strings}\}$

(c) $L_3 = \{\langle G\rangle \mid \text{Context-free grammar } G \text{ generates at least one string of even length}\}$

(2) (40 points) For each of the following languages, say whether it is decidable. Justify your answer in about 5–10 sentences.

(a) $L_1 = \{\langle M\rangle \mid \text{Turing machine } M \text{ accepts at least one string of even length}\}$

(b) $L_2 = \{\langle M\rangle \mid \text{Turing machine } M \text{ recognizes a regular language}\}$

(c) $L_3 = \{\langle G, k\rangle \mid \text{Every string of length at least } k \text{ can be generated by } G\}$
Here G is a context-free grammar over alphabet $\{\texttt{a}, \texttt{b}\}$

(d) $L_4 = \{\langle M, t\rangle \mid \text{Turing machine } M \text{ accepts some input in at most } t \text{ steps}\}$

(3) (20 points) The following special case of Post Correspondence Problem (PCP) is shown to be undecidable during lecture:

PCP$'$: Let's call a tile nonempty if both its top and bottom strings are not the empty string. (The encoding of) a collection C of nonempty tiles is in PCP$'$ if C contains a top-bottom match.

Show that the following variants of PCP are undecidable, by reducing from PCP$'$.

(a) PCP$_1$: flippable PCP. We say a tile is flipped if top and bottom strings are interchanged. (The encoding of) a collection C of tiles is in PCP$_1$ if C contains a top-bottom match, where each tile in the matching sequence may be optionally flipped.

(b) PCP$_2$: PCP with alphabet permutation. Given finite alphabets Σ_T and Σ_B of the same size, a (Σ_T, Σ_B)-tile has a top string over Σ_T and a bottom string over Σ_B. (The encoding of) a collection C of (Σ_T, Σ_B)-tiles is in PCP$_2$ if there is a bijection $p : \Sigma_T \to \Sigma_B$, such that after mapping all the top symbols according to p, the new collection of tiles contain a top-bottom match.

(4) (10 points) You just got hired by **Toktik**, the hottest software company of the 2010s. Your new boss has several project proposals for you. However, you suspect that some of her proposals may be a bit unrealistic. But you won't turn down a proposal just because

you don't like it, or else you might get fired pretty soon. You have to give a reason why you think it is not going to work.

For each of these software projects, say if you think the project is feasible or not. If you think it is feasible, say how you would approach it. If you think it is infeasible, explain why. Your explanation may be about 2-3 sentences. **You only need to answer two out of the following three parts below**; you may choose any two parts to answer.

(a) Toktik extensively uses a fairly new programming language called **Rust**. However most of their old programs, and there are many of them, are written in Java. Write an application which converts all their Java programs into Rust.

(b) Toktik solicits applications from developers that it then sells in its Toktik Store. But some of these developers are very clumsy and their applications tend to crash. It would be nice to have a tool that detects these crashing applications so Toktik can take them out of their store. Write a program called **crash detector**, which looks at the code of an application (written in Rust) and figures out if the application will ever crash.

(c) Some programs take a very long time to run. It would be nice to know roughly how long a program is going to run for ahead of time, so if it takes a long time you can go out and get lunch. Write an application called **timer**, that looks at a computer program and gives an estimate of its running time (say within a factor of two, so that if the actual running time is 10 seconds, your program should output 5-20 seconds).