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.

    > **Solution:** We can make the epsilon-NFA for the language $\{w | w \in Lossy(L)\}$. This is done by adding an epsilon transition alongside every transition in the original DFA. Then, we can construct a DFA for the same. Since we have a DFA accepting the above language, we construct a DFA that accepts the reverse of the language it accepts. We can simply reverse the transitions in the original DFA to obtain the required NFA.

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

    $$L_2 := \bigcup_{a_1a_2\cdots a_k = w \in L_1} S(a_1) \cdot S(a_2) \cdots S(a_k)$$

    where $a_i \in \{0,1\}$ are the letters 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.

    > **Solution:** Construct the original DFA for L_1. The example given can be done easily by constructing the DFA for $\{L = (0101+1010)^*01011010(0101+1010)^*10100101\}$. However, more generally, for any given homomorphism, refer to the proof here

    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 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.

Solution: We construct a new NFA where every state is represented by a tuple (s_1, s_2, b) where b is bit taking values over $\{0, 1\}$ and s_1 is a state in the state space of the DFA representing L_1 (call this DFA D_1) and s_2 is a state in the state space of the DFA representing L_2 (call this DFA D_2). The starting state is the set of starting states of D_1 and D_2. The transition function is as follows

- $\delta((s_1, s_2, 0), a) = \{(\delta_1(s_1, a), s_2, 1), (\delta_1(s_1, a), s_2, 0)\}$
- $\delta((s_1, s_2, 1), a) = \{(s_1, \delta_2(s_2, a), 0), (s_1, \delta_2(s_2, a), 1)\}$

Intuitively, the NFA has the ability to jump from D_1 to D_2 and vice versa at any time, while remembering the last state it was in D_1 (resp. D_2), when it jumps to D_2 (resp. D_1). This remembered state is where the NFA resumes when it jumps back to D_1 (resp. D_2).

The accepting states of the NFA are all states of the form $(f_1, f_2, 0)$ and $(f_1, f_2, 1)$ where f_1 is an accepting state of D_1, and f_2 is an accepting state of D_2.

Any word in interleave(L_1, L_2) induces an interleaving of paths in D_1 and D_2. We use this interleaving to jump from D_1 to D_2 in the NFA we have constructed above. This traces a path from the start state of the NFA to one of its accepting states. Hence this word is accepted by the NFA.

Similarly, any word accepted by the NFA traces out an interleaving of accepting paths in D_1 and D_2 and hence is in interleave(L_1, L_2)

The NFA constructed above therefore accepts interleave(L_1, L_2). Hence, interleave(L_1, L_2) is regular.

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 Pumping Lemma), yet can be pumped.

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

Solution: 1. Language $\{0^n1^m : n \geq m \geq 0\}$

The believer believes this language is regular and gives some value of $p > 0$ to the adversary.

The adversary considers the string $w = 0^p1^p$. This string is in the language since $p \geq p$ and $p \geq 0$, satisfying the condition of the language.

According to the pumping lemma, w can be divided, by the believer into three parts: $w = xyz$, such that:

1. $|xy| \leq p$
2. $|y| > 0$
3. For all $i \geq 0$, $xy^iz \in L$

Now, let's consider all possible partitions (that the believer can make:

The believer can partition the string as $x = 0^{k_1}, y = 0^{k_2}, z = 0^{p-k_1-k_2}1^p, k_1 + k_2 \leq p, k_2 \geq 1$. Observe that no other partitions are possible (x and y must contain only 0s as the length of xy is at most p)

In all cases, the adversary can pump the string y 0 times. Thus xy^nz becomes $xz = 0^{p-k_2}1^p$ which does not belong to the language (as $p - k_2 < p$. Thus, the language $\{0^n1^m : n \geq m \geq 0\}$ is not regular.

2. Language $\{0^{n^2} : n \geq 0\}$

The believer believes this language is regular and gives some value of $p > 0$ to the adversary.

Consider the string $w = 0^{p^2}$. This string is in the language since $p \geq 0$, satisfying the condition of the language.

According to the pumping lemma, w can be divided, by the believer into three parts: $w = xyz$, such that:

1. $|xy| \leq p$
2. $|y| > 0$
3. For all $i \geq 0$, $xy^iz \in L$

Now, let's consider an arbitrary partition that the believer can make. Any partition looks like: $x = 0^{k_1}, y = 0^{k_2}, z = 0^{p^2-k_1-k_2}, k_1 + k_2 \leq p, k_2 \geq 1$.

In all cases, the adversary can pump the string y 2 times. Thus xy^nz becomes $xy^2z = 0^{p^2+k_2}$ which does not belong to the language (as $p^2 + k_2 < (p+1)^2$). Thus, the language $\{0^{n^2} : n \geq 0\}$ is not regular.

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

Suppose L is regular. We know the Language $L_2 = \{ca^m b^n : m \geq 0, n \geq 0\}$ is regular. Hence, $L \cap L_2$ will also be regular. However, $L \cap L_2$ is $\{ca^n b^n, n \geq 0\}$, which is not regular. Hence, by contradiction, L is not regular

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.

Solution: Motivation: Consider the state $q \in Q$. Note that if some final state is reachable from q, then all strings whose traversal (beginning from q_0) ends at q will belong to $\text{init}(\mathcal{L})$. Now, note that the set of all strings whose traversal ends at q, is a regular language: Indeed, it is recognized by the automata $\mathcal{A} := (Q, \Sigma, \delta, q_0, \{q\})$! Now, it is also easy to see that for any string $w \in \text{init}(\mathcal{L})$, the traversal of w ends in a state q, from which some final state is reachable. Consequently, we have

$$\text{init}(\mathcal{L}) = \bigcup_{\mathcal{L}(q,\mathcal{F}) \neq \emptyset} \mathcal{L}(q_0, q)$$

Since regular languages are closed under union we're done. Similarly, for the cube-root language, we have:

$$\text{CubeRoot}(\mathcal{L}) = \bigcup_{q_1, q_2 \in Q} \mathcal{L}(q_0, q_1) \cap \mathcal{L}(q_1, q_2) \cap \mathcal{L}(q_2, \mathcal{F})$$

where we express $\mathcal{L}(\alpha, \{\beta\})$ simply as $\mathcal{L}(\alpha, \beta)$.
Since regular languages are closed under union and intersection, we're done.

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 $N_Z = 010_2 = 2$.

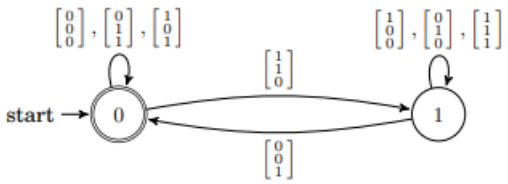
1. Prove that this language is regular, and draw a DFA for it. You can think of words accepted by this DFA as denoting the "solutions" to the equation $N_X + N_Y = N_Z$.

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?

Solution:

1. The idea is simple, just add numbers like we usually do. The input is given in reverse binary for intuition, since we usually add from the units digit, although the reversed language is also going to be regular (since reverse of regular languages are still regular). Suppose we do not have an outstanding carry. Then, as long as the X bit, Y bit and Z bit are all 0, or if exactly one of X and Y bits are 1 and Z is 1, we can simply continue to remain at the same state. If we get an invalid combination of X, Y, Z like $(0,0,1)$ or $(0,0,1)$ or $(1,1,1)$ then we can go to a trap non-accepting state. Finally, if we get a $(1,1,0)$, we must keep track of this carry by going to a new state, which is the carry state, and from which the acceptable transitions satisfy the carry bit added. Finally, the addition must end on a non-carry state, thus only the start state is accepting. The DFA (trap state not indicated) looks like:



2. We claim the satisfying solutions of any such formula form a regular language, and give a construction of the corresponding DFA. First, observe that for constants, like the formula $z = x + 1$ we can introduce a placeholder variable y and then take an intersection of the languages $z = x + y$ and $y = 1$, after which we can simply **drop**

(project out) the y coordinate from each letter to get an automata over Σ_2, whose accepted words satisfy $z = x + 1$. Similarly for multiple additions, we can evaluate them pair by pair, by first introducing a new variable for the first addition, and then taking an intersection. For example, $z = x + y + w$ can be written as an intersection $z = t + w$, $t = x + y$, and then we can project out the t-coordinate. Therefore, with standard methods for intersection, union, complement and projection, we can construct a DFA over Σ_k for the satisfying solutions of any such formula. Finally, checking for existence of a solution is simply checking if the DFA accepts any word.

In fact, some solvers which work for integer variables (like z3) use similar algorithms involving internal DFA constructions to find solutions to such formulas. It is crucial for these tool developers to have these constructions as optimized as possible. Such tools are called SMT-solvers and are an extension of SAT solvers. The study of which kinds of equations admit simple algorithms is thus an important area of research.

Note that our current setup allows us to multiply constants to variables, since it just corresponds to adding the variable that many times. However, we cannot express the multiplication of two variables in our setup. If we were to allow this, the language ceases to be regular, the set of words over Σ_3 satisfying $z = x \cdot y$ is not regular (try to prove this!). Therefore it is not possible to use a DFA to decide it. Is it possible with a more powerful algorithm maybe? The answer, in general, is no. The proof is somewhat advanced (it is due to Gödel, and follows from his incompleteness theorems), but the short answer is that if we could solve equations involving multiplication, then we would be able to solve a lot of things that are 'unsolvable'. We will see such proofs of 'unsolvability' when we talk about Turing Machines. However, by restricting the allowed multiplications, it is possible to get a subset of formulas with an algorithm to solve them.

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.

Solution: The claim is indeed true. For any regular language over a unary alphabet, we will construct the required finite union of sets. Suppose L is a regular language over a unary alphabet. Hence, we must have a DFA F If the language is empty, then the result is trivially true. If not, there exists exactly one path from the start state to an accepting state that contains that accepting state exactly once. For each such accepting state, we define c as the length of the loop from that accepting state to itself and d as the length of the path from the start state to that accepting state. Hence, we define the set $L_{c,d}$ as given for each accepting state, We take union over all such accepting states. The union of these sets is exactly the language accepted by the DFA. Since the accepting states are finite, this is a finite union and the result is proved