CS 361 Meeting 10 — 3/2/20

Announcements

1. Homework 4 online now, due Friday.

Review

1. Last time I introduced the FAMOUS Pumping Lemma:

   Lemma: Suppose L is a regular language. Then there exists a positive integer p such that any string $s \in L$ with length at least p may be partitioned into $s = xyz$ where

   (a) $|y| > 0$
   (b) $|xy| \leq p$
   (c) $xy^iz \in L$, for all $i \geq 0$.

2. Today, we will spend some time practicing how to use this result to prove that languages are not regular. If we have time, we will take a quick look at how to prove the lemma formally.

3. By way or review, last time we considered proofs that several languages were not regular:

   - $L_{UnaryAdd} = \{1^a + 1^b = 1^c \mid a + b = c\}$
   - $L_{EQ} = \{1^n = 1^n \mid n \geq 0\}$
   - $L_{EQ-RE} = \{e = e' \mid e \ \& \ e' \text{ are regular exprs and } L(e) = L(e')\}$

4. The point of the last example was to show that you can often avoid using the Pumping Lemma by using closure properties of regular languages in conjunction with known languages that are not regular.

   - Consider $L_{EQ-RE} = \{e = e' \mid e \ \& \ e' \text{ are regular exprs and } L(e) = L(e')\}$
   - Remember, that $\{1^n = 1^n \mid n \geq 0\} \subset L_{EQ-RE}$ and we know that this subset of L_{EQ-RE} is not regular.
   - Consider the language $1^* = 1^*$ (which is clearly regular).
   - The intersection of L_{EQ-RE} with $1^* = 1^*$ is just L_{EQ}.
   - If L_{EQ-RE} was regular, then since regular languages are closed under intersection, L_{EQ} would have to be regular.
   - So, L_{EQ-RE} must not be regular.

Click here to view the slides for this class

Thinking Negative Thoughts

1. In the last class, I also tried to emphasize that since we almost alway use the Pumping Lemma in proofs by contradiction, our goal is usually to show that the Lemma is not satisfied rather than showing it is. As a result, it is often useful to state the negation of the lemma rather than its positive statement. That is...

2. To show that L is not regular you must show that

   - **for every** sufficiently large p
   - **there exists** a string $s \in L$ of length $\geq p$ such that
   - **for every** possible partition of $s = xyz$ where
     (a) $|y| > 0$
     (b) $|xy| \leq p$
   - **there exists** $i \geq 0$, such that $xy^iz \notin L$.

Can You Pump?

1. Part of the reason most of the examples we have considered have been fairly easy is that the languages are fairly sparse. There are many possible strings that don't belong to these languages, so it is actually a bit hard to pump without bumping into something not in the language. In fact, for a language like $\{1^n = 1^n \mid n \geq 0\}$ all sufficiently long strings that belong to the language serve as counter examples.

2. Consider $L_{EQ-01} = \{w | w \in \{0,1\}^* \text{with equal number of 0s and 1s}\}$.

   - First, contrast this language with $\{0^n1^n | n \geq 0\}$.
   - Every possible permutation of a string in $\{0^n1^n | n \geq 0\}$ belongs to L_{EQ-01}. As a result, in some intuitive sense, L_{EQ-01} is a much bigger or denser language than $\{0^n1^n | n \geq 0\}$. Conceptually, this might make it harder to find a string that cannot be pumped.

- We can prove that L_{EQ-01} is not regular by first proving that the simple language $\{0^n1^n | n \geq 0\}$ is not regular and employing closure properties to conclude that this implies that L_{EQ-01} must be regular.
  - Consider the intersection of L_{EQ-01} with 0^*1^*.
  - This is just the language $\{0^n1^n | n \geq 0\}$.
  - We can prove that $\{0^n1^n | n \geq 0\}$ is not regular by observing we can't pump the 0s in 0^p1^p in much the same way we have for strings like $1^p = 1^p$. The boundary between the 0s and 1s is effectively equivalent to a delimiter like "=".
  - If L_{EQ-01} were regular, its intersection with 0^*1^* would have to be regular since regular languages are closed under intersection.
  - Thus, L_{EQ-01} must not be regular.

3. For practice, however, let's try to show that is not regular directly using the Pumping Lemma. Letting p be the value specified by the Pumping Lemma for this language, consider the following strings from the language:

   - $(01)^p$
   - 0^p1^p
   - $0^k1^p0^{p-k}$ with $p > k > p/2$
   - $(10)^p(01)^p$

   Which of these strings can or cannot be pumped?

   - $(01)^p$ can be pumped.
     Let $x = \epsilon, y = 01, z = (01)^{p-1}$ then $(01)^p = xyz$ and $xy^iz = (01)^{p+(i-2)} \in L_{EQ-01}$. Therefore, this string does not provide what we need to show that the language is not regular.
   - 0^p1^p cannot be pumped.
     Given that the string starts with p 0s, if for some x, y, z we have $0^p1^p = xyz$ and $|xy| \leq p$ then x and y must consist only of 0s and given $|y| > 0$, we know that xy^2z must contain more 0s than 1s so it does not belong in the language. This violates the conditions of the Pumping lemma and therefore shows that the language is not regular.
   - $0^k1^p0^{p-k}$ with $p > k > p/2$ can be pumped.
     If we let $x = 0^{k-1}, y = 01$ and $z = 1^{p-1}0^{p-k}$ then for all i, $xy^iz = 0^{k-1}(01)^i1^{p-1}0^{p-k}$ will have exactly $p + I - 1$ 0s and 1s so it does belong to the language. Again, this does not provide evidence that the language is non-regular.
   - $(10)^p(01)^p$ can be pumped.
     As with the first example, just let x be empty and y be the first 0101. This example is include to remind you that when you pick a pattern for a string to pump, pumping does not have to preserve the pattern, it just has to stay within the original language. Here, the pattern requires an even number of 01 pairs. When we pump, we may end up with an odd number of 01 pairs, but the number of 0s and 1s will still be equal so the pumped strings will still belong to the language.

4. So, if you wanted to prove this language was not regular, you could use 0^p1^p as a counter example in your argument, but you could not use the other three examples we just considered.

5. This example illustrates an important fact. Not all strings in a language will violate the Pumping lemma if the language is non-regular. One has to carefully search for an appropriate string.

A Needle in a Haystack

1. An example that gets even harder because the language is not at all sparse is $L_{NOTEQUAL} = \{1^n \neq 1^m \mid m \neq n\}$.

   - If you think about some examples we have considered like $L_{EQUAL} = \{1^n = 1^m \mid m = n\}$, it isn't hard to find an example that ends up outside the language no matter how you try to pump because there are very few strings in the langauge.
     - For odd length l, there is exactly 1 out of 2^l possible strings over the alphabet that fall in the language and 1 out of about $l - 2$ strings that have the right form.

- There are no strings of even length in the language.

- $L_{NOTEQUAL}$ is comparatively dense:
  - As suggested above, there are roughly $l-2$ strings of length l that fall within the language.

2. Let's assume that the pumping length for $L_{NOTEQUAL}$ is p and try to see if we can find ways to pump a few strings in the language.

   $1^p \neq 1^{p+1}$

   For this example, almost any partition you can think of will work. Basically, we can set $x = 1^l$, $y = 1^m$, $z = 1^{p-(l+m)} \neq 1^{p+1}$ as long as we make sure that $l + m \leq p$, $m > 0$ and for all $i \geq 0$, $l + im + p - (l + m) = (i-1)m + p \neq p + 1$.

   This will be satisfied as long as $(i-1)m \neq 1$ which we can ensure by making $m > 1$. That is, the only form of pumping that will get us into trouble is if we tried having $y = 1$.

   $1^p \neq 1^{p+p^2}$

   Showing that we can pump one type of string is not sufficient. We have to show that we can pump any string of sufficient length. So, until we can think of an argument that covers all possibilities, we have to keep looking.

   As above, we can say that no matter how we partition this string we will have $x = 1^k$, $y = 1^m$, $z = 1^{p-(l+m)} \neq 1^{p+p^2}$ and that therefore, all we need to ensure is that it is not possible to find an i such that $(i-1)m + p \neq p + p^2$.

   - That is, we have to make sure that we cannot set

     $$i = \frac{p^2}{m} + 1$$

   - We can accomplish this by letting $x = 1$ and $y = 1^{p-1}$ so that $m = p - 1$ which cannot divide p evenly.

   $1^p \neq 1^{p+p!}$

   Obviously, we saved the best for last.

   - Again, we have to choose k and m so that $x = 1^k$, $y = 1^m$, $z = 1^{p-(l+m)} \neq 1^{p+p^2}$ in such a way that $k+im+p-(l+m) = (i-1)m + p \neq p + p!$.
   - This means we need to choose m so that we cannot set $i = \frac{p!}{m} + 1$. Alas, since $m \leq p$, any value of m we choose will divide $p!$, so no matter what m we choose, there will be a value of i for which the Pumping Lemma fails.
   - Of course, the good news is that if our job was to show that $L_{NOTEQUAL}$ is not regular, this is just what we were looking for.

Note: The remainder of the material in these notes have not been covered in class, but I wanted to provide you with the details of the following proof based on $\hat{\delta}$ to complement the one you will find in Sipser's text.

Proving the Pumping Lemma

1. The proof of this Lemma is just a generalization of the argument I used in the last class to show that

   $$L_{UnaryAdd} = \{1^a + 1^b = 1^c | 1^k \text{ refers to a string of } k \text{ 1s and } a + b = c\}$$

   was not regular.

2. We need two technical lemmas I have put in a figure so that they don't separate the Pumping Lemma from its proof on the page.

   - The first lemma enables us to assume that if we break a string up into subparts, we will get the same final state by applying $\hat{\delta}$ to the whole string that we get if we apply $\hat{\delta}$ to the subparts in the appropriate order.
   - The second lemma formalizes the idea that if a given string leads a DFA through a loop, the repeating that string an arbitrary number of times starting in the first state of the loop will always bring the machine back to that state.

3. Given these lemmas, the proof of THE LEMMA is straightforward.

   (Pumping) Lemma : Suppose L is a regular language. Then there exists a positive integer p such that any string $s \in L$ with length at least p may be partitioned into $s = xyz$ where

---

Lemma 1: Given $\delta : Q \times \Sigma \to Q$, if we define $\hat{\delta} : Q \times \Sigma^* \to Q$ as:

$$\begin{aligned} \hat{\delta}(q, \varepsilon) &= q & (q \in Q) \\ \hat{\delta}(q, wx) &= \delta(\hat{\delta}(q, w), x) & (q \in Q,\, x \in \Sigma,\, w \in \Sigma^*) \end{aligned}$$

then $\hat{\delta}(q, wv) = \hat{\delta}(\hat{\delta}(q, w), v)$ for all $w, v \in \Sigma^*$.

Proof: The proof is by induction on the length of v. The basis is essentially the definition of $\hat{\delta}$. Namely that

$\hat{\delta}(q, w\epsilon) =$

$\hat{\delta}(q, w)$	by definition of concatenation
$\hat{\delta}(\hat{\delta}(q, w), \epsilon)$	by definition of $\hat{\delta}$

For the induction step, we assume the result is true for $|v| \leq k$ and try to show that $\hat{\delta}(q, w(vx)) = \hat{\delta}(\hat{\delta}(q, w), vx)$ for any $x \in \Sigma$. We know that $\hat{\delta}(q, wvx) = \delta(\hat{\delta}(q, wv), x)$. By our inductive assumption, we can therefore say that $\hat{\delta}(q, wvx) = \delta(\hat{\delta}(\hat{\delta}(q, w), v), x)$. By the definition of $\hat{\delta}$ we can conclude that $\hat{\delta}(q, wvx) = \hat{\delta}(\hat{\delta}(q, w), vx)$ as desired.

Lemma 2: Given $\delta : Q \times \Sigma \to Q$, if we define $\hat{\delta} : Q \times \Sigma^* \to Q$ as:

$$\begin{aligned} \hat{\delta}(q, \varepsilon) &= q & (q \in Q) \\ \hat{\delta}(q, wx) &= \delta(\hat{\delta}(q, w), x) & (q \in Q,\, x \in \Sigma,\, w \in \Sigma^*) \end{aligned}$$

and for some $w \in \Sigma*$ and $q \in Q$, $\hat{\delta}(q, w) = w$ then $\hat{\delta}(q, w^i) = q$ for all $i \geq 0$.

Proof: The proof is by induction on i. For $i = 0$, we know that $\hat{\delta}(q, w^0) = \hat{\delta}(q, \epsilon) = q$ by the definition of $\hat{\delta}$.
Now, assume that the result holds for all values less than or equal to i and consider the case of $i + 1$. $\hat{\delta}(q, w^{i+1}) = \hat{\delta}(\hat{\delta}(q, w^i), w)$ by Lemma 1. Thus, given our inductive assumption it is clear that $\hat{\delta}(q, w^{i+1}) = \hat{\delta}(\hat{\delta}(q, w^i), w) = \hat{\delta}(q, w) = q$ as required.

Figure 1: Lemmas related to the Pumping Lemma

(a) $|y| > 0$
(b) $|xy| \leq p$
(c) $xy^iz \in L$, for all $i \geq 0$.

Proof of Pumping Lemma: Suppose L is a regular language. If L is finite, then choose p to be longer than any string in L and the conditions of the lemma are satisfied trivially. Otherwise, let $M = (Q, \Sigma, \delta, s, F)$ be a DFA such that $L = L(M)$. Let p be the number of states in M.

Suppose $w = w_1w_2w_3...w_n \in L$ with $n \geq p$. When M processes w, it must pass through a sequence of states $q_0q_1...q_n$ such that

- $q_0 = s$
- $q_i = \hat{\delta}(s, w_1...w_i), 1 \leq i \leq n$

Since there are $p+1$ states in the sequence $q_0q_1...q_p$, we know that some state must appear twice in this sequence. Let l and k be two positions where such a repeated state occurs in $q_0q_1...q_p$ with $0 \leq k < l \leq p$. Let $x = w_1...w_k, y = w_{k+1}...w_l$, and $z = w_{l+1}...w_n$.

Clearly, the x and y we have selected satisfy conditions (a) and (b) in the statement of the Pumping Lemma. All we need to show is condition (c).

If we call the repeated state $\hat{q}$ then $\hat{\delta}(s, xy) = \hat{\delta}(s, x) = \hat{q}$ and $\hat{q} = \hat{\delta}(s, xy) = \hat{\delta}(\hat{\delta}(s, x), y) = \hat{\delta}(\hat{q}, y)$. Lemma 2 therefore allows us to conclude that for all $i \geq 0, \hat{q} = \hat{\delta}(\hat{q}, y^i)$. Lemma 1 then implies that $\hat{\delta}(\hat{q}, y^i) = \hat{\delta}(\hat{\delta}(s, x), y^i) = \hat{\delta}(s, xy^i) = \hat{q}$.

Using Lemma 1, we can write $\hat{\delta}(\hat{q}, z) = \hat{\delta}(\hat{\delta}(s, xy), z) = \hat{\delta}(s, xyz) \in F$.

Therfore, we can conclude that $\hat{\delta}(\hat{\delta}(s, xy^i), z)) = \hat{\delta}(s, xy^iz) \in F$ for all $i \geq 0$ as required by condition (c), completing the proof of the Lemma.