Lecture 2

Regular Expressions

2.1 Regular Languages

In the last lecture, we defined various operations on languages. Three of those operations are called the *regular operations*. These are union, concatenation, and star. That is, if A and B are languages, then we've seen that $A \cup B$, AB, and A^* are also languages. These three operations are the regular operations on languages.

A language is called *regular* if it can be defined using only the regular operations, starting from only the individual characters in the alphabet Σ. More specifically, the regular languages over an alphabet Σ are defined recursively:

1. $\varnothing$ and $\{\epsilon\}$ are regular.
2. $\{c\}$ is regular for each character $c \in \Sigma$.
3. If A and B are regular, then $A \cup B$ is regular.
4. If A and B are regular, then AB is regular.
5. If A is regular, then A^* is regular.

In the above definition, we consider only languages defined over a fixed alphabet Σ. The first two items function as base cases: they say that the empty language $\varnothing$ is regular, and the languages of size 1 consisting of a single string of length at most 1 are also regular. Taken together with the last three items, this definition is saying that regular languages are the ones we can form by starting with these basic size-1 languages, and using any combination of unions, concatenations, and stars in any order. If a language A can be constructed in this way, it is called regular; if it cannot, A is not regular.

Which languages are regular? We can start with a few observations. First, for every string $x \in \Sigma^*$, the language $\{x\}$ is regular. This holds even if the string x is not a single character but a longer string. To see why, recall that a string is a finite sequence of characters, and can therefore be written as a finite concatenation of length-1 strings: $x = x_1x_2x_3 \ldots x_n$, where $n = |x|$. It follows that the set $\{x\}$ is equal to the concatenation $\{x_1\}\{x_2\}\{x_3\} \ldots \{x_n\}$. Each of these sets $\{x_i\}$ contains just a single character in Σ, and is therefore regular. Moreover, concatenation of two regular languages gives a regular language. It therefore follows that the concatenation of all these strings, which equals $\{x\}$, is still regular (formally, we would have to prove this by induction).

We have concluded that $\{x\}$ is regular for every string x. This means every language of size 1 is regular. Now, recall that the union of two regular languages is regular. From this we can conclude that each language of size 2 is also regular: after all, such a language must have the form $\{x, y\}$ for some strings x and y, and $\{x, y\} = \{x\} \cup \{y\}$. Since $\{x\}$ and $\{y\}$ are regular, it must be the case that $\{x, y\}$ is regular as well.

Next, we can use the same argument to show that every language of size 3 is regular, and that every language of size 4 is regular, and so on. In fact, using induction, we can generalize this argument prove the following theorem.

Theorem 2.1. *Every finite language is regular.*

Of course, not only finite languages are regular. Another example of a regular language is the language of all strings Σ^*. Why is this regular? Well, the alphabet Σ is finite, and therefore regular, and the star operation preserves regularity (by the definition of regular languages). Another example of a regular language is the language A of all strings that have the form $00\ldots011\ldots1$, consisting of some number of zeros followed by some number of ones. This language is regular because it can be constructed as follows: $\{0\}^*\{1\}^*$.

Question 2.2. *Let A be the language of all strings in Σ^* of even length. Is A regular?*

The answer is yes: A can be written as $(\Sigma\Sigma)^*$, and is therefore regular. This is because $\Sigma\Sigma$ is the language that contains all strings of length 2, and applying star to it gives all strings that can be constructed out of length-2 pieces—that is, all even-length strings.

Are there languages that are not regular? It turns out that there are. One example is the language over $\{0, 1\}$ consisting of all strings with an equal number of ones and zeros. We will have to wait a few lectures before proving that this language is not regular. Later in the course, we will see a general way of proving irregularity of languages.

2.2 Regular Expressions

As we saw above, to show that a language is regular, all we need to do is to show how to decompose it into the elementary pieces (ϵ and characters in Σ) using the operations union, concatenation, and star. Regular expressions are a notation for such decompositions which is a little bit cleaner, removing clutter from excess brackets.

Formally, a regular expression is a string. If the alphabet is Σ, a regular expression over Σ will actually be a string over an expanded alphabet: in addition to the characters in Σ, a regular expression can also include the characters $\varnothing$, ϵ, $\cup$, *, (, and). Note that we will assume Σ did not already contain any of those characters; this assumption is not a problem, because the symbols used in Σ are arbitrary; we could always replace them by the numbers $0, 1, 2, \ldots, |\Sigma| - 1$ without affecting our analysis of the languages over Σ (all that would change is the notation we use to talk about such languages).

The formal definition of a regular expression over Σ is recursive, just like the definition of a regular language was recursive.

1. The symbol $\varnothing$ is a regular expression.
2. The symbol ϵ is a regular expression.
3. For each $c \in \Sigma$, the symbol c is a regular expression.
4. If R_1 and R_2 are regular expressions, then the string $(R_1 \cup R_2)$ is a regular expression.
5. If R_1 and R_2 are regular expressions, then the string (R_1R_2) is a regular expression.

6. If R is a regular expression, then the string (R^*) is a regular expression.

Note that in the above definition, $\varnothing$ is a *symbol* rather than being the empty set; regular expressions are always just strings, though they are strings that are meant to represent languages. It might sometimes be helpful to place regular expressions in quotes, like "$\varnothing$", to remind ourselves that we are talking about symbols or strings rather than sets (which share some of the notation).

One of the main ways in which regular expressions differ from standard set notation is that the curly brackets $\{\}$ are omitted. For example, if $\Sigma = \{0,1\}$, the set $\{0,1\}$ has regular expression $(0 \cup 1)$, even though in ordinary set notation we would have to write $\{0\} \cup \{1\}$. In effect, regular expressions are a shorthand that allows us to use 0 to denote $\{0\}$, 1 to denote $\{1\}$, ϵ to denote $\{\epsilon\}$, and so on.

Next, we formally define how to convert a regular expression into the set it represents. This conversion is intuitive, but it is nice to have a formal definition. If R is a regular expression over Σ, we denote the language of R by $L(R)$, which we define as follows.

1. If $R = \varnothing$, then $L(R) = \varnothing$.
2. If $R = \epsilon$, then $L(R) = \{\epsilon\}$.
3. If $R = c$ for some $c \in \Sigma$, then $L(R) = \{c\}$.
4. If $R = (R_1 \cup R_2)$ for some regular expressions R_1 and R_2, then $L(R) = L(R_1) \cup L(R_2)$.
5. If $R = (R_1R_2)$ for some regular expressions R_1 and R_2, then $L(R) = L(R_1)L(R_2)$.
6. If $R = (R_1^*)$ for some regular expression R_1, then $L(R) = L(R_1)^*$.

As you may have noticed, one annoying feature of the way we've defined regular expressions is that there are a lot of round brackets around everything. These round brackets are an artifact of the formal definition (in which we want to ensure the order of operations is not ambiguous), but they are often not necessary when writing informally. For example, in this class, we will simply write $0 \cup 1$ as a valid regular expression for $\{0,1\}$, even though formally it would have to be $(0 \cup 1)$.

To be able to discard even more parentheses, we introduce an order of operations for regular oprations:

- Star is applied before concatenation and union.
- Concatenation is applied before union.

This means that the expression $01^* \cup 0^*1$ should be interpreted as equivalent to the regular expression $((0(1^*)) \cup ((0^*)1))$. This convention cleans up a lot of clutter and makes regular expressions much more readable.

2.3 Examples of Regular Expressions

The art of constructing regular expressions takes some getting used to. Let's go through a few exercises to get a feel for how they work.

Question 2.3. *Let A be the language over $\{0,1\}$ consisting of all strings with an even number of ones. What is a regular expression for A?*

We want to use a regular expression to capture only the strings with an even number of ones. One attempt might be $(11)^*$. However, this gives us the set $\{\epsilon, 11, 1111, 111111, \dots\}$, and while this set contains only strings with an even number of ones, it is missing many other strings with an even number of ones, such as 0101.

The right way to approach this problem is to try to decompose the target language A into smaller pieces. How can we construct a string with an even number of ones out of smaller blocks? Well, one way to do so is to notice that a string with an even number of ones is a concatenation of smaller strings, each with exactly two ones. So if we had a regular expression for the strings with exactly two ones, we may be able to use it as a building block. In turn, a string with two ones can be further decomposed: it looks like a block of zeros, followed by a 1, followed by another block of zeros, followed by another 1, finally followed by another block of zeros. This is the regular expression $0^*10^*10^*$. Note that 0^* represents a block of zeros: it gives the language $L(0^*) = \{\epsilon, 0, 00, 000, \dots\}$, which contains any number of zeros in a row (such a block of 0s may even have size zero).

Putting this together, it suggests the regular expression $(0^*10^*10^*)^*$. This regular expression represents taking any number of strings with two 1s, and concatenating those strings together. This almost gives us what we want; however, this language is still missing some strings with an even number of ones. It is missing the strings 0, 00, 000, etc. (although it does contain the string ϵ). That's because there was a subtle error in the way we've decomposed our strings: we said that a string with an even number of ones is always a concatenation of strings with two ones, but this is only true of the original string had at least two ones. One way to fix this is to add back all the strings with no ones in them. Those strings have the simple regular expression 0^*, so we get the final expression $0^* \cup (0^*10^*10^*)^*$, which is correct.

A simpler regular expression for the same language is $0^*(10^*10^*)^*$. This regular expression employs a different way of breaking apart a string with an even number of ones: it is using the fact that a string with an even number of ones always looks like some number of zeros, then any number of blocks, with each of the blocks having exactly two ones and also starting with a one. Both of these regular expressions represent the same language, so both are valid regular expressions for A. Formally, we have $L(0^*(10^*10^*)^*) = L(0^* \cup (0^*10^*10^*)^*)$.

Question 2.4. *Let B be the language over $\{0, 1, 2\}$ consisting of all strings which contain the pattern 012 inside of them. What is a regular expression for B?*

As before, we should try to break apart the strings in B into smaller blocks. Here, the decomposition is clear: any string in B looks like some arbitrary string, followed by 012, followed by another arbitrary string. This has the regular expression $(0 \cup 1 \cup 2)^*012(0 \cup 1 \cup 2)^*$.

Question 2.5. *Let C be the language over $\{0, 1\}$ consisting of all strings x with an even number of 0s and an even number of 1s. What is a regular expression for C?*

As before, we would like to decompose strings in C into simpler blocks. One attempt might be to say that each block should contain exactly two zeros and two ones; this would suggest the regular expression $(0011 \cup 0101 \cup 0110 \cup 1001 \cup 1010 \cup 1100)^*$. However, the language generated by this regular expression is missing a lot of strings: for example, consider the string 00011101. It has an even number of zeros and an even number of ones, but its first four characters form 0001, which does not have an even number of zeros or ones. Therefore, the string 00011101 would not be generated by our candidate regular expression above, even though it is in C.

To find a regular expression for C, we need a different decomposition of a string $x \in C$ into simpler pieces. First, we note that x must have even length, say $2n$. Now, let's look at x as n blocks of size 2, each of which is one of 00, 11, 01, or 10. Observe that 00 and 11 may occur any number of times (since they don't change the parity of the number of zeros or the number of ones), but the blocks 01 and 10 must together occur an even number of times. We've already seen how to construct a regular expression for the language A above, which had an even number of ones and an arbitrary number of zeros in each string. We will now do something similar: we wish to have an even number of 01 or 10 blocks, and an arbitrary number of 00 or 11 blocks in our string. Copying from our regular expression for A, we then get the following regular expression for C: $(00 \cup 11)^*((01 \cup 10)(00 \cup 11)^*(01 \cup 10)(00 \cup 11)^*)^*$.

2.4 Are All Languages Regular?

So far, we have not proven any language to be irregular; we've only used regular expressions to show that a variety of languages *are* regular. One might wonder if all languages are regular, and if not, why not. In a previous offering of the course, some students came up with the following (wrong) proof that all langauges are regular.

Wrong Theorem 2.6. *All languages are regular.*

Wrong proof. Let Σ be an arbitrary alphabet, and let A be an arbitrary language over Σ. For each string $x \in A$, the set $\{x\}$ is finite, and therefore regular. Now, note that $A = \bigcup_{x \in A} \{x\}$ (that is, A is equal to the union, over all $x \in A$, of the sets $\{x\}$). Since a union of regular languages is regular, we conclude that A is regular. □

What is the problem with this proof? It is true that A is the union of all languages of the form $\{x\}$ for $x \in A$, and it is also true that each such language is regular (since it is finite). However, it is not true that a union of regular languages is regular—or at least, this is not true for *infinite* unions. Recall that in the definition of regularity, we said that a union of *two* regular languages is regular. This also implies (by induction) that a union of any finite number of regular languages is regular. But it does not mean that a union of *infinitely* many languages is regular! Be careful on this point: infinite unions often behave differently (and preserve different properties of languages) than finite unions.

Another common point of confusion is this: the language Σ^* is regular (since Σ is finite and star preserves regularity). Since every language A over Σ is a subset of Σ^*, doesn't that mean every language is regular?

No: a subset of a regular language need not be regular. It's true that languages are regular if they are in some sense "simple", but in many cases, subsets of a language are more complicated than the original language.

Finally, although we are not yet ready to prove that any individual language is not regular, here is an argument for why irregular languages must exist. Recall that each regular language over Σ can be represented by a regular expression over Σ. Recall also that a regular expression is just a string whose alphabet is $\Sigma \cup \{\varnothing, \epsilon, (,), \cup, ^*\}$. Let's call this expanded alphabet Σ'. Then Σ' is also finite. Since regular expressions are strings over Σ', the set of all regular expressions is a language over Σ' (and hence a subset of $(\Sigma')^*$). Now, as mentioned in the previous class, the set $(\Sigma')^*$ is infinite but countable, since Σ' is finite. This means there are only countably many regular expressions over Σ, and so only countably many regular languages over Σ.

On the other hand, we argued last class that there are uncountably many languages over Σ. This means that regular expressions cannot be matched up to the languages without there being languages "left over". In particular, there cannot be a regular expression for each language. Indeed, this argument actually implies that there are uncountably many irregular languages: in some sense, almost all the languages out there are actually irregular.

In the next few lectures, we will introduce a computational model which will provide an alternate characterization of regular languages. We will use this to deepen our understanding of which languages are and are not regular, and we'll introduce some tools for showing that languages are not regular.