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 $A \stackrel{*}{\Rightarrow}_{G'} w$ iff $A \stackrel{*}{\Rightarrow}_G w$ and $w \neq \epsilon$.

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

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 $A \rightarrow X_1 X_2 \cdots X_k$ in G, create rules $A \rightarrow \alpha_1 \alpha_2 \cdots \alpha_k$ 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 $S' \rightarrow S$. If S nullable in G, add $S' \rightarrow \epsilon$ also.

Correctness of the Algorithm

Leftmost Derivations

Before proving the correctness, we will introduce the notion of a leftmost derivation. A derivation $A \stackrel{*}{\Rightarrow} w$ 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 $A \stackrel{*}{\Rightarrow}_{\mathrm{lm}} w$.

Example 2. Let $G = (\{S, A, B\}, \{a, b\}, \{S \rightarrow AB,\ A \rightarrow aA \mid a,\ B \rightarrow bB \mid b\}, S)$. 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, $A \stackrel{*}{\Rightarrow} w$ iff $A \stackrel{*}{\Rightarrow}_{\mathrm{lm}} w$.
- 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 $A \stackrel{*}{\Rightarrow}_{\mathrm{lm}} w$ has the form

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

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 $X_i \overset{*}{\Rightarrow}_{\text{lm}} w_i$.

We are now ready to prove the correctness of the algorithm eliminating ϵ-rules.

Proof.

- By construction, there are no rules of the form $A \to \epsilon$ in G' (except possibly $S' \to \epsilon$), and S' does not appear in the RHS of any rule.
- $L(G) = L(G')$
  - $L(G') \subseteq L(G)$: For every rule $A \to w$ in G', we have $A \overset{*}{\Rightarrow}_G w$ (by expanding zero or more nullable variables in w to ϵ)
  - $L(G) \subseteq L(G')$: If $\epsilon \in L(G)$, then $\epsilon \in L(G')$. For $w \neq \epsilon$, we will prove by induction a stronger statement. We will show that for every $w \in \Sigma^*$ ($w \neq \epsilon$), and every variable A, if $A \overset{*}{\Rightarrow}^G_{\text{lm}} w$ then $A \overset{*}{\Rightarrow}^{G'}_{\text{lm}} w$ by induction on the number of steps in the derivation $A \overset{*}{\Rightarrow}^G_{\text{lm}} w$.
    - **Base Case:** If $A \overset{*}{\Rightarrow}^G_{\text{lm}} w$ in one step, then $A \to w$ is rule in G. Since $w \neq \epsilon$, $A \to w$ is also a rule in G', and so $A \overset{*}{\Rightarrow}^{G'}_{\text{lm}} w$.
    - **Ind. Step:** Consider $A \overset{*}{\Rightarrow}^G_{\text{lm}} w$. Then by the property of leftmost derivations, $A \overset{*}{\Rightarrow}^G_{\text{lm}} w$ is of the form

$$A \Rightarrow X_1X_2 \cdots X_k \overset{*}{\Rightarrow}_{\text{lm}} w_1X_2 \cdots X_k \overset{*}{\Rightarrow}_{\text{lm}} w_1w_2X_3 \cdots X_k \cdots \overset{*}{\Rightarrow}_{\text{lm}} w_1w_2 \cdots w_k = w$$

      where $X_i \overset{*}{\Rightarrow}^G_{\text{lm}} w_i$. Now if $w_i \neq \epsilon$, then by induction hypothesis we have $X_i \overset{*}{\Rightarrow}^{G'}_{\text{lm}} w_i$. Thus, if $i_1, \ldots i_n$ are the indices such that $w_i \neq \epsilon$, then we have $A \Rightarrow_{G'} X_{i_1}X_{i_2} \cdots X_{i_n}$ (as the other veriables are nullable, $X_{i_j} \overset{*}{\Rightarrow}^{G'}_{\text{lm}} w_{i_j}$ by induction hypothesis, and $w = w_{i_1} \cdots w_{i_n}$ (as the other w_js are ϵ). Putting it all together we have

$$A \Rightarrow^{G'} X_{i_1} \cdots X_{i_n} \overset{*}{\Rightarrow}^{G'}_{\text{lm}} w_{i_1}X_{i_2} \cdots X_{i_n} \overset{*}{\Rightarrow}^{G'}_{\text{lm}} \cdots \overset{*}{\Rightarrow}^{G'}_{\text{lm}} w_{i_1}w_{i_2} \cdots w_{i_n} = w$$

□

Eliminating ϵ-productions

An Example

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

- Nullables in G are A, B and S
- G' will have variables $\{S', S, A, B\}$ and rules:
  - $S \to AB|A|B$

- $A \to AaA|aA|Aa|a$
- $B \to BbB|bB|Bb|b$
- $S' \to S|\epsilon$

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 $A \to B$, where B is a non-terminal).
  - Note: $A \to a$ 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* $A \to B$ 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 $S' \to S$. 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 $A \stackrel{*}{\Rightarrow}_u B$, 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 $A \stackrel{*}{\Rightarrow} B$ 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* $\mathbf{L}(G') = \mathbf{L}(G)$

Proof. $\mathbf{L}(G') \subseteq \mathbf{L}(G)$**:** For every rule $A \to w$ in G', we have $A \stackrel{*}{\Rightarrow}_G w$ (by a sequence of zero or more unit productions followed by a nonunit production of G)

$L(G) \subseteq L(G')$**:** For $w \in L(G)$ consider a *leftmost derivation* $S \stackrel{*}{\Rightarrow}_{\text{lm}} w$ in G.

- All these derivation steps are possible in G' also, except the ones using the unit productions of G.

- Suppose $S \stackrel{*}{\Rightarrow} xA\alpha \Rightarrow_1 xB\alpha \Rightarrow_2 \cdots$, where $\Rightarrow_1$ corresponds to a unit rule. Then (in a leftmost derivation) $\Rightarrow_2$ must correspond to using a rule for B.

- So a leftmost derivation of w in G can be broken up into "big-steps" each consisting of zero or more unit productions on the leftmost variable, followed by a non-unit production.

- For each such "big-step" there is a single production rule in G' that yields the same result. $\square$

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 $S \overset{*}{\Rightarrow} \alpha X \beta \overset{*}{\Rightarrow} w$ 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 $S \overset{*}{\Rightarrow} \alpha X \beta$), or

Type 2: for all α, β such that $S \overset{*}{\Rightarrow} \alpha X \beta$, 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 $X \overset{*}{\Rightarrow} w$), 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 $S \overset{*}{\Rightarrow} \alpha X \beta$, 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.