March 8, 2021

Team homework submissions are now accepted! Starting with this homework, you are encouraged to form a team of up to four (4) people for a single joint homework submission. Benefits include sharing ideas with other people, reduced workload, and faster feedback (due to less grading). This is purely optional. If you do form a team, then elect one member of the team to submit the homework, and be sure all your names are on the submission.

Pumping Lemma Review

Here we review the Pumping Lemma for regular languages. This relates to Exercise 15, below.

Definition 1. We say that a language $L$ is pumpable iff

there exists an integer $p > 0$ such that

for all strings $w \in L$ with $|w| \geq p$,

there exist strings $x, y, z$ with $xyz = w$ and $|xy| \leq p$ and $|y| > 0$ such that

for every integer $i \geq 0$,

$xy^iz \in L$.

We prove this in class:

Lemma 2 (Pumping Lemma for Regular Languages). For any language $L$, if $L$ is regular, then $L$ is pumpable.

Here is the contrapositive, which is an equivalent statement:

Lemma 3 (Pumping Lemma (contrapositive form)). For any language $L$, if $L$ is not pumpable, then $L$ is not regular.

We use the contrapositive form to prove that certain languages are not regular by showing that they are not pumpable. By definition, a language $L$ is not pumpable iff

for any integer $p > 0$,

there exists a string $w \in L$ with $|w| \geq p$ such that

for all strings $x, y, z$ with $xyz = w$ and $|xy| \leq p$ and $|y| > 0$,

there exists an integer $i \geq 0$ such that

$xy^iz \notin L$. 

1
Here is a template for a proof that a language $L$ is not pumpable (and hence not regular). Parts in brackets are to be filled in with specifics for any given proof.

Given any $p > 0$,

let $w := \ldots$ with length $\geq p$.

Now for any $x, y, z$ with $xyz = w$ and $|xy| \leq p$ and $|y| > 0$,

let $i := \ldots$ which might depend on $p, w, x, y, and z$.

Then we have $xy^iz \notin L$ because $\ldots$.

Note:

- We cannot choose $p$. The value of $p$ could be any positive integer, and we have to deal with whatever value of $p$ is given to us.

- We can and do choose the string $w$, which will differ depending on the given value of $p$ (so the description of $w$ has to use $p$ somehow). We must choose $w$ to be in $L$ and with length $\geq p$, however.

- We cannot choose $x, y, or z$. These are given to us and could be any strings, except we know that they must satisfy $xyz = w$, $|xy| \leq p$, and $|y| > 0$.

- We get to choose $i \geq 0$ based on all the previous values.

Example: Let

$L = \{w \in \{0, 1\}^* \mid w \text{ has more 0’s than 1’s}\}$.

We show that $L$ is not pumpable using the template:

Given any $p > 0$,

let $w := 0^p1^{p-1}$. (Clearly, $w \in L$ and $|w| \geq p$.)

Now for any $x, y, z$ with $xyz = w$ and $|xy| \leq p$ and $|y| > 0$,

let $i = 0$.

Then we have $xy^iz = xy^0z = xz \notin L$, which can be seen as follows: Since $|xy| \leq p$ it must be that $x$ and $y$ consist entirely of 0’s, and so $y = 0^m$ for some $m$, and we further have $m \geq 1$ because $|y| > 0$. But then $xz = 0^{p-m}1^{p-1}$, and so because $p - m \leq p - 1$, the string $xz$ does not have more 0’s than 1’s, and thus $xz \notin L$.

Exercises

1. For the $\epsilon$-NFA of textbook Exercise 2.5.2,

\[
\begin{array}{c|ccc}
    & \epsilon & a & b \\
\hline
    \rightarrow p & \{q, r\} & \emptyset & \{q\} \\
    q & \emptyset & \{p\} & \{r\} \\
    \star r & \emptyset & \{r\} & \{p, q\} \\
\end{array}
\]

find an equivalent NFA (without $\epsilon$-moves) using the method explained in class. This is also Method 2 described in the COURSE NOTES (link from the course homepage) in Section 10.4.
2. Do Exercise 2.5.3(a): Design and $\epsilon$-NFA for the following language: the set of all strings consisting of zero or more $a$’s followed by zero or more $b$’s, followed by zero or more $c$’s. Try to use $\epsilon$-transitions to simplify your design.

3. Do Problem 2.3 (pp. 81–82). This illustrates a proof by string induction.

4. (a) Show that every regular language is recognized by an $\epsilon$-NFA where out of each state there is no more than one $\epsilon$-transition and no more than one non-$\epsilon$-transition (i.e., a transition on a symbol from the alphabet).

(b) Show that every regular language is recognized by an $\epsilon$-NFA where out of each state there is exactly one $\epsilon$-transition and exactly one non-$\epsilon$-transition (i.e., a transition on a symbol from the alphabet). (A solution to this part is obviously also a solution to the previous part.)

5. Do Exercise 3.1.1(b,c): Write regexes for the following languages:

b) The set of strings of 0’s and 1’s whose tenth symbol from the right end is 1.

c) The set of strings of 0’s and 1’s with at most one pair of consecutive 1’s.

6. (Optional) Do Exercises 3.1.2(b,c) and 3.1.3(a,b,c)

7. Write a regular expression for the language of strings over \{a,b,c\} where no $a$ appears after any $b$ or $c$.

8. Do Exercise 3.2.3: Convert the following DFA to a regular expression, using the state-elimination technique of Section 3.2.2.

\[
\begin{array}{c|cc}
& 0 & 1 \\
\hline
s & p & p \\
q & p & s \\
r & r & q \\
s & q & r \\
\end{array}
\]

9. Do Exercise 3.2.4(c): Convert the following regex to an $\epsilon$-NFA: $0(0 + 1)^*$.

10. Recall the DFA $D$ we constructed that accepts a binary string iff it has an odd number of 1’s:

\[
\begin{array}{c|cc}
& 0 & 1 \\
\hline
A & A & B \\
\star B & B & A \\
\end{array}
\]

(a) Convert $D$ into an equivalent clean $\epsilon$-NFA using the clean-up procedure in class (add a new start state, a new final state, and some $\epsilon$-transitions).

(b) Use the state elimination method to convert $D$ to a regular expression. Eliminate state $A$ first, then $B$.

11. Same exercise as before, except make $A$ the final state (so that $D$ accepts a string iff it has an even number of 1’s).
12. (Optional) Recall the product DFA $P$ that counts an even number of zeros and an odd number of ones:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rightarrow$ EE</td>
<td>OE</td>
<td>EO</td>
</tr>
<tr>
<td>OE</td>
<td>EE</td>
<td>OO</td>
</tr>
<tr>
<td>*EO</td>
<td>OO</td>
<td>EE</td>
</tr>
<tr>
<td>OO</td>
<td>EO</td>
<td>OE</td>
</tr>
</tbody>
</table>

Use the state elimination method to convert $P$ to a regular expression. (To control the complexity, you may wish to define names for intermediate regexes.)

13. Recall that a string $x$ is a subsequence of a string $y$ (written $x \preceq y$) if the symbols of $x$ appear in $y$ in order (although not necessarily contiguously). For language $L \subseteq \Sigma^*$, define

$$\text{SUBSEQ}(L) := \{ x \in \Sigma^* : (\exists y \in L)[x \preceq y] \},$$

that is, $\text{SUBSEQ}(L)$ is the set of all subsequences of strings in $L$. For example, if $L = \{aabc, cab\}$, then

$$\text{SUBSEQ}(L) = \{\epsilon, a, b, c, aa, ab, ac, bc, aab, aac, abc, aabc, ca, cb, cab\}.$$

Show that if $L$ is regular, then $\text{SUBSEQ}(L)$ is regular. [Hint: Two methods will work here: (1) transforming a regular expression for $L$ into a regular expression for $\text{SUBSEQ}(L)$; (2) transforming an $\epsilon$-NFA for $L$ into an $\epsilon$-NFA for $\text{SUBSEQ}(L)$. By the way, it is known that if $L$ is any language whatsoever, then $\text{SUBSEQ}(L)$ is regular, but the proof of this fact is not constructive.]

14. Write a regular expression for the language of strings over $\{a, b, c\}$ where no $a$ appears after any $b$ or $c$.

15. (Exercise 4.1.1 (selected items)): Prove that the following are not regular languages. For each, show that the given language is not pumpable. You may use the template given above.

(a) The set of strings of balanced parentheses. These are the strings of characters “(” and “)” that can appear in a well-formed arithmetic expression.

(b) $\{0^n1^n | n \geq 1\}$.

(c) $\{0^n1^m2^n | n \text{ and } m \text{ are arbitrary integers}\}$.

(d) $\{0^n1^{2n} | n \geq 1\}$. 

4