Computability - Exercise 4 - Solution

1. (a) The equivalence classes of L_1 are as follows.
   - For every $i \geq 0$, the set $\{0^i\}$ is an equivalence class.
   - For every $k > 0$, the set $\{0^i 1^j : j > 0, i - j = k\}$ is an equivalence class.
   - All other words, i.e., all the words not in L_1 except for ϵ, form an equivalence class.

   Since L_1 has infinitely many equivalence classes, it is not regular.

   (b) The language 0^*1^* is regular (in fact, it is given by regular expression). Recall that the class of regular languages is closed under intersection. Observe that the language L_1 is the intersection of 0^*1^* and L_2. Thus, if L_2 is regular then so is L_1, and we have reached a contradiction. Therefore, L_2 is not regular.

2. The language L has 4 equivalence classes:

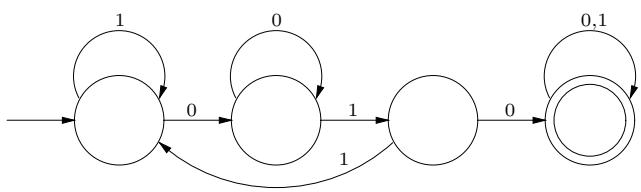

Figure 1: A DFA for L

- Words that contain 010 as a subword.
- Words that do not contain 010 as a subword, and their longest suffix that is also a prefix of 010 is of length 0 (e.g., words that end with 11).
- Words that do not contain 010 as a subword, and their longest suffix that is also a prefix of 010 is of length 1 (e.g., words that end with 00).
- Words that do not contain 010 as a subword, and their longest suffix that is also a prefix of 010 is of length 2 (e.g., words that end with 001).

3. The bound for the number of equivalence classes of L^m is k^n.
   The equivalence classes of L^m are the product of the equivalence classes of the languages in C. Note that the number does not depend on m. The parameter m just determines which classes are accepting.

   To prove the upper bound formally, construct the product automaton from the minimal DFA's for each language. Each state in the product is an n-tuple and the accepting states of the product automaton are the states that have exactly m accepting elements.

   The lower bound is a bit trickier for technical reasons. The problem is in ensuring that the various languages are "independent" in the following sense: we would like to ensure that knowing that a word $s \in \Sigma^*$ is in class c_i^1 of $\sim_{L_1}$ does not give any information on the class of s in $\sim_{L_2}$. To ensure such "independence", we work with an alphabet that is a Cartesian product of n alphabets, and let L_i care only about the i-th coordinate. In addition, to make sure the length of the word does not convey information, we pick L_i's in which for any (large enough) length of word j, there are words of length j both in L_i and outside L_i. For example, let $L'_1, \ldots, L'_n \subseteq \{0,1\}^*$ be languages over $\{0,1\}$ each with k equivalence classes such that for all sufficiently large j we have $L'_i \cap \{0,1\}^k \neq \emptyset$ and $L'_i \cap \{0,1\}^k \neq \{0,1\}^k$. We define L_i to be the language of words over $\Sigma = \{0,1\}^n$ in which for every word $x \in \Sigma^*$ we have $x \in L_i$ iff the projection of x on the i-th coordinate is in L'_i. It is not hard to see that the new languages $L_1, \ldots L_n$ are "independent" even if $L'_1, \ldots, L'_n$ were not. The formal proof is to define the equivalence classes of L^m as the Cartesian product of the equivalence classes of each language, and show separating words between each two of them. (Once the alphabet construction is understood, this is just easy technical writing.)

4. (a) $S \to 0S0|1S1|0|1|\epsilon$
   (b) $S \to 0S0|1S1|0A1|1A0$
   $A \to 0A|1A|\epsilon$
   (c) $S \to 0S1|1S0|SS|\epsilon$

5. Let G be in Chomsky normal form, such that it has k variables. Suppose that G generates the word w using 2^{k+1} derivation steps. The parse tree of w is a binary tree since the derivations are of the form $A \to BC$. As a binary tree with 2^{k+1} nodes, it must have a height of a least $k+1$. On the longest path there must be a variable that appears twice. The rest of the proof continues along the lines of the proof of the pumping lemma for CFLs.