CSCI 3130: Formal Languages and Automata Theory — Assignment 4
The Chinese University of Hong Kong, Fall 2021 — due 11:59 p.m. Thursday November 11

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) (35 points) Consider the following context-free grammar G:

$$S \to () \mid (S) \mid (SS)$$

It generates expressions like `()`, `(())`, `(()())` and so on.

(a) Every partially completed rule of the form $A \to \alpha \bullet \beta$ is known as an *item*. Construct an NFA for valid item updates of G.

(b) Convert the NFA to a DFA. Which of the states are shift states and which are reduce states? Are there any conflicts?

(c) Using the DFA, show an execution of the LR(0) parsing algorithm on the input

`(((())())())`

Show the stack of states, stack of processed input, and remaining input throughout the execution.

(d) Now consider the following context-free grammar G':

$$S \to \varepsilon \mid \texttt{(}S\texttt{)}$$

Show that G' is not an LR(0) grammar by giving the DFA of valid item updates.

(2) (25 points) This is the only question in this course concerning the state diagram of a Turing machine.

In this problem, you will design a Turing machine for the following language. Briefly explain how your Turing machine works (insufficient explanation may get zero points).

$$L = \{\texttt{a}^i\texttt{b}^j\texttt{c}^k \mid i \leqslant j \leqslant k\} .$$

Give both a high-level description and a state diagram of your Turing machine.

(3) (30 points) A *queue automaton* is like a push-down automaton without the input tape and the stack is replaced by a queue. A queue is a tape allowing symbols to be read only at the left end and written only at the right end. At each time step, the queue automaton may perform either a read or write operation. Each read operation (called a *pop*) reads and removes a symbol from the left and of the tape and each write operation (called a *push*) writes a symbol at the right end. For example, if the state of the tape is `abcaaab`, the operation **pop** `a` yields `bcaaab`. Now **push** `c` yields `bcaaabc`. The internal state of the queue automaton may change after each *pop* or *push* operation (and this transition may depend on the symbol pushed or popped). Initially, the queue contains the input followed by the special end-of-input symbol `$`. The automaton accepts (resp. rejects) by going into a special state q_{accept} (resp. q_{reject}). The transitions in a queue automaton are deterministic. You will argue that a queue automaton is equivalent to a Turing machine: Every queue automaton can be simulated on a Turing machine, and vice versa.

(a) Write a formal definition of a queue automaton. A formal definition of an automaton will look like page 17/slide 16 of Lecture 14 or page 9/slide 7 of Lecture 10.

(b) Show how to simulate a queue automaton on a Turing machine. For this, you need to specify

- how the tape of the Turing machine will be used to represent the queue automaton;
- how the Turing machine tape should be set up initially;
- what the Turing machine should do when the automaton performs a **push** or a **pop** (you may specify in 1-2 sentences the general idea, omitting the tedious details);
- what the Turing machine should do when the queue automaton accepts/rejects.

(c) Show how to simulate a Turing machine on a queue automaton. For concreteness, you may assume the Turing machine has tape alphabet $\Gamma = \{\mathtt{a}, \mathtt{b}, \square\}$. Again you should specify simulation details similar to those in part (b).

(4) (10 points) Argue that the collection of decidable languages is closed under concatenation.

In other words, given a Turing machine/algorithm M_1 that decides a language L_1, and a Turing machine/algorithm M_2 that decides L_2, how do you construct a Turing machine/algorithm M' that decides L_1L_2?