The Pumping Lemma for Regular Languages

"Pumping" is a second technique to prove nonregularity. It is interesting in its own right, although no more powerful than the analysis of $\equiv_L$ that we have discussed already in Sections ??. Pumping is important if for no other reason then because unlike the analysis of $\equiv_L$ in Myhill-Nerode, pumping arguments also work for the more powerful programs we will consider in the next chapter.

Lemma 3 Pumping Lemma for Regular Languages. *If L is a regular language then there is a constant k such that for every string w in L of length k or more the following is true.*

There are three strings, u, v, x, such that

1. *$w = uvx$, and*
2. *for all $i \geq 0$, $uv^ix \in L$, and*
3. *$|v| > 0$.*

Let's use this Lemma to prove that the language $L = \{a^nb^n : n \geq 1\}$ is not regular.

Assuming it was. Then the Lemma applies. Let k be the constant said to exist in the Lemma. Consider the string $w = a^kb^k$. The length of w is k or more (in fact, $2k$), hence statements (1) — (3) from the Lemma apply, implying that it is possible to split w into three parts, $w = uvx$, such that $|v| > 0$ and $uv^2x \in L$. But this is not true. No matter what three-way split of w one tries, uv^2x is never of the form of equal numbers of a's and b's.

(One can come up with a more detailed argument for this. The challenge is to cover all ways in which the strings could get divided up into three pieces. An added challenge is to come up with a *concise* argument. Here is an attempt:

The substring v cannot contain more than one type of character because otherwise v^2 contains characters in an order not permitted in L. Hence at most one type of character take part in the "pumping" from uvx to uv^2x, which therefore does not contain equal numbers of both characters anymore.)

This contradiction shows that L is not regular. □

Why is this so messy? The Pumping Lemma can be stated as follows.

$$L \text{ regular}$$

$$\Longrightarrow$$

$$\begin{array}{c}\text{There is a constant } k \text{ such that} \\ \text{for all strings } w \in L \text{ with } |w| \geq k \\ \text{there are strings } u, v, x \text{ with } w = uvx \text{ and } |v| > 0 \text{ such that} \\ \text{for all } i \geq 0,\ uv^ix \in L.\end{array}$$

The above statement is of the form "*L regular* $\Rightarrow B$," where B is that complicated statement with four alternating quantifiers. Since we are trying to prove that some languages are not regular, a more useful version of the statement is "*L not regular* $\Leftarrow \neg B$." Spelled out in detail, this version reads

$$L \text{ not regular}$$

$$\Longleftarrow$$

$$\begin{array}{c}\text{For all constants } k \\ \text{there is a string } w \in L \text{ with } |w| \geq k \text{ such that} \\ \text{for all strings } u, v, x \text{ with } w = uvx \text{ and } |v| > 0 \\ \text{there is an } i \geq 0 \text{ with } uv^i x \notin L.\end{array}$$

Thus, we can prove that some language L is not regular by proving the above four-line statement $\neg B$: *"For all ... $\notin L$"*. This is what we did in the previous example. Here is another one.

Another Example

Let's take the language that consists of strings like

```
1234+111=1345
10000+10000=20000
1020304+505=1020809
11111+22222=33333
```

The language consists of strings of the form $\alpha + \beta = \gamma$ that represent a *correct* equation between decimal integers.

This language L is not regular, a fact we prove by proving the statement "$\neg B$" above.

How do we prove something *"for all constants k"*? By starting out the proof by saying "Let k be an arbitrary constant" and carrying the k along as a parameter of the argument.

What's next? We need to show that *there exists* a string w with certain properties. To show this, all we need to do is exhibit *one* such string. In this example, let's pick w to be the string

$$\underbrace{1\cdots1}_{k}+\underbrace{2\cdots2}_{k}=\underbrace{3\cdots3}_{k}$$

(Note how the k plays its role.) What's the next step? We have to show that *for all* strings u, v, x which represent a three-way breakup of w ("$w = uvx$") and which don't have the second piece empty ("$|v| > 0$"), there is a constant i with $uv^i x \notin L$. This is the hard part of the proof because we somehow have to provide an argument that covers all possible such three-way breakups of w. It can be done, by dealing with an exhaustive set of cases. Here is such an exhaustive set of cases for this example.

Case 1. v contains the =. Then $uv^0 x \notin L$ if for no other reason then because it does not have an = in it.

Case 2. v does not contain the $=$. Then v lies entirely on one side of the $=$ and pumping (with any $i \neq 1$) changes that side of the equation but not the other, thus creating a string outside of L.

Exercises

Ex. 1. Draw the smallest finite-state machine M for the language of all bitstrings that contain the substring 011.

Ex. 2. Draw the smallest finite-state machine M for the language of all bitstrings that contain the substring 011. (Make sure you provide a 0 and a 1 transition out of every state.)

Minimizing finite-state machines

Ex. 3. Carry out the minimization algorithm of Figure 2.6 for the finite-state machine shown below. (Show the development of the "clusters" of states just like Figure 2.8 did.)

Ex. 4. Draw a finite-state machine on which the minimization algorithm runs through more than 10 passes.

The Myhill-Nerode Theorem

Ex. 5. Let L be the language of all bitstrings that contain a run of three consecutive 0s. Is it true that $01 \equiv_L 001$? Explain.

Ex. 6.

1. If L is the language of all bitstrings that start with a 1 and end with a 0, then $01 \equiv_L 00$.
2. If L is the language of all bitstrings of length 8, then $0001 \equiv_L 0011$.
3. Let L be the language of all bitstrings x such that x contains at least one 0. (For example, $01 \in L$, $1111 \notin L$.) How many equivalence classes does $\equiv_L$ induce?
4. Let L be the language of all bitstrings x such that x contains at least one 0 and at least one 1. (For example, $01 \in L$, $1111 \notin L$, $00 \notin L$.) How many equivalence classes does $\equiv_L$ induce?

Ex. 7. Let L be the language of binary strings whose third bit from the end is a 1. Give one string from each of the equivalence classes induced by $\equiv_L$.

Ex. 8. Same for the language of bitstrings whose first two bits are equal to its last two bits.

Ex. 9. Give one representative of each equivalence class of $\equiv_L$ for $L = \{x : x$ *contains the substring* $000\}$.

Ex. 10. Consider the following language L over the alphabet $\Sigma = \{a, b, \$\}$:

$$L = \{x\$y : x, y \in \{a, b\}^+ \textit{ and } x \textit{ does not start with the same letter as } y\}$$

For every equivalence class of $\equiv_L$, give one shortest string from the class.

Ex. 11. Is it true that $x \equiv_L y$ implies $xz \equiv_L yz$? Is it true that $xz \equiv_L yz$ implies $x \equiv_L y$? When your answer is no, give a counterexample.

Ex. 12. Are the two relations $\equiv_L$ and $\equiv_{\bar{L}}$ (that's $\bar{L}$, the complement of L) always the same, for any given L? Either argue that they are, or show an example were they aren't?

Ex. 13. *(b)* Do the partitions that go with $\equiv_L$ and with $\equiv_{L^{rev}}$ (that's $L^{rev} = \{x^{rev} : x \in L\}$) always have the same equivalence classes? Either argue that they always do, or show an example were they don't? (x^{rev} is x written backwards.)

Ex. 14. Which of the following languages is regular? (Just circle "Yes" or "No" for each language. Do not give explanations.)

1. The language described by $(ab(a^\star)^\star(b^\star \cup (aab)^\star))$
2. Strings containing the word `while`
3. Strings containing two opening parentheses with no closing parenthesis in between the two, e.g. "`(a)b)c(d(e(f`"
4. Strings containing more opening parentheses than closing ones, e.g. "`())(((`"
5. Strings containing all their opening parentheses before all their closing ones, e.g. "`((x(y-/)`"
6. Strings of even length whose first half and second half both start with the same letter, e.g. "`abcabaeeee`"

Ex. 15. Consider two finite-state machines that accept languages A and B and consider their cross-product machine. As discussed in class, one can always choose the accepting states of the cross-product machine such that it accepts $A \cap B$.

Can one always choose the accepting states of the cross-product machine such that it accepts ...

1. $A \cup B$?
2. $A - B$? (That's $\{x : x \in A \text{ and } x \notin B\}$.)
3. AB ? (That's A concatenated with B: $\{xy : x \in A \text{ and } y \in B\}$.)
4. Strings in $A \cup B$ of length five or more?
5. A ?

Regular Expressions

Ex. 16. Derive an asymptotic bound on the length of the expression constructed from a finite-state machine in the proof of Kleene's Theorem. Assume we are dealing with bitstrings only.