Lecture III: Normal Forms and Properties for CFL's

Jeffrey Ullman
Stanford University

Eliminating Useless Variables

Removing Epsilon

Removing Unit Productions

Chomsky Normal Form

Properties of CFL's

Useless Symbols

We say that $X \in \mathbf{V_N}$ is useful if:

$$S \Rightarrow^* \alpha X \beta \Rightarrow^* w$$

with $w \in \mathbf{V_T}^*$ and $\alpha, \beta \in \mathbf{V}^*$

A symbol is useless if it does not participate in any derivation and can be eliminated.

Useless Symbols (cont.)

- $X \in \mathbf{V_N}$ is generating if:

$$X \Rightarrow^* w, \quad \text{for} \quad w \in \mathbf{V_T}^*$$

- $X \in \mathbf{V_N}$ is reachable if:

$$S \Rightarrow^* \alpha X \beta, \quad \text{for} \quad \alpha, \beta \in \mathbf{V}^*$$

`Definition.` We say that a symbol X is useful if it is both generating and reachable.

Variables That Derive Nothing "Non-Generating"

- Consider: S -> AB

  A -> aA | a

  B -> AB
- Although A derives all strings of a's, B derives no terminal strings (can you prove this fact?).
- Thus, S derives nothing, and the language is empty.

Testing Whether a Variable Derives Some Terminal String

- Basis: If there is a production A -> w, where w has no variables, then A derives a terminal string.
- Induction: If there is a production A -> α, where α consists only of terminals and variables known to derive a terminal string, then A derives a terminal string.

Eliminating Non-Generating Symbols

To eliminate Non-Generating Symbols we need to:

1. Compute the set H of generating symbols, and then
2. Eliminate all productions containing a symbol in $NG = \mathsf{V_N} \setminus H$ (set of Non-Generating symbols).

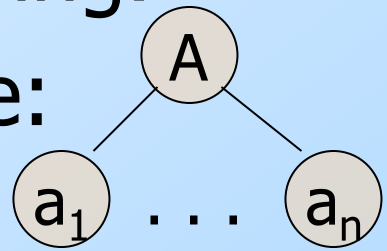
```
GENERATING-SYMBOLS(G)
H = V_T;
while there is a change in H do
    for each production A → X_1 ... X_k in P do
        if {X_1, ..., X_k} ⊆ H then
            H = H ∪ {A};
return H
```

Testing – (2)

- Eventually, we can find no more variables.
- An easy induction on the order in which variables are discovered shows that each one truly derives a terminal string.
- Conversely, any variable that derives a terminal string will be discovered by this algorithm.

Proof of Converse

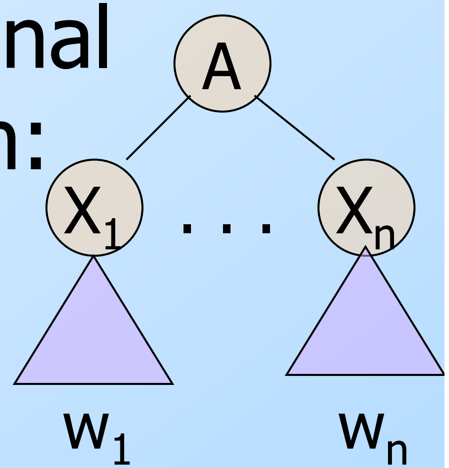
- The proof is an induction on the height of the least-height parse tree by which a variable A derives a terminal string.
- Basis: Height = 1. Tree looks like:



- Then the basis of the algorithm tells us that A will be discovered.

Induction for Converse

- Assume IH for parse trees of height < h, and suppose A derives a terminal string via a parse tree of height h:



- By IH, those X_i's that are variables are discovered.
- Thus, A will also be discovered, because it has a right side of terminals and/or discovered variables.

Algorithm to Eliminate Non-Generating Variables

1. Discover all variables that derive terminal strings.
2. For all other variables, remove all productions in which they appear either on the left or the right.

Example: Eliminate Variables

S -> AB | C

A -> aA | a

B -> bB

C -> c

- Basis: A and C are identified because of A -> a and C -> c.
- Induction: S is identified because of S -> C.
- Nothing else can be identified.
- Result: S -> C, A -> aA | a, C -> c

Unreachable Symbols

- Another way a terminal or variable deserves to be eliminated is if it cannot appear in any derivation from the start symbol.
- Basis: We can reach S (the start symbol).
- Induction: if we can reach A, and there is a production A -> α, then we can reach all symbols of α.

To eliminate Non-Reachable Symbols we need to:

1. Compute the set R of reachable symbols, and then
2. Eliminate all productions containing a symbol in $NR = \mathrm{V_N} \setminus R$ (set of Non-Reachable symbols).

```
REACHABLE-SYMBOLS(G)
R = {S};
while there is a change in R do
    for each production A → X_1 ... X_k in P do
        if A ∈ R then
            R = R ∪ {X_1, ..., X_k};
return R
```

Unreachable Symbols – (2)

- Easy inductions in both directions show that when we can discover no more symbols, then we have all and only the symbols that appear in derivations from S.
- Algorithm: Remove from the grammar all symbols not discovered reachable from S and all productions that involve these symbols.

Eliminating Useless Symbols

- A symbol is *useful* if it appears in some derivation of some terminal string from the start symbol.
- Otherwise, it is *useless*.
  Eliminate all useless symbols by:
  1. Eliminate non-generating symbols;
  2. Eliminate unreachable symbols.

Example: Useless Symbols – (2)

S -> AB | b

A -> C

C -> c

B -> bB

- If we eliminated unreachable symbols first, we would find everything is reachable.
- A, C, and c would never get eliminated.

Why It Works

- After step (1), every symbol remaining derives some terminal string.
- After step (2) the only symbols remaining are all derivable from S.
- In addition, they still derive a terminal string, because such a derivation can only involve symbols reachable from S.

Epsilon Productions

- We can almost avoid using productions of the form A -> ϵ (called *ϵ-productions*).
  - The problem is that ϵ cannot be in the language of any grammar that has no ϵ–productions.
- Theorem: If L is a CFL, then L-{ϵ} has a CFG with no ϵ-productions.

Nullable Symbols

- To eliminate ϵ-productions, we first need to discover the *nullable variables* = variables A such that A =>* ϵ.
- Basis: If there is a production A -> ϵ, then A is nullable.
- Induction: If there is a production A -> α, and all symbols of α are nullable, then A is nullable.

The following algorithm computes the set N of nullable symbols.

NULLABLE-SYMBOLS(G)
$N = \emptyset$;
for *each production* $A \to \epsilon$ *in* P **do**
    $N = N \cup \{A\}$
while *there is a change in* N **do**
    for *each production* $A \to X_1 \ldots X_k$ *in* P **do**
        if $\{X_1, \ldots, X_k\} \subseteq N$ **then**
            $N = N \cup \{A\}$;
return N

Example: Nullable Symbols

S -> AB, A -> aA | ϵ, B -> bB | A

- Basis: A is nullable because of A -> ϵ.
- Induction: B is nullable because of B -> A.
- Then, S is nullable because of S -> AB.

Proof of Nullable-Symbols Algorithm

- The proof that this algorithm finds all and only the nullable variables is very much like the proof that the algorithm for symbols that derive terminal strings works.
- Do you see the two directions of the proof?
- On what is each induction?

Eliminating ϵ-Productions

- Key idea: turn each production $A \text{ -> } X_1 \ldots X_n$ into a **family of productions**.
- For each subset of nullable X's, there is one production with those eliminated from the right side "in advance."
  - Except, if all X's are nullable, do not make a production with ϵ as the right side.

Finally, eliminate all ε-Productions except the one for S.

Example: Eliminating ϵ-Productions

S -> ABC, A -> aA | ϵ, B -> bB | ϵ, C -> ϵ

- A, B, C, and S are all nullable.
- New grammar:

  S -> ~~ABC~~ | AB | ~~AC~~ | ~~BC~~ | A | B ~~| C~~

  A -> aA | a

  B -> bB | b

Note: C is now useless.
Eliminate its productions.

Why it Works

- Prove that for all variables A:
  1. If $w \neq \epsilon$ and $A \Rightarrow^*_{old} w$, then $A \Rightarrow^*_{new} w$.
  2. If $A \Rightarrow^*_{new} w$ then $w \neq \epsilon$ and $A \Rightarrow^*_{old} w$.
- Then, letting A be the start symbol proves that L(new) = L(old) – {ϵ}.
- (1) is an induction on the number of steps by which A derives w in the old grammar.

Proof of 1 – Basis

- If the old derivation is one step, then A -> w must be a production.
- Since $w \neq \epsilon$, this production also appears in the new grammar.
- Thus, $A =>_{new} w$.

Proof of 1 – Induction

- Let $A =>^*_{old} w$ be an n-step derivation, and assume the IH for derivations of less than n steps.
- Let the first step be $A =>_{old} X_1 \ldots X_n$.
- Then w can be broken into $w = w_1 \ldots w_n$,
- where $X_i =>^*_{old} w_i$, for all i, in fewer than n steps.

Induction – Continued

- By the IH, if $w_i \neq \epsilon$, then $X_i =>*_{new} w_i$.
- Also, the new grammar has a production with A on the left, and just those X_i's on the right such that $w_i \neq \epsilon$.
  - Note: they all can't be ϵ, because $w \neq \epsilon$.
- Follow a use of this production by the derivations $X_i =>*_{new} w_i$ to show that A derives w in the new grammar.

Proof of Converse

- We also need to show part (2) – if w is derived from A in the new grammar, then it is also derived in the old.
- Induction on number of steps in the derivation.
- We'll leave the proof for reading in the text.

Unit Productions

- A *unit production* A -> B, with B ϵ V_N.
- These productions can be eliminated.

1. Key idea:

   - Remove ϵ–productions

   - If A =>* B by a series of unit productions, and B -> α is a non-unit-production, then add production A -> α

- Then, drop all unit productions.

Unit Productions (cont.)

To check that $A \Rightarrow^* B$, by a series of unit productions, note that:

- Since we have not ϵ-predictions then $A \Rightarrow^* B$ iff:

$$A \Rightarrow B_1 \Rightarrow B_2 \Rightarrow \ldots \Rightarrow B_{k-1} \Rightarrow B_k \Rightarrow B$$

- Each single derivation, $B_i \Rightarrow B_{i+1}$ must correspond to a unit production $B_i \rightarrow B_{i+1}$ in P.
- We can construct the Graph of Unit Productions and check whether B is reachable from A:
  - There is a node for each symbol in $\mathrm{V_N}$;
  - There is an edge (X, Y) in the graph if the unit production $X \rightarrow Y$ is in P.

Cleaning Up a Grammar

- Theorem: if L is a CFL, then there is a CFG for $L - \{\epsilon\}$ that has:
  1. No useless symbols.
  2. No ϵ-productions.
  3. No unit productions.
- I.e., every right side is either a single terminal or has length $\geq$ 2.

Cleaning Up – (2)

- Proof: Start with a CFG for L.
- Perform the following steps in order:
  1. Eliminate ϵ-productions.
  2. Eliminate unit productions.
  3. Eliminate variables that derive no terminal string.
  4. Eliminate variables not reached from the start symbol.

Must be first. Can create unit productions or useless variables.

Chomsky Normal Form

- A CFG is said to be in *Chomsky Normal Form* if every production is of one of these two forms:
  1. A -> BC (right side is two variables).
  2. A -> a (right side is a single terminal).
- Theorem: If L is a CFL, then L – {ϵ} has a CFG in CNF.

Summary of Decision Properties

- As usual, when we talk about "a CFL" we really mean "a representation for the CFL, e.g., a CFG or a PDA (Push-Down Automata) accepting by final state or empty stack.
- There are algorithms to decide if:
  1. String w is in CFL L: Parsers.
  2. CFL L is empty: Check if S is useless.
  3. CFL L is infinite.

Non-Decision Properties

- Many questions that can be decided for regular languages cannot be decided for CFL's.
- Example: Are two CFL's the same?
- Example: Are two CFL's disjoint?
- Need theory of Turing machines and decidability to prove no algorithm exists.

Closure Properties of CFL's

- CFL's are closed under union, concatenation, and Kleene closure.
- But not under intersection or difference.

Closure of CFL's Under Union

- Let L and M be CFL's with grammars G and H, respectively.
- Assume G and H have no variables in common.
  - Names of variables do not affect the language.
- Let S_1 and S_2 be the start symbols of G and H.

Closure Under Union – (2)

- Form a new grammar for $L \cup M$ by combining all the symbols and productions of G and H.
- Then, add a new start symbol S.
- Add productions $S \to S_1 \mid S_2$.

Closure Under Union – (3)

- In the new grammar, all derivations start with S.
- The first step replaces S by either S_1 or S_2.
- In the first case, the result must be a string in L(G) = L, and in the second case a string in L(H) = M.

Closure of CFL's Under Concatenation

- Let L and M be CFL's with grammars G and H, respectively.
- Assume G and H have no variables in common.
- Let S_1 and S_2 be the start symbols of G and H.

Closure Under Concatenation – (2)

- Form a new grammar for LM by starting with all symbols and productions of G and H.
- Add a new start symbol S.
- Add production S -> S_1S_2.
- Every derivation from S results in a string in L followed by one in M.

Closure Under Kleen Closure (Star)

- Let L have grammar G, with start symbol S_1.
- Form a new grammar for L* by introducing to G a new start symbol S and the productions $S \to S_1S \mid \epsilon$.
- A derivation from S generates a sequence of zero or more S_1's, each of which generates some string in L.

Nonclosure Under Intersection

- Unlike the regular languages, the class of CFL's is not closed under $\cap$.
- We know that $L_1 = \{0^n1^n2^n \mid n \geq 1\}$ is not a CFL (use the pumping lemma).
- However, $L_2 = \{0^n1^n2^i \mid n \geq 1, i \geq 1\}$ is.
  - CFG: S -> AB, A -> 0A1 | 01, B -> 2B | 2.
- So is $L_3 = \{0^i1^n2^n \mid n \geq 1, i \geq 1\}$.
- But $L_1 = L_2 \cap L_3$.

Nonclosure Under Difference

- We can prove something more general:
  - Any class of languages that is closed under difference is closed under intersection.
- Proof: $L \cap M = L - (L - M)$.
- Thus, if CFL's were closed under difference, they would be closed under intersection, but they are not.