Announcements
1. Homework 3 due Friday.

Regular Language are Repetitious

1. Last time, we began to think about how we could determine whether or not a language was regular.

2. I suggested that to make progress on thinking about how to show a language was not regular it would be best to start with a language that was very simple:

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

is regular.

3. I argued that this language could not be regular because any machine that recognized the language would have to loop through some state on sufficiently large inputs and that we could construct strings by replicating the symbols consumed while traversing the loop such that the strings constructed did not belong to the desired but would have to lead the machine to a final state.

4. I wasn’t confident that you all saw the logic behind this argument, so before going any further I want to solidify your thinking about how following loops in the state diagram of a DFA leads to repeatable substrings in the language of the machine.

- Consider the following machine (which happens to be the “5” version of the “binary numbers that are multiples of N” DFA we presented using our formal notation earlier).

- What I would like to do with this machine is look for simple cycles. To be more precise I would like to look for strings of varying lengths that take the machine from start state to final state while repeating at least one but as few as possible of the states in the path.

- Here are some examples:

<table>
<thead>
<tr>
<th>input</th>
<th>path</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>( \varepsilon \rightarrow 0 \rightarrow 0 )</td>
</tr>
<tr>
<td>11001</td>
<td>( \varepsilon \rightarrow 1 \rightarrow 3 \rightarrow 1 \rightarrow 0 )</td>
</tr>
<tr>
<td>101101</td>
<td>( \varepsilon \rightarrow 1 \rightarrow 2 \rightarrow 0 \rightarrow 1 \rightarrow 2 \rightarrow 0 )</td>
</tr>
<tr>
<td>111101</td>
<td>( \varepsilon \rightarrow 1 \rightarrow 3 \rightarrow 2 \rightarrow 0 \rightarrow 1 \rightarrow 2 \rightarrow 0 )</td>
</tr>
</tbody>
</table>

- I haven’t included inputs that caused simple cycles of length 5
and 6 because there are no such simple cycles in this machine’s state graph.

- An alternate, interesting questions we can ask about these lengths is whether there are inputs of length 5 or 6 that traverse paths in the state diagram that contain no cycles.
  - For 5 we get:
    
    | input | path |
    |-------|------|
    | 01110 | ϵ → 0 → 1 → 3 → 2 → 4 |
  
    - This is a path that visits every state in the machine (but ultimately ends in a non-final state.)

- For 6, there is no such input because an input of length 6 must visit 7 states (including the start state). Since the machine only has 6 states, any path of length 7 must include at least one repeat.

5. The interesting thing about the sorts of loops in the state diagram we have been looking at is that they correspond to substrings in the input that can be repeated (or omitted) without changing the final state of the corresponding path. This means from each of our examples, we can generate an infinite set of inputs that lead to the same final state by just “starring” the input subsequence that leads the machine through the loop. The following table show how this would be done for the examples discussed above.

<table>
<thead>
<tr>
<th>input</th>
<th>related strings</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00*</td>
</tr>
<tr>
<td>11001</td>
<td>1(10)*01</td>
</tr>
<tr>
<td>101101</td>
<td>1(011)*01</td>
</tr>
<tr>
<td>1111101</td>
<td>1(1111)*01</td>
</tr>
</tbody>
</table>

6. It was this property of strings that lead a DFA through a cycle that I was using last time to show that the language

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

is not regular.

7. If this language were regular, then there would be some DFA M such that \( L_{UnaryAdd} = L(M) \). if \( M = (Q, \Sigma, \delta, s, F) \), and \( n = |Q| \) then the input \( 1^{2n} + 1 = 1^{2n+1} \):

a) must lead to a final state in \( M \), and

b) must lead \( M \) through at least one cycle in its set of states (since its length requires more states be visited than there are distinct states).

8. In case the notation \( 1^{2n} + 1 = 1^{2n+1} \) isn’t sufficiently illuminating, we can show the structure of the input we have in mind as:

<table>
<thead>
<tr>
<th>input</th>
<th>related strings</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>2n</td>
</tr>
<tr>
<td>+1</td>
<td>3</td>
</tr>
<tr>
<td>111...111</td>
<td>2n+1</td>
</tr>
</tbody>
</table>

9. The “cloudy” diagram below was intended to give some names to the “dimensions” of this loop. In particular we are assuming that the first loop begins after \( k \) 1’s have been read and that it is completed after \( l \) steps implying that the number of 1s in the input that drive the machine around the loop is \( l - k > 0 \).

10. This suggests another way we can partition the input:
11. Now, the key observation is that since the string of \( l - k \) ones leads the machine \( M \) through a loop, we can describe an infinite set of strings that must belong to \( L(M) \) by “starring” this sequence of 1’s.

\[
\begin{array}{|c|c|c|c|c|}
\hline
\text{input} & 11\ldots11 & (11\ldots11)^* & 11\ldots11 & +1 = 111\ldots111 \\
\hline
\text{length} & k & l - k & 2n - l & 3 & 2n + 1 \\
\hline
\end{array}
\]

12. Since the closure (*) means we can either repeat or delete (i.e., repeat 0 times) a substring, this would allow us to conclude that the string

\[
\begin{array}{|c|c|c|c|c|}
\hline
\text{input} & 11\ldots11 & 11\ldots11 & +1 = 111\ldots111 \\
\hline
\text{length} & k & 2n - l & 3 & 2n + 1 \\
\hline
\end{array}
\]

which is just

\[
\begin{array}{|c|c|c|c|c|}
\hline
\text{input} & 11\ldots11 & +1 = 111\ldots111 \\
\hline
\text{length} & 2n - (l - k) & 3 & 2n + 1 \\
\hline
\end{array}
\]

which is \( 2^{2n - (l - k)} + 1 = 2^{2n+1} \in L_{\text{UnaryAdd}} \). This, however, should only be true if \( 2n - (l - k) + 1 = 2n + 1 \) which would imply \( l - k = 0 \) contrary to our conclusion that there must be a cycle of length 1 or greater in the path followed processing a string longer than the number of states in the machine \( M \).

13. This contradiction allows us to conclude that our assumption that \( L(M) = L_{\text{UnaryAdd}} \) was false, so \( L_{\text{UnaryAdd}} \) must not be regular.

**The Pumping Lemma (for regular languages)**

1. We can generalize the partitioning we performed on \( 2^{2n + 1} \) in a way that leads to an understanding of a more general result that can be used to show that certain languages are not regular.

- The key to our discussion of \( 2^{2n + 1} \) was the subsequence of 1’s in \( 2^n \) that could be repeated. That is, there were really three key parts to our partition as shown below:

\[
\begin{array}{|c|c|c|c|c|}
\hline
\text{input} & 11\ldots11 & (11\ldots11)^* & 11\ldots11 + 1 = 111\ldots111 \\
\hline
\text{length} & k & l - k & 4n - l + 4 \\
\hline
\end{array}
\]

- Better yet, rather than counting all the digits so carefully, we can just name subparts as in:

\[
\begin{array}{|c|c|c|c|}
\hline
\text{input} & 11\ldots11 & (11\ldots11)^* & 11\ldots11 + 1 = 111\ldots111 \\
\hline
\text{name} & x & y & z \\
\hline
\end{array}
\]

- Now, we can summarize the logic behind our argument by saying that for any string \( w \) that is long enough, we must be able to write \( w = xyz \) in such a way that \( y \) corresponds to a string that leads \( M \) through a loop and therefore, it must be the case that \( xy^iz \subseteq L \) (or equivalently \( xy^iz \in L, i \geq 0 \)).

2. This generalization of our approach to \( L_{\text{UnaryAdd}} \) is encapsulated in the FAMOUS Pumping Lemma:

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

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

3. It may at times be useful to use a slightly different statement of this result:

**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) \( xy^iz \subseteq L \), for all \( i \geq 0 \).
4. The proof of this lemma is in the text. For not, we will just assume the lemma is true and work to make sure we know how to use it to show that languages are not regular.

**Pumping Iron**

1. I started with unary addition because I had previously mentioned that at least one language encoding addition (binary addition) was not regular in a homework assignment.

2. As our first exercise with the Pumping Lemma, let's consider an even simpler language.

3. Let's show that $L_{EQ} = \{1^n = 1^n \mid n \geq 0\}$ is not regular.
   - Using the Pumping Lemma, all we need to do is given a possible $p$ for this language, show how to find a string that cannot be pumped (i.e., a string that turns into strings that don’t belong in $L_{EQ}$ when pumped).
   - Consider $1^p = 1^p$.
   - The pumping lemma requires that we be able to remove or duplicate some substring $y$ which is non-empty ($|y| > 0$) that appears within the first $p$ symbols ($|xy| \leq p$) of $1^p = 1^p$.
   - Any prefix $xy$ of $1^p = 1^p$ of length at most $p$ must contain only 1s. So, $x = 1^k$ and $y = 1^l$ for some $l$ and $k$ such that $l > 0$ and $l + k \leq p$.
   - The Pumping lemma allows us to conclude that if $L_{EQ}$ is regular all strings of the form $1^k (1^l)^* 1^{p-(l+k)} = 1^p \in L_{EQ}$. However, $1^k 1^{p-(l+k)} = 1^p \notin L_{EQ}$.
   - Therefore, $L_{EQ}$ must not be regular!

4. This brings us back to the example I started with to remind you that decision problems can be interesting.

5. It is worth observing that similar tricks can be used to show that other language that initially seem complex can be shown non-regular by focusing on simple subsets.
   - For any alphabet $\Sigma$, consider $L_{ValidRE} = \{e \mid e$ is a valid regular expression over $\Sigma\}$.
   - Consider the intersection of this set with the language $(^*x)^*$ (this is intended to be a regular expression describing strings of the form $((\ldots((=))\ldots)))$ where the parentheses are not necessarily balanced).
   - The intersection of $L_{ValidRE}$ with $(^*=)^*$ is just $(^nx)^n$ which is a lot like $1^n = 1^n$. You could easily prove $(^nx)^n$ is not regular using the same approach we used for $1^n = 1^n$ (or once you finish the homework assignment you could use the fact that regular languages are closed under homomorphism).
   - This would enable you to conclude that $L_{ValidRE}$ is not regular.

$L_{EQ-RE} = \{e = e' \mid e & e'$ are regular expressions over $\Sigma$ and $L(e) = L(e')\}$