CS 208 Tutorial 4

---

1. Let $\Sigma = \{0, 1\}$. In each of the following problems, you have to prove the regularity of a desired language.

    1. For every word w in Σ^*, we say $w' \in \Sigma^*$ is a subword of w iff w' is obtained by replacing some letters in w with ε. Thus, 011 is a subword of 0010111, but is not a subword of 0001. Furthermore, for $w \in \Sigma^*$, define w^R as the string w read in reverse. For example, 011^R is 110. Now, let L be a regular language over Σ. Define

    $$\text{Lossy}(L) := \{w' : w' \text{ is a subword of some word in } L\}$$

    The need for such languages arise in real life: Imagine a channel on some network, on which we are transmitting bit streams. However, the channel is lossy, and some bits are lost in transmission. Therefore, if we transmit a word w, we may receive a subword of w. In particular, if we want to transmit words from a regular language L [1], we'll actually end up receiving subwords of words in L [2].
    Show that if L is regular, so is $\{w^R : w \in \text{Lossy}(L)\}$. by constructing an **NFA** for this language.

    2. Let $L_1 := L((0+1)^*01(0+1)^*10)$. Define

    $$L_2 := \bigcup_{a_1 a_2 \cdots a_k = w \in L_1} S(a_0) \cdot S(a_1) \cdots S(a_k)$$

    where $a_i \in \{0, 1\}$ are the alphabets of the word $w \in L_1$, and $S(0) := \{0101\}, S(1) := \{1010\}$ are functions mapping each letter of Σ to a singleton language. Construct a **DFA** for L_2.
    Motivation: Such transformations are used in networks, where we encode individual bits into larger chunks for redundancy in our transformation, so that even if a few bits are corrupted, the transmitted word can still be recovered. Also look up string homomorphisms (and inverse homomorphisms) from Hopcroft, Motwani and Ullmann.

    3. Let L_1, L_2 be regular languages over Σ. Define

    $$\text{interleave}(L_1, L_2) := \{\alpha_1\beta_1\alpha_2\beta_2 \ldots \alpha_k\beta_k : \alpha_1, \ldots, \alpha_k, \beta_1, \ldots, \beta_k \in \Sigma^*, \alpha_1\alpha_2 \ldots \alpha_k \in L_1, \beta_1\beta_2 \ldots \beta_k \in L_2\}$$

    In the above definition, $\alpha_1\alpha_2 \ldots \alpha_k$ represents the concatenation of strings $\alpha_1, \alpha_2, \ldots, \alpha_k$, and similarly for $\beta_1\beta_2 \ldots \beta_k$.
    Prove that if L_1, L_2 are regular, then $\text{interleave}(L_1, L_2)$ is regular.
    Motivation: Imagine a network where packets are arriving from two different channels (look up TDM channels). Then what is received on the other side is an interleaving of the packets received from both the channels.

---

[1] Why might we want to do so? Recall that regular expressions and hence regular languages can be used to describe sets of words containing specific patterns.

[2] If you are more interested about such things, you can look up *erasure codes* for further reading.

2. Recall the Pumping Lemma described in class (without mentioning the name as such). To recall: Let L be a regular language. Then there is an integer $p \geq 1$ (depending only on L) such that for any $w \in L$ such that $|w| \geq p$, we can write $w = xyz$, such that:

   1. $|y| \geq 1$.
   2. $|xy| \leq p$. Note that x *may* be ε.
   3. $xy^n z \in L$ for all $n \geq 0$. In particular, $xz \in L$.

   More informally, for any regular language, and for any long enough word in it, after removing some small enough prefix and some suffix, the rest of the word is just some small enough word repeated over and over again, i.e. 'pumped' (that's where the lemma gets its name from).

   The Pumping Lemma is extremely useful to show that a given language is **not regular**. Basically, if we show that a given language L doesn't satisfy the Pumping Lemma, then it can't be regular (since every regular language satisfies the Pumping Lemma).

   The strategy to show a language L non-regular using Pumping Lemma is best viewed as a turn-based game between an adversary (who wants to show that the language is not regular) and a believer (who believes that the language is regular). The game goes as follows:

   - The believer chooses an integer $p > 0$. She claims this is the count of states in the DFA that she believes recognizes the language.
   - The adversary chooses a (often carefully constructed) string $w \in L$ such that $|w| > p$.
   - The believer then splits w into three parts $w = x \cdot y \cdot z$, where $|xy| \leq p$. The believer thinks this is the shortest prefix of w that ends up looping back to a state of the p-state DFA.
   - The adversary now chooses an (often carefully constructed) integer $n \geq 0$ such that $xy^n z \notin L$, thereby winning the game

   If the **adversary can win** the above game (i.e. can choose w in step 2 and n in step 4) *for every choice of p and for every decomposition of w and $x \cdot y \cdot z$ chosen by the believer*, then the language L must be non-regular. Why so? Because if L was indeed regular, there must exist a DFA with a certain number of states accepting L. If the believer chooses p to be this count of states in step 1, then we know from the Pumping Lemma that for every $w \in L$ such that $|w| > p$ which the adversary can choose in step 2, there must exist a decomposition $x \cdot y \cdot z$ with $|xy| \leq p$ such that $xy^n z \in L$ for all $n \geq 0$.

   Take a few moments to understand the above sequence of steps in the game.

   Using the above idea, show that the following languages are **not** regular:

   1. $\{0^n 1^m : n \geq m \geq 0\}$.
   2. $\{0^{n^2} : n \geq 0\}$.

   It is important to note that the pumping lemma is not and "if and only if" statement: If a language is regular, it satisfies the lemma, but not necessarily the other way arond. Indeed, there are languages that are not regular, yet satisfy the conditions of the pumping lemma.

   Try to prove that the following language is not regular (with knowledge of the fact that $\{a^n b^n \mid n \geq 0\}$ is not regular – a fact that can be proved again using the Pumpling Lemma), yet can be pumped.

$$L := \{c^m a^n b^n : m \geq 1, n \geq 0\} \cup a^* b^*$$

3. **Takeaway**: In this question, we'll see an almost templatized way of proving that certain transformations of regular languages are regular.
   Let $\mathcal{L}$ be a regular language, and let $\mathcal{A} := (Q, \Sigma, \delta, q_0, \mathcal{F})$ be a DFA recognizing $\mathcal{L}$. Note that Q is the set of states of $\mathcal{A}$, $q_0 \in Q$ is the start state, $\mathcal{F} \subseteq Q$ is the set of final states, Σ is our alphabet, and $\delta : Q \times \Sigma \mapsto Q$ is the transition function.
   For any $\alpha \in Q, B \subseteq Q$, define $\mathcal{L}(\alpha, B)$ to be the regular language recognized by the DFA $(Q, \Sigma, \delta, \alpha, B)$, i.e. we take $\mathcal{A}$ and change the starting state to α and the end state(s) to B.
   Let $\mathcal{L}$ be a regular language. Prove that the following languages are regular:

   1. $\text{init}(\mathcal{L}) := \{w : wx \in \mathcal{L} \text{ for some } x \in \Sigma^*\}$
   2. $\text{CubeRoot}(\mathcal{L}) := \{w : w^3 \in \mathcal{L}\}$

   [Hint: Try to express these languages as union/intersections/concatenations of $L(\alpha, B)$'s for various α, B.]
   See section 4.2 of Hopcroft, Motwani, and Ullmann for more problems of this type.

   These questions can also be solved by converting the DFA of $\mathcal{L}$ to NFAs accepting the given languages. However, many of those conversions are clumsy, and it requires some care to show that the transformed automaton accepts precisely the desired language. The above method, however, is short and much more transparent.

4. **Takeaway**: Consider the 8-letter alphabet

$$\Sigma_3 = \{(0,0,0),(0,0,1),(0,1,0),(0,1,1),(1,0,0),(1,0,1),(1,1,0),(1,1,1)\}$$

Each letter of the alphabet is a 3-tuple of bits. Consider the language formed by words in this language that satisfy the following property: If all the X-coordinates of the letters are considered as a binary number N_X in reverse, and all the Y-coordinates of the letters are considered as a binary number N_Y in reverse, and all the Z-coordinates of the letters are considered as a binary number N_Z in reverse, then $N_X + N_Y = N_Z$.

For example, the word $(0,0,0)(0,1,1)(1,0,1)$ is in the language, since we have $N_X = 100_2 = 4$, $N_Y = 010_2 = 2$ and $N_Z = 110_2 = 6$. However, the word $(0,0,0)(0,1,1)(1,1,0)$ is not in the lanuage, since $N_X = 100_2 = 4$, $N_Y = 110_2 = 6$ but $N_Z = 010_2 = 2$.

1. Prove that this language is regular, and draw a DFA for it. You can think of words accepted by this DFA as denoting the "solutions" to the equation $N_X + N_Y = N_Z$.

Consider any "formula" which is written exclusively using the symbols $+, =, \vee, \neg$, variable symbols, and constant symbols 0 and 1 along with appropriate parenthesis. One such expression is $((x + y = z) \vee (x - y = 1 + 1 + 1)) \wedge (z = 1 + 1 + x + x)$. What are the set of solutions of the expression? For the above expression, the possible satisfying solutions are all tuples $\{(a, a-3, 2+2a) : a \geq 3\}$, and all tuples $\{(a, a+2, 2+2a) : a \geq 0\}$ Here, we investigate whether we can come up with a simple way to check for solutions to such an equation.

1. Prove that the set of satisfying solutions of such a formula in k variables $F(x_1, x_2 \cdots x_k)$ can be written as a regular language over Σ_k where each coordinate is interpreted in reverse binary. Does this give you an algorithm to decide if any such formula has a solution? Could you do such a thing if multiplication was also allowed?

5. **Takeaway**: A student claims that every regular language over a unary alphabet (say, $\Sigma = \{0\}$) is a finite union of languages of the form $Ł_{c,d} = \{0^{c.n+d} \mid n \geq 0\}$ for constant integers $c, d \geq 0$. Either prove the student wrong by providing a regular language over $\Sigma = \{0\}$ that can't be expressed in the above form (you must give a proof of this), or prove that the student's claim is correct.