CS 361 Meeting 14 — 3/11/20

Announcements

1. Homework 5 is due Friday.
2. Take-home/self-scheduled midterm planned for next week!

Review

1. I want to remind you of two examples we considered last class.
   - The second is:

   $$L_{\text{UnaryAddition}} = \{1^a + 1^b = 1^{a+b} \mid a, b \geq 0\}$$

   - The key to the this example is to recognize that any string of the form $1^n + w1^n$ where $w \in L_{EQ}$ belongs in $L_{\text{UnaryAddition}}$.
   - Therefore, the grammar:

     A → 1 A 1
     A → + E
     E → 1 E 1
     E → =

   describes $L_{\text{UnaryAddition}}$.

2. The third is a good "real" example of the use of context-free grammars to formalize the recursive description of a language.

   $$L_{\text{RE}} = \{e \mid e \text{ is a valid regular expression over } \{0, 1\}\}$$

   It may help to recall that:

   Definition: Given some finite alphabet Σ, we define e to be a regular expression if e is

   - a for some $a \in \Sigma$
   - $\emptyset$
   - ε
   - $e_0 \cup e_1$, where e_0 and e_1 are regular expressions
   - $e_0 e_1$ where e_0 and e_1 are regular expressions
   - e_0^* where e_0 is a regular expression.
   - (e_0) where e_0 is a regular expression.

3. The grammar for L_{RE} must include rules for the base cases of the definition of regular expressions:

   R → 0
   R → 1
   R → $\emptyset$
   R → ϵ

   together with rules for the recursive steps:

   $R \rightarrow (R)$
   $R \rightarrow RR$
   $R \rightarrow R \cup R$
   $R \rightarrow R^*$

4. Any language that can be described by a context-free grammar is called a context-free language.
5. The examples we have considered raise an interesting question. We know that there are non-regular languages that are context-free languages. We also know that there are regular languages that are context-free. What we don't know is whether there are regular languages that are not context-free.
6. The big question is whether the set of regular languages is a subset of the set of context-free languages.
7. In fact, all regular languages can be described by context-free grammars. The divisible by three example suggests we might go about proving this using the approach outlined below:

Click here to view the slides for this class

- **Proof:** Given a regular langauge L, we know that there is some DFA D such that L = L(D). Given D, we can construct a grammar G with...

  ...

  and clearly the grammar G describes the language L.
- **Proof:** Given a regular langauge L, we know that there is some regular expression e such that L = L(e). Given e, we can construct a grammar G with ...

  ...

  and clearly the grammar G describes the language L.
- To do this, however, we need more precise definitions of what a grammar is and what language it describes.

Formal Grammars

1. Like all good entities in the world of formal languages, a grammar is a tuple. In this case, a 4-tuple.

2. Formally:

   Context-free Grammar A context free grammar is composed of:

   (a) A finite alphabet V of variables (or non-terminals).

   (b) A distinct finite alphabet Σ called the terminals or terminal symbols.

   (c) $S \in V$ referred to as the start symbol.

   (d) A finite set R of pairs composed of one element from V and one element from $(V \cup \Sigma)^*$ called the rules. Rules are usually written in the form:

   $$A \rightarrow X_1 X_2 ... X_m$$

   rather than $(A, X_1 X_2 ... X_m)$.

3. The association between a context free grammar and the language it describes is formalized through the notion of a derivation:

   Yields Given a grammar, G $= (V, \Sigma, S, R)$, and two strings x and y in $(V \cup \Sigma)^*$ such that $x = \alpha A \beta$ and $y = \alpha \gamma \beta$ where $\alpha, \gamma, \beta \in (V \cup \Sigma)^*$ and $(A, \gamma) \in R$ we say that x *yields (or directly derives)* y. In this case we write

   $$x \Longrightarrow y$$

4. Examples of direct derivations.

   Consider $G = (\{B, G\}, \{a, x, z\}, B, \{(B, xGBy), (B, z), (G, aG), (G, \epsilon)\})$ or less formally

   $B \rightarrow$ x G B y
   $B \rightarrow$ z
   $G \rightarrow$ a G
   $G \rightarrow$ ϵ

   The following are examples of the "yields" relation for this grammar:

   - $B \Longrightarrow$ x G B y.
   - x G a $\Longrightarrow$ x a G a

5. More on the notion of a derivation:

   Derivation Given a grammar, G $= (V, \Sigma, S, R)$, and two strings x and y in $(V \cup \Sigma)^*$ we say that x *derives* y if there exists a sequence of string $\alpha_0, \alpha_1, \alpha_2, ..., \alpha_m$ all in $(V \cup \Sigma)^*$ such that

   (a) for all $i < m$, $\alpha_i \Longrightarrow \alpha_{i+1}$,

   (b) $x = \alpha_0$, and

   (c) $y = \alpha_m$.

   In this case we write

   $$x \stackrel{*}{\Longrightarrow} y$$

   The sequence $\alpha_0, \alpha_1, \alpha_2, ..., \alpha_m$ is called a derivation of length m of y from x.

6. Using the grammar G shown above we can say that that $B \stackrel{*}{\Longrightarrow}$ xaxzyy since:

   - $B \Longrightarrow$ x G B y

- x G B y $\implies$ x a G B y
- x a G B y $\implies$ x a B y
- x a B y $\implies$ x a x G B y y
- x a x G B y y $\implies$ x a x B y y
- x a x B y y $\implies$ x a x z y y

7. Time for more definitions:

   Sentential form Given a grammar, $G = (V, \Sigma, S, R)$, a string w is called a *sentential form* of G if $S \overset{*}{\Longrightarrow} w$.

   Sentence A sentential form containing only symbols from the terminal vocabulary of a language is called a *sentence*.

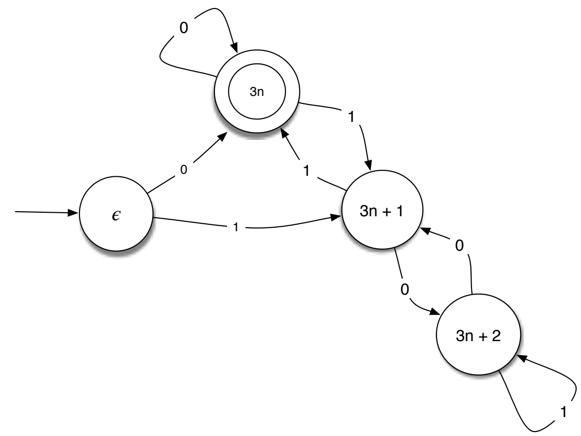
   L(G) The language defined by a grammar G is the set of all sentences.

$$L(G) = \{w \mid S \overset{*}{\Longrightarrow} w \text{ and } w \in \Sigma^*\}$$

8. To make the advantages of this formalism a bit more apparent, consider how it can be used to clarify the arguments one might use to show that all regular languages are context-free.

9. First, to give you an intuitive understanding of how me might convert a DFA that recognizes a language into a context-free grammar for the same language, consider the following example.

   - The example is one of my favorites. Binary strings that represent values that are multiples of 3. You may recall that the following FSA recognized this language:



   - This language can be recognized by a grammar whose variables correspond roughly to the states of the FSA for the language:

     $M \to 0Z \mid 1U$
     $Z \to 0Z \mid 1U \mid \epsilon$
     $U \to 0D \mid 1Z$
     $D \to 0U \mid 1D$

     Here, Z generates suffixes that can be added to a binary string that is divisible by 3 to yield a longer binary string that is divisible by 3. U generates suffixes that can be added to a binary string representing a value that equals 1 mod 3 to turn it into a binary string that is divisible by 3. Similarly, D generates appropriate suffixes for strings of binary digits that equal 2 mod 3.

10. Given a regular language L, we know that for some DFA, $D = (Q, \Sigma, \delta, s, F)$ $L = L(D)$. From D, we can construct a grammar $G = (Q, \Sigma, s, R)$ where $R = \{(q, xq') \mid q' = \delta(q, x)\} \cup \{(q, \epsilon) \mid q \in F\}$ such that $L(G) = L(D) = L$.

    To establish this we must show that $\hat{\delta}(s, w) = q$ in D if and only if $s \overset{*}{\Longrightarrow} wq$ in G. The proof is by induction on the length of w. For the basis, it is clear that for $w = \epsilon$, $\hat{\delta}(s, \epsilon) = s$ and $s \overset{*}{\Longrightarrow} s$ as required.

    Now, we assume the result holds for $|w| < n$ and consider any string wx with $x \in \Sigma$ and $|wx| = n$. If $\hat{\delta}(s, wx) = q$ then by the definition of $\hat{\delta}$, $\delta(\hat{\delta}(s, w), x) = q$. Let $q' = \hat{\delta}(s, w)$. Then, we know that $\delta(q', x) = q$

so by our definition of G, $(q', xq) \in R$. By our inductive assumption, $s \overset{*}{\Longrightarrow} wq'$. Therefore, by adding one step using the rule (q', xq) we can conclude that $s \overset{*}{\Longrightarrow} wxq$. On the other hand, suppose we know that $s \overset{*}{\Longrightarrow} wxq$. Given the way we have defined G, we know that this derivation must end with the application of a rule of the form $(q', xq) \in R$. That is, we can write $s \overset{*}{\Longrightarrow} wq' \Longrightarrow wxq$. Such a rule, however, would only be included if $\delta(q', x) = q$. Moreover, given that $s \overset{*}{\Longrightarrow} wq'$ our inductive assumption implies that $\hat{\delta}(s, w) = q'$. From this, we can conclude that $\hat{\delta}(s, wx) = \delta(\hat{\delta}(s, w), x) = \delta(q', x) = q$ as desired.

Finally, we note that we can extends a derivation of the form $S \overset{*}{\Longrightarrow} wq \Longrightarrow w$ to generate a sentence in G if and only if G contains a rule of the form $(q, \epsilon) \in R$. G contains such a rule for q if and only if $q \in F$ in D. Therefore, $S \overset{*}{\Longrightarrow} wq \Longrightarrow w$ if and only if $w \in L(D)$.

Pushdown Automata

1. We first encountered regular languages as those languages recognized by a class of automata (DFAs) and then encountered a generative notation (regular expressions) that was associated with exactly the same class of languages.

2. With context-free languages, we are taking the opposite approach. First, I presented a generative notation (context-free grammars) that describes these languages. Now, we will examine a type of automaton that recognizes the same set of languages.

3. The new type of automata is not finite! Instead, we will allow for a potentially infinite memory.

   - We will still limit our attention to machines that accept or reject finite input strings.
   - We will restrict access to the machine's memory in a way that makes them more powerful than finite automata, but still limited in computational power.

4. The model we will consider next is called a pushdown automata.

   - The control of a pushdown automata will be a lot like a finite automata, The machine will have a finite set of states and a transition function that specifies the possible moves.
   - It is nondeterministic.
   - The control has final and non-final states and accepts if there is a way that it can be in a final state when the input is exhausted.
   - In addition to its input, the machine will have the ability to read and write symbols on an unbounded stack.
   - The symbol on the top of the stack can help determine the possible transitions from the current state and can be removed from the stack as part of the transition.
   - In addition to a new state, a transition can specify a new symbol that should be pushed onto the stack.
   - The alphabet used on the stack can be distinct from the alphabet of the input language.

5. An example should make all of this clear. Recall the language $\{1^n = 1^n | n \geq 0\}$.

   - This was one of the early examples of a language we showed was not regular.
   - Consider how this language can be accepted by a pushdown automata.
     - The diagram below provides an informal description of a pushdown automaton that recognizes this language.
     - When we draw a state diagram for a PDA, each edge is labeled by a triple "$i, p/s$" where i is the current input symbol, p is the symbol to be popped from the stack while taking the transition (or ϵ if nothing needs to be popped while taking the transition), and s is either a symbol to be pushed on the stack or ϵ.
     - Basically, a pushdown automata can count up by sticking symbols on the stack as it reads one part of the input and count down by popping those symbols later.

– Note that this machine has one feature that is somewhat an artifact of the way Sipser chooses to describe PDAs — namely, his formalism provides no way for the machine to sense if its stack is empty. This will lead most of our machines to include a start state with just one transition that puts a recognizable symbol at the bottom of the stack before processing any input.
– Also note that this trick depends on epsilon-transitions. In particular, for now, all PDAs are non-deterministic.