1 Three Simplifications

Motivation for Grammar Simplification

Parsing Problem

Given a CFG G and string w, determine if $w \in \mathbf{L}(G)$.

- Fundamental problem in compiler design and natural language processing.

If G is in general form then the procedure maybe very inefficient. So the grammar is "transformed" into a simpler form to make the parsing problem easier.

1.1 Eliminating ϵ-productions

Eliminating ϵ-productions

- Often would like to ensure that the length of the intermediate strings in a derivation are not longer than the final string derived
- But a long intermediate string can lead to a short final string if there are *ϵ-productions* (rules of the form $A \to \epsilon$).
- Can we rewrite the grammar not to have ϵ-productions?

Eliminating ϵ-production

The Problem

Given a grammar G produce an equivalent grammar G' (i.e., $\mathbf{L}(G) = \mathbf{L}(G')$) such that G' *has no rules of the form* $A \to \epsilon$, except possibly $S \to \epsilon$, and S does not appear on the right hand side of any rule.

Note: If S can appear on the RHS of a rule, say $S \to SS$, then when there is the rule $S \to \epsilon$, we can again have long intermediate strings yielding short final strings.

We will first introduce a concept that will be useful in this transformation.

Nullable Variables

Definition 1. A variable A (of grammar G) is *nullable* if $A \overset{*}{\Rightarrow} \epsilon$.

How do you determine if a variable is nullable?

- If $A \to \epsilon$ is a production in G then A is nullable
- If $A \to B_1 B_2 \cdots B_k$ is a production and each B_i is nullable, then A is nullable.
- Repeat the above steps until no new nullable variables can be found.

Using nullable variables

Intuition

For every variable A in G have a variable A in G' such that $A \overset{*}{\Rightarrow}_{G'} w$ iff $A \overset{*}{\Rightarrow}_G w$ and $w \neq \epsilon$.

For every rule $B \to CAD$ in G, where A is nullable, add two rules in G': $B \to CD$ and $B \to CAD$.

The Algorithm

- If $G = (V, \Sigma, R, S)$ then $G' = (V \cup \{S'\}, \Sigma, R', S')$ where $S' \notin V$.
- And the set R' will be defined as follows. For each rule $A \to X_1 X_2 \cdots X_k$ in G, create rules $A \to \alpha_1 \alpha_2 \cdots \alpha_k$ where

$$\alpha_i = \begin{cases} X_i & \text{if } X_i \text{ is a non-nullable variable/terminal in } G \\ X_i \text{ or } \epsilon & \text{if } X_i \text{ is nullable in } G \end{cases}$$

  and not all α_i are ϵ

- Add rule $S' \to S$. If S nullable in G, add $S' \to \epsilon$ also.

Correctness of the Algorithm

Leftmost Derivations

Before proving the correctness, we will introduce the notion of a leftmost derivation. A derivation $A \overset{*}{\Rightarrow} w$ is a *leftmost derivation* if every step of the derivation is obtained by applying a rule to the leftmost variable; we will denote this by $A \overset{*}{\Rightarrow}_{\mathrm{lm}} w$.

Example 2. Let $G = (\{S, A, B\}, \{a, b\}, \{S \to AB,\ A \to aA \mid a,\ B \to bB \mid b\}, S)$. The derivation $S \Rightarrow AB \Rightarrow aB \Rightarrow ab$ is a leftmost derivation. However, $S \Rightarrow AB \Rightarrow Ab \Rightarrow ab$ is not a leftmost derivation.

A few properties of leftmost derivations are useful to observe.

- Our proof constructing a derivation corresponding to a parse tree constructed a leftmost derivation.
- Therefore, $A \overset{*}{\Rightarrow} w$ iff $A \overset{*}{\Rightarrow}_{\mathrm{lm}} w$.
- A grammar $G = (V, \Sigma, R, S)$ is ambiguous iff there is $w \in \Sigma^*$ such that w has two (different) parse trees with root S and yield w iff there is $w \in \Sigma^*$ such that there are two (different) leftmost derivation of w from S.
- For $w \in \Sigma^*$, a leftmost derivation $A \overset{*}{\Rightarrow}_{\mathrm{lm}} w$ has the form

$$A \Rightarrow X_1 X_2 \cdots X_k \overset{*}{\Rightarrow}_{\mathrm{lm}} w_1 X_2 \cdots X_k \overset{*}{\Rightarrow}_{\mathrm{lm}} w_1 w_2 X_3 \cdots X_k \cdots \overset{*}{\Rightarrow}_{\mathrm{lm}} w_1 w_2 \cdots w_k = w$$

where $w_i \in \Sigma^*$, and $w_i = X_i$ if $X_i \in \Sigma$. That is, the derivation applies a rule to A, and then applies a sequence of steps to the leftmost symbol until we get a string of terminals (and no steps if the leftmost symbol is not a variable), and then sequence of steps the second symbol, and so on. Thus, here we have $X_i \stackrel{*}{\Rightarrow}_{\mathrm{lm}} w_i$.

Eliminating ϵ-productions
An Example

Example 3. Let $G = (\{S, A, B\}, \{a, b\}, R, S)$ where R is given by: $S \rightarrow AB$; $A \rightarrow AaA|\epsilon$; and $B \rightarrow BbB|\epsilon$.

- Nullables in G are A, B and S
- G' will have variables $\{S', S, A, B\}$ and rules:
  - $S \rightarrow AB|A|B$
  - $A \rightarrow AaA|aA|Aa|a$
  - $B \rightarrow BbB|bB|Bb|b$
  - $S' \rightarrow S|\epsilon$

1.2 Eliminating Unit Productions

Eliminating Unit Productions

- Often would like to ensure that the number of steps in a derivation are not much more than the length of the string derived
- But can have a long chain of derivation steps that make little or no "progress," if the grammar has *unit productions* (rules of the form $A \rightarrow B$, where B is a non-terminal).
  - Note: $A \rightarrow a$ is not a unit production
- Can we rewrite the grammar not to have unit-productions?

Eliminating unit-productions
Given a grammar G produce an equivalent grammar G' (i.e., $\mathbf{L}(G) = \mathbf{L}(G')$) such that G' *has no rules of the form* $A \rightarrow B$ where $B \in V'$.

Role of Unit Productions

Unit productions can play an important role in designing grammars:

- While eliminating ϵ-productions we added a rule $S' \rightarrow S$. This is a unit production.

- We have used unit productions in building an unambiguous grammar:

$$\begin{array}{ll} I \to a \mid b \mid Ia \mid Ib & T \to F \mid T * F \\ N \to 0 \mid 1 \mid N0 \mid N1 & E \to T \mid E + T \\ F \to I \mid N \mid -N \mid (E) & \end{array}$$

But as we shall see now, they can be (safely) eliminated

Eliminating Unit Productions

Basic Idea
Introduce new "look-ahead" productions to replace unit productions: look ahead to see where the unit production (or a chain of unit productions) leads to and add a rule to directly go there.

Example 4. $E \to T \to F \to I \to a|b|Ia|Ib$. So introduce new rules $E \to a|b|Ia|Ib$

But what if the grammar has *cycles of unit productions*? For example, $A \to B|a$, $B \to C|b$ and $C \to A|c$. You cannot use the "look-ahead" approach, because then you will get into an infinite loop.

The Algorithm

1. Determine pairs $\langle A, B\rangle$ such that $A \overset{*}{\Rightarrow}_u B$, i.e., A derives B using only unit rules. Such pairs are called *unit pairs.*

   - Easy to determine unit pairs: Make a directed graph with vertices = V, and edges = unit productions. $\langle A, B\rangle$ is a unit pair, if there is a directed path from A to B in the graph.
   - Note, it is possible to $A \overset{*}{\Rightarrow} B$ without using unit productions. Example, $A \to BC$ and $C \to \epsilon$.

2. If $\langle A, B\rangle$ is a unit pair, then add production rules $A \to \beta_1|\beta_2|\cdots\beta_k$, where $B \to \beta_1|\beta_2|\cdots|\beta_k$ are all the non-unit production rules of B

3. Remove all unit production rules.

Proposition 5. *Let G' be the grammar obtained from G using this algorithm to eliminate unit productions. Then* $\mathbf{L}(G') = \mathbf{L}(G)$

1.3 Eliminating Useless Symbols

Eliminating Useless Symbols

- Ideally one would like to use a compact grammar, with the fewest possible variables
- But a grammar may have "useless" variables which do not appear in any valid derivation

- Can we identify all the useless variables and remove them from the grammar? (Note: there may still be other redundancies in the grammar.)

Useless Symbols

Definition 6. A symbol $X \in V \cup \Sigma$ is useless in a grammar $G = (V, \Sigma, S, P)$ if there is no derivation of the form $S \stackrel{*}{\Rightarrow} \alpha X \beta \stackrel{*}{\Rightarrow} w$ where $w \in \Sigma^*$ and $\alpha, \beta \in (V \cup \Sigma)^*$.

Removing useless symbols (and rules involving them) from a grammar does not change the language of the grammar.

We can say X is useless iff either

Type 1: X is *not "reachable"* from S (i.e., no α, β such that $S \stackrel{*}{\Rightarrow} \alpha X \beta$), or

Type 2: for all α, β such that $S \stackrel{*}{\Rightarrow} \alpha X \beta$, either α, X or β cannot yield a string in Σ^*. i.e., either

Type 2a: X is *not "generating"* (i.e., no $w \in \Sigma^*$ such that $X \stackrel{*}{\Rightarrow} w$), or

Type 2b: α or β contains a non-generating symbol

Algorithm to Remove Useless Symbols

Algorithm
So, in order to remove useless symbols,

1. First remove all symbols that are not generating (Type 2a)
   - If X was useless, but reachable and generating (i.e., Type 2b) then X becomes unreachable after this step
     - Type 2b: for all α, β such that $S \stackrel{*}{\Rightarrow} \alpha X \beta$, α or β contains a non-generating symbol. Then in the new grammar all such derivations disappear (because some variable in α or β is removed).
2. Next remove all unreachable symbols in the new grammar.
   - Removes Type 1 (originally unreachable) and Type 2b useless symbols now

Doesn't remove any useful symbol in either step (Why?)

Only remains to show how to do the two steps in this algorithm

---

Generating and Reachable Symbols

Generating symbols

- If $A \to x$, where $x \in \Sigma^*$, is a production then A is generating
- If $A \to \gamma$ is a production and all variables in γ are generating, then A is generating.

Reachable symbols

- S is reachable
- If A is reachable and $A \to \alpha B \beta$ is a production, then B is reachable

---

1.4 Putting Together the Three Simplifications

The Three Simplifications, Together

Proposition 7. *Given a grammar G, such that $\mathbf{L}(G) \neq \emptyset$, we can find a grammar G' such that $\mathbf{L}(G') = \mathbf{L}(G)$ and G' has no ϵ-productions (except possibly $S \to \epsilon$), unit productions, or useless symbols, and S does not appear in the RHS of any rule.*

Proof. Apply the following 3 steps *in order*:

1. Eliminate ϵ-productions
2. Eliminate unit productions
3. Eliminate useless symbols. □

Note: Applying the steps in a different order may result in a grammar not having all the desired properties.

---

2 Chomsky Normal Form

Normal Forms for Grammars

It is typically easier to work with a context free language if given a CFG in a *normal form*.

Normal Forms
A grammar is in a normal form if its production rules have a special structure:

- *Chomsky Normal Form:* Productions are of the form $A \to BC$ or $A \to a$, where A, B, C are variables and a is a terminal symbol.
- *Greibach Normal Form* Productions are of the form $A \to a\alpha$, where $\alpha \in V^*$ and $A \in V$.

If ϵ is in the language, we allow the rule $S \to \epsilon$. We will require that S does not appear on the right hand side of any rules.

We will restrict our discussion to Chomsky Normal Form.

Main Result

Proposition 8. *For any non-empty context-free language L, there is a grammar G, such that $L(G) = L$ and each rule in G is of the form*

1. $A \to a$ where $a \in \Sigma$, or

2. $A \to BC$ where neither B nor C is the start symbol, or

3. $S \to \epsilon$ where S is the start symbol (iff $\epsilon \in L$)

Furthermore, G has no useless symbols.

Outline of Normalization

Given $G = (V, \Sigma, S, P)$, convert to CNF

- Let $G' = (V', \Sigma, S, P')$ be the grammar obtained after eliminating ϵ-productions, unit productions, and useless symbols from G.
- If $A \to x$ is a rule of G', where $|x| = 0$, then A must be S (because G' has no other ϵ-productions). If $A \to x$ is a rule of G', where $|x| = 1$, then $x \in \Sigma$ (because G' has no unit productions). In either case $A \to x$ is in a valid form.
- All remaining productions are of form $A \to X_1 X_2 \cdots X_n$ where $X_i \in V' \cup \Sigma$, $n \geq 2$ (and S does not occur in the RHS). We will put these rules in the right form by applying the following two transformations:
  1. Make the RHS consist only of variables
  2. Make the RHS be of length 2.

Make the RHS consist only of variables

Let $A \to X_1 X_2 \cdots X_n$, with X_i being either a variable or a terminal. We want rules where all the X_i are variables.

Example 9. Consider $A \to BbCdefG$. How do you remove the terminals?

For each $a, b, c \ldots \in \Sigma$ add variables $X_a, X_b, X_c, \ldots$ with productions $X_a \to a$, $X_b \to b$, ... Then replace the production $A \to BbCdefG$ by $A \to BX_bCX_dX_eX_fG$

For every $a \in \Sigma$

1. Add a new variable X_a

2. In every rule, if a occurs in the RHS, replace it by X_a

3. Add a new rule $X_a \to a$

Make the RHS be of length 2

- Now all productions are of the form $A \to a$ or $A \to B_1 B_2 \cdots B_n$, where $n \geq 2$ and each B_i is a variable.

- How do you eliminate rules of the form $A \to B_1 B_2 \dots B_n$ where $n > 2$?
- Replace the rule by the following set of rules

$$\begin{array}{rcl} A & \to & B_1 B_{(2,n)} \\ B_{(2,n)} & \to & B_2 B_{(3,n)} \\ B_{(3,n)} & \to & B_3 B_{(4,n)} \\ & \vdots & \\ B_{(n-1,n)} & \to & B_{n-1} B_n \end{array}$$

where $B_{(i,n)}$ are "new" variables.

An Example

Example 10. Convert: $S \to aA|bB|b$, $A \to Baa|ba$, $B \to bAAb|ab$, into Chomsky Normal Form.

1. Eliminate ϵ-productions, unit productions, and useless symbols. This grammar is already in the right form.
2. Remove terminals from the RHS of long rules. New grammar is: $X_a \to a$, $X_b \to b$, $S \to X_a A|X_b B|b$, $A \to B X_a X_a|X_b X_a$, and $B \to X_b A A X_b|X_a X_b$
3. Reduce the RHS of rules to be of length at most two. New grammar replaces $A \to B X_a X_a$ by rules $A \to B X_{aa}$, $X_{aa} \to X_a X_a$, and $B \to X_b A A X_b$ by rules $B \to X_b X_{AAb}$, $X_{AAb} \to A X_{Ab}$, $X_{Ab} \to A X_b$
