Introduction to Theoretical Computer Science

Exercise Sheet: Week 3

Here are a few routine exercises.

1. Consider the language $L = \{ww \mid w \in \{\mathtt{a}, \mathtt{b}\}^*\}$. Show that it is not regular.

   Consider the sequence $u_i = \mathtt{a}^i\mathtt{b}$ for $i \in \mathbb{N}$. Then for each i and j (where $i \neq j$), we have the word $w_{ij} = u_i$ such that $u_i w_{ij} \in L$ and $u_j w_{ij} \notin L$. Thus as the sequence u_i is infinite, there are infinite $\equiv_L$ equivalence classes, and therefore L is not regular by the Myhill-Nerode theorem.

   This question could also be done via pumping using $\mathtt{a}^p\mathtt{b}\mathtt{a}^p\mathtt{b}$ as your choice of string where p is the pumping length.

2. Is the language $L' = \{ww \mid w \in \{\mathtt{a}, \mathtt{b}\}^* \wedge |w| < 4\}$ regular? Why or why not?

   Yes. It is a finite language (there are finitely many strings of length < 4), and thus is regular, as all finite languages are regular.

3. We saw in lectures that the language L is not context-free. Show that, by contrast, its complement *is* context-free.

   It can be described by the following CFG:

   $$\begin{array}{lcl} S & \to & A \mid B \mid AB \mid BA \\ A & \to & \mathtt{a} \mid \mathtt{a}A\mathtt{a} \mid \mathtt{b}A\mathtt{a} \mid \mathtt{a}A\mathtt{b} \mid \mathtt{b}A\mathtt{b} \\ B & \to & \mathtt{b} \mid \mathtt{a}B\mathtt{a} \mid \mathtt{b}B\mathtt{a} \mid \mathtt{a}B\mathtt{b} \mid \mathtt{b}B\mathtt{b} \end{array}$$

   A and B are any odd-length string with $\mathtt{a}$ and $\mathtt{b}$ in the centre, respectively. The only words we must exclude are words that are even length, comprised of two identical odd-length strings. But the only even-length strings accepted here are comprised of two odd-length strings with different centre characters (AB and BA).

   This one may take a while for everyone to understand.

The following questions/comments are intended as prompts for discussion. Of course, you can ask/discuss about anything. Some of these topics we've touched on in discussion in lectures – this is an opportunity to think about them a bit more.

4. Consider https://www.geeksforgeeks.org/theory-of-computation/concatenation-process-in-dfa/.

   It proposes a construction for sequential concatenation of two DFAs A_1, A_2 (recognizing languages L_1 and L_2, resp.) into a new DFA A that recognizes the language L_1L_2. That construction is explained by example and not formally specified.

   a) Specify this construction formally.

b) Explain why this construction is wrong in general, and give a counterexample. (E.g., by minimally modifying their example).

c) Explain why this wrong construction works on their particular example.

Though the description mentions transitioning into the second DFA, it is not described when this is supposed to happen, and how this should be deterministic.

Taking an ε-transition from a state in F_1 to the start state of A_2 is not deterministic.

The example suggests building a new DFA by merging the accepting states of A_1 with the initial state(s) of A_2, taking the union of the transition functions and the accepting states of A_2. This does not work, because A_1 could have more than one accepting state (and generalized automata with multiple initial states are not deterministic). Worse, this would allow going back from A_2 into A_1. (E.g., add a c-loop at the accepting state of A_1 and a d-loop at the accepting state of A_2 in their example. One must not have any letter c after reading a letter d.)

Their example works because of the a,b-loop at the single accepting state of A_1 and no back transitions elsewhere in A_1. (Even so, their new automaton removes the b-loop at that state to be deterministic.)

Intuitively, the fundamental problem is that the combined automaton needs to nondeterministically guess when the first part of the word (accepted by A_1) ends and the second part (accepted by A_2) begins. So one needs to go via NFAs and later determinize by using the subset construction to get a DFA for the composition.

5. Suppose we augmented a finite automaton with an additional feature: a single mutable variable. On each transition, it may read from or write to this variable.

   a) If the variable is of type Σ (that is, it can contain one alphabet symbol), what class of languages can such automata recognize?

   Regular languages. The number of Σ symbols is finite, so we could convert these automata to regular automata by having $|\Sigma|$ copies of every state.

   b) If the variable is of type $\mathbb{N}$, what class of languages can they recognize?

   Assuming we can do basic arithmetic operations, this automaton could recognize any Turing-recognizable (recursively enumerable) language. This is because almost all reasonable domains can be encoded into a single natural number. Discuss how to do this.

   If the operations on this type $\mathbb{N}$ variable are more limited then the class of language would be smaller. E.g., only increment/decrement by 1 and test for zero yields a subclass of context-free languages.

6. A queue automaton is like a pushdown automaton, but with a queue (fifo) instead of a stack (lifo). These can recognize any Turing-recognizable language. Now suppose we instead defined a pushdown automaton with an additional stack. What class of languages can these 2-stack PDAs recognize? *By simulating a queue with*

two stacks, you can emulate a queue automaton. Hence they also recognize the Turing-recognizable languages.