Normal Forms for CFG's

Eliminating Useless Variables

Removing Epsilon

Removing Unit Productions

Chomsky Normal Form

Variables That Derive Nothing

- 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

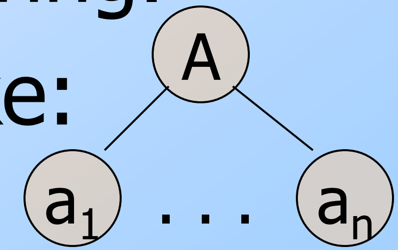
- Basis: If there is a production A -> w, where w has no variables, then A derives a terminal string.
- Induction: If there is a production A -> α, where α consists only of terminals and variables known to derive a terminal string, then A derives a terminal string.

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

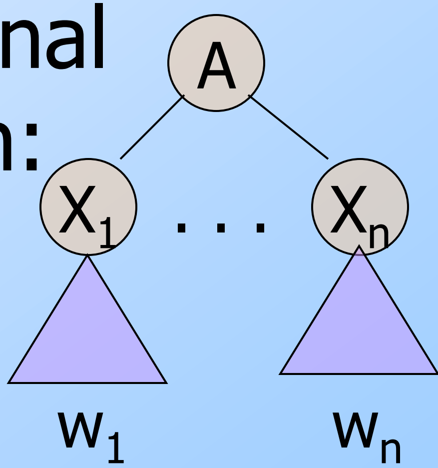
- The proof is an induction on the height of the least-height parse tree by which a variable A derives a terminal string.
- Basis: Height = 1. Tree looks like:



- 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 Variables That Derive Nothing

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.

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 symbols that derive no terminal string.
  2. Eliminate unreachable symbols.

Example: Useless Symbols – (2)

S -> AB, 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.

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 $A \to X_1 \ldots X_n$ 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.

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 $L(new) = L(old) - \{\epsilon\}$.
- (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* is one whose right side consists of exactly one variable.
- These productions can be eliminated.
- Key idea: 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 – (2)

- Find all pairs (A, B) such that A =>* B by a sequence of unit productions only.
- Basis: Surely (A, A).
- Induction: If we have found (A, B), and B -> C is a unit production, then add (A, C).

Proof That We Find Exactly the Right Pairs

- By induction on the order in which pairs (A, B) are found, we can show A =>* B by unit productions.
- Conversely, by induction on the number of steps in the derivation by unit productions of A =>* B, we can show that the pair (A, B) is discovered.

Proof The the Unit-Production-Elimination Algorithm Works

- Basic idea: there is a leftmost derivation $A =>*_{lm} w$ in the new grammar if and only if there is such a derivation in the old.
- A sequence of unit productions and a non-unit production is collapsed into a single production of the new grammar.

Cleaning Up a Grammar

- Theorem: if L is a CFL, then there is a CFG for L – {ε} 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.

Proof of CNF Theorem

- Step 1: "Clean" the grammar, so every production right side is either a single terminal or of length at least 2.
- Step 2: For each right side $\neq$ a single terminal, make the right side all variables.
  - For each terminal *a* create new variable A_a and production A_a -> a.
  - Replace *a* by A_a in right sides of length > 2.

Example: Step 2

- Consider production A -> BcDe.
- We need variables A_c and A_e. with productions A_c -> c and A_e -> e.
  - Note: you create at most one variable for each terminal, and use it everywhere it is needed.
- Replace A -> BcDe by A -> BA_cDA_e.

CNF Proof – Continued

- Step 3: Break right sides longer than 2 into a chain of productions with right sides of two variables.
- Example: A -> BCDE is replaced by A -> BF, F -> CG, and G -> DE.
  - F and G must be used nowhere else.

Example of Step 3 – Continued

- Recall A -> BCDE is replaced by A -> BF, F -> CG, and G -> DE.
- In the new grammar, A => BF => BCG => BCDE.
- More importantly: Once we choose to replace A by BF, we must continue to BCG and BCDE.
  - Because F and G have only one production.

CNF Proof – Concluded

- We must prove that Steps 2 and 3 produce new grammars whose languages are the same as the previous grammar.
- Proofs are of a familiar type and involve inductions on the lengths of derivations.