Normal Forms

CS154
Chris Pollett
Outline

• s-grammars
• Methods for transforming grammars
• Chomsky Normal Form
s-grammars

- Last day we gave a brute force algorithm for checking if a string could be generated by a CFG.
- It had a worst case exponential run-time.
- We would like parsing algorithms which run in linear time.
- One way to achieve this is to restrict the kind of grammars we consider:

**Definition.** A context-free grammar $G=(V, T, S, P)$ is said to be a simple grammar or s-grammar if all its productions are of the form:

\[ A \rightarrow ax \]

where $A$ is in $V$, $a$ is in $T$, $x$ is in $V^*$, and any pair $(A,a)$ occurs at most once in $P$.

- For example, $S \rightarrow aSbSSc$ is an s-grammar, but $S \rightarrow aSbSSlaSSlc$ is not because $(S,a)$ occurs in $S \rightarrow aS$ and $S \rightarrow aSS$.
- Our brute force parsing algorithm will run in linear time with an s-grammar since at any given step there is at most one production which can be used. Further since the right hand side of a production always starts with a terminal we match at least one character of the input with each substitution.
- So after at most linearly many substitution we know if the string is in the language.
- s-grammars tend to be too restrictive to specify practical programming languages; nevertheless, they show the form of the rule is important to get efficient parsers.
Methods for Transforming Grammars

• We are now going to work towards some normal forms which will be useful in obtaining parsing algorithms for general CFGs.

• To do this we will look at different ways to simplify our grammars.

• To start sometimes it is useful to get rid of the empty string from our language in order to make our proofs easier. It turns out this won’t cause a loss of generality in the statements we can say about CFGs.

• To see this, suppose $L$ is a language and let $L' = L - \{ \lambda \}$.

• If $G'=(V,T,S,P)$ is a CFG for $L'$, then $G = (V,T,S_0,P \cup \{ S_0 \rightarrow S \mid \lambda \} )$ will be a CFG for $L$.

• So we will for now restrict our attention to grammars without $\lambda$. 
More Methods of Transforming Grammars

- Suppose we have a CFG \( G=(V,T,S,P) \) and let \( A \rightarrow x_1Bx_2 \) be in \( P \). Suppose the variable \( B \) occurs in the following productions in \( G \): \( B \rightarrow y_1 \mid y_2 \mid \ldots \mid y_n \). Then if \( G' \) is the CFG obtained by replacing \( A \rightarrow x_1Bx_2 \) by \( A \rightarrow x_1y_1x_2 | x_1y_2x_2 | \ldots | x_1y_nx_2 \), we will have \( L(G')=L(G) \).

**Definition.** Let \( G=(V,T,S,P) \) be a CFG. A variable \( A \) in \( V \) is said to be **useful** iff there is at least one \( w \) in \( L(G) \) such that \( S \Rightarrow^* xAy \Rightarrow^* w \). Otherwise, \( A \) is said to be **useless**. A production is useless if it involves any useless variables.

- For example, \( S \rightarrow A, A \rightarrow aA | \lambda, B \rightarrow bA \). Then \( B \) is useless as it is not reachable from the start variable. So the production \( B \rightarrow bA \) is useless.

- Given a CFG if we eliminate all its useless productions we still get a smaller CFG with the same language.

- To determine the useful variables and productions we can start with \( V_1 = \text{empty set} \). Then repeat the following until there are no more variables added to \( V_1 \): For each production \( A \rightarrow x_1 \ldots x_n \), with all \( x_i \)'s that are variables in \( V_1 \), add \( A \) to \( V_1 \). If the start variable is not in \( V_1 \) then we know the language is empty, so we can delete all productions.

- Otherwise, if \( S \) is in \( V_1 \), it still might not be the case that every variable in \( V_1 \) is useful, so we set \( V_2 = \{S\} \). Then repeat the following until there are no more variables added to \( V_2 \): For each production \( A \rightarrow x_1 \ldots x_n \), with all \( x_i \)'s that are variables in \( V_1 \) and with add \( A \) to \( V_2 \), add each variable on the left hand side to \( V_2 \). After this procedure terminates, take \( V_2 \) to be the set of useful variables. All other variables and production they are involved in are useless.
Removing $\lambda$-rules/productions

- A production (rule) of a CFG of the form $A --> \lambda$ is called a $\lambda$-production or $\lambda$-rule. Any variable for which $A \Rightarrow^* \lambda$, is called nullable.
- Even though a CFG might generate a language not containing $\lambda$, it still might have nullable productions. In which case these productions can be removed.
- For example, in $S --> aCb, C --> aCbl \lambda$, the variable $C$ is nullable. We can eliminate the $\lambda$-rule by doing substitutions to get: $S --> aCblab, C --> aCblab$.
- To find the set $N$ of nullable variables of a CFG, we can first put all variables $A$ which occur in productions of the form $A --> \lambda$ into $N$. Then repeat until no new variables are added the following step: if $B$ occurs in a production $B --> A_1A_2..A_n$ where each $A_i$ is in $N$, then add $B$ to $N$.
- Once we have the set of nullable variables, we can eliminate any $\lambda$-rules from our grammar and for each rule $C --> C_1C_2..C_n$ where a nullable variables occur add a rule with each possible substitution of a nullable variable by $\lambda$. 
Eliminate Unit Productions

- A **unit production** is a production of the form $A \rightarrow B$.
- In general, using the reachability algorithm we can determine if $A \Rightarrow^\ast C$ for any two variables $A$ and $B$.
- If $C$ occurs in the rules $C \rightarrow y_1 \mid y_2 \mid \ldots \mid y_n$, then we can add the rule $A \rightarrow y_1 \mid y_2 \mid \ldots \mid y_n$ to our grammar without effecting the strings it generates. If we do this for all variables involved on the right hand side of a unit rule and for each $C$ for which $A \Rightarrow^\ast C$, then we have eliminated unit rules.
Chomsky Normal Form

- To get an efficient parsing algorithm for general CFGs it is convenient to have them in some kind of normal form.
- Chomsky Normal Form is often used.
- A CFG is in **Chomsky Normal Form** if every rule is of the form $A \rightarrow BC$ or of the form $A \rightarrow a$, where $A, B, C$ are any variables and $a$ is a terminal. In addition the rule $S \rightarrow \lambda$ is permitted.