Lecture 5

Showing a language is not regular

In this lecture, we will develop tools for showing that a language is not regular, and hopefully gain a better understanding of what the regular languages are. We will start with some closure properties of regular languages.

5.1 Closure properties

We say that a class of languages is *closed under* an operation on languages if applying this operation to any language(s) in the class gives another language in the class.

This definition is best understood by examples. Consider the class Regular consisting of all regular languages over some alphabet Σ. This is a *class* of languages, rather than a language; that is to say, Regular is a set of sets of strings over Σ. We say Regular is closed under union if $A \cup B \in \mathsf{Regular}$ for all $A, B \in \mathsf{Regular}$. We know the regular languages are closed under union because they are characterized by regular expressions (and the union of two regular expressions defines a new regular expression).

Similarly, we also know that Regular is closed under concatenation and star (again due to the regular expression characterization of regular languages). This means that $AB \in \mathsf{Regular}$ for all $A, B \in \mathsf{Regular}$, and $A^* \in \mathsf{Regular}$ for all $A \in \mathsf{Regular}$.

Are the regular languages closed under complement? Complement is not one of the regular operations, so it is not clear how to take a regular expression for a language A and construct a regular expression for $\overline{A}$. Is this possible to do?

While it is hard to see with the regular expression characterization, it is actually easy to show that Regular is closed under complement if we use the DFA characterization. To see this, let $A \in \mathsf{Regular}$, and let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA with $L(M) = A$. Now construct a new DFA $M' = (Q, \Sigma, \delta, q_0, Q \setminus F)$ which is the same as M except with the accept states flipped: every state that is accepting in M is not accepting in M' and vice versa. Then for every string $x \in \Sigma^*$, M accepts x if and only if $\delta^*(q_0, x) \in F$, which is if and only if M' rejects x. Hence $L(M') = \overline{L(M)} = \overline{A}$, which means that $\overline{A}$ is recognized by a DFA and is therefore regular.

This shows that Regular is closed under complement. This also means that for every regular expression R, there exists a regular expression R' which generates exactly the strings not generated by R. However, even though we know R' must exist, it is often tricky to construct it from R.

What about intersection? Is Regular closed under intersection? It turns out that there is a simple trick to show that it is. If $A, B \in \mathsf{Regular}$, then $A \cap B = \overline{\overline{A} \cup \overline{B}}$, and since Regular is closed under complement and union, we conclude that $\overline{\overline{A} \cup \overline{B}} \in \mathsf{Regular}$ and therefore $A \cap B \in \mathsf{Regular}$.

How about set difference: if $A, B \in \mathsf{Regular}$, does it hold that $A \setminus B \in \mathsf{Regular}$? The set $A \setminus B$ is the set of all strings in A but not in B; this can be written $A \cap \overline{B}$. Since Regular is closed under complement and intersection, it is also closed under set difference.

5.2 The pumping lemma

The pumping lemma characterizes a special property that is satisfied by all regular languages. It is often the case that this property is *not* satisfied by irregular languages, which means that the pumping lemma can be used to show a language is irregular (just show it doesn't satisfy this special property).

Theorem 5.1 (Pumping lemma)**.** *Let A be a regular language over an alphabet Σ. Then there exists a positive integer $n \in \mathbb{N}$ such that for every string w in A of length at least n, there exist strings $x, y, z \in \Sigma^*$ such that:*

1. $w = xyz$,
2. $y \neq \epsilon$,
3. $|xy| \leq n$,
4. $xy^i z \in A$ *for all* $i \in \mathbb{N}$.

(The number n above is called the *pumping length* of the language A.)

The statement of the pumping lemma can be confusing at first. What it is saying is that if A is regular, then all sufficiently long strings w in A have some substring y that can be *duplicated* any number of times, and the resulting string will still be in A. That is to say, y can be found somewhere in the middle of w, so w can be written $w = xyz$ for some strings x and z, and the special property is that xy^iz (the string with y duplicated i times) must be in the language A for all values of i. This property of w must hold for *all* sufficiently long strings $w \in A$ (those of length at least n for some fixed n that may depend on A). Additionally, the substring y in the middle of w will not be empty (so that $xy^iz \neq w$ when $i \neq 1$), and can always be found within the first n symbols of w (i.e. $|xy| \leq n$).

Why is the pumping lemma true? It turns out to follow from the DFA characterization of regular languages. Every regular language A over Σ has some DFA M recognizing A. This DFA has some nonzero finite number of states n. When the DFA reads a string w of length at least n, it starts at the start state q_0 and makes at least n transitions, meaning it visits at least $n+1$ total states in its path. Since there are only n unqiue states in the DFA, it must have visited some state more than once. But this means the DFA "forgot" something: the only memory a DFA maintains is which state it is in, and it visited some state q twice. This means when the DFA is at q, it cannot tell whether it's visiting q for the first time or for the second time. We can use this confusion to argue that the DFA must accept additional strings other than w (as it cannot tell them apart from w).

We now formally prove the pumping lemma.

Proof. Let A be a regular language over Σ, and let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA recognizing A. Let $n = |Q|$; then n is a positive integer. Let $w \in A$ be a string with $|w| \geq n$. Let $p_0, p_1, \ldots, p_{|w|}$ be the sequence of states that M visits when run on w; that is, p_0 is the start state q_0, p_1 is the state $\delta(q_0, w_1)$, p_2 is the state $\delta(\delta(q_0, w_1), w_2)$, and so on. The last state in the sequence is an accept state (since M accepts $w \in A$), so $p_{|w|} \in F$.

Consider the first $n+1$ states in the sequence, that is, $p_0, p_1, \ldots, p_n$ (note that $|w| \geq n$ so p_n exists). Since there are $n+1$ states in this sequence but $|Q| = n$, there must be some state q that has been visited twice, say $p_i = q$ and $p_j = q$ with $0 \leq i < j \leq n$. Let x be the string consisting of the first i symbols in w, that is, $x = w_1 w_2 \ldots w_i$ if $i \geq 1$ and $x = \epsilon$ if $i = 0$. Let y be the next $j - i$ symbols in w, that is $y = w_{i+1} w_{i+2} \ldots w_j$. Since $j > i$, we have $|y| = j - i > 0$, so $y \neq \epsilon$. Let z be the string consisting of the rest of the symbols in w, that is $z = w_{j+1} w_{j+2} \ldots w_{|w|}$ if $|w| \geq j+1$, or $z = \epsilon$ if $|w| = j$. It is clear that $w = xyz$ by our choice of strings $x, y, z \in \Sigma^*$. We also have $|xy| = j \leq n$.

Observe that when M runs on x, it reaches state p_i; that is, $\delta^*(q_0, x) = p_i$. Also, when M runs on y starting from p_i, it reaches p_j, meaning $\delta^*(p_i, y) = p_j$. Similarly, $\delta^*(p_j, z) = p_{|w|} \in F$. Recall that $p_i = p_j = q$. Thus $\delta(q_0, x) = q$, $\delta^*(q, y) = q$, and $\delta^*(q, z) \in F$. That is, reading x from the start state makes M reach q, reading y starting from q makes M return to q, and reading z starting from q makes M reach an accept state. Now consider the string $xy^i z$ for any $i \in \mathbb{N}$. When M is run on $xy^i z$, it reads x to reach q, then reads y some number of times (i times), returning to q each time, then reads z, reaching an accept state. We conclude that $xy^i z \in L(M) = A$ for all $i \in \mathbb{N}$. Note that this also holds when $i = 0$: in that case, $xy^i z = xz$, the DFA M reaches q when reading x, and then reaches an accept state when reading z. □

5.3 Using the pumping lemma

Let us now show how to use the pumping lemma to show that a language is not regular.

Example 1. Consider the language $A = \{0^n 1^n : n \in \mathbb{N}\}$ over the alphabet $\{0, 1\}$. This is the language of all strings consisting of a sequence of 0s followed by a sequence of 1s of the same length. We will use the pumping lemma to show that A is not regular by showing that it doesn't satisfy the properties of the lemma.

That is, suppose by contradiction that A was regular. Then there would be some positive integer n for which the statement of the pumping lemma holds. Let $w = 0^n 1^n$; then $w \in A$ and $|w| \geq n$. By the pumping lemma, there must be some strings $x, y, z \in \{0, 1\}^*$ such that $xyz = w$, $|xy| \leq n$, $y \neq \epsilon$, and $xy^i z \in A$ for all $i \in \mathbb{N}$. Observe that since $xyz = w = 0^n 1^n$ and $|xy| \leq n$, it must be the case that xy consist of only 0s. Moreover, since $y \neq \epsilon$, we must have $y = 0^k$ for some $k \in \mathbb{N}$ with $k \geq 1$ and $k \leq n$. Now, pick $i = 2$. The string $xy^i z = xy^2 z$ has k extra zeros; that is, it is the string $0^{n+k} 0^n$. Since $k \geq 1$, this string does not have the same number of ones and zeros, so $xy^2 z \notin A$. This contradicts the pumping lemma, so A is not regular.

Note how this argument worked: the pumping lemma gave us n (which we had no control over). Then we picked the string w in a way that depends on n; we picked $w = 0^n 1^n$. The pumping lemma gave us $x, y, z \in \Sigma^*$, which we had no control over except for the guarantees of the lemma (i.e. $xyz = w$, $|xy| \leq n$, and $y \neq \epsilon$); and finally, we picked $i \in \mathbb{N}$, and showed that $xy^i z \notin A$, which contradicted the pumping lemma and completed the proof that A is not regular.

Example 2. Let $B = \{0^n 1^m : n \geq 2m\}$. We will use the pumping lemma to show that B is not regular.

To do so, suppose by contradiction that B is regular, and let n be its pumping length. We pick $w = 0^{2n} 1^n$. Then $w \in A$ and $|w| \geq n$. Let $x, y, z \in \Sigma^*$ be such that $xyz = w$, $|xy| \leq n$, and $y \neq \epsilon$. Then the string xy is all-0, since it is an initial segment of $0^{2n} 1^n$ of length at most n. Hence $y = 0^k$ for some k. Since $y \neq \epsilon$, we have $k \geq 1$.

Now, the string xy^iz is the string $0^{2n-k+ik}1^n$. Note that this string is in B for all $i \geq 1$, since we've only made the number of zeros larger, not smaller. On the other hand, when $i = 0$, the string xy^iz is xz, which is $0^{2n-k}1^n$. The number of zeros in this string is less than twice the number of ones, so it is not in B. This gives us the desired contradiction, showing B is not regular. Note that we had to use the pumping lemma with $i = 0$ to get this to work.

Example 3. Let $C\{w \in \{0,1,2\}^* : |w|_2 \geq |w|_1 + |w|_0\}$. Here we use the notation $|w|_c$ to denote the number of times the symbol c occurs inside the string w; for instance, $|w|_0$ is the number of 0s in w.

To show that C isn't regular, we again start by assuming it is and using the pumping lemma. Let n be the pumping length. The main decision we need to make is how to choose the string $w \in C$ of length least n. It is usually a good idea to pick w to be a string which is "just barely" in C. In this case, since the constraint for being in C is that $|w|_2 \geq |w|_1 + |w|_0$, it will be a good idea to pick a w with $|w|_2 = |w|_1 + |w|_0$. Given this constraint, it will help to pick w to be as simple as possible. We can pick w to not have any 0s, and have all its 1s come before all its 2s, so that it has the form 1^m2^m. Finally, we will pick $m = n$, so that the string w is at least length n and so that its first n symbols are as simple as possible (in this case, all 1s).

Then $w = 1^n2^n \in C$. The pumping lemma gives us x, y, z with $xyz = w$, $|xy| \leq n$, and $y \neq \epsilon$. As before, this will imply that $y = 1^k$ for some $k \geq 1$. The string xz will then be $1^{n-k}2^n$, so the string xy^iz will be $1^{n-k+ik}2^n$. This is not in C when $i = 2$, contradicting the pumping lemma and completing the proof that C is not regular.

5.4 Using closure properties to show irregularity

We seen that regular languages have many nice closure properties, which can be used to show that languages are regular. For example, to show that the language of all strings over $\{0,1\}$ that contain at least one 1 is regular, we could simply note it is equal to $\overline{L(0^*)}$. That is, 0^* is the regular expression generating all strings of the form 0^n, $L(0^*)$ is the set of strings generated by 0^* (the set of all all-zero strings), and $\overline{L(0^*)}$ is the set of strings not in $L(0^*)$, which is the set of all strings with at least one 1. Since the language generated by a regular expression is always regular, and since regular languages are closed under complement, this shows that the language of all strings that contain at least one 1 is regular.

Interestingly, we can also use closure properties to show that languages are *not* regular. As an example, consider the language $D = \{w \in \{0,1\}^* : |w|_0 = |w|_1\}$, the set of all strings w over $\{0,1\}$ which have the same number of 0s and 1s. We could directly apply the pumping lemma to show that D is irregular, but there is actually a simpler way. Consider the language $D \cap L(0^*1^*)$. The language of 0^*1^* is the language of all strings consisting of some number of 0s follows by some number of 1s. Since D is the set of strings with the same number of 0s and 1s, we have $D \cap L(0^*1^*) = A$, where $A = \{0^n1^n : n \in \mathbb{N}\}$.

Now, suppose by contradiction that D was regular. Then since regular languages are closed under intersection, the language $A = D \cap L(0^*1^*)$ would be regular. But we already showed that A is not regular, giving a contradiction.

What happened here is that the closure properties allowed us to reduce the question of whether D is regular to the simpler question of whether A is regular. In principle, any closure property of a regular language can be used in such reductions. In practice, however, intersection is one of the most useful closure properties for showing that languages are not regular.

As a side note, there is sometimes confusion about the difference between the languages $L(0^*1^*)$ and $A = \{0^n1^n : n \in \mathbb{N}\}$. These look similar, but they are not the same language. The former contains all strings that have some number of 0s followed by some number of 1s; this includes 00111, for example. The latter contains only the strings with the *same* number of 0s and 1s. This means that $A \subseteq L(0^*1^*)$, but the two languages are not equal. Remember that a subset of a regular language can be irregular, as is happening here (A is irregular while $L(0^*1^*)$ is regular).

5.5 Conclusion

We have now seen two methods of showing that languages are not regular: the pumping lemma and closure properties. Intuitively, the regular languages are those that can be recognized by a machine with a constant amount of memory—that is, a DFA with a finite number of states. Note that this number of states is fixed and cannot depend on the input size. The reason languages like $\{0^n1^n : n \in \mathbb{N}\}$ are not regular is that recognizing them requires counting the number of 0s the machine sees as it reads the string, and remembering this count n. Since DFAs only have a constant amount of memory (which can be much less than the input size), they cannot do this for sufficiently large n.

What can DFAs do? We have already seen some examples. The main two common tasks that DFAs succeed at is either (a) finding some pattern in a string (such as deciding whether a string contains the substring 010), or (b) counting things, but only modulo some number (e.g. count the number of 1s in the string modulo 2, to decide whether the string has even or odd parity). If a language can be defined in terms of such tasks (searching and counting modulo a constant), chances are it is regular – though proving this requires constructing a regular expression, NFA, DFA, or using closure properties. If the language appears to require comparing two arbitrarily-long numbers (e.g. is the number of 0s in the string greater than the number of 1s), chances are it is not regular – though to prove this you'll likely need to use closure properties or the pumping lemma. Of course, there are many other types of languages that may be regular or irregular, but these types of tasks are important starting points.