Selected solutions week 4 - IN2080

The following are solutions to some of the weekly tasks which we discuss in the group sessions.

2.2

a.

Example 2.36 tells us that the language $C = \{a^n b^n c^n \mid n \geq 0\}$ is not context-free (as we've not been through the pumping lemma for CFLs yet, don't worry about how this is shown). If we can give two CFLs A and B, such that $C = A \cap B$, we've shown that the class of CFLs is not closed under intersection; if it was, then C would be a CFL as well. We let $A = \{a^m b^n c^n \mid m, n \geq 0\}$ and $B = \{a^n b^n c^m \mid m, n \geq 0\}$. These two languages are clearly CFLs (try making a PDA for either), and the intersection between them is C (as the overlap is only the strings containing the same amount of the symbols). As such, the class of CFLs are not closed under intersection.

b.

As we're going to show in the next exercise, the class of CFLs **are** closed under union, i.e. $A \cup B$ is a CLF (for some CLFs A and B). If the class of CLFs were to be closed under complementation, then it would also be closed under intersection, as $A \cap B = \overline{\overline{A} \cup \overline{B}}$. This contradicts our previous proof however, and thus CLFs cannot be closed under complementation.

2.16

As this week focuses on PDAs, we'll use them for this exercise, even though it can also be done with CFGs. Let $M = (Q_M, \Sigma_M, \Gamma_M, \delta_M, q_{0_M}, F_M)$ and $N = (Q_N, \Sigma_N, \Gamma_N, \delta_N, q_{0_N}, F_N)$ be two PDAs.

Union

For the class of context-free languages to be closed under union, there must be a PDA $K = (Q_K, \Sigma_K, \Gamma_K, \delta_K, q_{0_K}, F_K)$ such that $L(K) = L(M) \cup L(N)$, where

M and N are PDAs as well. In this case, we employ the simple strategy we saw with NFAs; either machine must succeed, so we non-deterministically guess which machine to simulate. Let K be defined, such that:

- $Q_K = Q_M \cup Q_N \cup \{q_0\}$
- $\Sigma_K = \Sigma_M \cup \Sigma_N$
- $\Gamma_K = \Gamma_M \cup \Gamma_N$
- $\delta_K(q,a,s) = \begin{cases} \delta_M(q,a,s) & \text{if } q \in Q_M \\ \delta_N(q,a,s) & \text{if } q \in Q_N \\ \{(q_{0_M},\epsilon),(q_{0_N},\epsilon)\} & \text{if } q = q_{0_K}, a = \epsilon, s = \epsilon \end{cases}$
- $q_{0_K} = q_0$
- $F_K = F_M \cup F_N$

Concatenation

This construction is also similar to the NFA case; we first simulate the one machine, then non-deterministically guess when we're done with the first part of the string, before simulating the second machine on the second part of the string. The difference is that we introduce an intermediate state between the two machines, which is tasked with emptying the stack from the first machine. As such, we also need a new start state, which pushes a special symbol unto the stack before simulating the first machine, so we know when the stack is empty.

Star

Again, we essentially just copy our homework from NFAs; a brand new start- and final state which has an ϵ-transition to the old start state, and ϵ-transitions from the final states to the old start state. The only difference, is that instead of going directly to the old start state, we enter an intermediate state which, you guessed it, empties the stack. In addition, the ϵ-transition from our new start state to the old one also pushes a special "empty-stack" symbol.

2.58

a.

I'll only give an informal description of the PDA here.
The PDA will read any symbol, and push x on the stack for each of them. Then we'll non-deterministically guess when we've reached the halfway point. When we do, we'll start reading any symbol, while popping xs off the stack. To reach the final state, we must read a 1 while popping from the stack. After that is done, we read symbols as long as the stack is not empty, and only reject if we read anything after the stack is empty.
By guessing that we've reached the middle of the string, and only reading that

many symbols afterwards, we guarantee that the 1 is read in the second half of the string.

b.

$$S \to ASA \mid AF1$$
$$F \to AF \mid \epsilon$$
$$A \to 0 \mid 1$$

This grammar works by first generating an even amount of symbols. By placing the S variable in the middle, we guarantee that we'll place our 1 in the second half of the string. Afterwards, we'll generate any symbol, as well as a 1. We must remember to at least generate a string of minimum two symbols. The variable F is placed between these, such that the 1 will be in the second half. Afterwards, any symbol is generated. As these symbols push the 1 further and further away, it will remain in the second half regardless of what we generate.