CSE 105: Introduction to the Theory of Computation Fall 2015

Lecture Notes: A nonregular language

Instructor: *Daniele Micciancio* UCSD CSE

This lecture notes are provided as a supplement to the textbook. In the textbook you have read about the pumping lemma for regular languages, a very useful tool to prove that certain languages are not regular. Here we consider a different method, called "diagonalization", that will be very useful later on in the course. The method involves the construction of a specific language which is not regular almost by definition. The language is not particularly meaninful, I know of no application where you would want to design a finite automaton for this language. The goal of this method is just to establish the existence of some languages which are not regular. The method is interesting because of its generality: you can use this same method to define computational problems that are unsolvable by virtually any computational model! So, no matter how powerful is your computer (or model of computation), there is always some well defined problem that is beyond its computational ability.

1 Encoding regular expressions

For concreteness, let us consider the set of regular languages over the binary alphabet $\{0,1\}$. We know that a language is regular if and only if it is the language of a regular expression R. Consider the set $\mathcal{R}$ of all regular expressions over the set of basic symbols $0,1$. These regular expressions can be represented as strings over the larger alphabet $\Sigma = \{0,1,+,\circ,(,),^\star,\emptyset\}$. (Notice that no special symbol is needed for the the empty string ϵ because it can be represented by the equivalent expression $\emptyset^\star$.) For example the set of all binary strings can be represented by the binary expression $E = (0+1)^\star$. Since the alphabet Σ has size 8, we may encode its symbols as triplets of bits, just like 8-bit bytes are used to represent characters on conventional computers. The way we map the elements of Σ to bits is largely arbitrary, but for concreteness let us consider a specific encoding $\phi\colon \Sigma \to \{0,1\}^3$ as defined by the following table:

a	0	1	$+$	$\circ$	(	)	$\star$	$\emptyset$
$\phi(a)$	000	001	010	011	100	101	110	111

Using this encoding, regular expressions can be also represented as binary strings, e.g., $\phi(E) = 100\ 000\ 010\ 001\ 101\ 110$. Of course, not every binary string is the representation of a syntactically valid regular expression, just like not every file represents a valid C program. For example, "000 010 101" does not represents a regular expression because it encodes the string "0+)". Also "0110" does not encodes any string over Σ because its length is not a multiple of 3. All we care about is that any regular expression $E \in \mathcal{R}$ is represented by a binary string $\phi(E) \in \{0,1\}^*$, and that any binary string encodes at most one regular expression. (I.e., the mapping $\phi\colon \mathcal{R} \to \Sigma^*$ is injective.) So, we may consider languages over $\{0,1\}$ (i.e., sets of binary strings) corresponding to specific sets of regular expressions.

Notice that each regular expression $E \in \mathcal{R}$ is represented by a binary string $\phi(E) \in \{0,1\}^*$, and it also represents a language $\mathcal{L}(E) \subseteq \{0,1\}^*$, i.e., a set of binary strings. So, shall we think of each regular expression as a string or as (the representation of) a sets of strings? Well, it is useful to do both. There is nothing special, or to be confused about here. This is just the same as a computer program being represented by a string (possibly including special “new line” characters to make the string more readable), and the same program representing a set of strings, e.g., the set of input strings for which the program outputs 1.

2 A nonregular language

We are now ready to formally prove that there is some language that is not regular, i.e., it cannot be described by a regular expression.

Let L be the set of all binary strings of the form $\phi(E)$ where $E \in \mathcal{R}$ is a binary regular expression such that $\phi(E) \notin \mathcal{L}(E)$. In English, you can describe this as the set of regular expressions that do not generate their own encoding. For example, 111 is in L because it encodes the “empty set” regular expression $\emptyset$, and clearly $\phi(\emptyset) = 111 \notin \emptyset = \mathcal{L}(\emptyset)$. We claim that this language is not regular. In fact, the proof is very simple, as the language L was defined with the specific goal of not being regular. Here is the proof.

Theorem 1 *The language $L = \{\phi(E) \colon E \in \mathcal{R} \wedge \phi(E) \notin \mathcal{L}(E)\}$ is not regular.*

Proof: Assume for contradiction that L is regular. Since $L \subseteq \{0,1\}^*$ is a binary language, there is a regular expression $E \in \mathcal{R}$ such that $\mathcal{L}(E) = L$. Now consider the following question: $\phi(E) \in L$? i.e., does the string $w = \phi(E)$ belongs to the set L. We do not know the answer to this question, but sure the answer must be either “yes” or “no”. We will show that in either case we get a contradiction: $w \in L$ if and only if $w \notin L$. This is proved by a chain of implications:

- By definition of L, we have $w \in L$ if and only if $w = \phi(E')$ for some regular expression $E' \in \mathcal{R}$ such that $\phi(E') \notin \mathcal{L}(E')$
- Since the function ϕ is injective, and recalling that $w = \phi(E)$, the condition $w = \phi(E')$ is satisfied if and only if $E = E'$.
- It follows that $w \in L$ if and only if the string $w = \phi(E) = \phi(E')$ is not in $\mathcal{L}(E') = \mathcal{L}(E) = L$

This proves that $w \in L$ if and only if $w \notin L$. This is a contradiction. So, our contradiction hypothesis must be false and L is not regular. $\square$

3 Why "diagonalization"

The technique used by the above construction and proof is called "diagonalization", and it was first discovered and used by mathematician Georg Cantor in 1873 to prove that there are infinite sets that cannot be put in one-to-one correspondence with the infinite set of natural numbers. You can read about Cantor's diagonal argument in the textbook (Theorem 4.17). Here we illustrated why it is called diagonalization in the context of our proof that there are nonregular languages. Think building an infinite table with rows and columns indexed by all possible binary strings and all table entries filled with 0s and 1s:

$$
\begin{array}{c|cccccccc}
 & \epsilon & 0 & 1 & 00 & \cdots & 111 & \cdots & \phi(E) \\
\epsilon & \mathbf{1} & 1 & 1 & 1 & \cdots & 1 & \cdots & 1 \\
0 & 1 & \mathbf{1} & 1 & 1 & \cdots & 1 & \cdots & 1 \\
1 & 1 & 1 & \mathbf{1} & 1 & \cdots & 1 & \cdots & 1 \\
00 & 1 & 1 & 1 & \mathbf{1} & \cdots & 1 & \cdots & 1 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots & & & \vdots \\
\phi(\emptyset) & 0 & 0 & 0 & 0 & 0 & \mathbf{0} & 0 & 0 \\
\vdots & & & & & & & \ddots & \vdots \\
\phi(E) & 1 & 0 & 0 & 1 & \cdots & 1 & \cdots & \mathbf{1}
\end{array}
$$

Each row $T[r, \epsilon], T[r, 0] \ldots$ represents a language: the set of strings x for which $T[r, x] = 1$. So, for example, the row labeled with $\phi(E)$ represents a language that contains ϵ, 00 and 111, but not 0 or 1. We are interested in the rows that represent regular expressions. (All other rows can be filled arbitrarily, but for concreteness we filled them with 1s.) For each row $\phi(E)$ representing a regular expression E, we fill the corresponding table entries so that the row represents the language $L(E)$ of the regular expression. For example, the row indexed by $\phi(\emptyset) = 111$ should be the all zero row because $L(\emptyset)$ does not contain any string. Notice that all regular languages are listed as a row in the table, because any regular language (over the alphabet $\{0, 1\}$) is represented by a regular expression. So, if we can come up with a language D which is different from all rows in the table, the language D is certainly not regular. We can come up with such a language by selecting a row which differs from the first row in its first entry, differents from the second row in its second entry, and so on. For each string x, we put it in $x \in D$ if $T[x, x] = 0$, and leave it out $x \notin D$ if $T[x, x] = 1$. In other words, the language D is obtained by taking all the diagonal entries of the table, and flipping them. It is easy to see that this language is precisely the diagonal language $D = \{\phi(E) \colon E \in \mathcal{R} \wedge \phi(E) \notin \mathcal{L}(E)\}$ built in the previous section.