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 meaningful, 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 \(R\) of all
regular expressions over the set of basic symbols 0, 1. These regular expressions can be represented as strings
over the larger alphabet \(\Sigma = \{0, 1, +, \circ, (, ), \star, \emptyset\}\), where . For example the set of all binary strings can be
represented by the binary expression \(E = (0 + 1)^*\). Since the alphabet \(\Sigma\) has size 8, we may encode its
symbols are triplets of bits, just like 8-bit bytes are used to represent symbols from larger alphabets. The
way we map the elements of \(\Sigma\) to bits is largely arbitrary, but for concreteness let us consider a specific
encoding \(\phi: \Sigma \to \{0, 1\}^3\) as defined by the following table:

<table>
<thead>
<tr>
<th>(a)</th>
<th>(\phi(a))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
</tr>
<tr>
<td>+</td>
<td>010</td>
</tr>
<tr>
<td>\circ</td>
<td>011</td>
</tr>
<tr>
<td>(</td>
<td>100</td>
</tr>
<tr>
<td>)</td>
<td>101</td>
</tr>
<tr>
<td>\star</td>
<td>110</td>
</tr>
<tr>
<td>\emptyset</td>
<td>111</td>
</tr>
</tbody>
</table>

Using this encoding, regular expressions can be also represented as binary strings, e.g., \(\phi(E) = 100000010001101110\).
Of course, every binary string is the representation of a regular expression, just like not every string over \(\Sigma\)
is a syntactically valid regular expression. But we may consider binary languages corresponding to specific
sets of regular expressions, as we do next. Notice that each regular expression \(E \in R\) is represented by a
single binary string \(\phi(E) \in \{0, 1\}^*\), and it also represents a language \(L(E) \subseteq \{0, 1\}^*\), i.e., a set of binary
strings. There is nothing special, or to be confused about. 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

Let \(L\) be the set of all binary strings of the form \(\phi(E)\) where \(E \in R\) is a binary regular expression such that
\(\phi(E) \notin L(E)\). 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:** The language \(L = \{\phi(E): E \in R \land \phi(E) \notin L(E)\}\) 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 \(E \in R\) such that \(L(E) = L\). Now consider the following question: \(\phi(E) \in L'\) i.e., does
the string \(w = \phi(E)\) belong 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 $\phi$ 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. □

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:

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

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 $\epsilon, 00$ 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 $\mathcal{L}(E)$ of the regular expression. For example, the row indexed by $\phi(\emptyset) = 111$ should be the all zero row because $\mathcal{L}(\emptyset)$ 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, different 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 $D = \{\phi(E): E \in \mathcal{R} \land \phi(E) \notin \mathcal{L}(E)\}$ built in the previous section.