CS 4510 Automata and Complexity February 21

Lecture 8

Lecturer: Frederic Faulkner *Scribe(s):*

1 Review

Previously, we introduced the notion of a *context-free grammar*, which is basically a set of replacement rules that can be applied to the *non-terminals* to eventually produce some string of *terminals*. All strings of terminals produced by starting with the *starting non-terminal* constitute the langauge of the grammar. For example, the following grammar produces all strings of the form $0^m 1^n$ where $m > n$:

$$S \to 0S1$$
$$S \to 0S$$
$$S \to 0$$

This grammar has three rules, a single non-terminal, and two terminals (0 and 1).

2 Closure

Theorem 1 *Let L_1 and L_2 be context-free languages. Then $L_1 \cup L_2$, $L_1 \circ L_2$, and L_1^* are also context-free.*

Proof. Let G_1 and G_2 be grammars for L_1 and L_2 with starting symbols S_1 and S_2 respectively. Then we can create a grammar G with a new start symbol S, and all of the rules, terminals, and non-terminals of G_1 and G_2. If we add the rules $S \to S_1 | S_2$, G will generate $L_1 \cup L_2$; on the other hand, if we add the rule $S \to S_1 S_2$, we will generate $L_1 \circ L_2$. The proof for L_1^* is left as an exercise for the reader.

3 Chomsky Normal Form

Often when dealing with grammars in the real world it is convenient to have the grammar in some standardized form where all the rules behave similarly. The most popular standardized form for a grammar is *Chomsky Normal Form*.

Definition 2 *A grammar is in Chomsky Normal Form (CNF) if all of its rules have exactly one of three forms:*

- $A \to BC$
- $A \to a$

- $S \to \varepsilon$

where A, B, C, S are nonterminals, S is the start terminal, B and C are NOT the start terminal, and a is a terminal.

Why is Chomsky Normal Form useful? Well, note that every rule (other than $S \to \varepsilon$) either increases the number of nonterminals in the string by 1, or decreases the number of non-terminals by 1 while increasing the number of terminals by 1. This makes it easier for us to reason about the effects of applying the rules of this grammar. For example, if G is a Chomsky Normal Form grammar that produces the (non-empty) string x, then it does so in exactly $2|x| - 1$ steps.

It would be nice to know which context-free languages are generated by CNF grammars. Fortunately, we have a theorem for that.

Theorem 3 *Every context-free language is generated by a Chomsky Normal Form grammar.*

Proof. We show how to convert an arbitrary context-free non-CNF grammar G to an equivalent grammar G' that is CNF.

First, we must make sure that the start variable is never on the right hand side of a rule. We do this by setting a new start variable S_0 for G', and adding the rule $S_0 \to S$, where S was the old start state.

Next, we remove all the rules of the form $A \to \varepsilon$ unless A is the start variable. Once we remove a rule like so, we must update all the rules that produce A on the right hand side, so that for each rule of the form $B \to wAv$ we update it to $B \to wAv \mid wv$ unless doing so would add a rule $B \to \varepsilon$ that we already removed. (w, v are strings of terminals and/or non-terminals).

Similarly, we remove all rules of the form $A \to C$. Then for each rule of the form $C \to w$, we add the rule $A \to w$ unless this adds in a rule $A \to B$ that was already removed. (Again, w is a string of terminals and/or non-terminals.)

Next, if we have any rules of the form $A \to x_1 x_2 ... x_k$, we can replace these by rules of the form $A \to x_1 A_1$, $A_1 \to x_2 A_2$, ... $A_{k-2} \to x_{k-1} x_k$. (Each A_i is a new non-terminal, and each x_i is a single terminal or non-terminal.)

Finally, in the previous step, if any of the x_i are terminals, we replace them by a new nonterminal U_i and add the rule $U_i \to x_i$.

Let's see an example of how to apply these rules to convert a grammar into Chomsky Normal Form. We begin with the following grammar:

$$S \to BSBq \mid Aa$$
$$A \to B \mid aa \mid \varepsilon$$
$$B \to b \mid c \mid \varepsilon$$

We see that S appears on the right hand side, which is not allowed. Following the algorithm above, we fix this by adding a new start terminal S_0, and modifying the grammar as follows:

$$
\begin{aligned}
S_0 &\to S \\
S &\to BSBq \mid Aa \\
A &\to B \mid aa \mid \varepsilon \\
B &\to b \mid c \mid \varepsilon
\end{aligned}
$$

The next step, accoring to the algorithm above, is to handle the rules $A \to \varepsilon$ and $B \to \varepsilon$, which are not allowed. We remove $A \to \varepsilon$ and upate the one rule containing A on the right hand side:

$$
\begin{aligned}
S_0 &\to S \\
S &\to BSBq \mid Aa \mid a \\
A &\to B \mid aa \\
B &\to b \mid c \mid \varepsilon
\end{aligned}
$$

Now we remove $B \to \varepsilon$, and update accordingly. Note that in the rule $S \to BSBq$, we must account for the fact that either or both of the B's could have become ε:

$$
\begin{aligned}
S_0 &\to S \\
S &\to BSBq \mid BSq \mid SBq \mid Sq \mid Aa \mid a \\
A &\to B \mid aa \mid \varepsilon \\
B &\to b \mid c
\end{aligned}
$$

Note that the red ε is NOT added back, because we have already deleted the rule $A \to \varepsilon$.

Now that the ε's have been removed, we handle the cases where some rules are too short. We must remove $A \to B$:

$$
\begin{aligned}
S_0 &\to S \\
S &\to BSBq \mid BSq \mid SBq \mid Sq \mid Aa \mid a \\
A &\to aa \mid b \mid c \\
B &\to b \mid c
\end{aligned}
$$

and $S_0 \to S$:

$$
\begin{aligned}
S_0 &\to BSBq \mid BSq \mid SBq \mid Sq \mid Aa \mid a \\
S &\to BSBq \mid BSq \mid SBq \mid Sq \mid Aa \mid a \\
A &\to aa \mid b \mid c \\
B &\to b \mid c
\end{aligned}
$$

Now, we handle all rules whose right-hand sides are too long. Note that it will take multiple iterations of this step to handle $BSBq$, since each iteration creates a new rule whose right-hand side is only one character shorter than the previous rule.

After one iteration:

$$\begin{aligned}
S_0 &\to BU_1 \mid BU_2 \mid SU_3 \mid Sq \mid Aa \mid a \\
S &\to BU_4 \mid BU_5 \mid SU_6 \mid Sq \mid Aa \mid a \\
U_1 &\to SBq \\
U_2 &\to Sq \\
U_3 &\to Bq \\
U_4 &\to SBq \\
U_5 &\to Sq \\
U_6 &\to Bq \\
A &\to aa \mid b \mid c \\
B &\to b \mid c
\end{aligned}$$

Note that we could have reused some of the U_i, but we are trying to implement the above algorithm exactly, rather than find an efficient transformation for this particular example.

After one more iteration of shortening, all of the rules have right-hand sides consisting of a single terminal, or exactly two characters in any combination of terminal or nonterminal:

$$\begin{aligned}
S_0 &\to BU_1 \mid BU_2 \mid SU_3 \mid Sq \mid Aa \mid a \\
S &\to BU_4 \mid BU_5 \mid SU_6 \mid Sq \mid Aa \mid a \\
U_1 &\to SU_7 \\
U_2 &\to Sq \\
U_3 &\to Bq \\
U_4 &\to SU_8 \\
U_5 &\to Sq \\
U_6 &\to Bq \\
U_7 &\to Bq \\
U_8 &\to Bq \\
A &\to aa \mid b \mid c \\
B &\to b \mid c
\end{aligned}$$

Finally, we must handle the case where any length-2 right-hand sides contain terminals, by making the new nonterminals U_a and U_q to replace a and q where needed.

$$\begin{aligned}
S_0 &\rightarrow BU_1 \mid BU_2 \mid SU_3 \mid SU_q \mid AU_a \mid a\\
S &\rightarrow BU_4 \mid BU_5 \mid SU_6 \mid SU_q \mid AU_a \mid a\\
U_1 &\rightarrow SU_7\\
U_2 &\rightarrow SU_q\\
U_3 &\rightarrow BU_q\\
U_4 &\rightarrow SU_8\\
U_5 &\rightarrow SU_q\\
U_6 &\rightarrow BU_q\\
U_7 &\rightarrow BU_q\\
U_8 &\rightarrow BU_q\\
A &\rightarrow U_aU_a \mid b \mid c\\
B &\rightarrow b \mid c\\
U_a &\rightarrow a\\
U_q &\rightarrow q
\end{aligned}$$

We have now converted our original grammar into Chomsky Normal Form. Note that we have not ended up with the most efficient form of this grammar (for example, we could at least combine U_3, U_6, U_7 and U_8 into a single nonterminal).