CS 373: Theory of Computation
Sariel Har-Peled and Madhusudan Parthasarathy

Lecture 12: Cleaning up CFGs and Chomsky Normal form

3 March 2009

In this lecture, we are interested in transforming a given grammar into a cleaner form. We start by describing how to clean up a grammar. Then, we show how to transform a cleaned up grammar into a grammar in Chomsky Normal Form.

1 Cleaning up a context-free grammar

The main problem with a general context-free grammar is that it might be complicated, contained parts that can not be used, and not necessarily effective.

It might be useful to think about CFG as a program, that we would like to manipulate (in a similar way that a compiler handles programs). If want to cleanup a program, we need to understand its structure. To this end, for CFGs, we will show a sequence of algorithms that analyze and cleanup a given CFG. Interestingly, these procedures uses similar techniques to the one used by compilers to manipulate programs being compiled.

Note, that some of the cleanup steps are not necessary if one just wants to transform a grammar into Chomsky Normal Form. In particular, Section 1.2 and Section 1.3 are not necessary for the CNF conversion. Note however, that the algorithm of Section 1.3 gives us am immediate way to decide if a grammar is empty or not, see Theorem 1.2.

1.1 Example of a messy grammar

For example, consider the following strange very strange grammar.

$$
(G1)\quad \boxed{\begin{array}{ll}
\Rightarrow & S_0 \rightarrow S \mid X \mid Z \\
& S \rightarrow A \\
& A \rightarrow B \\
& B \rightarrow C \\
& C \rightarrow Aa \\
& X \rightarrow C \\
& Y \rightarrow aY \mid a \\
& Z \rightarrow \epsilon .
\end{array}}
$$

This grammar is bad. How bad? Well, it has several problems:

(i) The variable Y can never be derived by the start symbol S_0. It is a ***useless*** variable.

(ii) The rule $\mathsf{S} \to \mathsf{A}$ is redundant. We can replace any appearance of S by A, and reducing the number of variables by one. Rule of the form $\mathsf{S} \to \mathsf{A}$ is called a ***unit production*** (or ***unit rule***.

(iii) The variable A is also ***useless*** since we can note derive any word in Σ^* from A (because once we starting deriving from A we get into an infinite loop).

(iv) We also do not like Z, since one can generate ϵ from it (that is $\mathsf{Z} \overset{*}{\Rightarrow} \epsilon$. Such a variable is called ***nullable***. We would like to have the property that only the start variable can be derived to ϵ.

We are going to present a sequence of algorithms that transform the grammar to not have these drawbacks.

1.2 Removing useless variables unreachable from the start symbol

Given a grammar $\mathcal{G} = (\mathcal{V}, \Sigma, \mathcal{R}, \mathsf{S})$, we would like to remove all variables that are not derivable from S. To this end, consider the following algorithm.

```
compReachableVars $\left(\mathcal{G} = (\mathcal{V}, \Sigma, \mathcal{R}, \mathsf{S})\right)$
        $V_{old} \leftarrow \emptyset$
        $V_{new} \leftarrow \{\mathsf{S}\}$
        while $V_{old} \neq V_{\mathrm{new}}$ do
            $V_{\mathrm{old}} \leftarrow V_{\mathrm{new}}$.
            for  $\mathsf{X} \in V_{old}$ do
                for  $(\mathsf{X} \to w) \in \mathcal{R}$ do
                    Add all variables appearing in $w$ to $V_{\mathrm{new}}$.
        return $V_{\mathrm{new}}$.
```

Clearly, this algorithm returns all the variables that are derivable form the start symbol S. As such, settings $\mathcal{V}' = \mathbf{compReachableVars}(\mathcal{G})$ we can set our new grammar to be $\mathcal{G}' = (\mathcal{V}', \Sigma, \mathcal{R}', \mathsf{S})$, where $\mathcal{R}'$ is the set of rules of $\mathcal{R}$ having only variables in $\mathcal{V}'$.

1.3 Removing useless variables that do not generate anything

In the next step we remove variables that do not generate any string.

Given a grammar $\mathcal{G} = (\mathcal{V}, \Sigma, \mathcal{R}, \mathsf{S})$, we would like to remove all variables that are not derivable from S. To this end, consider the following algorithm.

```
compGenerating Vars $\left(\mathcal{G} = (\mathcal{V}, \Sigma, \mathcal{R}, \mathsf{S})\right)$
    $V_{\text{old}} \leftarrow \emptyset$
    $V_{\text{new}} \leftarrow V_{\text{old}}$
    do
        $V_{\text{old}} \leftarrow V_{\text{new}}$.
        for  $\mathsf{X} \in \mathcal{V}$ do
            for  $(\mathsf{X} \rightarrow w) \in \mathcal{R}$ do
                if $w \in (\Sigma \cup V_{\text{old}})^*$ then
                    $V_{\text{new}} \leftarrow V_{\text{new}} \cup \{\mathsf{X}\}$
    while $(V_{old} \neq V_{\text{new}})$
    return $V_{\text{new}}$.
```

As such, settings $\mathcal{V}' =$ **compReachableVars**$(\mathcal{G})$ we can set our new grammar to be $\mathcal{G}' = (\mathcal{V}', \Sigma, \mathcal{R}', \mathsf{S})$, where $\mathcal{R}'$ is the result of removing all rules that uses variables not in $\mathcal{V}'$. In the new grammar, every variable can derive some string.

Lemma 1.1 *Given a context-free grammar (CFG) $\mathcal{G} = (\mathcal{V}, \Sigma, \mathcal{R}, \mathsf{S})$ one can compute an equivalent CFG $\mathcal{G}'$ such that any variable of $\mathcal{G}'$ can derive some string in Σ^*.*

Note, that if a grammar $\mathcal{G}$ has an empty language, then the equivalent grammar generated by Lemma 1.1 will have no variables in it. Namely, given a grammar we have an algorithm to decide if the language it generates is empty or not.

Theorem 1.2 (CFG emptiness.) *Given a CFG $\mathcal{G}$, there is an algorithm that decides if the language of $\mathcal{G}$ is empty or not.*

Applying the algorithm of Section 1.2 together with the algorithm of Lemma 1.1 results in a CFG without any useless variables.

Lemma 1.3 *Given a CFG one can compute an equivalent CFG without any useless variables.*

2 Removing ϵ-productions and unit rules from a grammar

Next, we would like to remove ***ϵ-production*** (i.e., a rule of the form $\mathsf{X} \rightarrow \epsilon$) and ***unit-rules*** (i.e., a rule of the form $\mathsf{X} \rightarrow \mathsf{Y}$) from the language. This is somewhat subtle, and one needs to be careful in doing this removal process.

2.1 Discovering nullable variables

Given a grammar $\mathcal{G} = (\mathcal{V}, \Sigma, \mathcal{R}, \mathsf{S})$, we are interested in discovering all the nullable variables. A variable $\mathsf{X} \in \mathcal{V}$ is ***nullable***, if there is a way derive the empty string from X in $\mathcal{G}$. This can be done with the following algorithm.

```
compNullableVars $\left(\mathcal{G} = (\mathcal{V}, \Sigma, \mathcal{R}, \mathsf{S})\right)$
    $V_{\text{null}} \leftarrow \emptyset$
    do
        $V_{\text{old}} \leftarrow V_{\text{null}}$.
        for  $\mathsf{X} \in \mathcal{V}$ do
            for  $(\mathsf{X} \rightarrow w) \in \mathcal{R}$ do
                if $w = \epsilon$  or  $w \in (V_{\text{null}})^*$ then
                    $V_{\text{null}} \leftarrow V_{\text{null}} \cup \{\mathsf{X}\}$
    while $(V_{\text{null}} \neq V_{\text{old}})$
    return $V_{\text{null}}$.
```

2.2 Removing ϵ-productions

A rule is an ***ϵ-production*** if it is of the form $VX \rightarrow \epsilon$. We would like to remove all such rules from the grammar (or almost all of them).

To this end, we run **compNullableVars** on the given grammar $\mathcal{G} = (\mathcal{V}, \Sigma, \mathcal{R}, \mathsf{S})$, and get the set of all nullable variable V_{null}. If the start variable is nullable (i.e., $\mathsf{S} \in V_{\text{null}}$), then we create a new start state S', and add the rules to the grammar

$$\mathsf{S}' \rightarrow \mathsf{S} \mid \epsilon.$$

We also now remove all the other rules of the form $\mathsf{X} \rightarrow \epsilon$ from $\mathcal{R}$. Let $\mathcal{G}' = (\mathcal{V}', \Sigma, \mathcal{R}', \mathsf{S}')$ be the resulting grammar. The grammar $\mathcal{G}'$ is not equivalent to the original rules, since we missed some possible productions. For example, if we had the rule

$$\mathsf{X} \rightarrow \mathsf{ABC},$$

where B is nullable, then since B is no longer nullable (we removed all the ϵ-productions form the language), we missed the possibility that $\mathsf{B} \overset{*}{\Rightarrow} \epsilon$. To compensate for that, we need to add back the rule

$$\mathsf{X} \rightarrow \mathsf{AC},$$

to the set of rules.

So, for every rule $\mathsf{A} \rightarrow X_1 X_2 \ldots X_m$ is in $\mathcal{R}'$, we add the rules of the form $\mathsf{A} \rightarrow \alpha_1 \ldots \alpha_m$ to the grammar, where

(i) If X_i is not nullable (its a character or a non-nullable variable), then $\alpha_i = X_i$.

(ii) If X_i is nullable, then α_i is either X_i or ϵ.

(iii) Not all α_is are ϵ.

Let $\mathcal{G}'' = (\mathcal{V}, \Sigma, \mathcal{R}', \mathsf{S}')$ be the resulting grammar. Clearly, no variable is nullable, except maybe the start variable, and there are no ϵ-production rules (except, again, for the special rule for the start variable).

Note, that we might need to feed $\mathcal{G}''$ into our procedures to remove useless variables. Since this process does not introduce new rules or variables, we have to do it only once.

3 Removing unit rules

A ***unit rule*** is a rule of the form $\mathsf{X} \rightarrow \mathsf{Z}$. We would like to remove all such rules from a given grammar.

3.1 Discovering all unit pairs

We have a grammar $\mathcal{G} = (\mathcal{V}, \Sigma, \mathcal{R}, \mathsf{S})$ that has no useless variables or ϵ-predictions. We would like to figure out all the unit pairs. A pair of variables Y and X is a ***unit pair*** if $\mathsf{X} \overset{*}{\Rightarrow} \mathsf{Y}$ by $\mathcal{G}$. We will first compute all such pairs, and their we will remove all unit

Since there are no ϵ transitions in $\mathcal{G}$, the only way for $\mathcal{G}$ to derive Y from X, is to have a sequence of rules of the form

$$\mathsf{X} \rightarrow \mathsf{Z}_1, \mathsf{Z}_1 \rightarrow \mathsf{Z}_2, \ldots, \mathsf{Z}_{k-1} \rightarrow \mathsf{Z}_z = \mathsf{Y},$$

where all these rules are in $\mathcal{R}$. We will generate all possible such pairs, by generating explicitly the rules of the form $\mathsf{X} \rightarrow \mathsf{Y}$ they induce.

```
compUnitPairs $\left(\mathcal{G} = (\mathcal{V}, \Sigma, \mathcal{R}, \mathsf{S})\right)$
    $R_{\text{new}} \leftarrow \left\{ \mathsf{X} \rightarrow \mathsf{Y} \;\middle|\; (\mathsf{X} \rightarrow \mathsf{Y}) \in \mathcal{R} \right\}$
    do
        $R_{\text{old}} \leftarrow R_{\text{new}}$.
        for $(\mathsf{X} \rightarrow \mathsf{Y}) \in R_{\text{new}}$ do
            for $(\mathsf{Y} \rightarrow \mathsf{Z}) \in R_{\text{new}}$ do
                $R_{\text{new}} \leftarrow R_{\text{new}} \cup \{\mathsf{X} \rightarrow \mathsf{Z}\}$.
    while $(R_{\text{new}} \neq R_{\text{old}})$
    return $R_{\text{new}}$.
```

3.2 Removing unit rules

If we have a rule $\mathsf{X} \rightarrow \mathsf{Y}$, and $\mathsf{Y} \rightarrow w$, then if we want to remove the unit rule $\mathsf{X} \rightarrow \mathsf{Y}$, then we need to introduce the new rule $\mathsf{X} \rightarrow w$. We want to do that for all possible unit pairs.

```
removeUnitRules $\left(\mathcal{G} = (\mathcal{V}, \Sigma, \mathcal{R}, \mathsf{S})\right)$
    $U \leftarrow$ compUnitPairs$(\mathcal{G})$
    $\mathcal{R} \leftarrow \mathcal{R} \setminus U$
    for $(\mathsf{X} \rightarrow \mathsf{A}) \in U$ do
        for $(\mathsf{A} \rightarrow w) \in R_{\text{old}}$ do
            $\mathcal{R} \leftarrow \mathcal{R} \cup \{\mathsf{X} \rightarrow w\}$.
    return $(\mathcal{V}, \Sigma, \mathcal{R}, \mathsf{S})$.
```

We thus established the following result.

Theorem 3.1 *Given an arbitrary CFG, one can compute an equivalent grammar $\mathcal{G}'$, such that $\mathcal{G}'$ has no unit rules, no ϵ-productions (except maybe a single ϵ-production for the start variable), and no useless variables.*

4 Chomsky Normal Form

Chomsky Normal Form requires that each rule in the grammar is either

(C1) of the form $\mathsf{A} \rightarrow \mathsf{BC}$, where A, B, C are all variables and neither B nor C is the start variable.

(That is, a rule has exactly two variables on its right side.)

(C2) $\mathsf{A} \rightarrow \mathsf{a}$, where A is a variable and a is a terminal.

(A rule with terminals on its right side, has only a single character.)

(C3) $\mathsf{S} \rightarrow \epsilon$, where S is the start symbol.

(The start variable can derive ϵ, but this is the only variable that can do so.)

Note, that rules of the form $\mathsf{A} \rightarrow \mathsf{B}$, $\mathsf{A} \rightarrow \mathsf{BCD}$ or $\mathsf{A} \rightarrow \mathsf{aC}$ are all illegal in a CNF.

Also a grammar in CNF never has the start variable on the right side of a rule.

Why should we care for CNF? Well, its an effective grammar, in the sense that every variable that being expanded (being a node in a parse tree), is guaranteed to generate a letter in the final string. As such, a word w of length n, must be generated by a parse tree that has $O(n)$ nodes. This is of course not necessarily true with general grammars that might have huge trees, with little strings generated by them.

4.1 Outline of conversion algorithm

All context-free grammars can be converted to CNF. We did most of the steps already. Here is an outline of the procedure:

(i) Create a new start symbol S_0, with new rule $\mathsf{S}_0 \rightarrow \mathsf{S}$ mapping it to old start symbol (i.e., S).

(ii) Remove nullable variables (i.e., variables that can generate the empty string).

(iii) Remove unit rules (i.e., variables that can generate each other).

(iv) Restructure rules with long righthand sides.

The only step we did not describe yet is the last one.

4.2 Final restructuring of a grammar into CNF

Assume that we already cleaned up a grammar by applying the algorithm of Theorem 3.1 to it. So, we now want to convert this grammar $\mathcal{G} = (\mathcal{V}, \Sigma, \mathcal{R}, \mathsf{S})$ into CNF.

Removing characters from right side of rules. As a first step, we introduce a variable V_c for every character $c \in \Sigma$ and it to $\mathcal{V}$. Next, we add the rules $\mathsf{V}_c \rightarrow c$ to the grammar, for every $c \in \Sigma$.

Now, for any string $w \in (\mathcal{V} \cup \Sigma)^*$, let $\widehat{w}$ denote the string, such that any appearance of a character c in w, is replaced by V_c.

Now, we replace every rule $\mathsf{X} \rightarrow w$, such that $|w| > 1$, by the rule $\mathsf{X} \rightarrow \widehat{w}$.

Clearly, (C2) and (C3) hold for the resulting grammar, and furthermore, any rule having variables on the right side, is made only of variables.

Making rules with only two variables on the right side. The only remaining problem, is that in the current grammar, we might have rules that are too long, since they have long string on the right side. For example, we might have a rule in the grammar of the form

$$\mathsf{X} \rightarrow \mathsf{B}_1\mathsf{B}_3 \ldots \mathsf{B}_k.$$

To make this into a binary rule (with only two variables on the right side, we remove this rule from the grammar, and replace it by the following set of rules

$$\begin{aligned} &\mathsf{X} \rightarrow \mathsf{B}_1\mathsf{Z}_1 \\ &\mathsf{Z}_1 \rightarrow \mathsf{B}_2\mathsf{Z}_2 \qquad\qquad \mathsf{Z}_2 \rightarrow \mathsf{B}_3\mathsf{Z}_3 \\ \ldots& \\ &\mathsf{Z}_{k-3} \rightarrow \mathsf{B}_{k-2}\mathsf{Z}_{k-2} \\ &\mathsf{Z}_{k-2} \rightarrow \mathsf{B}_{k-1}\mathsf{B}_k, \end{aligned}$$

where $\mathsf{Z}_1, \ldots, \mathsf{Z}_{k-2}$ are new variables.

We repeat this process, till all rules in the grammar is binary. This gramamr is now in CNF. We summarize our result.

Theorem 4.1 (CFG → CNF.) *Any context-free grammar can be converted into Chomsky normal form.*

4.3 An example of converting a CFG into CNF

Let us look at an example grammar with start symbol S.

$$\text{(G0)} \qquad \boxed{\begin{array}{ll} \Rightarrow & \mathsf{S} \rightarrow \mathsf{ASA} \mid \mathtt{a}\mathsf{B} \\ & \mathsf{A} \rightarrow \mathsf{B} \mid \mathsf{S} \\ & \mathsf{B} \rightarrow \mathtt{b} \mid \epsilon \end{array}}$$

After adding the new start symbol S_0, we get the following grammar.

$$\text{(G1)} \qquad \boxed{\begin{array}{ll} \Rightarrow & \mathsf{S}_0 \rightarrow \mathsf{S} \\ & \mathsf{S} \rightarrow \mathsf{ASA} \mid \mathtt{a}\mathsf{B} \\ & \mathsf{A} \rightarrow \mathsf{B} \mid \mathsf{S} \\ & \mathsf{B} \rightarrow \mathtt{b} \mid \epsilon \end{array}}$$

Removing nullable variables In the above grammar, both A and B are the nullable variables. We have the rule $\mathsf{S} \rightarrow \mathsf{ASA}$. Since A is nullable, we need to add $\mathsf{S} \rightarrow \mathsf{SA}$ and $\mathsf{S} \rightarrow \mathsf{AS}$ and $\mathsf{S} \rightarrow \mathsf{S}$ (which is of course a silly rule, so we will not waste our time putting it in). We also have $\mathsf{S} \rightarrow \mathsf{aB}$. Since B is nullable, we need to add $\mathsf{S} \rightarrow \mathsf{a}$. The resulting grammar is the following.

$$\text{(G2)} \quad \boxed{\begin{array}{ll} \Rightarrow & \mathsf{S_0} \rightarrow \mathsf{S} \\ & \mathsf{S} \rightarrow \mathsf{ASA} \mid \mathsf{aB} \mid \mathsf{a} \mid \mathsf{SA} \mid \mathsf{AS} \\ & \mathsf{A} \rightarrow \mathsf{B} \mid \mathsf{S} \\ & \mathsf{B} \rightarrow \mathsf{b} \end{array}}$$

Removing unit rules. The unit pairs for this grammar are $\{\mathsf{A} \rightarrow \mathsf{B}, \mathsf{A} \rightarrow \mathsf{S}, \mathsf{S_0} \rightarrow \mathsf{S}\}$. We need to copy the productions for S up to $\mathsf{S_0}$, copying the productions for S down to A, and copying the production $\mathsf{B} \rightarrow \mathsf{b}$ to $\mathsf{A} \rightarrow \mathsf{b}$.

$$\text{(G3)} \quad \boxed{\begin{array}{ll} \Rightarrow & \mathsf{S_0} \rightarrow \mathsf{ASA} \mid \mathsf{aB} \mid \mathsf{a} \mid \mathsf{SA} \mid \mathsf{AS} \\ & \mathsf{S} \rightarrow \mathsf{ASA} \mid \mathsf{aB} \mid \mathsf{a} \mid \mathsf{SA} \mid \mathsf{AS} \\ & \mathsf{A} \rightarrow \mathsf{b} \mid \mathsf{ASA} \mid \mathsf{aB} \mid \mathsf{a} \mid \mathsf{SA} \mid \mathsf{AS} \\ & \mathsf{B} \rightarrow \mathsf{b} \end{array}}$$

Final restructuring. Now, we can directly patch any places where our grammar rules have the wrong form for CNF. First, if the rule has at least two symbols on its righthand side but some of them are terminals, we introduce new variables which expand into these terminals. For our example, the offending rules are $\mathsf{S_0} \rightarrow \mathsf{aB}$, $\mathsf{S} \rightarrow \mathsf{aB}$, and $\mathsf{A} \rightarrow \mathsf{aB}$. We can fix these by replacing the a's with a new variable U, and adding a rule $\mathsf{U} \rightarrow \mathsf{a}$.

$$\text{(G4)} \quad \boxed{\begin{array}{ll} \Rightarrow & \mathsf{S_0} \rightarrow \mathsf{ASA} \mid \mathsf{UB} \mid \mathsf{a} \mid \mathsf{SA} \mid \mathsf{AS} \\ & \mathsf{S} \rightarrow \mathsf{ASA} \mid \mathsf{UB} \mid \mathsf{a} \mid \mathsf{SA} \mid \mathsf{AS} \\ & \mathsf{A} \rightarrow \mathsf{b} \mid \mathsf{ASA} \mid \mathsf{UB} \mid \mathsf{a} \mid \mathsf{SA} \mid \mathsf{AS} \\ & \mathsf{B} \rightarrow \mathsf{b} \\ & \mathsf{U} \rightarrow \mathsf{a} \end{array}}$$

Then, if any rules have more than two variables on their righthand side, we fix that with more new variables. For the grammar (G4), the offending rules are $\mathsf{S_0} \rightarrow \mathsf{ASA}$, $\mathsf{S} \rightarrow \mathsf{ASA}$, and $\mathsf{A} \rightarrow \mathsf{ASA}$. We can rewrite these using a new variable Z and a rule $\mathsf{Z} \rightarrow \mathsf{SA}$. This gives us the CNF grammar shown on the right.

$$\boxed{\text{(G5)} \quad \begin{array}{ll} \Rightarrow & \mathsf{S_0} \rightarrow \mathsf{AZ} \mid \mathsf{UB} \mid \mathsf{a} \mid \mathsf{SA} \mid \mathsf{AS} \\ & \mathsf{S} \rightarrow \mathsf{AZ} \mid \mathsf{UB} \mid \mathsf{a} \mid \mathsf{SA} \mid \mathsf{AS} \\ & \mathsf{A} \rightarrow \mathsf{b} \mid \mathsf{AZ} \mid \mathsf{UB} \mid \mathsf{a} \mid \mathsf{SA} \mid \mathsf{AS} \\ & \mathsf{B} \rightarrow \mathsf{b} \\ & \mathsf{U} \rightarrow \mathsf{a} \\ & \mathsf{Z} \rightarrow \mathsf{SA} \end{array}}$$

We are done!