CSCE 355, Assignment 4

Pumping Lemma Review

Here we review the Pumping Lemma for regular languages. This relates to Exercise 8, 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 $s \in L$ with $|s| \geq p$ such that
> for all strings x, y, z with $xyz = s$ and $|xy| \leq p$ and $|y| > 0$,
> there exists an integer $i \geq 0$ such that
> $xy^iz \notin L$.

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 $s :=$ [describe some string in L with length $\geq p$].
> Now for any x, y, z with $xyz = s$ and $|xy| \leq p$ and $|y| > 0$,
> let $i :=$ [give some integer ≥ 0 which might depend on p, s, x, y, and z].
> Then we have $xy^iz \notin L$ because [give some reason/explanation].

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 s, which will differ depending on the given value of p (so the description of s has to use p somehow). We must choose s 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 $xyz = s$, $|xy| \leq p$, and $|y| > 0$.
- We get to choose $i \geq 0$ based on all the previous values.

Example: Let

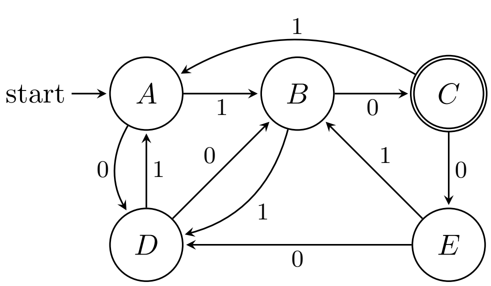
$$L = \{w \in \{0,1\}^* \mid w \text{ has more 0's than 1's}\}.$$

We show that L is not pumpable using the template:

> Given any $p > 0$,
> let $s := 0^p 1^{p-1}$. (Clearly, $s \in L$ and $|s| \geq p$.)
> Now for any x, y, z with $xyz = s$ 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. Consider the DFA N (below left) over the alphabet $\{0, 1\}$:



	A	B	C	D
B				
C				
D				
E				

   (a) Fill in the distiguishability table to the right with X in each entry corresponding to a pair of distinguishable states.
   (b) Draw the minimal DFA equivalent to N.

2. Using the sets-of-states method described in class or in the book, convert the following NFA N (no ϵ-moves) to an equivalent DFA D:

	a	b
$\rightarrow 1$	$\{1,2\}$	$\{1\}$
2	$\{2\}$	$\{1,3\}$
$*3$	$\emptyset$	$\emptyset$

Only give states of D that are reachable from its start state, and label each state of D with the states of N that it contains. Include all dead states (if there are any), and do not merge indistinguishable states.

3. Consider the regex $r := (\mathtt{a}+\mathtt{b})^*(\mathtt{b}+\mathtt{c})^*$ over the alphabet $\Sigma := \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$. Find a regex $\overline{r}$ such that $L(\overline{r}) = \overline{L(r)}$, the complement of $L(r)$ in Σ^*. Do this as follows:

   (a) Convert r to an equivalent ϵ-NFA N. (You may contract ϵ-transitions provided it is sound to do so.)

   (b) Remove ϵ-transitions from N to get an equivalent NFA N' using the method described in class and the course notes (Method 2).

   (c) Using the sets-of-states construction described in class, convert N' into an equivalent DFA D. (Only include states of D reachable from its start state.)

   (d) (Optional) Minimize D by merging indistinguishable states, if any.

   (e) Form the complementary DFA $\neg D$.

   (f) Starting with a clean ϵ-NFA equivalent to $\neg D$, find the equivalent regex $\overline{r}$ by the state elimination method described in class.

   As far as anyone knows, there is no general procedure for negating (complementing) a regex that is significantly faster than going through the steps above. The same holds for finding a regex for the intersection of two languages given by regexes, which would involve the product construction on two ϵ-NFAs or DFAs.

4. For any string $w \neq \epsilon$, the *principal suffix* of w is the string resulting by removing the first symbol from w. We will denote this string by $ps(w)$. For any language L, define $ps(L) := \{ps(w) : w \in L \wedge w \neq \epsilon\}$. Show that if L is regular, then $ps(L)$ is regular. (The underlying alphabet is arbitrary.)

5. (not in the textbook; optional) 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

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

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

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

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

6. (! (not in the textbook; optional)) Fix a finite alphabet Σ. Given string $w \in \Sigma^*$, a *cyclic shift* of w is any string of the form yx where $x, y \in \Sigma^*$ are such that $w = xy$. Given language $L \subseteq \Sigma^*$, define

   $$\mathrm{cyclicShift}(L) \ := \ \{yx \mid x, y \in \Sigma^* \wedge xy \in L\} ,$$

the language of all cyclic shifts of strings in L. Show that if L is regular, then cyclicShift(L) is regular. [Hint: Using an n-state ϵ-NFA recognizing L, you can construct an ϵ-NFA recognizing cyclicShift(L) with about $2n^2$ many states.]

7. (! (not in the textbook; optional)) Let x and y be any two strings over an alphabet Σ. A *merge* of x and y is any string over Σ obtained by merging the symbols of x with those of y in some arbitrary way, maintaining the order of the symbols from each string. More exactly, a string $z \in \Sigma^*$ is a *merge* of x and y iff there exist strings $x_1, \ldots, x_k$ and $y_1, \ldots, y_k$ in Σ^* (for some $k \geq 0$) such that

   - $x = x_1 x_2 \cdots x_k$,
   - $y = y_1 y_2 \cdots y_k$, and
   - $z = x_1 y_1 x_2 y_2 \cdots x_k y_k$.

   For example, there are five different merges of the strings `ab` and `bc`:

   `abbc`, `abcb`, `babc`, `bacb`, `bcab`

   Let A and B be any languages over Σ. Define

   $$A \, \mathit{merge} \, B := \{z \in \Sigma^* \mid z \text{ is a merge of some } x \in A \text{ and some } y \in B\}.$$

   Show that if A and B are both regular, then $A \, \mathit{merge} \, B$ is regular. *Hint*: Given a DFA for A with r many states and an DFA for B with s many states, you can construct an *NFA* for $A \, \mathit{merge} \, B$ with rs many states.

8. (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) $\{0^n 1 0^n \mid n \geq 1\}$.

   (c) $\{0^n 1^m 2^n \mid n \text{ and } m \text{ are arbitrary integers}\}$.

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

9. Consider the following grammar generating the language of strings of well-balanced parentheses:

   $$S \to (S)S \mid \epsilon$$

   Give a leftmost derivation of the string (()) and a rightmost derivation of the string ()((()()). Also give a parse tree yielding each string (two parse trees in all).

10. Describe briefly in words the language $L(G)$, where $G = (\{A, B\}, \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}, A, P)$ is a context-free grammar and the productions in P are

   $$A \to \mathtt{a}A\mathtt{c} \mid B$$
   $$B \to \epsilon \mid B\mathtt{c}$$

11. Give a context-free grammar for the language $\{\mathtt{a}^\ell \mathtt{b}^m \mathtt{c}^n \mid \ell \le m \text{ or } m \le n\}$. (Note that the connective is "or," not "and.")

12. Consider the grammar of Exercise 5.1.8:

$$S \to \mathtt{a}S\mathtt{b}S \mid \mathtt{b}S\mathtt{a}S \mid \epsilon$$

Show that `abba` is generated by the grammar but `aba` is *not* generated by the grammar. (This is a special case of the full exercise.)