CSE 322 Lecture Notes
Monday, Feb. 14

Simplifying CFGs

We would like to be able to simplify a grammar, removing *useless symbols* and *annoying productions.*

Annoying productions are things like unit productions $A \rightarrow B$, where A, B are variables, or ϵ-productions, $A \rightarrow \epsilon$ where A is not the start variable. The process for removing these is covered in the textbook, Sipser p. 99. These notes will discuss removing useless symbols, which is not covered in the book.

Suppose you have a grammar $G = (V, \Sigma, P, S)$ where $V = \{S, A, B, C\}$, $\Sigma = \{a, b\}$ and P is the following:

$$\begin{array}{lcl} S & \rightarrow & AB \mid a \\ B & \rightarrow & bB \\ A & \rightarrow & a \mid BC \end{array}$$

What is $L(G)$? By examination, we see that $L(G) = \{a\}$. But perhaps it would be easier to determine this from a simpler grammar. We note that B, C are both useless, as neither derive a terminal string.

Definition: Given a CFG $G = (V, \Sigma, P, S)$, a variable $A \in V$ is *useful* if and only if there exists $w \in \Sigma^*$, and $\alpha, \beta \in (V \cup \Sigma)^*$ such that $S \stackrel{*}{\Rightarrow} \alpha A \beta \stackrel{*}{\Rightarrow} w$.

This means that A participates in some derivation of some string in the language. We observe the following:

(i) We want useful variable X to derive some $w \in \Sigma^*$

(ii) We want useful variable X to be reachable from S.

In order to make G only have useful variables, we want to be sure all variables satisfy both of these properties. Note that 1 and 2 do not guarantee that we only have useful variables. These rules are analogous to insisting that each state in a DFA is in a path from the start state to some final state.

Lemma 1: Given a CFG $G = (V, \Sigma, P, S)$, where $L(G) \neq \phi$, there is an equivalent grammar $G' = (V', \Sigma, P', S)$ such that for every $A \in V', A \stackrel{*}{\Rightarrow} w$, for some $w \in \Sigma^*$. Equivalent grammar means that $L(G) = L(G')$.

Proof: The proof will be constructive. We will build V' and from it we define $P' \subseteq P$, where $P' = \{p \in P \mid \text{p only uses symbols from } (V' \cup \Sigma)^*\}$ We iteratively build V' using the following algorithm:

1. Initialize $V' = \{A \in V \mid A \rightarrow w \text{ is a rule in } P, \text{ and } w \in \Sigma^*\}$.

2. Repeat until there is no change: If $A \rightarrow \alpha$ is a rule in P such that $\alpha \in (V' \cup \Sigma)^*$, then add A to V' (that is, $V' = V' \cup \{A\}$).

Claim: This does step (i) above. We are bootstrapping V', adding to it variables that we know will be able to derive strings of terminals using variables already in V'. For example, on G given above, the algorithm proceeds as follows: We initalize $V' = \{S, A\}$ since both clearly derive a string of terminals. Now we try step two, S never appears on the right side of a rule, and A does not appear alone, so cannot add to V' and the algorithm stops. This gives $G' = (\{S, A\}, \Sigma, \{S \rightarrow a, A \rightarrow a\}, S)$

Also notice that this gives an algorithm for testing $L(G) = \phi$. If $S \notin V'$, then $L(G) = \phi$, since no string of terminals is derivable from S. (The reason we stipulate $L(G) \neq \phi$, is that otherwise this algorithm won't produce a correct grammar, since S isn't in the variable set if the language is empty. However, if $L(G) = \phi$, then $G' = (\{S\}, \Sigma, \phi, S)$ is a suitable grammar for $L(G)$.)

Lemma 2: Given a CFG $G = (V, \Sigma, P, S)$, where $L(G) \neq \phi$, there exists an equivalent CFG, $G' = (V', \Sigma, P', S)$ such that for every $A \in V'$, there exists $\alpha, \beta \in (V' \cup \Sigma)^*$ with $S \stackrel{*}{\Rightarrow} \alpha A \beta$.

Proof: Again we give a construction. We build V' and then define $P' = \{p \in P \mid p \text{ uses symbols from } (V' \cup \Sigma)^*\}$.

1. Intialize $V' = \{S\}$
2. Repeat until no change: For every $A \in V'$, if $A \rightarrow \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_k$ are productions in P, add all variables in $\alpha_1, \alpha_2, \ldots, \alpha_k$ to V'.

This does item (ii) above. From S we add all variables reachable from it, and then variables reachable from those, and so on. In particular, suppose we applied Lemma 2 to the result G' from Lemma 1. Then we intialize $V' = \{S\}$. There are no variables in the strings S produces, so the algorithm stops. Note that $S \rightarrow AB$ was removed by Lemma 1. Thus the result of applying both these lemmas is $G'' = (\{S\}, \Sigma, \{S \rightarrow a\}, S)$ which is what we determined it should be.

Individually, neither lemma will produce a grammar with only useful variables. But applying Lemma 1 and then Lemma 2, does the trick. However, it is important to note that applying Lemma 2 and then Lemma 1 will not necessarily work. On our original grammar, if we apply Lemma 2 first then we get: First $V' = \{S\}$. Then we add A, B to get $V' = \{S, A, B\}$. Repeating once more we get $V' = \{S, A, B, C\}$, and therefore, we get back our original grammar. Then applying lemma 1, gives the grammar G' as above, which still contains the useless production $A \rightarrow a$.