Properties of Context-Free Languages

COMP 455 – 002, Spring 2019

Simplification of CFGs

We can simplify CFGs by removing:

- **Useless symbols**.
  - X is **generating** if $X \overset{*}{\Rightarrow} w$, where $w \in T^*$.
  - X is **reachable** if $S \overset{*}{\Rightarrow} \alpha X \beta$ (S is the start symbol).
  - X is **useful** only if it is both reachable and generating.
- **ε-productions**, of the form $A \to \varepsilon$.
  - If ε is in the language, we will still need *one* ε-production.
- **Unit productions**, of the form $A \to B$.

Finding Generating Variables

Theorem 7.4: The following algorithm correctly finds all generating variables.

old_vars := $\emptyset$;
new_vars := $\{A \mid A \rightarrow w$ exists, and $w \in T^*\}$;
while old_vars $\neq$ new_vars:
    old_vars = new_vars;
    new_vars = old_vars $\cup \{A \mid A \rightarrow \alpha, \alpha \in (T \cup$ old_vars$)^*\}$;
return new_vars;

Finding Generating Variables

```
old_vars := ∅;
new_vars := {A | A → w exists,  and w ∈ T*};
while old_vars ≠ new_vars:
        old_vars = new_vars;
        new_vars = old_vars ∪ {A | A → α, α ∈ (T ∪ old_vars)*};
return new_vars;
```

Proof of Theorem 7.4:

We want to show that X is added to new_vars if and only if $X \overset{*}{\Rightarrow} w$ for some $w \in T^*$.

"Only if": We must show that if X is added to new_vars then $X \overset{*}{\Rightarrow} w$.

This can be proven by induction on the number of iterations of the algorithm (specifics are left as an exercise).

Finding Generating Variables

```
old_vars := ∅;
new_vars := {A | A → w exists,  and w ∈ T*};
while old_vars ≠ new_vars:
        old_vars = new_vars;
        new_vars = old_vars ∪ {A | A → α, α ∈ (T ∪ old_vars)*};
return new_vars;
```

Proof of Theorem 7.4:

We want to show that X is added to new_vars if and only if $X \overset{*}{\Rightarrow} w$ for some $w \in T^*$.

"If": We must show that if $X \overset{*}{\Rightarrow} w$, then X is eventually added to new_vars.

This can be proven by induction on the length of the derivation (specifics are left as an exercise).

Finding Reachable Variables

Theorem 7.5: There exists an iterative algorithm that will correctly find all *reachable* symbols.

This is similar to the previous algorithm, except this time you'll start with a set containing the start symbol and look for new reachable symbols in each iteration.

```
old_vars := ∅;
new_vars := {S};
while old_vars ≠ new_vars:
        old_vars = new_vars;
        new_vars = old_vars ∪
          {A | A is produced by something in old_vars};
return new_vars;
```

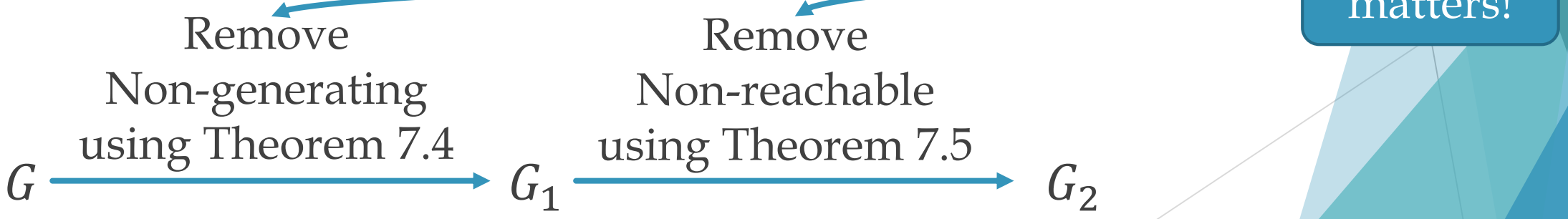
Eliminating Useless Symbols

Theorem 7.2: (Abbreviated) Every nonempty CFL is generated by a CFG with no useless symbols.

Proof:

Let L be the language of some CFG G, where $L \neq \emptyset$.

Define:



Eliminating Useless Symbols

$G \xrightarrow{\text{Remove non-generating}} G_1 \xrightarrow{\text{Remove unreachable}} G_2$

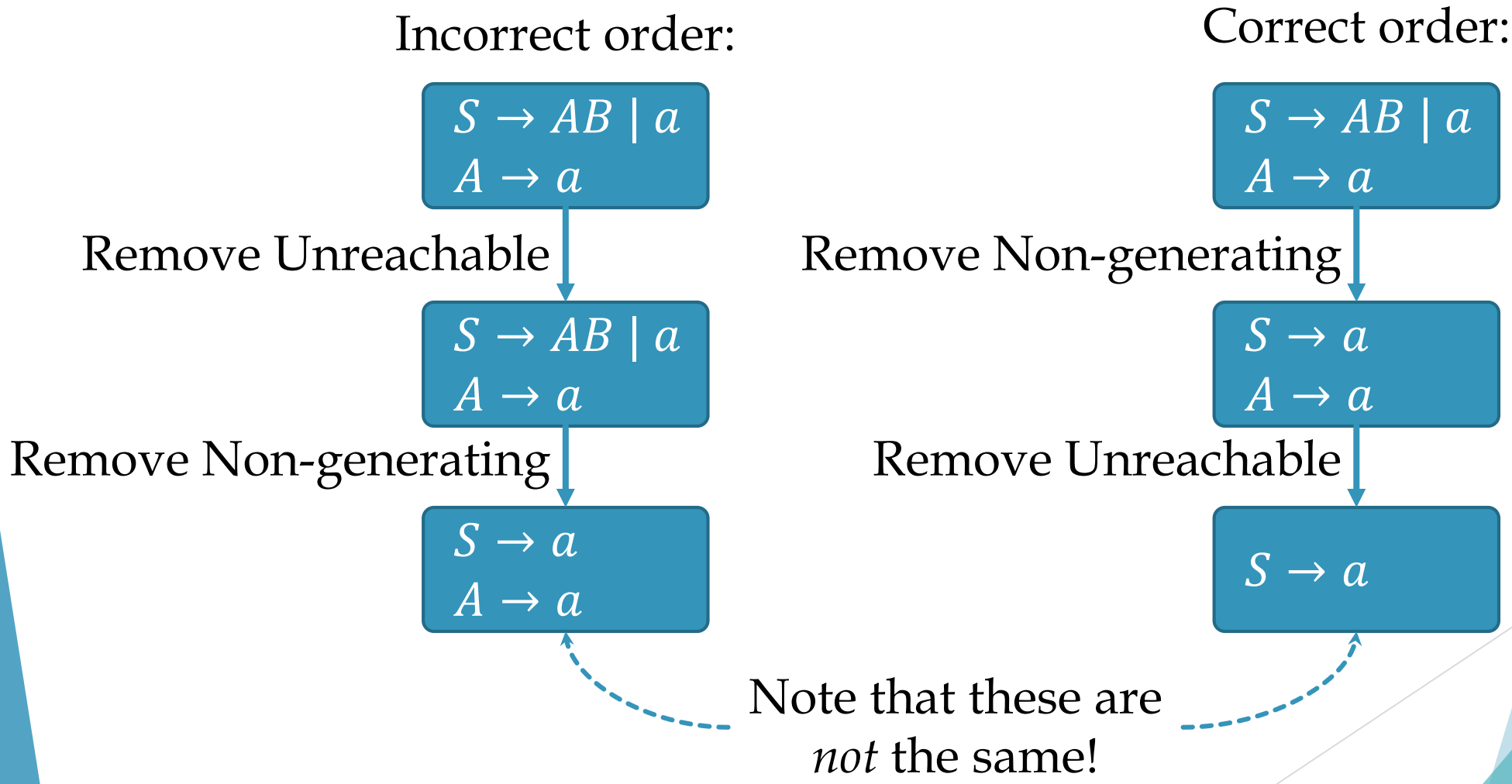
Proof, continued:

Assume G_2 contains a useless variable, X.

- By Theorem 7.5, $S \overset{*}{\underset{G_2}{\Rightarrow}} \alpha X \beta$.
  - In other words, we know that X is reachable in G_2.
- Any production in G_2 must be a production in G_1, so $S \overset{*}{\underset{G_1}{\Rightarrow}} \alpha X \beta$ must be a production in G_1.
- By Theorem 7.4, $S \overset{*}{\underset{G_1}{\Rightarrow}} \alpha X \beta \overset{*}{\underset{G_1}{\Rightarrow}} w$.
  - In other words, we know that X is producing in G_1.
- Every symbol in this derivation is reachable from S, so none will be eliminated by Theorem 7.5. So, $S \overset{*}{\underset{G_2}{\Rightarrow}} \alpha X \beta \overset{*}{\underset{G_2}{\Rightarrow}} w$. This contradicts the assumption that X was useless in G_2.

It should be intuitively clear that removing useless symbols won't change the language of a grammar.

Eliminating Useless Symbols: Example



Removing Nullable Symbols

A symbol A is *nullable* if $A \overset{*}{\Rightarrow} \varepsilon$.

Theorem 7.7: There exists an algorithm that will correct identify all nullable symbols.

We will not prove this—it should be intuitively similar to what we've done before.

```
old_vars := ∅;
new_vars := {A | A → ε exists};
while old_vars ≠ new_vars:
        old_vars = new_vars;
        new_vars = old_vars ∪ {A | A → α, α ∈ old_vars*};
return new_vars;
```

Removing ε-Productions

Theorem 7.9, reworded: If $L = L(G)$ for a CFG G, then there exists a CFG G_1 with no ε-Productions such that $L(G_1) = L(G) - \{\varepsilon\}$.

Proof:

To construct G_1: If $A \to X_1 \ldots X_k$ is in P, then add all productions of the form $A \to \alpha_1 \ldots \alpha_k$ to P_1, where:

1. If X_i is *not* nullable, then $\alpha_i = X_i$,
2. If X_i is nullable, then α_i is *either* X_i or ε, and
3. Not all α_i's are ε.

This requires adding *two* production rules for each nullable X_i.

Removing ε-Productions: Example

CFG P:

- $S \to ABC$
- $A \to \varepsilon$
- $B \to b \mid \varepsilon$
- $C \to c$

A and B are nullable. CFG P_1:

- $S \to ABC \mid BC \mid AC \mid C$
- $B \to b$
- $C \to c$

A is now useless in P_1. (If you want to eliminate both ε-productions and useless symbols, you must remove ε-productions *first*.

Removing ε-Productions: Proof

G_1: The CFG without ε-productions producing $L(G) - \{\varepsilon\}$.

Proof, continued:

Claim: For all $A \in V$ and $w \in T^*$, $A \underset{G_1}{\overset{*}{\Rightarrow}} w$ if and only if $w \neq \varepsilon$ and $A \underset{G}{\overset{*}{\Rightarrow}} w$.

"**If**": Assume $A \underset{G}{\overset{i}{\Rightarrow}} w$ and $w \neq \varepsilon$. We prove by induction on i that $A \underset{G_1}{\overset{*}{\Rightarrow}} w$.

Base case: $i = 1$ (one derivation step)

$A \to w$ must be a production in P. Because $w \neq \varepsilon$, $A \to w$ is also a production in G_1.

Removing ε-Productions: Proof

Inductive step: $i > 1$ (more than one derivation step)

Assume $A \underset{G}{\Rightarrow} Y_1 \ldots Y_m \overset{i-1}{\underset{G}{\Rightarrow}} w$. Then $Y_j \overset{*}{\underset{G}{\Rightarrow}} w_j$ and $w = w_1 \ldots w_m$.

Claim: For all $A \in V$ and $w \in T^*$, $A \overset{*}{\underset{G_1}{\Rightarrow}} w$ **if** and only if $w \neq \varepsilon$ and $A \overset{*}{\underset{G}{\Rightarrow}} w$.

If $w_j \neq \varepsilon$, then $Y_j \overset{*}{\underset{G_1}{\Rightarrow}} w_j$, by the induction hypothesis.

If $w_j = \varepsilon$, then Y_j is nullable.

Therefore, $A \rightarrow \beta_1 \ldots \beta_m$ is in the productions of G_1, where

- $\beta_j = Y_j$ if $w_j \neq \varepsilon$,
- $\beta_j = \varepsilon$ if $w_j = \varepsilon$.

And we have the following derivation in G_1:

$$A \Rightarrow \beta_1\beta_2 \ldots \beta_m \overset{*}{\Rightarrow} w_1\beta_2 \ldots \beta_m \overset{*}{\Rightarrow} w_1w_2 \ldots w_m = w.$$

Removing ε-Productions: Proof

Claim: For all $A \in V$ and $w \in T^*$, $A \overset{*}{\underset{G_1}{\Rightarrow}} w$ if and **only if** $w \neq \varepsilon$ and $A \overset{*}{\underset{G}{\Rightarrow}} w$.

"Only if": Assume $A \overset{i}{\underset{G_1}{\Rightarrow}} w$. Then, $w \neq \varepsilon$.

We will prove by induction on i that $A \overset{*}{\underset{G}{\Rightarrow}} w$.

Base case: $i = 1$.

$A \to w$ is in the productions of G_1. Therefore, $A \to \alpha$ is in the productions of G where $w = \alpha$ with nullable symbols replaced by ε.

Removing ε-Productions: Proof

"Only if", continued:

We must show that the derivation $A \underset{G}{\Rightarrow} \alpha \overset{*}{\underset{G}{\Rightarrow}} w$ exists in G.

Inductive step: Suppose that $A \underset{G_1}{\Rightarrow} X_1 \ldots X_k \overset{i-1}{\underset{G_1}{\Rightarrow}} w$.

Then, $A \rightarrow \beta$ is in the productions of G, where $X_1 \ldots X_k = \beta$ with some nullable symbols removed.

As in the base case, $A \overset{*}{\underset{G}{\Rightarrow}} X_1 \ldots X_k$. And, by the inductive hypothesis, we can show that $X_1 \ldots X_k \overset{*}{\underset{G}{\Rightarrow}} w$.

This is where nullable symbols are replaced by ε.

"Assume the derivation with $i-1$ steps is correct…" etc.

Removing Unit Productions

Theorem 7.13 (reworded): If $L = L(G)$ for a CFG G, then there exists a CFG G_1 with no unit productions such that $L = L(G_1)$.

Proof: Let $G = (V, T, P, S)$

To construction G_1, first add all non-unit productions in P to P_1.

Next, if $A \underset{G}{\overset{*}{\Rightarrow}} B$ and $B \rightarrow \alpha$ is a non-unit production in P, then add $A \rightarrow \alpha$ to P_1.

We can find all pairs of (A, B) where $A \underset{G}{\overset{*}{\Rightarrow}} B$ using an iterative algorithm like before (see Section 7.1.4 of the book).

Removing Unit Productions: Example

Consider the CFG G with the following productions:

- $S \rightarrow A \mid b$
- $A \rightarrow AAa$

After removing the unit production $S \rightarrow A$, this becomes:

- $S \rightarrow AAa \mid b$
- $A \rightarrow AAa$

Removing Unit Productions: Proof

G: The grammar containing unit productions.
G_1: The grammar with unit productions removed.

Claim: $L(G_1) \subseteq L(G)$

Proof:

If $A \to \alpha$ is in P_1, then $A \underset{G}{\overset{*}{\Rightarrow}} \alpha$. Therefore, $A \underset{G_1}{\overset{*}{\Rightarrow}} \alpha$ implies $A \underset{G}{\overset{*}{\Rightarrow}} \alpha$.

Removing Unit Productions: Proof

G: The grammar containing unit productions.
G_1: The grammar with unit productions removed.

Claim: $L(G) \subseteq L(G_1)$

Proof:

Suppose $w \in L(G)$.

Let $S = \alpha_0 \underset{G}{\Rightarrow} \alpha_1 \underset{G}{\Rightarrow} \ldots \underset{G}{\Rightarrow} \alpha_n = w$ be a leftmost derivation.

If $\alpha_i \underset{G}{\Rightarrow} \alpha_{i+1}$ is due to a *non-unit* production, then $\alpha_i \underset{G_1}{\Rightarrow} \alpha_{i+1}$.

Removing Unit Productions: Proof

G: The grammar containing unit productions.
G_1: The grammar with unit productions removed.

Claim: $L(G) \subseteq L(G_1)$

Consider the following leftmost derivation *with* unit productions in G: $\alpha_{i-1} \underset{G}{\Rightarrow} \alpha_i \underset{G}{\Rightarrow} \alpha_{i+1} \underset{G}{\Rightarrow} \ldots \underset{G}{\Rightarrow} \alpha_j \underset{G}{\Rightarrow} \alpha_{j+1}$.

Non-unit production

Unit productions

Non-unit production

Unit productions just replace the symbol at the same (leftmost) position, so $\alpha_{i-1} \Rightarrow \alpha_{j+1}$ will also hold by some production in $P_1 - P$.

Putting it all Together

Theorem 7.14: If L is the language of a CFG G, and L contains at least one string other than ε, then there exists a CFG G_1 with no ε-productions, unit productions, or useless symbols such that $L(G_1) = L - \{\varepsilon\}$.

(See the book for a formal proof.)

The *order* in which we apply the previous results is important.

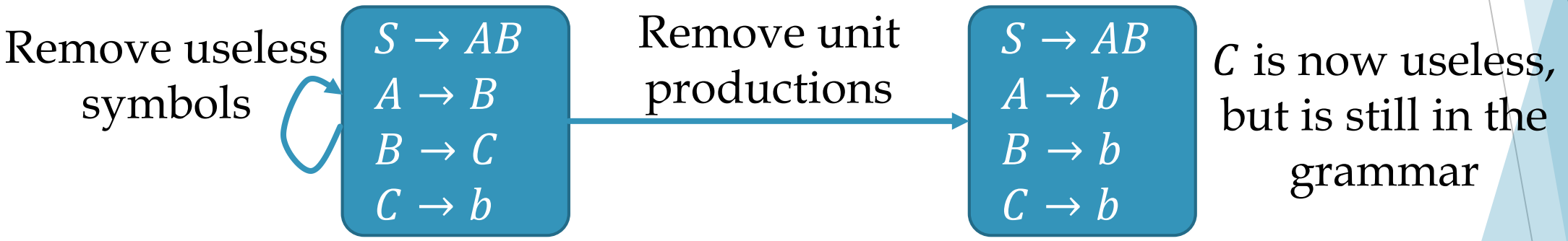
Putting it all Together

The order of previous results:

- We saw on slide 12 that eliminating ε-productions may cause some symbols to become useless, so we must eliminate ε-productions before removing useless symbols.
- In the same example on slide 12, removing ε-productions also introduced a unit production ($S \rightarrow C$), so ε-productions must be eliminated before unit productions.

Putting it all Together

Finally, unit productions must be eliminated before useless symbols. Consider this example:



So, the only viable order is: **1)** Eliminate ε-productions, **2)** Eliminate unit productions, and **3)** Eliminate useless symbols.

Chomsky Normal Form (CNF)

- A CFG is in *Chomsky Normal Form* if all of its productions are of the form $A \to BC$ or $A \to a$, and it contains no useless symbols.

Theorem 7.16 (reworded): Any CFL that doesn't include ε can be generated by a CFG in CNF.

Proof: Let $L = L(G)$ for some CFG G, where $\varepsilon \notin L$. Use Theorem 7.14 to convert G into $G_1 = (V, T, P, S)$, where G_1 contains no ε-productions, unit productions, or useless symbols.

Converting to Chomsky Normal Form

Otherwise $A \to X$ would be a unit production, and have been eliminated already.

Proof (continued):

If $A \to X$ is a production, then $X \in T$ (which is already in the correct form).

Otherwise, consider $A \to X_1 X_2 \ldots X_m$, where $m \geq 2$.

- If X_i is a terminal, introduce a new variable C_a and a new production $C_a \to a$, then replace X_i by C_a.
- Call the resulting grammar G_2 (after making all such replacements).

Claim: $L(G_1) = L(G_2)$. (The proof is left as an exercise.)

Converting to Chomsky Normal Form

The remaining problem is that we need to replace productions of the form $A \rightarrow B_1 B_2 \ldots B_m$ (where $m \geq 3$).

Replace such a production by:

$A \rightarrow B_1 D_1$, $D_1 \rightarrow B_2 D_2$, …, $D_{m-1} \rightarrow B_{m-1} B_m$, using newly added D_i variables.

Call the resulting grammar G_3.

Claim: $L(G_3) = L(G_2)$. (The proof is left as an exercise.)

Conversion to CNF: Example

Starting CFG with no ε-productions, unit productions, or useless symbols.

$$S \to bA \mid aB$$
$$A \to bAA \mid aS \mid a$$
$$B \to aBB \mid bS \mid b$$

$$S \to C_bA \mid C_aB$$
$$A \to C_bAA \mid C_aS \mid a$$
$$B \to C_aBB \mid C_bS \mid b$$
$$C_a \to a$$
$$C_b \to b$$

Replace $A \to C_bAA$ by:
$A \to C_bD_1$, $D_1 \to AA$

Replace $B \to C_aBB$ by:
$B \to C_aD_2$, $D_2 \to BB$

$$S \to C_bA \mid C_aB$$
$$A \to C_bD_1 \mid C_aS \mid a$$
$$B \to C_aD_2 \mid C_bS \mid b$$
$$D_1 \to AA$$
$$D_2 \to BB$$
$$C_a \to a$$
$$C_b \to b$$

CFG in CNF.

The Pumping Lemma for CFLs

Theorem 7.18 (the Pumping Lemma for CFLs): Let L be any CFL. Then there exists a value n such that for all $z \in L$, where $|z| \geq n$, there exist strings u, v, w, x, y such that:

1. $z = uvwxy$,
2. $|vx| \geq 1$,
3. $|vwx| \leq n$, and
4. For all $i \geq 0$, $uv^iwx^iy \in L$.

Proof of the Pumping Lemma for CFLs

If L is a CFL, let G be a CFG generating $L - \{\varepsilon\}$.

Claim: Let $z \in L - \{\varepsilon\}$. If a parse tree for z in G has no path longer than n, then $|z| \leq 2^{n-1}$. (This is Theorem 7.17 in the book.)

The Pumping Lemma only applies to strings longer than n, so removing ε doesn't matter.

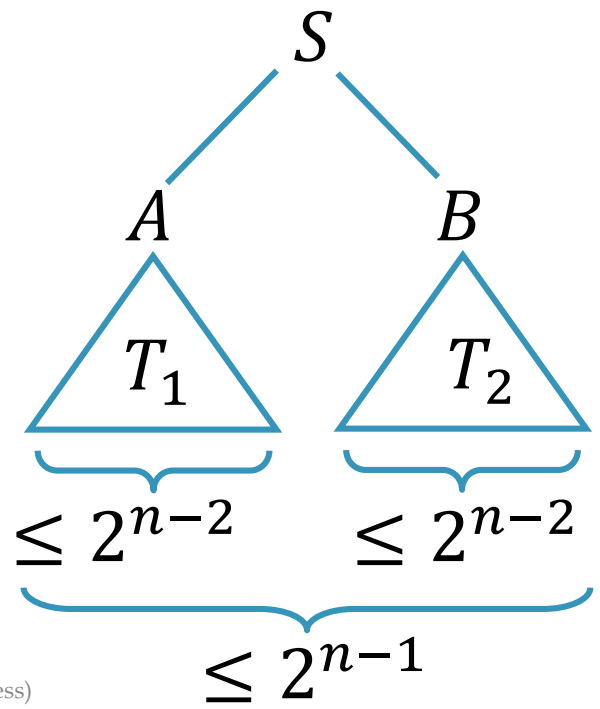
Proof, by induction on n:

Base case: $n = 1$. String $z = a$. Tree: S — a $\quad |z| = 1 = 2^{n-1}$

Proof of the Pumping Lemma for CFLs

Inductive step: $n > 1$.

Suppose that a tree exists with some path of length n, but no path exceeding a length n. It looks like this:



Proof of the Pumping Lemma for CFLs

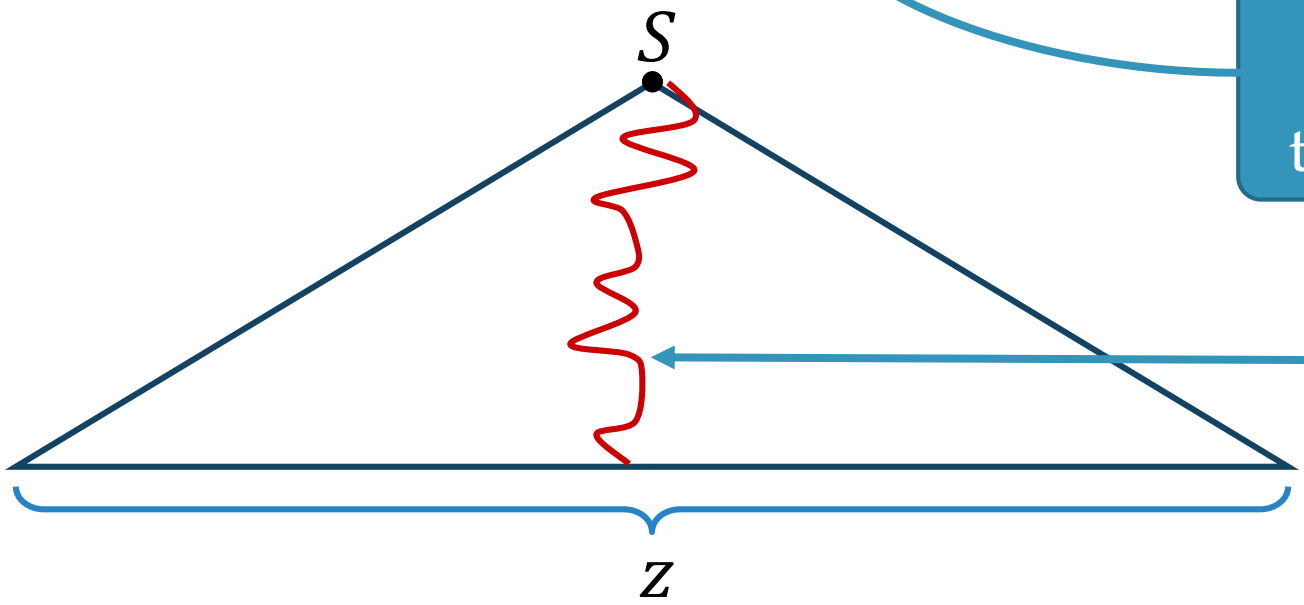
- $z = uvwxy$
- $|vx| \geq 1$
- $|vwx| \leq n$
- For all $i \geq 0$, $uv^i wx^i y \in L$.

- Let m equal the number of variables in the CFG G.
- Let $n = 2^m$.
- Suppose $z \in L(G)$, where $|z| \geq n$.
  - **Note**: $|z| > 2^{m-1}$
- We claim that any parse tree for z has a path of length $\geq m + 1$.
  - To see this, suppose all paths in a tree are shorter than $m + 1$ (no path is $> m$). Then, $|z| \leq 2^{m-1}$, contradicting the claim that $|z| > 2^{m-1}$.

Proof of the Pumping Lemma for CFLs

- $z = uvwxy$
- $|vx| \geq 1$
- $|vwx| \leq n$
- For all $i \geq 0$, $uv^i wx^i y \in L$.

- As stated on the previous slide, any parse tree for z has a path of length $\geq m + 1$.
- Such a path has at least $m + 2$ nodes, $m + 1$ of which are variables.



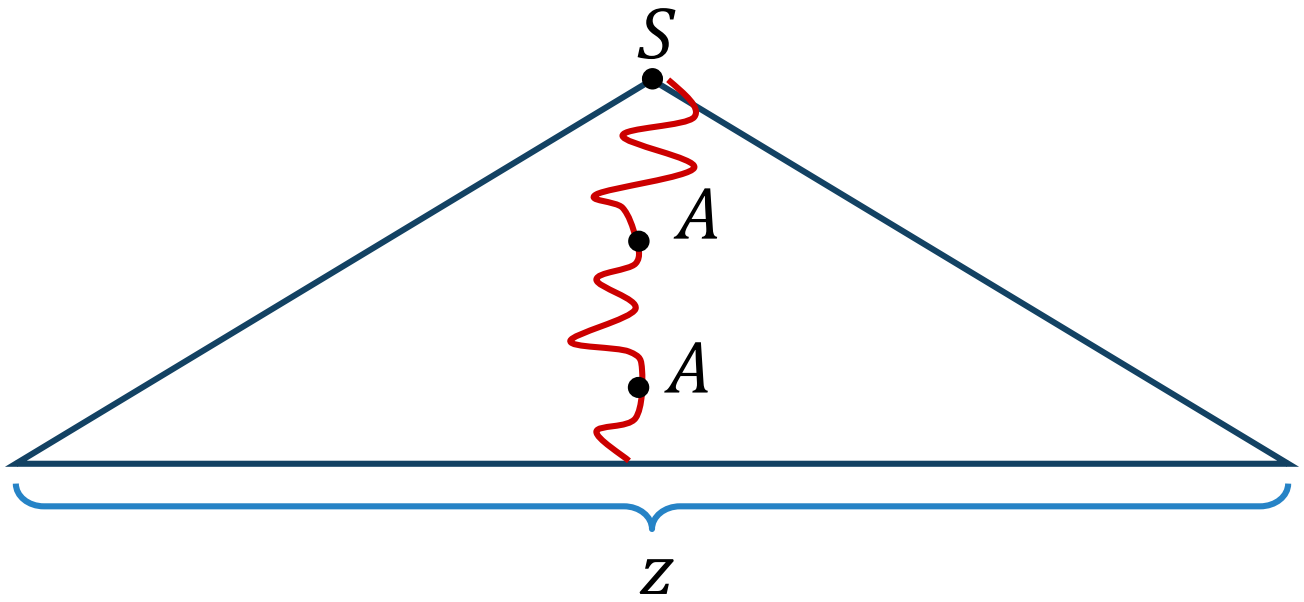
The CNF grammar requires replacing variables with a terminal at the end of the path.

Some path from the start symbol to a terminal in z must contain $m + 1$ variables.

Proof of the Pumping Lemma for CFLs

- $z = uvwxy$
- $|vx| \geq 1$
- $|vwx| \leq n$
- For all $i \geq 0$, $uv^i wx^i y \in L$.

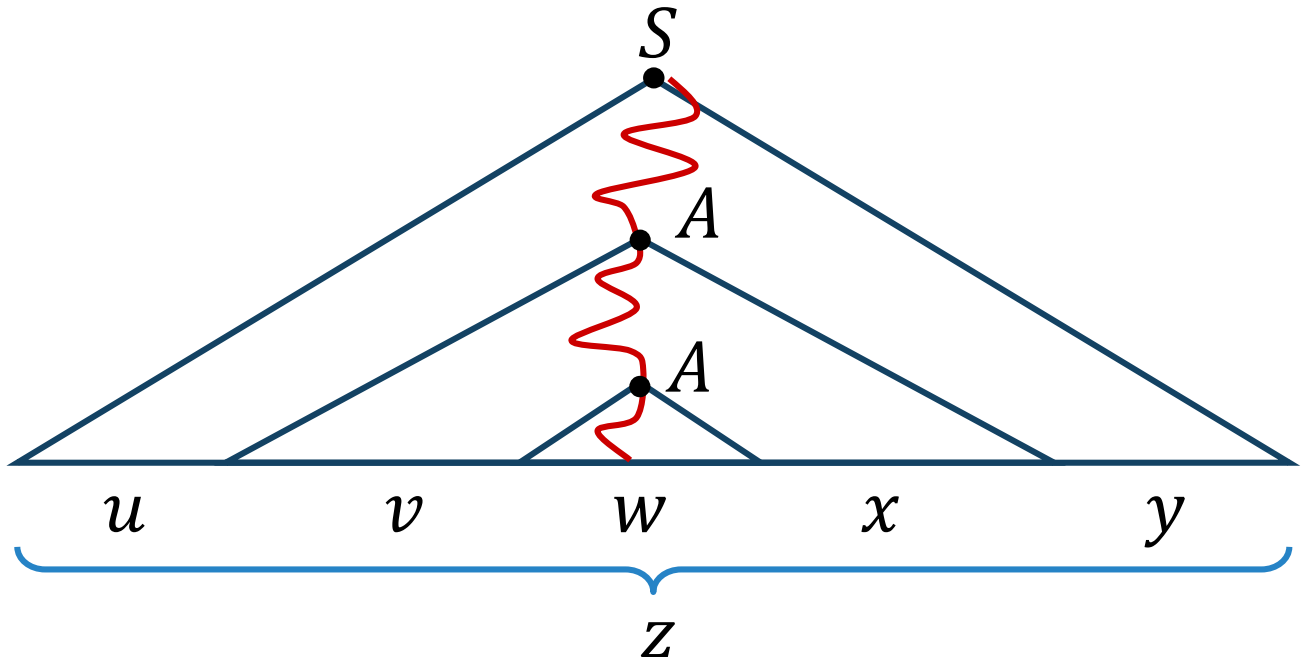
- Since the CFG contains m variables, but the path in z's parse tree contains $m + 1$ variables, *at least one variable must be repeated in the path*.
- If A is the repeated variable, the path looks like this:



Proof of the Pumping Lemma for CFLs

- $z = uvwxy$
- $|vx| \geq 1$
- $|vwx| \leq n$
- For all $i \geq 0$, $uv^iwx^iy \in L$.

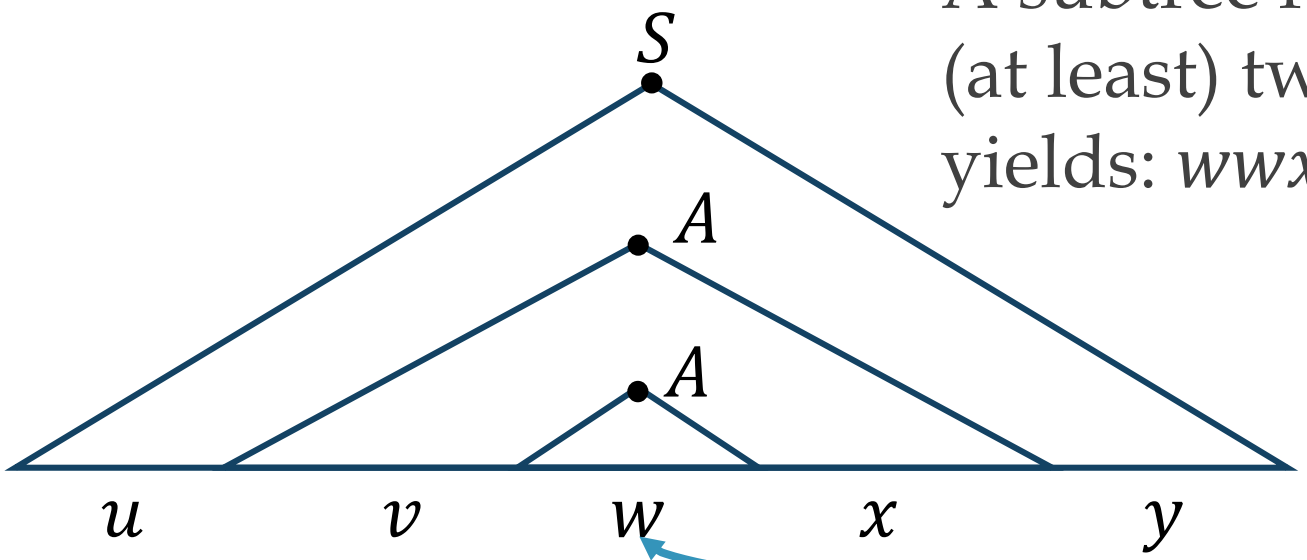
- Consider the subtrees rooted at each occurrence of A:



Proof of the Pumping Lemma for CFLs

- $z = uvwxy$
- $|vx| \geq 1$
- $|vwx| \leq n$
- For all $i \geq 0$, $uv^iwx^iy \in L$.

A subtree rooted at A has (at least) two possible yields: wwx and w.



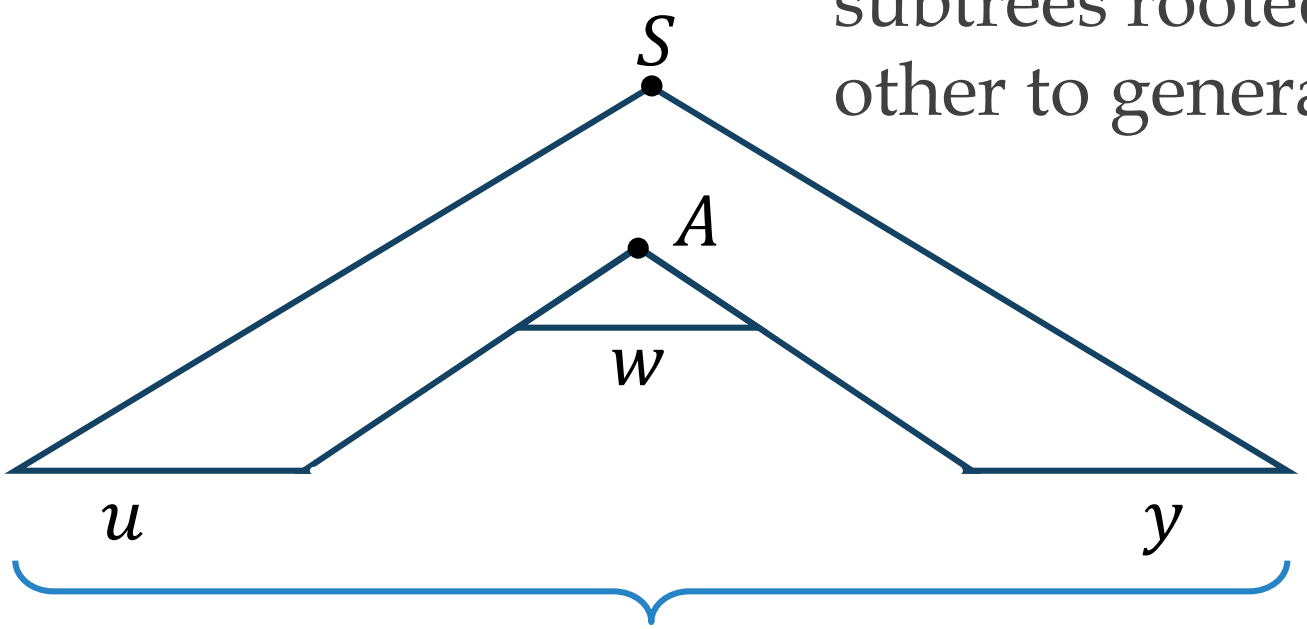
vwx = yield of first subtree of A

w = yield of second subtree of A

Proof of the Pumping Lemma for CFLs

- $z = uvwxy$
- $|vx| \geq 1$
- $|vwx| \leq n$
- For all $i \geq 0$, $uv^i wx^i y \in L$.

We can replace the possible subtrees rooted at A with each other to generate different strings.

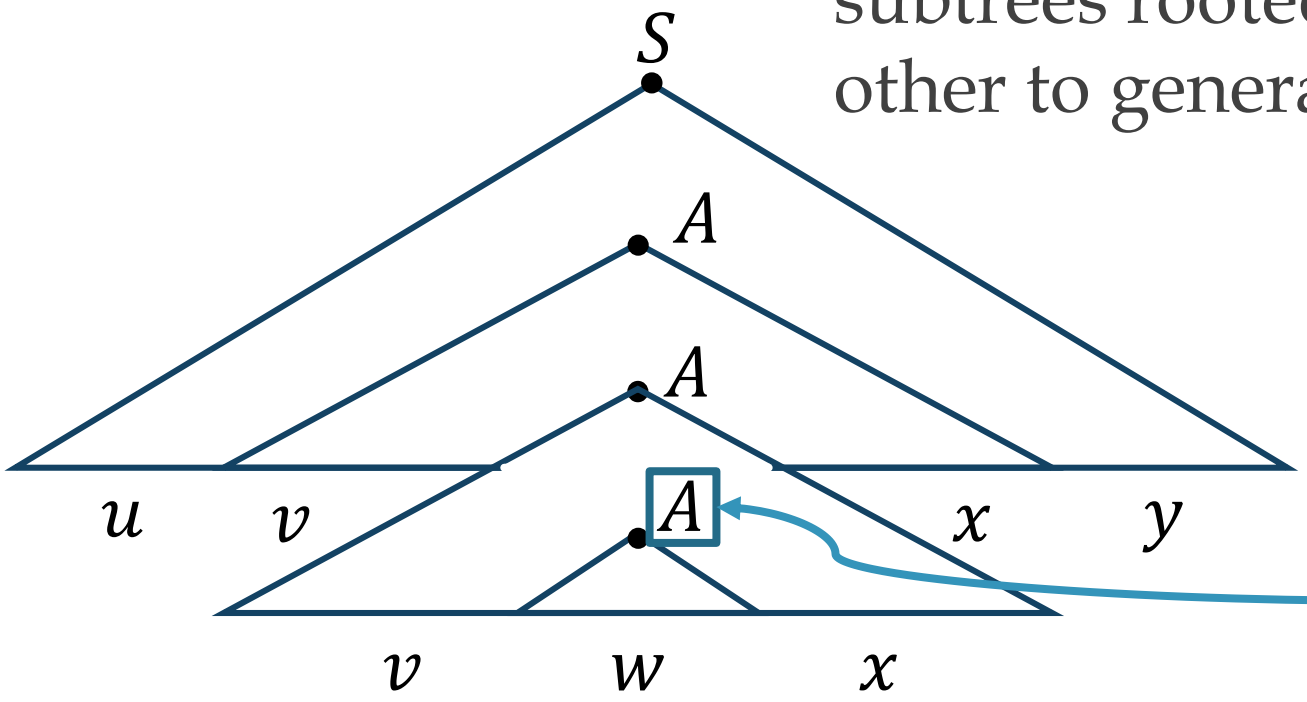


This tree has a yield $uwy = uv^0wx^0y$
This string must also be in L.

Proof of the Pumping Lemma for CFLs

- $z = uvwxy$
- $|vx| \geq 1$
- $|vwx| \leq n$
- For all $i \geq 0$, $uv^i wx^i y \in L$.

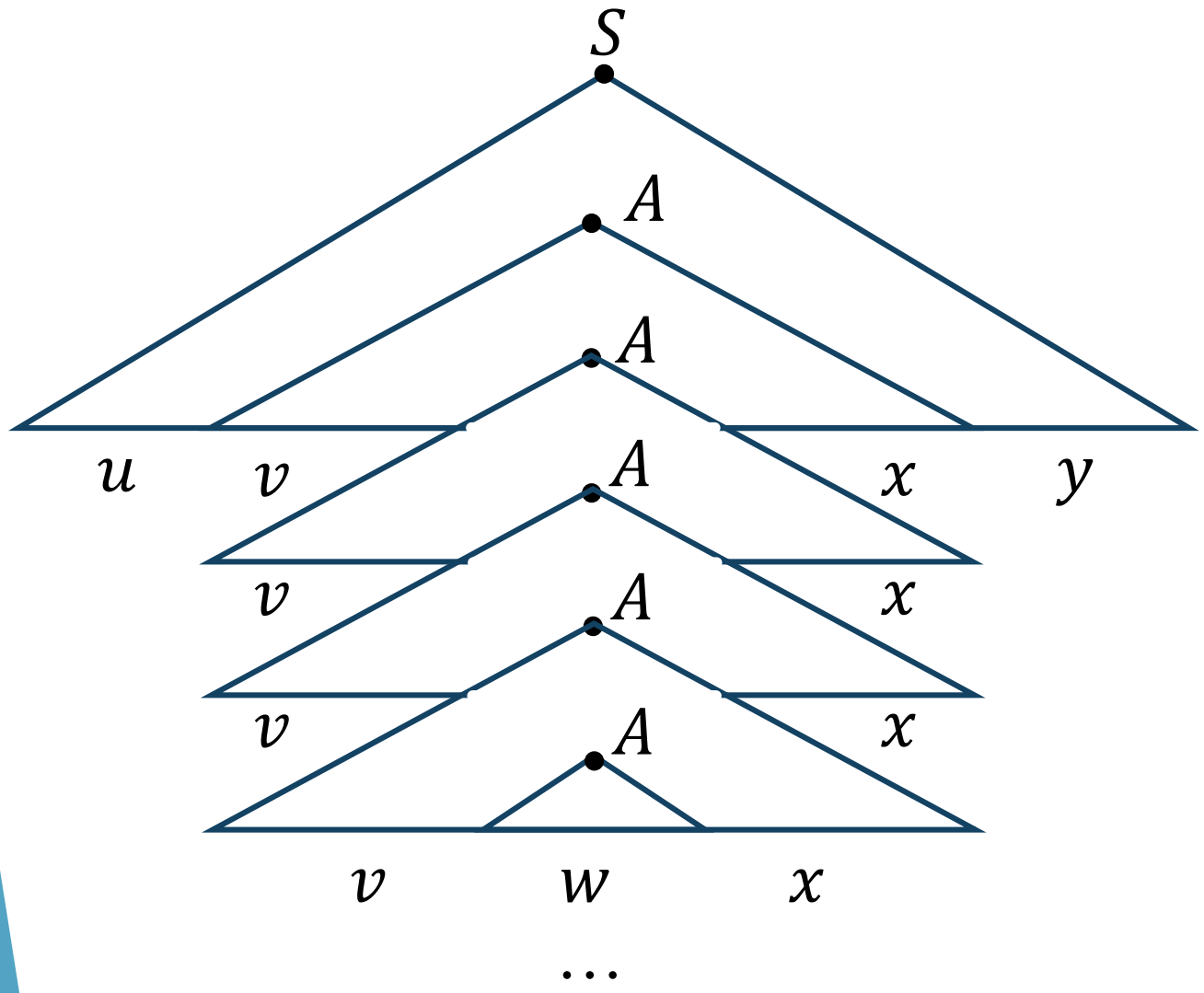
We can replace the possible subtrees rooted at A with each other to generate different strings.



Note that this introduces another occurrence of A.

This tree has a yield $uvvwxxy = uv^2wx^2y$
This string must also be in L.

Proof of the Pumping Lemma for CFLs



- $z = uvwxy$
- $|vx| \geq 1$
- $|vwx| \leq n$
- For all $i \geq 0$, $uv^iwx^iy \in L$.

We can repeat this process indefinitely to keep generating strings in L of the form uv^iwx^iy.

So, $uv^iwx^iy \in L$ for all $i \geq 0$.

Application of the CFL Pumping Lemma

- $L = \{a^i b^i c^i \mid i \geq 1\}$.
- Let the string $z = a^n b^n c^n$, for an arbitrary $n > 0$. Clearly, $z \in L$ for any n.
- Let $z = uvwxy$, $|vx| \geq 1$, and $|vwx| \leq n$. This means:
  - vx is all a's, b's, or c's, or
  - vx is all a's and b's or all b's and c's
  - (The key point is that vx can not possibly contain a's, b's, and c's all at the same time.)
- If vx contains only one type of symbol, uv^0wx^0y will contain too few of that symbol.
- If vx contains two types of symbols, uv^0wx^0y will contain too many of the symbol not in vx.
- Therefore, $uv^0wx^0y \notin L$, and the Pumping Lemma for CFLs does not hold for L.

Application of the CFL Pumping Lemma

- $L = \{ww | \, w \in (\mathbf{0} + \mathbf{1})^*\}$.
- Let $z = 0^n 1^n 0^n 1^n$. Clearly, $z \in L$ for any n.
- Let $z = uvwxy$, $|vx| \geq 1$, and $|vwx| \leq n$. This means:
  - vx is all 0s or all 1s
  - vx is some 0s followed by some 1s
  - vx is some 1s followed by some 0s.
- If vx contains only 0s or only 1s, uv^0wx^0y will contain too few 0s or 1s in one half of the string.
- If vx contained 0s followed by 1s, then either the first or second half of uv^0wx^0y will contain fewer 0s *and* 1s than the other half.
- If vx contained 1s followed by 0s (vx is in the middle of z), then uv^0wx^0y will have more 0s in the first group of 0s than the second group of 0s. (The number of 1s will also be similarly imbalanced.)
- In any of these cases, $uv^0wx^0y \notin L$.

Closure of CFLs under Substitution

- Recall that a homomorphism maps characters in some alphabet Σ to strings over another alphabet Δ.
- A homomorphism is actually a special case of *substitution*, which maps characters in one alphabet to *any string in a language* over another alphabet.
- Consider this example substitution f:
  - $\Sigma = \{0,1\}$, $\Delta = \{a, b\}$, $f(0) = \mathbf{a} + \mathbf{b}^*$, $f(1) = \mathbf{a}^*\mathbf{b}$.
  - $f(\mathbf{0}^*\mathbf{1}^*) = (\mathbf{a} + \mathbf{b}^*)^*(\mathbf{a}^*\mathbf{b})^*$.
  - (Note that this particular example uses regular languages.)

Closure of CFLs under Substitution

Theorem 7.23 (reworded): The CFLs are closed under substitution and, by extension, homomorphism.

Proof:

The main idea is to replace all terminals in a CFG with start symbols of another CFG.

- Let L be a CFL, and $L \subseteq \Sigma^*$. For all $a \in \Sigma$, let L_a be a CFL.
- Let $L = L(G)$. For all $a \in \Sigma$, let $L_a = L(G_a)$.
- Assume these grammars have distinct variables.

Closure under Substitution, Proof contd.

Let $G = (V, T, P, S)$, and for all $a \in \Sigma$, $G_a = (V_a, T_a, P_a, S_a)$

Define $G' = (V', T', P', S')$, where:

- $V' = (\bigcup_{a \in \Sigma} V_a) \cup V$
- $T' = \bigcup_{a \in \Sigma} T_a$
- $S' = S$
- $P' = \bigcup_{a \in \Sigma} P_a \cup \{A \to \alpha' \mid A \to \alpha$ is in P, and $\alpha' = \alpha$ with each $a \in \Sigma$ replaced by $S_a\}$.

The language defined by substitution equals $L(G')$. (The proof is left as an exercise—or see Theorem 7.23 in the book.)

Union, Concatenation, and Closure

Theorem 7.24: CFLs are closed under Union, Concatenation, *-closure, and $^+$-closure.

Notation similar to *, but meaning "1 or more repetitions".

Proof:

Union: Let L_1 and L_2 be CFLs. $L_1 \cup L_2 = s(L)$, where $L = \{1, 2\}$ (which is clearly a CFL), and s is the substitution defined by $s(1) = L_1$ and $s(2) = L_2$.

The proofs for the others are similar.

Closure under Reversal

Theorem 7.25: CFLs are closed under reversal.

Proof:

If $L = L(G)$, where $G = (V, T, P, S)$, then $L^R = L(G^R)$, where $G^R = (V, T, P^R, S)$, and $P^R = \{A \to \alpha^R \mid \text{A} \to \alpha \text{ is a production in } P\}$.

(The full formal proof is left as an exercise.)

The productions in G^R are just the productions in G written backwards.

(Lack of) Closure under Intersection

Theorem: CFLs are *not* closed under intersection.

Proof:

- $L_1 = \{a^i b^i c^i \mid i \geq 1\}$. We know L_1 isn't a CFL from earlier slides.
- $L_2 = \{a^i b^i c^j \mid i \geq 1, j \geq 1\}$. This *is* a CFL.
- $L_3 = \{a^i b^j c^j \mid i \geq 1, j \geq 1\}$. This is also a CFL.
- However, $L_1 = L_2 \cap L_3$.

See example 7.26 in the book for CFGs.

(Lack of) Closure under Complementation

Corollary to the previous theorem: CFLs are *not* closed under complementation.

Proof:

- CFLs are closed under union.
- $L_1 \cap L_2 \equiv \overline{\overline{L_1} \cup \overline{L_2}}$.
- So, if CFLs are closed under complementation, they would be closed under intersection, too.

Intersection with Regular Languages

Theorem 7.26: If L is a CFL and R is a regular language, then $L \cap R$ is a CFL.

Proof:

Let L be the language of some PDA $P = (Q_P, \Sigma, \Gamma, \delta_P, q_P, Z_0, F_P)$.

Let R be the language of some DFA $A = (Q_A, \Sigma, \delta_A, q_A, F_A)$.

Idea: Create a new PDA combining the states of P and A, similar to combining two DFAs.

Proof: Intersection with Reg. Languages

Let $P' = (Q_P \times Q_A, \Sigma, \Gamma, \delta, (q_P, q_A), Z_0, F_P \times F_A)$, where:

- $\delta\big((q,p),a,X\big)$ contains $\big((r,s),\gamma\big)$ if and only if $\delta_A(p,a) = s$ and $\delta_P(q,a,X)$ contains (r,γ).

Claim: $\big((q_P,q_A),w,Z_0\big) \overset{i}{\underset{P'}{\vdash}} \big((q,p),\varepsilon,\gamma\big)$ if and only if $(q_P,w,Z_0) \overset{i}{\underset{P}{\vdash}} (q,\varepsilon,\mathrm{Y})$ and $\delta(q_A,w) = p$.

(This can be proven by induction on i.)

Closure under Inverse Homomorphism

Theorem 7.30: CFLs are closed under inverse homomorphism.

Proof:

Consider L, where L is the language of some PDA P.

$P = (Q, \Delta, \Gamma, \delta, q_0, Z_0, F)$.

Let $h: \Sigma \rightarrow \Delta^*$.

Construct a PDA P' that accepts $h^{-1}(L)$.

Closure under Inverse Homomorphism

Proof, continued:

The key idea is the same as for regular languages: When processing an input a, P' simulates P on the input $h(a)$.

In A DFA, simulation required just a single state transition.

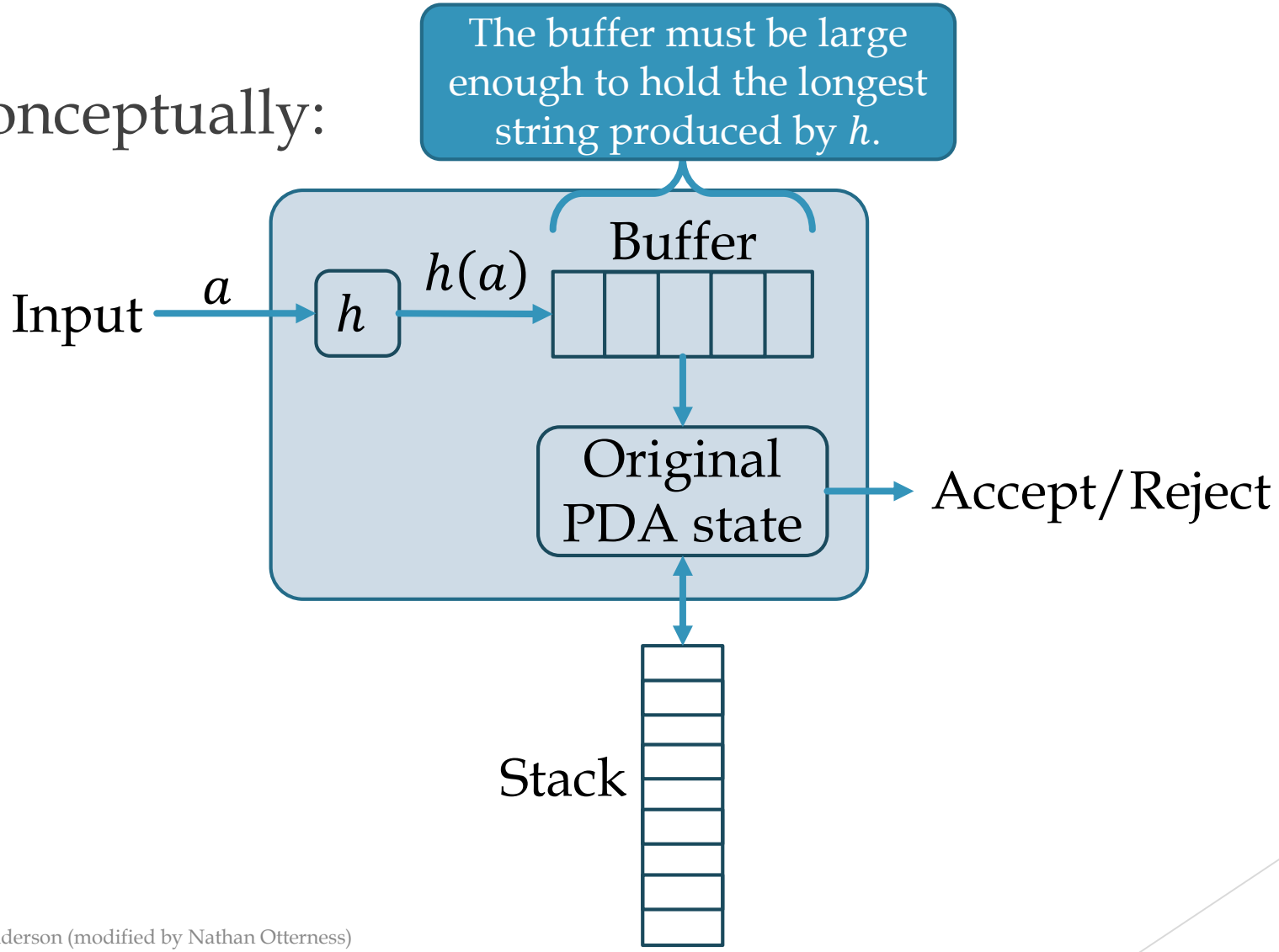
However, P, being a PDA, does more than just change state on input $h(a)$--it may change the stack contents or make nondeterministic choices.

Solution: Use a *buffer* to hold the symbols of $h(a)$.

This buffer will really be part of P''s (*finite!*) state.

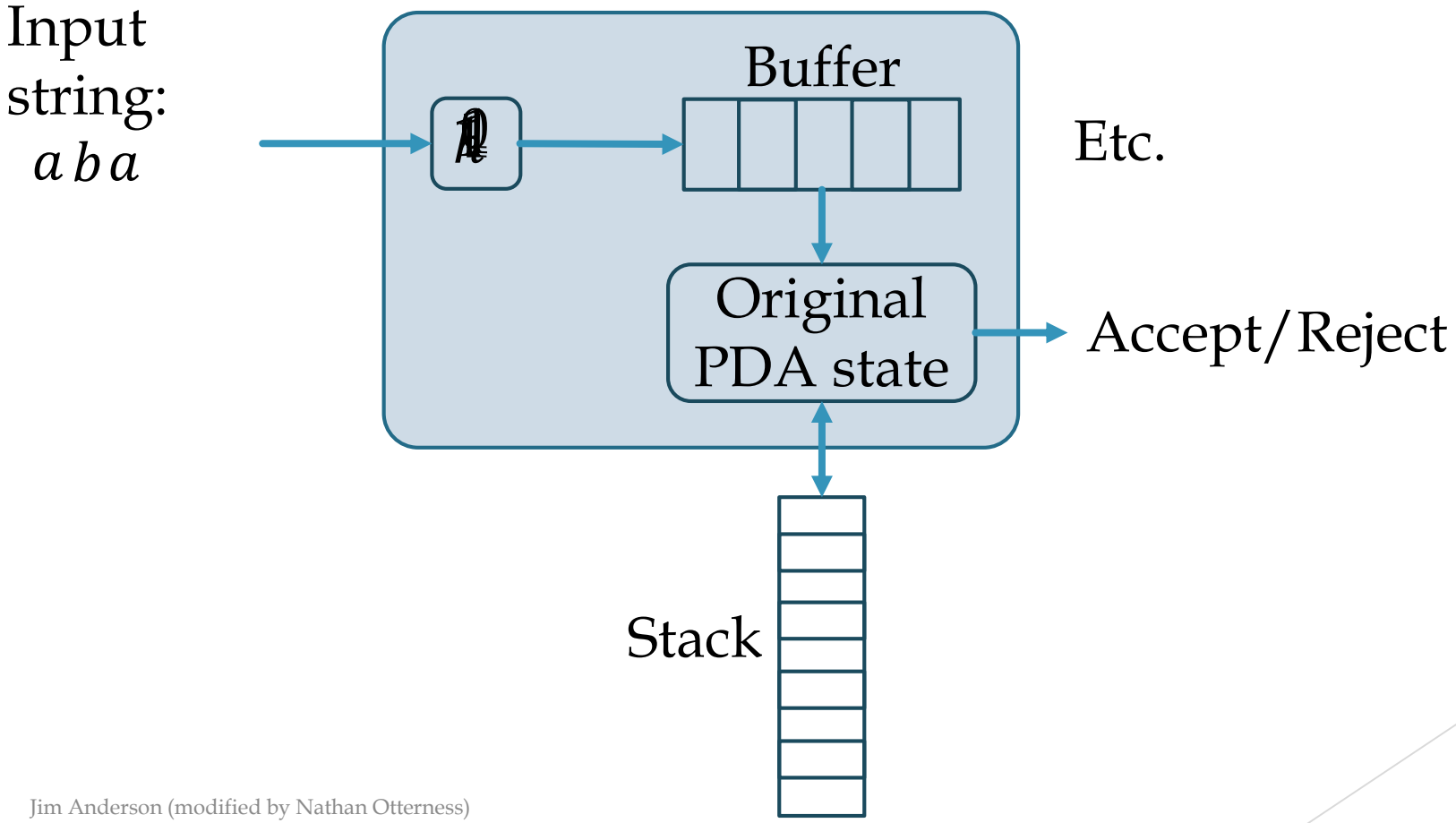
Closure under Inverse Homomorphism

Conceptually:



Closure under Inverse Homomorphism

Short example: $h(a) = 01$, and $h(b) = 111$



Closure under Inverse Homomorphism

- We have a homomorphism $h: \Sigma \to \Delta^*$.
- Recall that $P = (Q, \Delta, \Gamma, \delta, q_0, Z_0, F)$
- Let $P' = (Q', \Sigma, \Gamma, \delta', \boxed{(q_0, \varepsilon)}, Z_0, \boxed{F \times \{\varepsilon\}})$.
- States in Q' is a set of pairs (q, x) such that
  - q is a state in Q, and
  - x is the finite "buffer" — $h(a)$ or a suffix of $h(a)$ for some symbol $a \in \Sigma$.

Start in the original start state, with an empty buffer.

Accept in any of the original accepting states, if the buffer is empty.

Closure under Inverse Homomorphism

Transitions in P':

- $\delta'\big((q,x),\varepsilon,X\big)$ contains all $\big((p,x),\gamma\big)$ such that $\delta(q,\varepsilon,X)$ contains (p,γ).
  - "Simulate ε-transitions"
- $\delta'\big((q,bx),\varepsilon,X\big)$ contains all $\big((p,x),\gamma\big)$ such that $\delta(q,b,X)$ contains (p,γ).
  - "Simulate non-ε transitions"
- $\delta'\big((q,\varepsilon),a,X\big)$ contains $\Big(\big(q,h(a)\big),X\Big)$ for all $a \in \Sigma$ and $X \in \Gamma$.
  - "Load the buffer"

Closure under Inverse Homomorphism

Claim: If $(q_0, h(w), Z_0) \overset{*}{\underset{P}{\vdash}} (p, \varepsilon, \gamma)$, then

$\big((q_0, \varepsilon), w, Z_0\big) \overset{*}{\underset{P'}{\vdash}} \big((p, \varepsilon), \varepsilon, \gamma\big)$.

Proof sketch:

For each $a \in \Sigma$, whatever sequence of moves P makes on $h(a)$, P' can make a corresponding sequence of moves on input a.

Closure under Inverse Homomorphism

Claim: If $\big((q_0, \varepsilon), w, Z_0\big) \overset{*}{\underset{P'}{\vdash}} \big((p, \varepsilon), \varepsilon, \gamma\big)$, then $(q_0, h(w), Z_0) \overset{*}{\underset{P}{\vdash}} (p, \varepsilon, \gamma)$.

Proof sketch:

P' can only process w one character at a time. For each character a, P' does what P does on $h(a)$.

The first claim implied that $L(P') \supseteq h^{-1}\big(L(P)\big)$.

This claim implies that $L(P') \subseteq h^{-1}\big(L(P)\big)$.

Decision Properties of CFLs

As with regular languages, we'll focus less on efficiency and more on simplicity than what's done in the book.

- **Theorem**: There exist algorithms to determine if a CFL is **(a)** empty, **(b)** finite, or **(c)** infinite.

Detecting if a CFL is Empty

Nonemptiness:

- Use the iterative algorithm for detecting useless symbols to test if $A \overset{*}{\Rightarrow} w$ for all $A \in V$.
- The CFL is nonempty if and only if $S \overset{*}{\Rightarrow} w$ for some terminal string w.

Detecting if a CFL is Finite or Infinite

Assume L does not contain ε. (If $\varepsilon \in L$, then consider $L - \{\varepsilon\}$ instead.)

$L - \{\varepsilon\}$ will be finite if and only if L is finite.

Suppose $L = L(G)$, where $G = (V, T, P, S)$ is in CNF (and therefore has no useless symbols).

Consider the directed graph (V, E), where $(A, B) \in E$ if $A \to BC$ or $A \to CB$ is in P for some variable C.

Claim: L is finite if and only if (V, E) has no cycles.

Detecting if a CFL is Finite or Infinite

Claim (again): L is finite if and only if (V, E) has no cycles.

"Only if": A cycle has the form $A_0, A_1, \ldots, A_n, A_0$.

Therefore, we have $A_0 \Rightarrow \alpha_1 A_1 \beta_1 \Rightarrow \alpha_2 A_2 \beta_2 \Rightarrow \cdots \Rightarrow \alpha_n A_n \beta_n \Rightarrow \alpha_{n+1} A_0 \beta_{n+1}$, where $|\alpha_i \beta_i| = i$.

This is due to the grammar being in CNF—we can only produce i symbols in i derivation steps.

Since G has no useless symbols:

- $\alpha_{n+1} \overset{*}{\Rightarrow} w$,
- $\beta_{n+1} \overset{*}{\Rightarrow} x$,
- $S \overset{*}{\Rightarrow} y A_0 z$, and
- $A_0 \overset{*}{\Rightarrow} v$,

where w, x, y, z, and v are terminal strings.

Detecting if a CFL is Finite or Infinite

"Only if" proof, continued:

From:

- $\alpha_{n+1} \overset{*}{\Rightarrow} w$
- $\beta_{n+1} \overset{*}{\Rightarrow} x$
- $S \overset{*}{\Rightarrow} yA_0z$
- $A_0 \overset{*}{\Rightarrow} v$

we have:

$S \overset{*}{\Rightarrow} yA_0x \overset{*}{\Rightarrow} ywA_0xz \overset{*}{\Rightarrow} yw^2A_0x^2z \overset{*}{\Rightarrow} \ldots \overset{*}{\Rightarrow} yw^iA_0x^iz \overset{*}{\Rightarrow} yw^ivx^iz.$

Therefore, L is infinite.

Detecting if a CFL is Finite or Infinite

Claim (again): L is finite if and only if (V, E) has no cycles.

"If": Suppose (V, E) has no cycles.

- **Definition**: The *rank* of $A \in V$ is the longest path beginning from A. If a graph has no cycles, then all ranks are finite.
- **Claim**: If A has a rank r, then A derives no terminal string of length exceeding 2^r.
  - This can be proven by induction on r, similarly to some claims prior to the Pumping Lemma.
- Since (V, E) has no cycles, S has a finite rank. Therefore, the longest string derivable from S has a finite length, so L must be finite.

Membership: CYK Algorithm

We want to know if a string x is in L.

- Let $L - \{\varepsilon\} = L(G)$ for a CFG G in CNF.
- Let $|x| = n$.
- Let x_{ij} be a substring of x of length j beginning at position i.
- Inductively determine all variables A such that $A \overset{*}{\Rightarrow} x_{ij}$.
- $x \in L$ if and only if $S \overset{*}{\Rightarrow} x_{1n}$.

Membership: CYK Algorithm

$j = 1$, so $x_{ij} = x_{i1}$. x_{i1} has a length of 1, so it's a terminal symbol.

Base case: $j = 1$. $A \overset{*}{\Rightarrow} x_{ij}$ if and only if $A \rightarrow x_{ij}$.

Inductive step: $j > 1$. $A \overset{*}{\Rightarrow} x_{ij}$ if and only if there exists $A \rightarrow BC$ and k, $1 \leq k < j$, such that $B \overset{*}{\Rightarrow} x_{ik}$ and $C \overset{*}{\Rightarrow} x_{i+k\ j-k}$.

Membership: CYK Algorithm

Pseudocode for determining the sets of variables, V_{ij}, that can produce the substring x_{ij}:

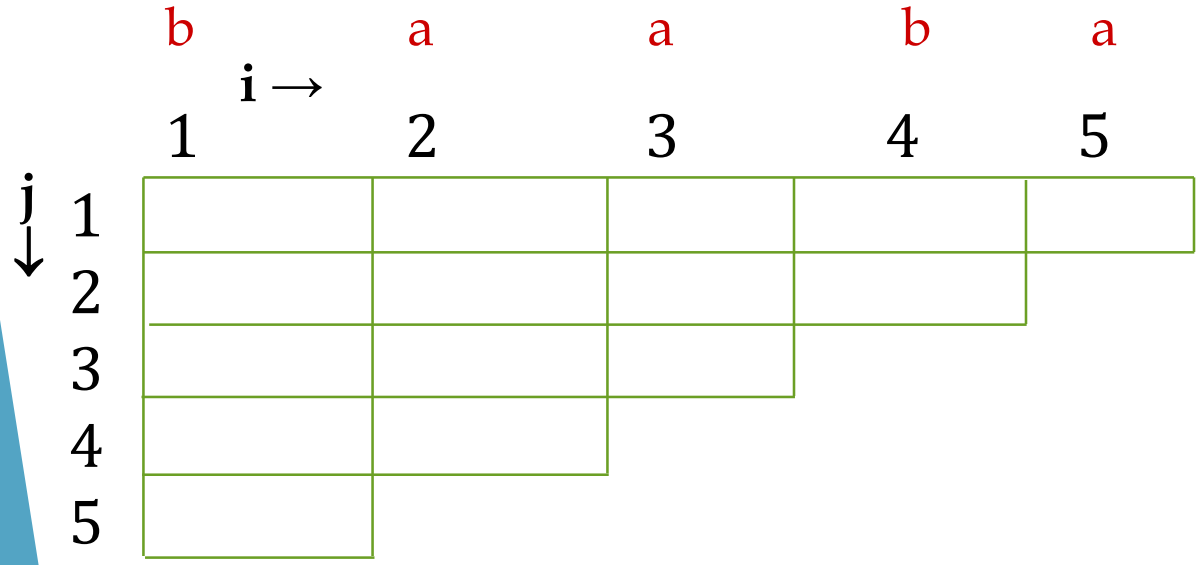
```
for i := 1 to n:
  V_i1 := {A | A → x_i1 }

for j := 2 to n:
  for i := 1 to n – j + 1:
    V_ij := ∅;
    for k := 1 to j – 1:
      V_ij := V_ij ∪ {A | A → BC, B ∈ V_ik, C ∈ V_i+k j-k}
```

The time complexity is $O(n^3)$.

CYK Algorithm: Example

$S \rightarrow AB \mid BC$
$A \rightarrow BA \mid a$
$B \rightarrow CC \mid b$
$C \rightarrow AB \mid a$
$x = baaba\ (n = 5)$




```
for i := 1 to n:
  V_i1 := {A | A → x_i1 }

for j := 2 to n:
  for i := 1 to n – j + 1:
    V_ij := ∅;
    for k := 1 to j – 1:
      V_ij := V_ij ∪ {A | A → BC, B ∈ V_ik, C ∈ V_i+k j-k}
```

CYK Algorithm: Example

$S \rightarrow AB \mid BC$
$A \rightarrow BA \mid a$
$B \rightarrow CC \mid b$
$C \rightarrow AB \mid a$
$x = baaba\ (n = 5)$

```
for i := 1 to n:
   V_i1 := {A | A → x_i1 }

for j := 2 to n:
   for i := 1 to n – j + 1:
      V_ij := ∅;
      for k := 1 to j – 1:
         V_ij := V_ij ∪ {A | A → BC, B ∈ V_ik, C ∈ V_i+k j-k}
```

	b	a	a	b	a
	i → 1	2	3	4	5
j ↓ 1	B	A, C	A, C	B	A,C
2					
3					
4					
5					

CYK Algorithm: Example

$S \rightarrow AB \mid BC$
$A \rightarrow BA \mid a$
$B \rightarrow CC \mid b$
$C \rightarrow AB \mid a$
$x = baaba\ (n = 5)$

```
for i := 1 to n:
   V_i1 := {A | A → x_i1 }

for j := 2 to n:
   for i := 1 to n – j + 1:
      V_ij := ∅;
      for k := 1 to j – 1:
         V_ij := V_ij ∪ {A | A → BC, B ∈ V_ik, C ∈ V_i+k j-k}
```

	i → 1	2	3	4	5
	b	a	a	b	a
j ↓ 1	*B*	*A, C*	*A, C*	*B*	*A,C*
2	*S, A*	*B*	*S, C*	*S, A*	
3					
4					
5					

CYK Algorithm: Example

$S \rightarrow AB \mid BC$
$A \rightarrow BA \mid a$
$B \rightarrow CC \mid b$
$C \rightarrow AB \mid a$
$x = baaba\ (n = 5)$

```
for i := 1 to n:
   V_i1 := {A | A → x_i1 }

for j := 2 to n:
   for i := 1 to n – j + 1:
      V_ij := ∅;
      for k := 1 to j – 1:
         V_ij := V_ij ∪ {A | A → BC, B ∈ V_ik, C ∈ V_i+k j-k}
```

	i → 1	2	3	4	5
	b	a	a	b	a
j ↓ 1	B	A, C	A, C	B	A,C
2	S, A	B	S, C	S, A	
3	$\emptyset$	B	B		
4					
5					

CYK Algorithm: Example

$S \rightarrow AB \mid BC$
$A \rightarrow BA \mid a$
$B \rightarrow CC \mid b$
$C \rightarrow AB \mid a$
$x = baaba\ (n = 5)$

```
for i := 1 to n:
   V_i1 := {A | A → x_i1 }

for j := 2 to n:
   for i := 1 to n – j + 1:
      V_ij := ∅;
      for k := 1 to j – 1:
         V_ij := V_ij ∪ {A | A → BC, B ∈ V_ik, C ∈ V_i+k j-k}
```

	i → 1	2	3	4	5
	b	a	a	b	a
j ↓ 1	B	A, C	A, C	B	A,C
2	S, A	B	S, C	S, A	
3	$\varnothing$	B	B		
4	$\varnothing$	S, A, C			
5					

CYK Algorithm: Example

$S \rightarrow AB \mid BC$
$A \rightarrow BA \mid a$
$B \rightarrow CC \mid b$
$C \rightarrow AB \mid a$
$x = baaba\ (n = 5)$

```
for i := 1 to n:
   V_i1 := {A | A → x_i1 }

for j := 2 to n:
   for i := 1 to n – j + 1:
      V_ij := ∅;
      for k := 1 to j – 1:
         V_ij := V_ij ∪ {A | A → BC, B ∈ V_ik, C ∈ V_i+k j-k}
```

	i → 1	2	3	4	5
	b	a	a	b	a
j ↓ 1	B	A, C	A, C	B	A,C
2	S, A	B	S, C	S, A	
3	$\emptyset$	B	B		
4	$\emptyset$	S, A, C			
5	S, A, C				

$S \in V_{1n}$, so $x \in L$.