# COMP-330
# Theory of Computation

Fall 2019 -- Prof. Claude Crépeau

# Lec. 13 :
# Pumping Lemma for CFLs

Announcements

## EXCEPTIONAL OFFICE HOURS

Posted Oct 14, 2019 11:44 AM

On Wednesday (Oct 16) I will be attending an external meeting during most of the day. My office hours will exceptionally be held from 11:00 to 14:00, same place as usual, McConnell 110N.

Claude

---

## Office Hours

Posted Oct 11, 2019 6:06 PM

Hi,

As next Monday is a holiday. I will have the OH for next week on Wednesday between 11.30am - 12.30pm.

Regards,

Anirudha

# PDA to CFG

**PROOF** Say that $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{\text{accept}}\})$ and construct $G$. The variables of $G$ are $\{A_{pq} | \, p, q \in Q\}$. The start variable is $A_{q_0, q_{\text{accept}}}$. Now we describe $G$'s rules.

- For each $p, q, r, s \in Q$, $t \in \Gamma$, and $a, b \in \Sigma_\varepsilon$, if $\delta(p, a, \varepsilon)$ contains $(r, t)$ and $\delta(s, b, t)$ contains $(q, \varepsilon)$, put the rule $A_{pq} \rightarrow a A_{rs} b$ in $G$.
- For each $p, q, r \in Q$, put the rule $A_{pq} \rightarrow A_{pr} A_{rq}$ in $G$.
- Finally, for each $p \in Q$, put the rule $A_{pp} \rightarrow \varepsilon$ in $G$.

# PDA to CFG

**CLAIM 2.30**

If $A_{pq}$ generates $x$, then $x$ can bring $P$ from a state $p$ (with an empty stack) to a state $q$ (with an empty stack).

We prove this claim by induction on the number of steps in the derivation of $x$ from $A_{pq}$.

If $A_{pq}$ generates $x$, then $x$ can bring $P$ from a state $p$ (with an empty stack) to a state $q$ (with an empty stack).

***Basis:*** The derivation has 1 step.

A derivation with a single step must use a rule whose right-hand side contains no variables. The only rules in $G$ where no variables occur on the right-hand side are $A_{pp} \to \varepsilon$. Clearly, input $\varepsilon$ takes $P$ from $p$ with empty stack to $p$ with empty stack so the basis is proved.

If $A_{pq}$ generates $x$, then $x$ can bring $P$ from a state $p$ (with an empty stack) to a state $q$ (with an empty stack).

***Basis:*** The derivation has 1 step.
A derivation with a single step must use a rule whose right-hand side contains no variables. The only rules in $G$ where no variables occur on the right-hand side are $A_{pp} \to \varepsilon$. Clearly, input $\varepsilon$ takes $P$ from $p$ with empty stack to $p$ with empty stack so the basis is proved.

***Induction step:*** Assume true for derivations of length at most $k$, where $k \geq 1$, and prove true for derivations of length $k + 1$.

If $A_{pq}$ generates $x$, then $x$ can bring $P$ from a state $p$ (with an empty stack) to a state $q$ (with an empty stack).

***Basis:*** The derivation has 1 step.
A derivation with a single step must use a rule whose right-hand side contains no variables. The only rules in $G$ where no variables occur on the right-hand side are $A_{pp} \to \varepsilon$. Clearly, input $\varepsilon$ takes $P$ from $p$ with empty stack to $p$ with empty stack so the basis is proved.

***Induction step:*** Assume true for derivations of length at most $k$, where $k \geq 1$, and prove true for derivations of length $k+1$.

Suppose that $A_{pq} \overset{*}{\Rightarrow} x$ with $k+1$ steps. The first step in this derivation is either $A_{pq} \Rightarrow aA_{rs}b$ or $A_{pq} \Rightarrow A_{pr}A_{rq}$. We handle these two cases separately.

If $A_{pq}$ generates $x$, then $x$ can bring $P$ from a state $p$ (with an empty stack) to a state $q$ (with an empty stack).

In the first case, consider the portion $y$ of $x$ that $A_{rs}$ generates, so $x = ayb$. Because $A_{rs} \overset{*}{\Rightarrow} y$ with $k$ steps, the induction hypothesis tells us that $P$ can go from $r$ on empty stack to $s$ on empty stack. Because $A_{pq} \rightarrow aA_{rs}b$ is a rule of $G$, $\delta(p, a, \varepsilon)$ contains $(r, t)$ and $\delta(s, b, t)$ contains $(q, \varepsilon)$, for some stack symbol $t$. Hence, if $P$ starts at $p$ with an empty stack, after reading $a$ it can go to state $r$ and push $t$ onto the stack. Then reading string $y$ can bring it to $s$ and leave $t$ on the stack. Then after reading $b$ it can go to state $q$ and pop $t$ off the stack. Therefore $x$ can bring it from $p$ with empty stack to $q$ with empty stack.

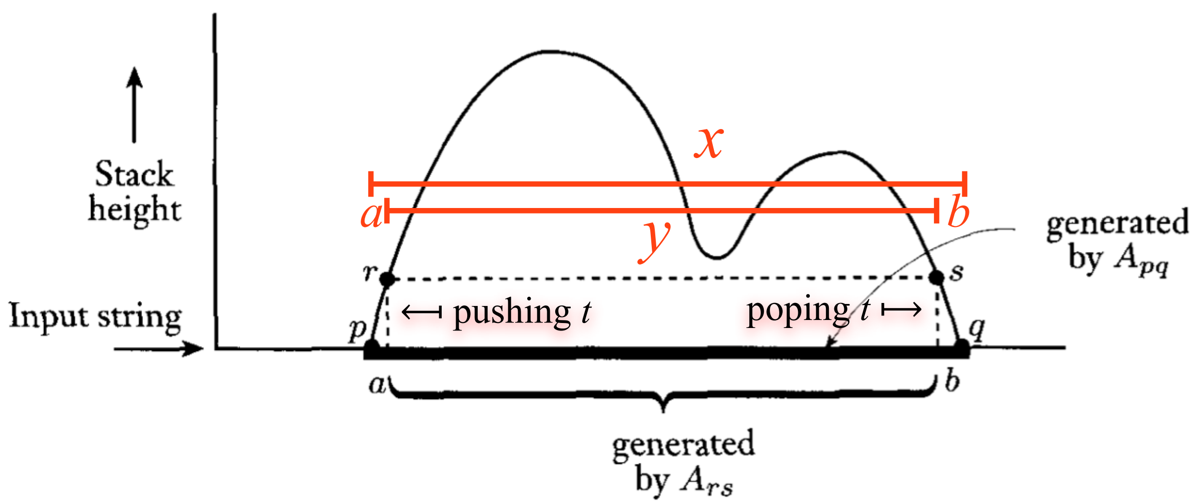



**FIGURE 2.29**
PDA computation corresponding to the rule $A_{pq} \rightarrow aA_{rs}b$

If $A_{pq}$ generates $x$, then $x$ can bring $P$ from a state $p$ (with an empty stack) to a state $q$ (with an empty stack).

$A_{pq} \Rightarrow aA_{rs}b$

In the first case, consider the portion $y$ of $x$ that $A_{rs}$ generates, so $x = ayb$. Because $A_{rs} \overset{*}{\Rightarrow} y$ with $k$ steps, the induction hypothesis tells us that $P$ can go from $r$ on empty stack to $s$ on empty stack. Because $A_{pq} \to aA_{rs}b$ is a rule of $G$, $\delta(p, a, \varepsilon)$ contains $(r, t)$ and $\delta(s, b, t)$ contains $(q, \varepsilon)$, for some stack symbol $t$. Hence, if $P$ starts at $p$ with an empty stack, after reading $a$ it can go to state $r$ and push $t$ onto the stack. Then reading string $y$ can bring it to $s$ and leave $t$ on the stack. Then after reading $b$ it can go to state $q$ and pop $t$ off the stack. Therefore $x$ can bring it from $p$ with empty stack to $q$ with empty stack.

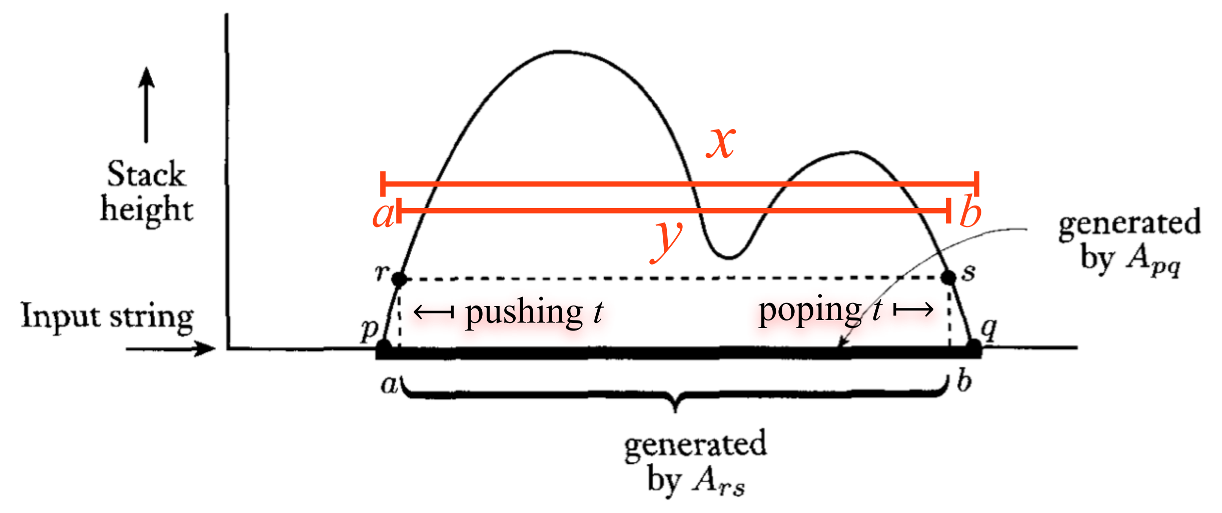



**FIGURE 2.29**
PDA computation corresponding to the rule $A_{pq} \to aA_{rs}b$

If $A_{pq}$ generates $x$, then $x$ can bring $P$ from a state $p$ (with an empty stack) to a state $q$ (with an empty stack).

In the second case, consider the portions $y$ and $z$ of $x$ that $A_{pr}$ and $A_{rq}$ respectively generate, so $x = yz$. Because $A_{pr} \overset{*}{\Rightarrow} y$ in at most $k$ steps and $A_{rq} \overset{*}{\Rightarrow} z$ in at most $k$ steps, the induction hypothesis tells us that $y$ can bring $P$ from $p$ to $r$, and $z$ can bring $P$ from $r$ to $q$, with empty stacks at the beginning and end. Hence $x$ can bring it from $p$ with empty stack to $q$ with empty stack. This completes the induction step.

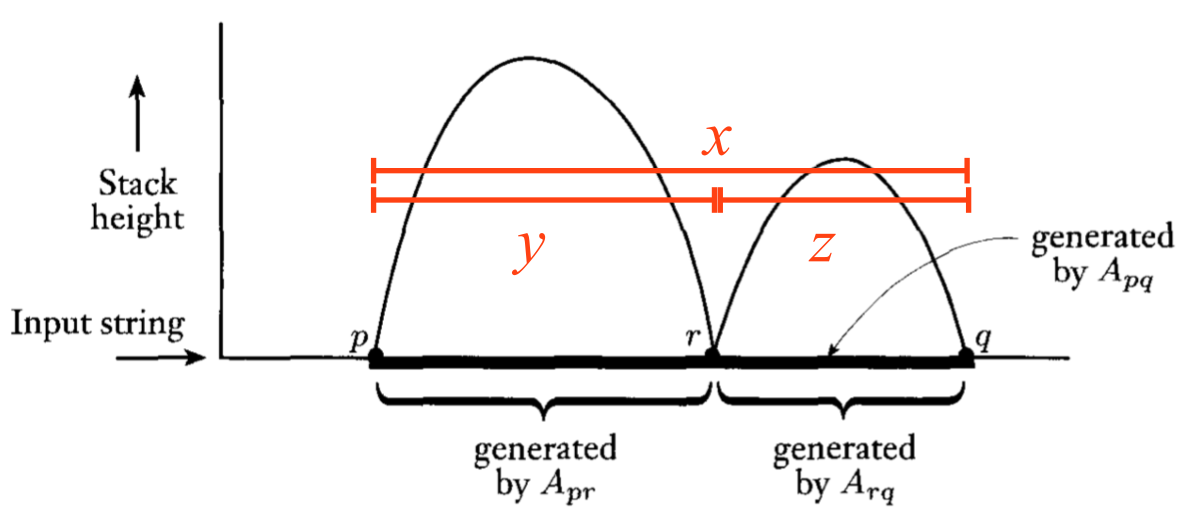



**FIGURE 2.28**
PDA computation corresponding to the rule $A_{pq} \rightarrow A_{pr}A_{rq}$

If $A_{pq}$ generates $x$, then $x$ can bring $P$ from a state $p$ (with an empty stack) to a state $q$ (with an empty stack).

$A_{pq} \Rightarrow A_{pr} A_{rq}$

In the second case, consider the portions $y$ and $z$ of $x$ that $A_{pr}$ and $A_{rq}$ respectively generate, so $x = yz$. Because $A_{pr} \overset{*}{\Rightarrow} y$ in at most $k$ steps and $A_{rq} \overset{*}{\Rightarrow} z$ in at most $k$ steps, the induction hypothesis tells us that $y$ can bring $P$ from $p$ to $r$, and $z$ can bring $P$ from $r$ to $q$, with empty stacks at the beginning and end. Hence $x$ can bring it from $p$ with empty stack to $q$ with empty stack. This completes the induction step.

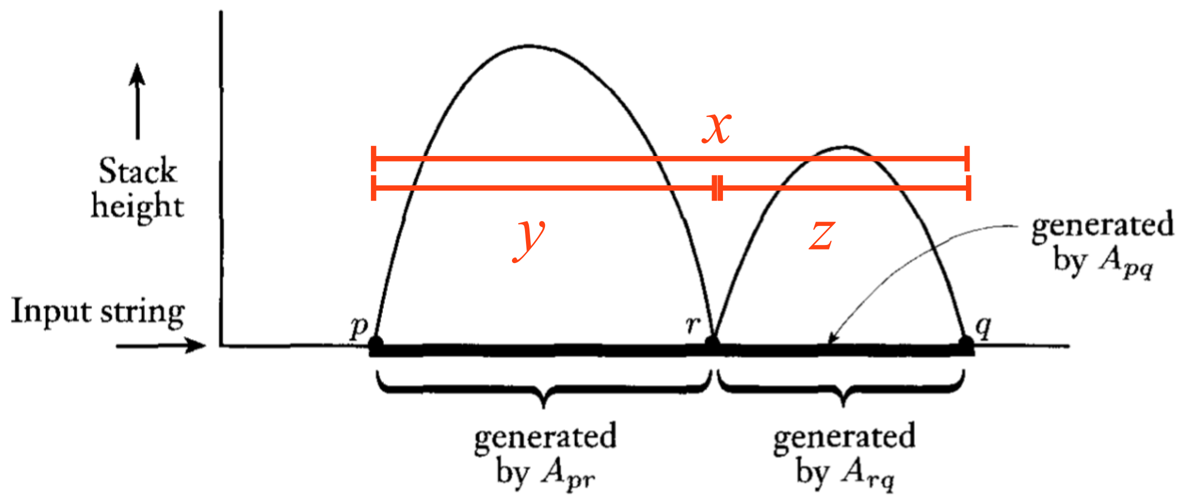



**FIGURE 2.28**
PDA computation corresponding to the rule $A_{pq} \rightarrow A_{pr} A_{rq}$

# PDA to CFG

**CLAIM 2.31**

If $x$ can bring $P$ from a state $p$ (with an empty stack) to a state $q$ (with an empty stack), then $A_{pq}$ generates $x$.

We prove this claim by induction on the number of steps in the computation of $P$ that goes from $p$ to $q$ with empty stacks on input $x$.

If $x$ can bring $P$ from a state $p$ (with an empty stack) to a state $q$ (with an empty stack), then $A_{pq}$ generates $x$.

***Basis:*** The computation has 0 steps.
If a computation has 0 steps, it starts and ends at the same state—say, $p$. So we must show that $A_{pp} \overset{*}{\Rightarrow} x$. In 0 steps, $P$ only has time to read the empty string, so $x = \varepsilon$. By construction, $G$ has the rule $A_{pp} \rightarrow \varepsilon$, so the basis is proved.

If $x$ can bring $P$ from a state $p$ (with an empty stack) to a state $q$ (with an empty stack), then $A_{pq}$ generates $x$.

***Basis:*** The computation has 0 steps.
If a computation has 0 steps, it starts and ends at the same state—say, $p$. So we must show that $A_{pp} \overset{*}{\Rightarrow} x$. In 0 steps, $P$ only has time to read the empty string, so $x = \varepsilon$. By construction, $G$ has the rule $A_{pp} \rightarrow \varepsilon$, so the basis is proved.

***Induction step:*** Assume true for computations of length at most $k$, where $k \geq 0$, and prove true for computations of length $k + 1$.

If $x$ can bring $P$ from a state $p$ (with an empty stack) to a state $q$ (with an empty stack), then $A_{pq}$ generates $x$.

***Basis:*** The computation has 0 steps.
If a computation has 0 steps, it starts and ends at the same state—say, $p$. So we must show that $A_{pp} \overset{*}{\Rightarrow} x$. In 0 steps, $P$ only has time to read the empty string, so $x = \varepsilon$. By construction, $G$ has the rule $A_{pp} \rightarrow \varepsilon$, so the basis is proved.

***Induction step:*** Assume true for computations of length at most $k$, where $k \geq 0$, and prove true for computations of length $k + 1$.

Suppose that $P$ has a computation wherein $x$ brings $p$ to $q$ with empty stacks in $k + 1$ steps. Either the stack is empty only at the beginning and end of this computation, or it becomes empty elsewhere, too.

If $x$ can bring $P$ from a state $p$ (with an empty stack) to a state $q$ (with an empty stack), then $A_{pq}$ generates $x$.

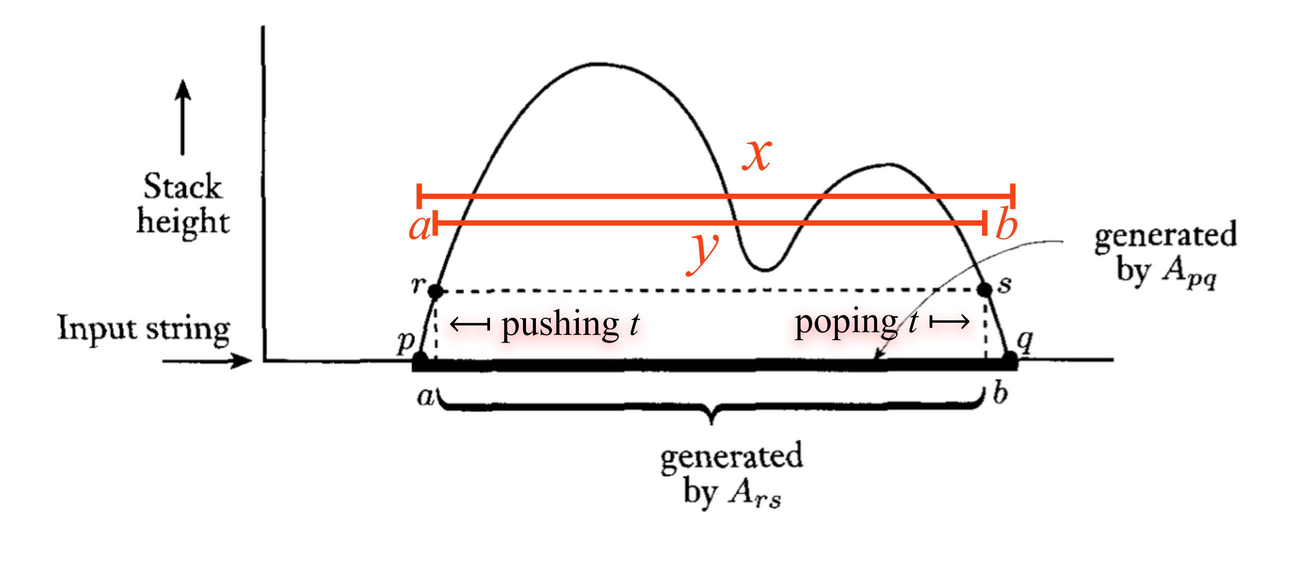



**FIGURE 2.29**
PDA computation corresponding to the rule $A_{pq} \to aA_{rs}b$

If $x$ can bring $P$ from a state $p$ (with an empty stack) to a state $q$ (with an empty stack), then $A_{pq}$ generates $x$.

In the first case, the symbol that is pushed at the first move must be the same as the symbol that is popped at the last move. Call this symbol $t$. Let $a$ be the input read in the first move, $b$ be the input read in the last move, $r$ be the state after the first move, and $s$ be the state before the last move. Then $\delta(p, a, \varepsilon)$ contains $(r, t)$ and $\delta(s, b, t)$ contains $(q, \varepsilon)$, and so rule $A_{pq} \to aA_{rs}b$ is in $G$.

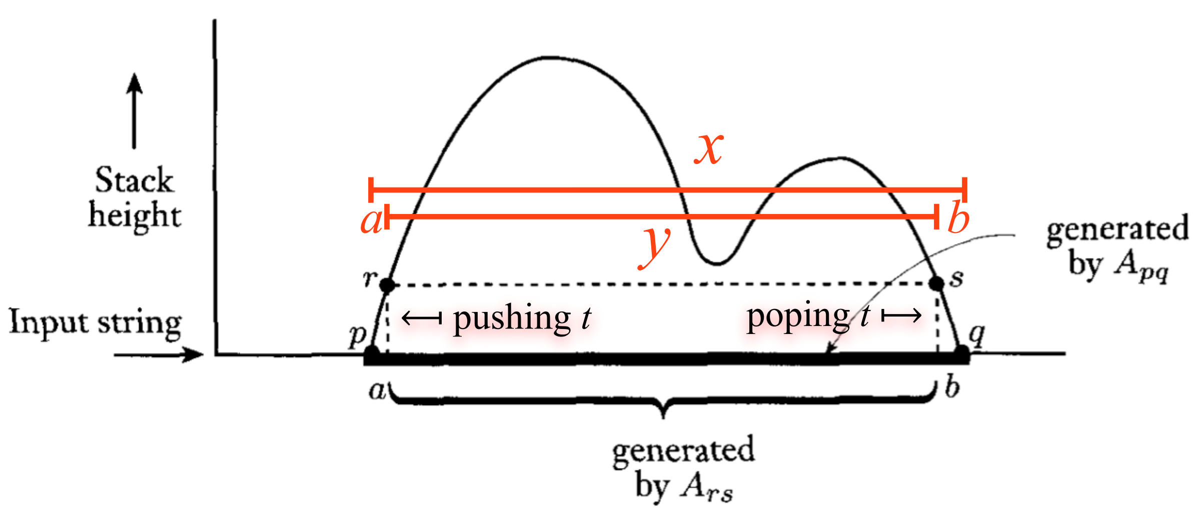



**FIGURE 2.29**
PDA computation corresponding to the rule $A_{pq} \to aA_{rs}b$

If $x$ can bring $P$ from a state $p$ (with an empty stack) to a state $q$ (with an empty stack), then $A_{pq}$ generates $x$.

the stack is empty only at the beginning and end

In the first case, the symbol that is pushed at the first move must be the same as the symbol that is popped at the last move. Call this symbol $t$. Let $a$ be the input read in the first move, $b$ be the input read in the last move, $r$ be the state after the first move, and $s$ be the state before the last move. Then $\delta(p, a, \varepsilon)$ contains $(r, t)$ and $\delta(s, b, t)$ contains $(q, \varepsilon)$, and so rule $A_{pq} \rightarrow aA_{rs}b$ is in $G$.

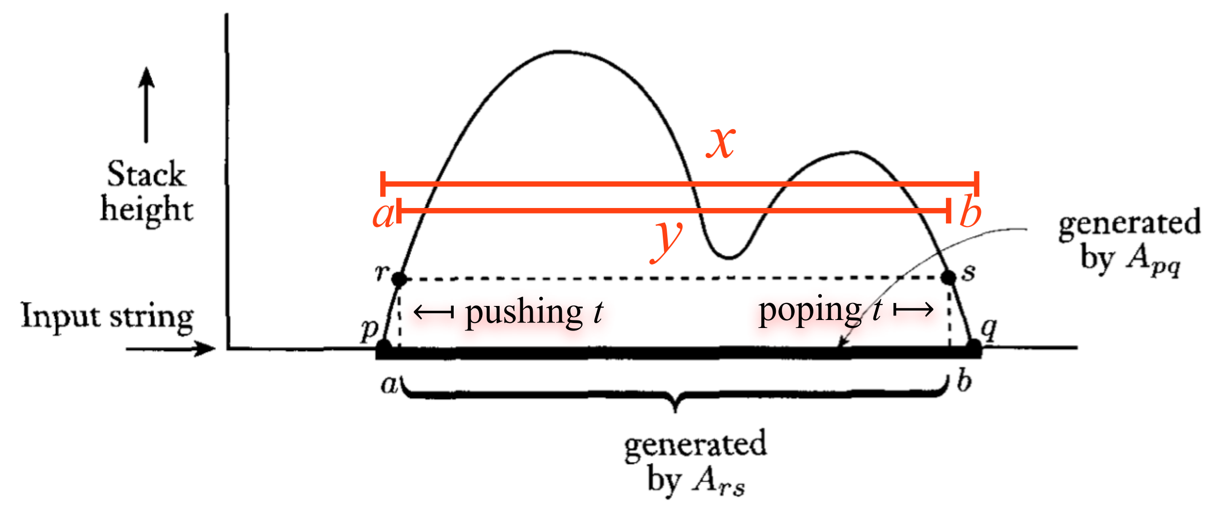



**FIGURE 2.29**
PDA computation corresponding to the rule $A_{pq} \rightarrow aA_{rs}b$

If $x$ can bring $P$ from a state $p$ (with an empty stack) to a state $q$ (with an empty stack), then $A_{pq}$ generates $x$.

the stack is empty only at the beginning and end

In the first case, the symbol that is pushed at the first move must be the same as the symbol that is popped at the last move. Call this symbol $t$. Let $a$ be the input read in the first move, $b$ be the input read in the last move, $r$ be the state after the first move, and $s$ be the state before the last move. Then $\delta(p, a, \varepsilon)$ contains $(r, t)$ and $\delta(s, b, t)$ contains $(q, \varepsilon)$, and so rule $A_{pq} \to aA_{rs}b$ is in $G$.

- For each $p, q, r, s \in Q$, $t \in \Gamma$, and $a, b \in \Sigma_\varepsilon$, if $\delta(p, a, \varepsilon)$ contains $(r, t)$ and $\delta(s, b, t)$ contains $(q, \varepsilon)$, put the rule $A_{pq} \to aA_{rs}b$ in $G$.

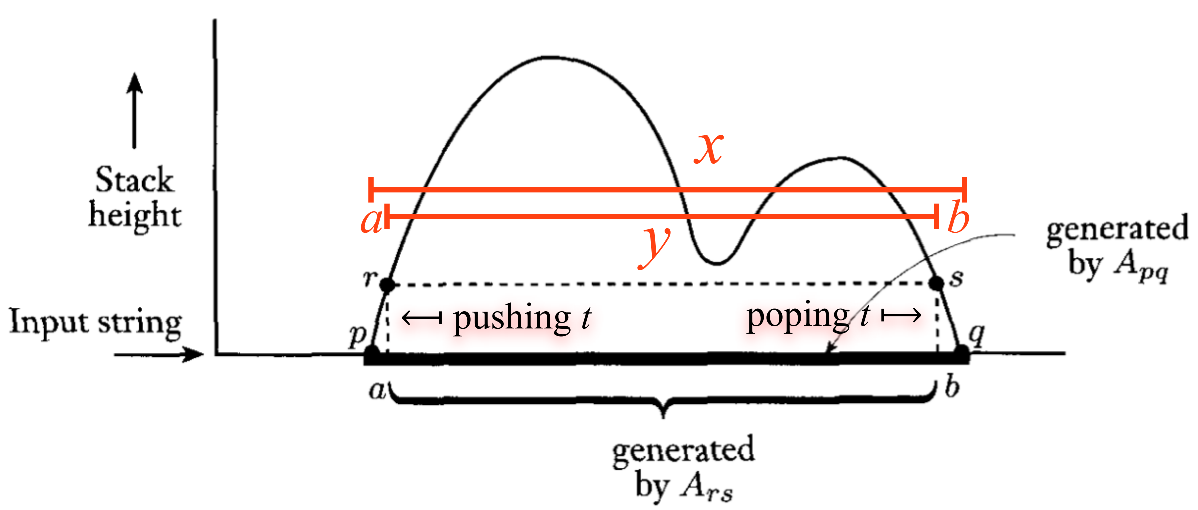



**FIGURE 2.29**
PDA computation corresponding to the rule $A_{pq} \to aA_{rs}b$

If $x$ can bring $P$ from a state $p$ (with an empty stack) to a state $q$ (with an empty stack), then $A_{pq}$ generates $x$.

the stack is empty only at the beginning and end

In the first case, the symbol that is pushed at the first move must be the same as the symbol that is popped at the last move. Call this symbol $t$. Let $a$ be the input read in the first move, $b$ be the input read in the last move, $r$ be the state after the first move, and $s$ be the state before the last move. Then $\delta(p, a, \varepsilon)$ contains $(r, t)$ and $\delta(s, b, t)$ contains $(q, \varepsilon)$, and so rule $A_{pq} \to aA_{rs}b$ is in $G$.

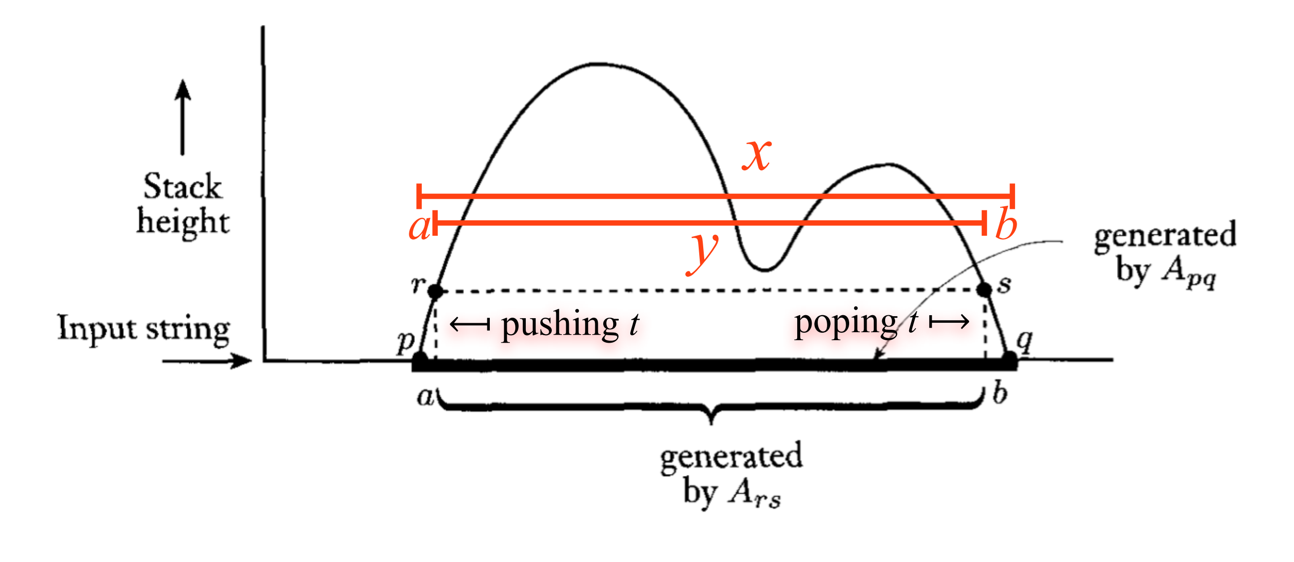



**FIGURE 2.29**
PDA computation corresponding to the rule $A_{pq} \to aA_{rs}b$

If $x$ can bring $P$ from a state $p$ (with an empty stack) to a state $q$ (with an empty stack), then $A_{pq}$ generates $x$.

the stack is empty only at the beginning and end

In the first case, the symbol that is pushed at the first move must be the same as the symbol that is popped at the last move. Call this symbol $t$. Let $a$ be the input read in the first move, $b$ be the input read in the last move, $r$ be the state after the first move, and $s$ be the state before the last move. Then $\delta(p, a, \varepsilon)$ contains $(r, t)$ and $\delta(s, b, t)$ contains $(q, \varepsilon)$, and so rule $A_{pq} \to aA_{rs}b$ is in $G$.

Let $y$ be the portion of $x$ without $a$ and $b$, so $x = ayb$. Input $y$ can bring $P$ from $r$ to $s$ without touching the symbol $t$ that is on the stack and so $P$ can go from $r$ with an empty stack to $s$ with an empty stack on input $y$. We have removed the first and last steps of the $k+1$ steps in the original computation on $x$ so the computation on $y$ has $(k+1) - 2 = k - 1$ steps. Thus the induction hypothesis tells us that $A_{rs} \overset{*}{\Rightarrow} y$. Hence $A_{pq} \overset{*}{\Rightarrow} x$.

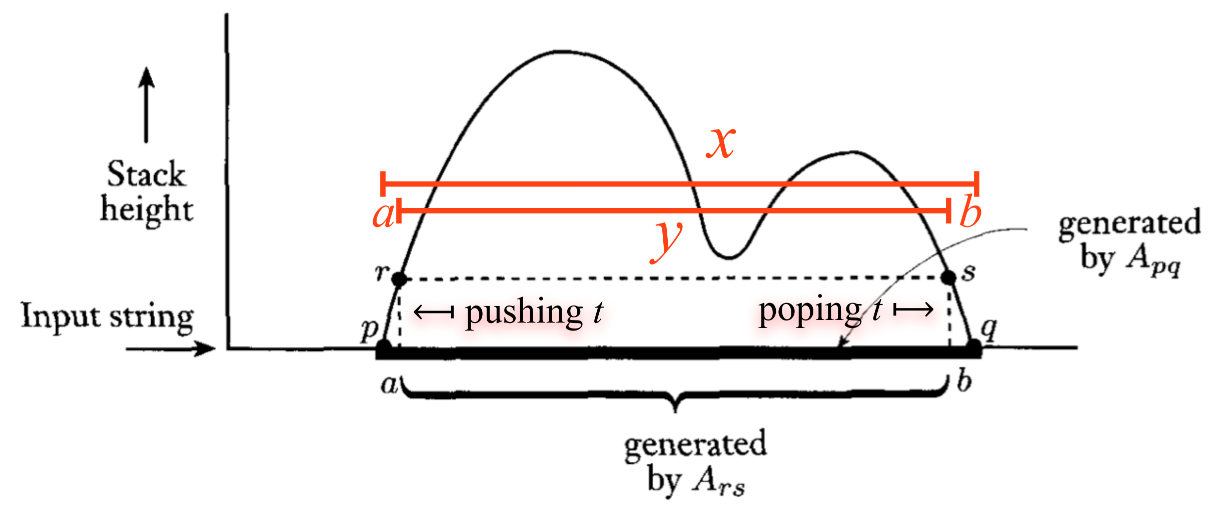



**FIGURE 2.29**
PDA computation corresponding to the rule $A_{pq} \to aA_{rs}b$

If $x$ can bring $P$ from a state $p$ (with an empty stack) to a state $q$ (with an empty stack), then $A_{pq}$ generates $x$.

In the second case, let $r$ be a state where the stack becomes empty other than at the beginning or end of the computation on $x$. Then the portions of the computation from $p$ to $r$ and from $r$ to $q$ each contain at most $k$ steps. Say that $y$ is the input read during the first portion and $z$ is the input read during the second portion. The induction hypothesis tells us that $A_{pr} \overset{*}{\Rightarrow} y$ and $A_{rq} \overset{*}{\Rightarrow} z$. Because rule $A_{pq} \rightarrow A_{pr}A_{rq}$ is in $G$, $A_{pq} \overset{*}{\Rightarrow} x$, and the proof is complete.

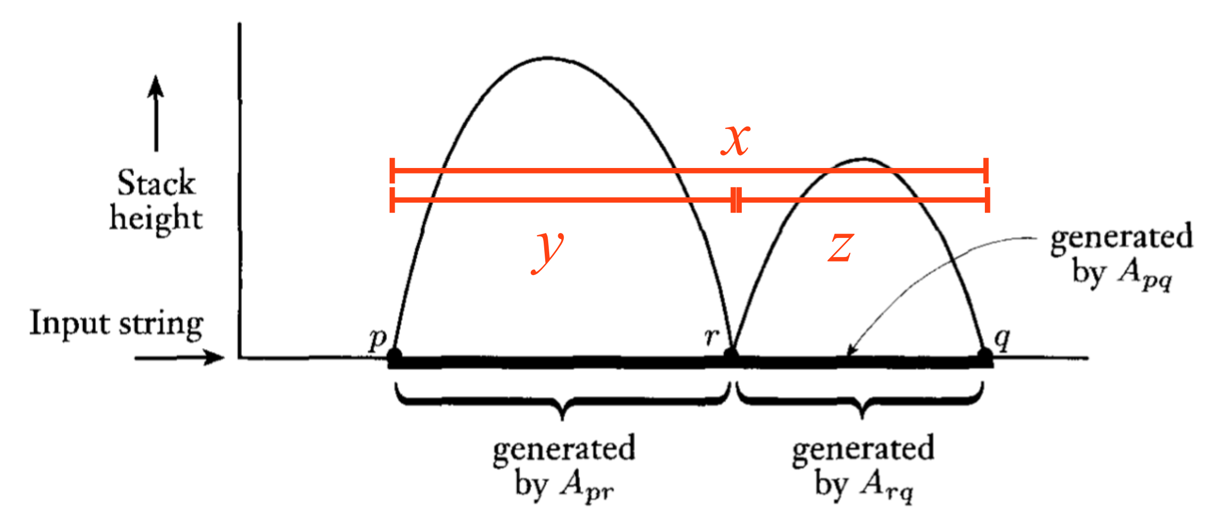



**FIGURE 2.28**
PDA computation corresponding to the rule $A_{pq} \rightarrow A_{pr}A_{rq}$

If $x$ can bring $P$ from a state $p$ (with an empty stack) to a state $q$ (with an empty stack), then $A_{pq}$ generates $x$.

it becomes empty elsewhere, too.

In the second case, let $r$ be a state where the stack becomes empty other than at the beginning or end of the computation on $x$. Then the portions of the computation from $p$ to $r$ and from $r$ to $q$ each contain at most $k$ steps. Say that $y$ is the input read during the first portion and $z$ is the input read during the second portion. The induction hypothesis tells us that $A_{pr} \overset{*}{\Rightarrow} y$ and $A_{rq} \overset{*}{\Rightarrow} z$. Because rule $A_{pq} \to A_{pr}A_{rq}$ is in $G$, $A_{pq} \overset{*}{\Rightarrow} x$, and the proof is complete.

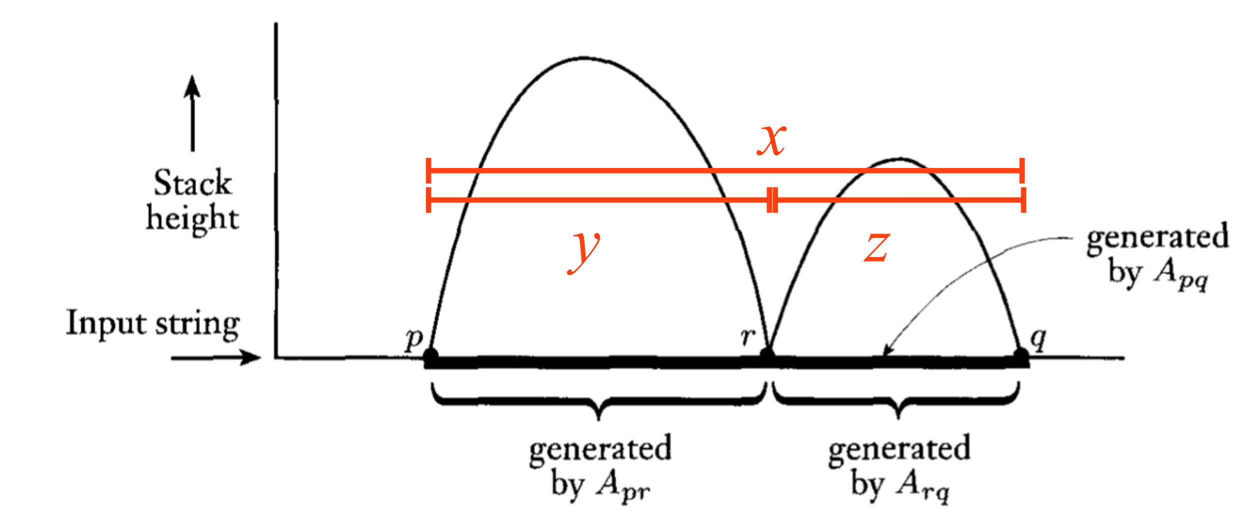



**FIGURE 2.28**
PDA computation corresponding to the rule $A_{pq} \to A_{pr}A_{rq}$

# PDA vs CFG

# PDA vs CFG

**LEMMA 2.21**

If a language is context free, then some pushdown automaton recognizes it.

# PDA vs CFG

**LEMMA 2.21**

If a language is context free, then some pushdown automaton recognizes it.

**LEMMA 2.27**

If a pushdown automaton recognizes some language, then it is context free.

# PDA vs CFG

**LEMMA 2.21**

If a language is context free, then some pushdown automaton recognizes it.

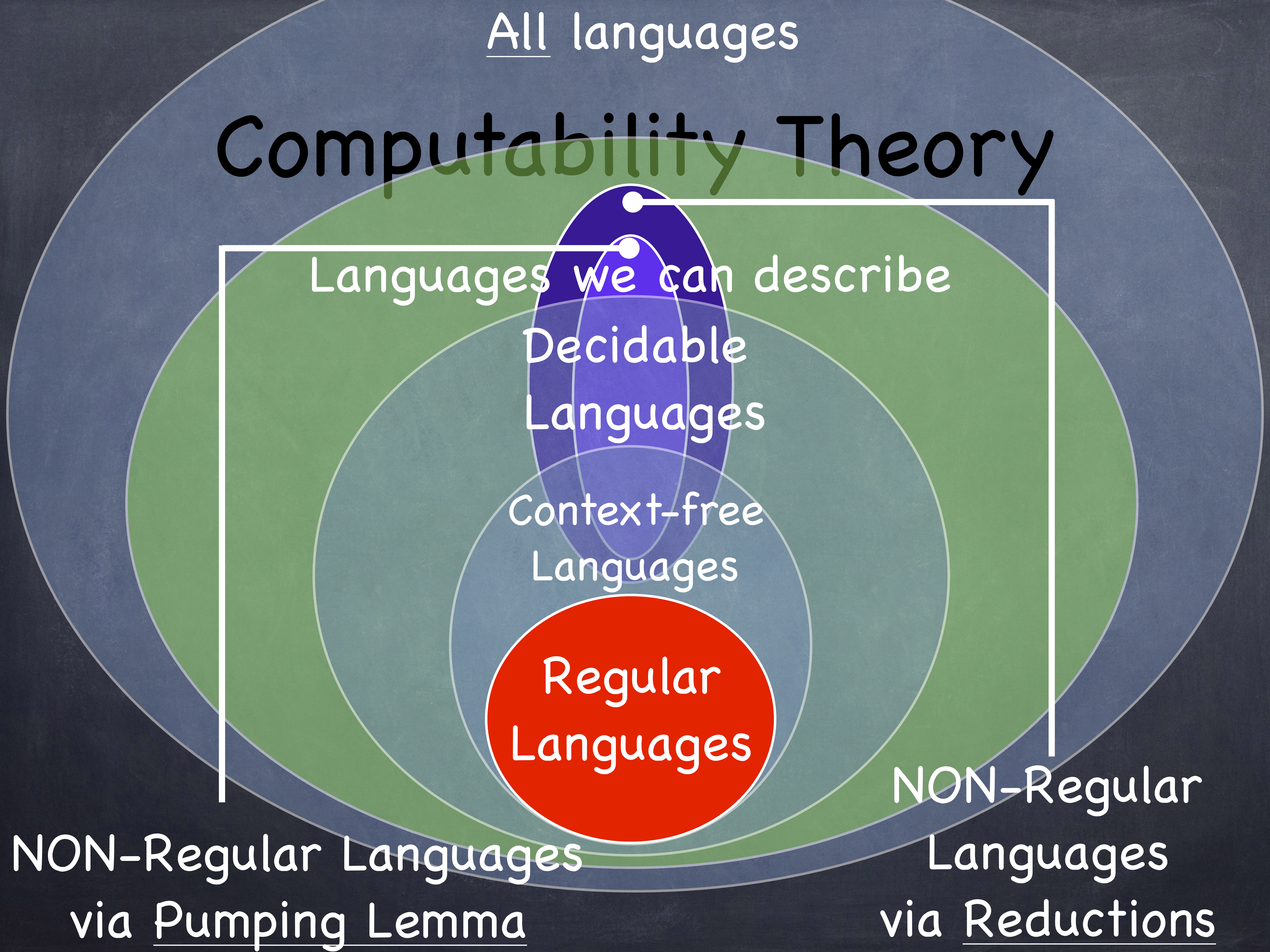
**LEMMA 2.27**

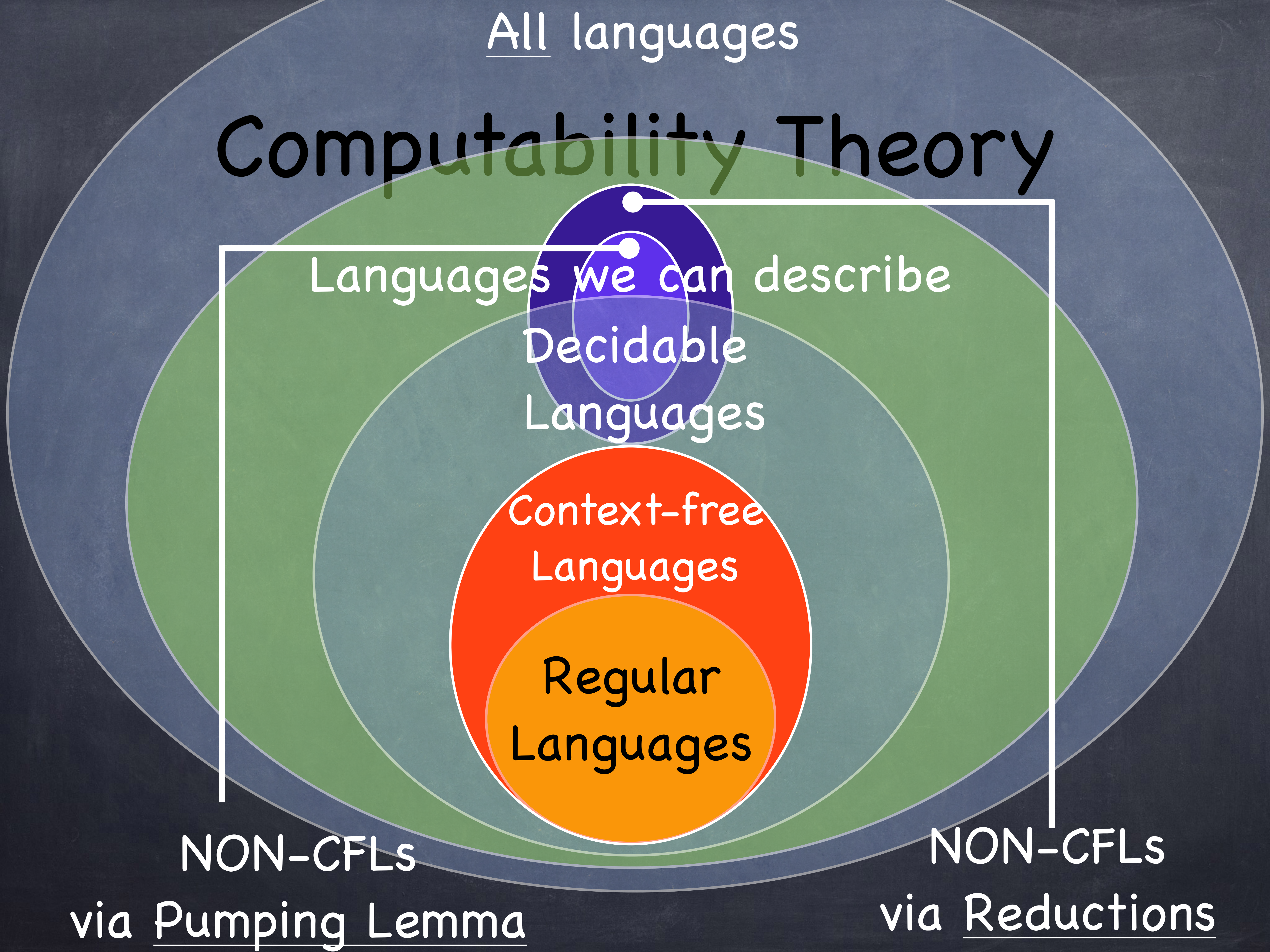
If a pushdown automaton recognizes some language, then it is context free.

**THEOREM 2.20**

A language is context free if and only if some pushdown automaton recognizes it.

# Computability Theory




All languages
Computability Theory
Languages we can describe
Decidable Languages
Context-free Languages
Regular Languages
NON-CFLs via Pumping Lemma
NON-CFLs via Reductions

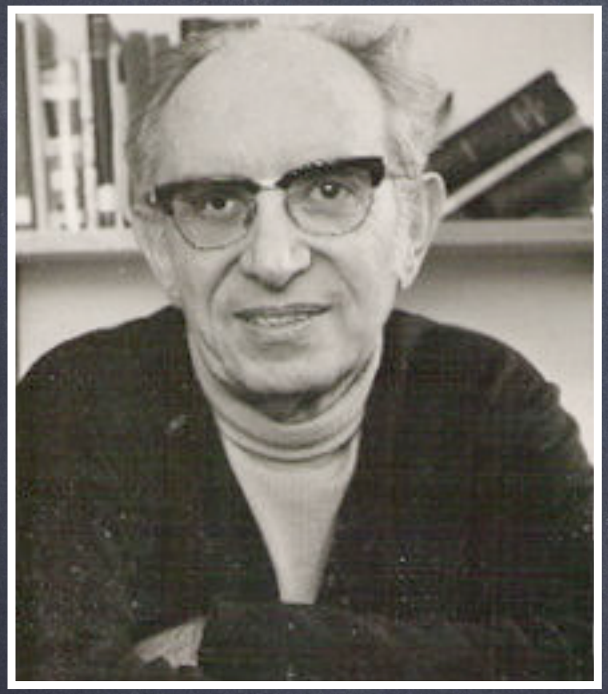
# Pumping Lemma for CFLs

**THEOREM 2.34**

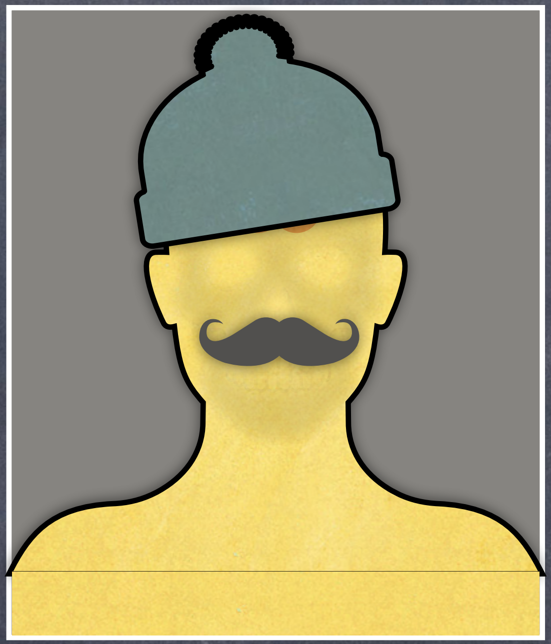
**Pumping lemma for context-free languages** If $A$ is a context-free language, then there is a number $p$ (the pumping length) where, if $s$ is any string in $A$ of length at least $p$, then $s$ may be divided into five pieces $s = uvxyz$ satisfying the conditions

1. for each $i \geq 0$, $uv^ixy^iz \in A$,
2. $|vy| > 0$, and
3. $|vxy| \leq p$.

# Pumping Lemma for CFLs



Yehoshua Bar-Hillel



Micha A. Perles

Eli Shamir

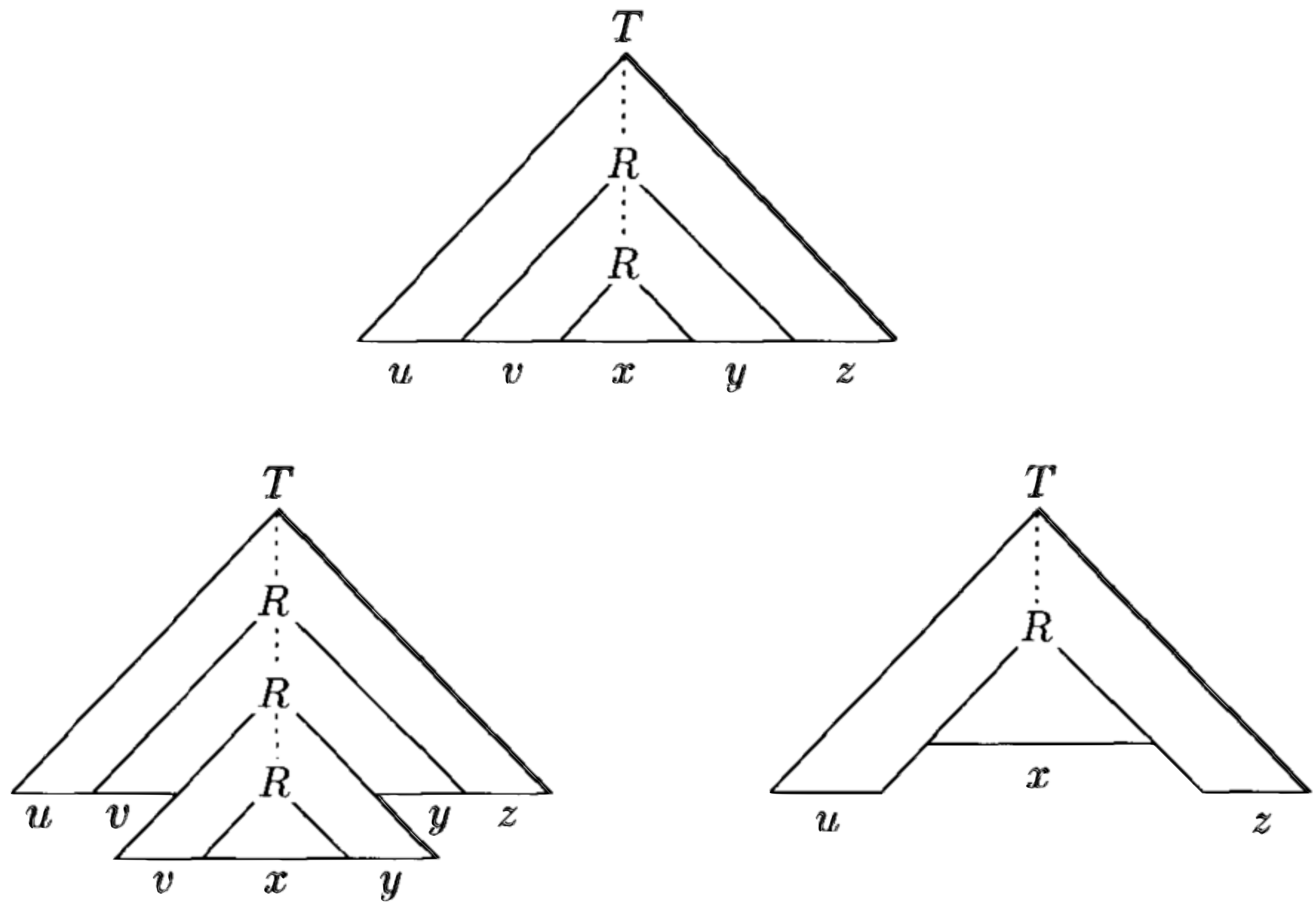



**FIGURE 2.35**
Surgery on parse trees

# Pumping Lemma

**PROOF** Let $G$ be a CFG for CFL $A$. Let $b$ be the maximum number of symbols in the right-hand side of a rule. In any parse tree using this grammar we know that a node can have no more than $b$ children. In other words, at most $b$ leaves are 1 step from the start variable; at most $b^2$ leaves are within 2 steps of the start variable; and at most $b^h$ leaves are within $h$ steps of the start variable. So, if the height of the parse tree is at most $h$, the length of the string generated is at most $b^h$. **Reciprocally,** if a generated string is at least $b^h + 1$ long, each of its parse trees must be at least $h + 1$ high.

# Pumping Lemma

**PROOF** Let $G$ be a CFG for CFL $A$. Let $b$ be the maximum number of symbols in the right-hand side of a rule. In any parse tree using this grammar we know that a node can have no more than $b$ children. In other words, at most $b$ leaves are 1 step from the start variable; at most $b^2$ leaves are within 2 steps of the start variable; and at most $b^h$ leaves are within $h$ steps of the start variable. So, if the height of the parse tree is at most $h$, the length of the string generated is at most $b^h$. **Reciprocally,** if a generated string is at least $b^h + 1$ long, each of its parse trees must be at least $h + 1$ high.

Say $|V|$ is the number of variables in $G$. We set $p$, the pumping length, to be $b^{|V|+1}$. Now if $s$ is a string in $A$ and its length is $p$ or more, its parse tree must be at least $|V| + 1$ high, **because $b^{|V|+1} \geq b^{|V|}+1$.**

# Pumping Lemma

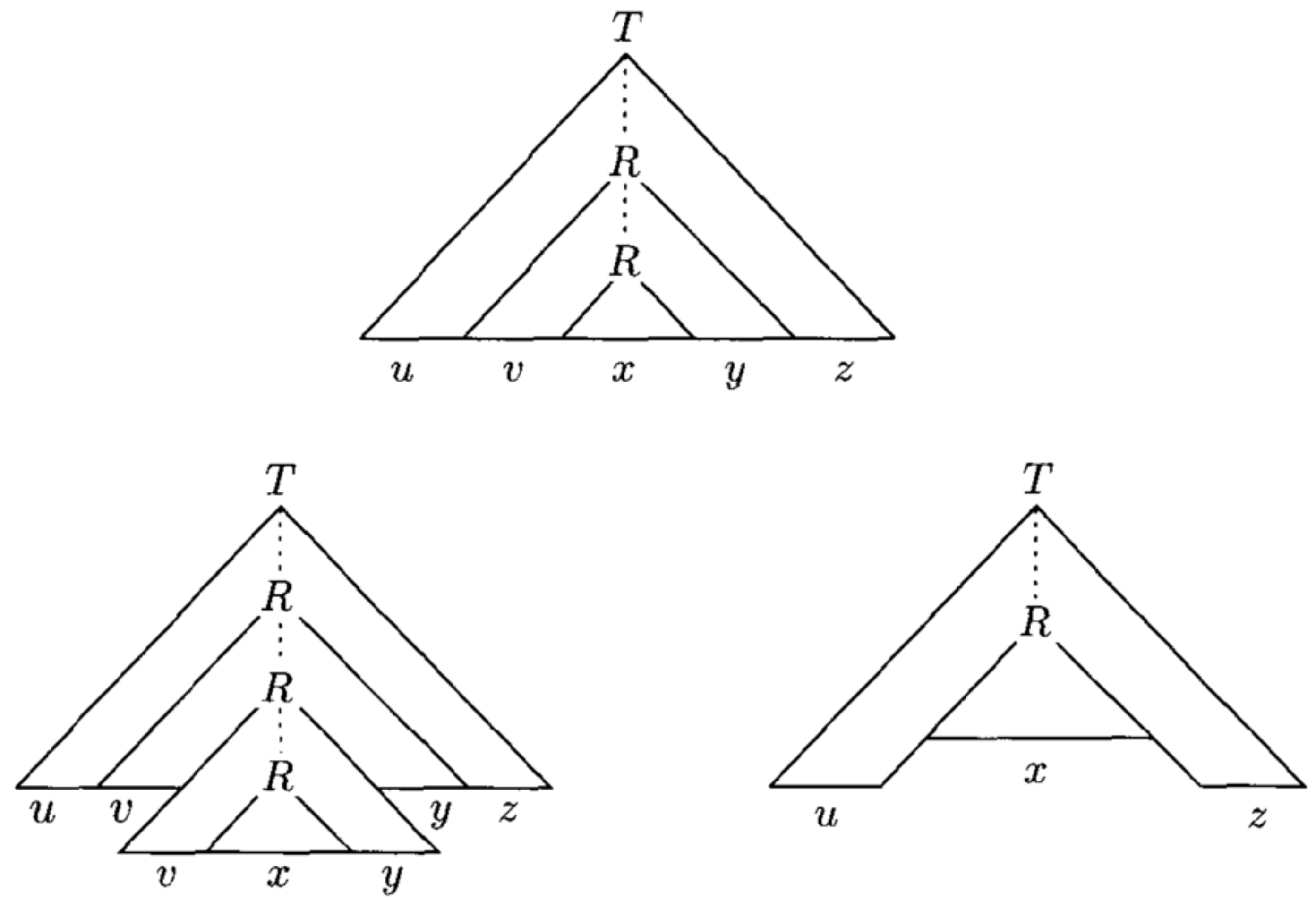



**FIGURE 2.35**
Surgery on parse trees

To see how to pump any such string $s$, let $\tau$ be one of its parse trees. If $s$ has several parse trees, choose $\tau$ to be a parse tree that has the smallest number of nodes. We know that $\tau$ must be at least $|V| + 1$ high, so it must contain a path from the root to a leaf of length at least $|V| + 1$. That path has at least $|V| + 2$ nodes; one at a terminal, the others at variables. Hence that path has at least $|V| + 1$ variables. With $G$ having only $|V|$ variables, some variable $R$ appears more than once on that path. For convenience later, we select $R$ to be a variable that repeats among the lowest $|V| + 1$ variables on this path.

# Pumping Lemma

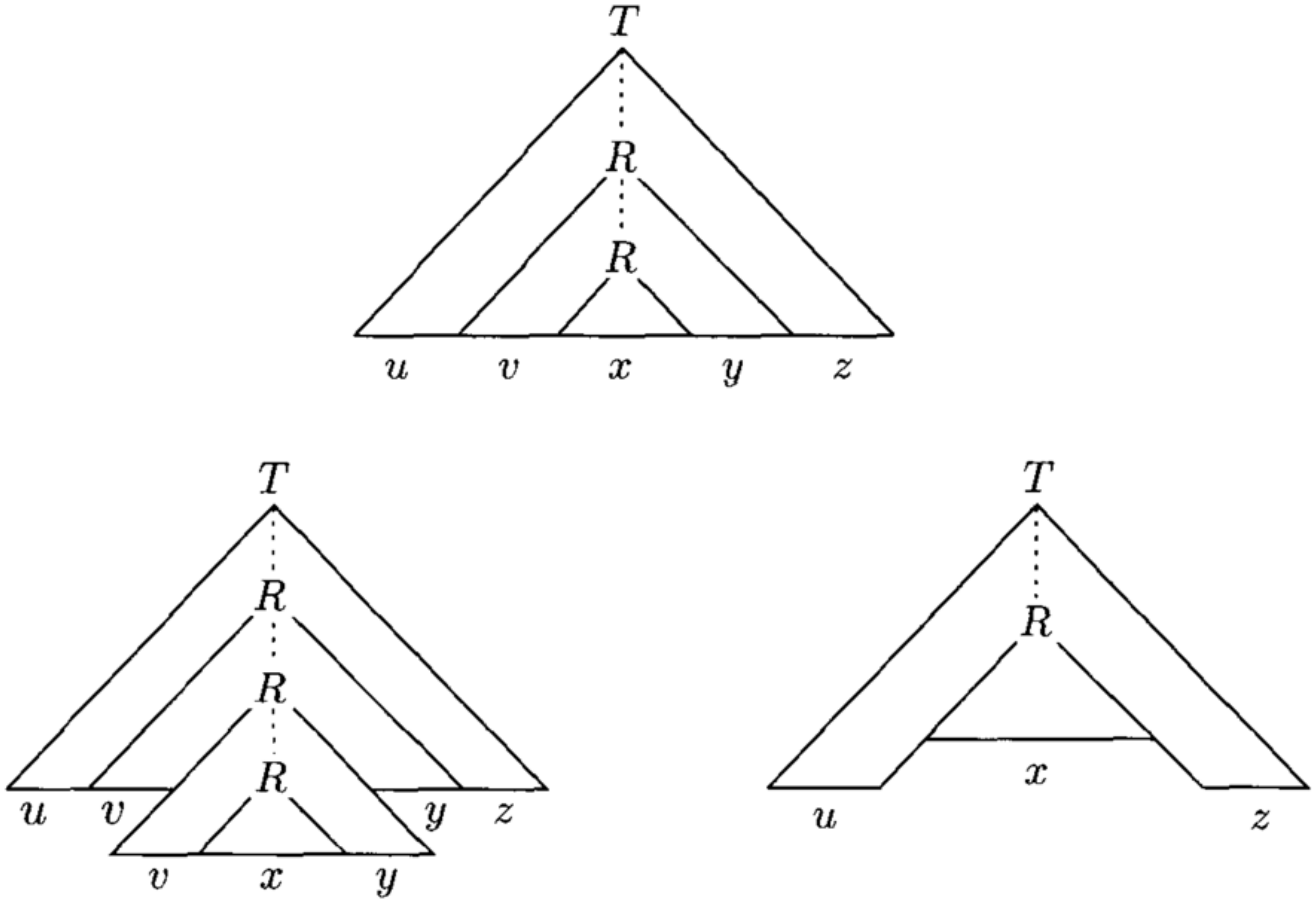



**FIGURE 2.35**
Surgery on parse trees

We divide $s$ into $uvxyz$ according to Figure 2.35. Each occurrence of $R$ has a subtree under it, generating a part of the string $s$. The upper occurrence of $R$ has a larger subtree and generates $vxy$, whereas the lower occurrence generates just $x$ with a smaller subtree. Both of these subtrees are generated by the same variable, so we may substitute one for the other and still obtain a valid parse tree. Replacing the smaller by the larger repeatedly gives parse trees for the strings $uv^ixy^iz$ at each $i > 1$. Replacing the larger by the smaller generates the string $uxz$. That establishes condition 1 of the lemma. We now turn to conditions 2 and 3.

# Pumping Lemma

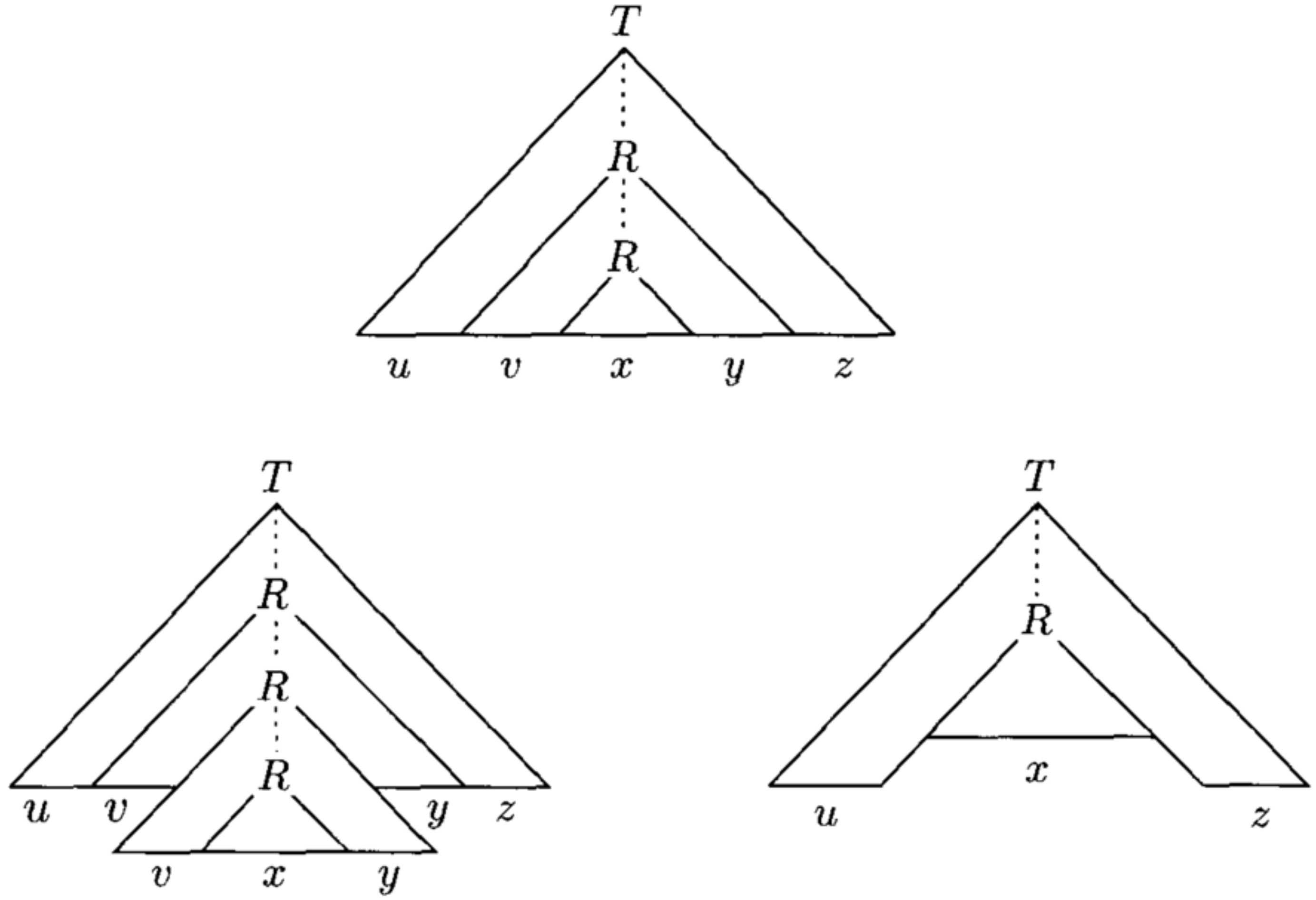



**FIGURE 2.35**
Surgery on parse trees

To get condition 2 we must be sure that both $v$ and $y$ are not $\varepsilon$. If they were, the parse tree obtained by substituting the smaller subtree for the larger would have fewer nodes than $\tau$ does and would still generate $s$. This result isn't possible because we had already chosen $\tau$ to be a parse tree for $s$ with the smallest number of nodes. That is the reason for selecting $\tau$ in this way.

# Pumping Lemma

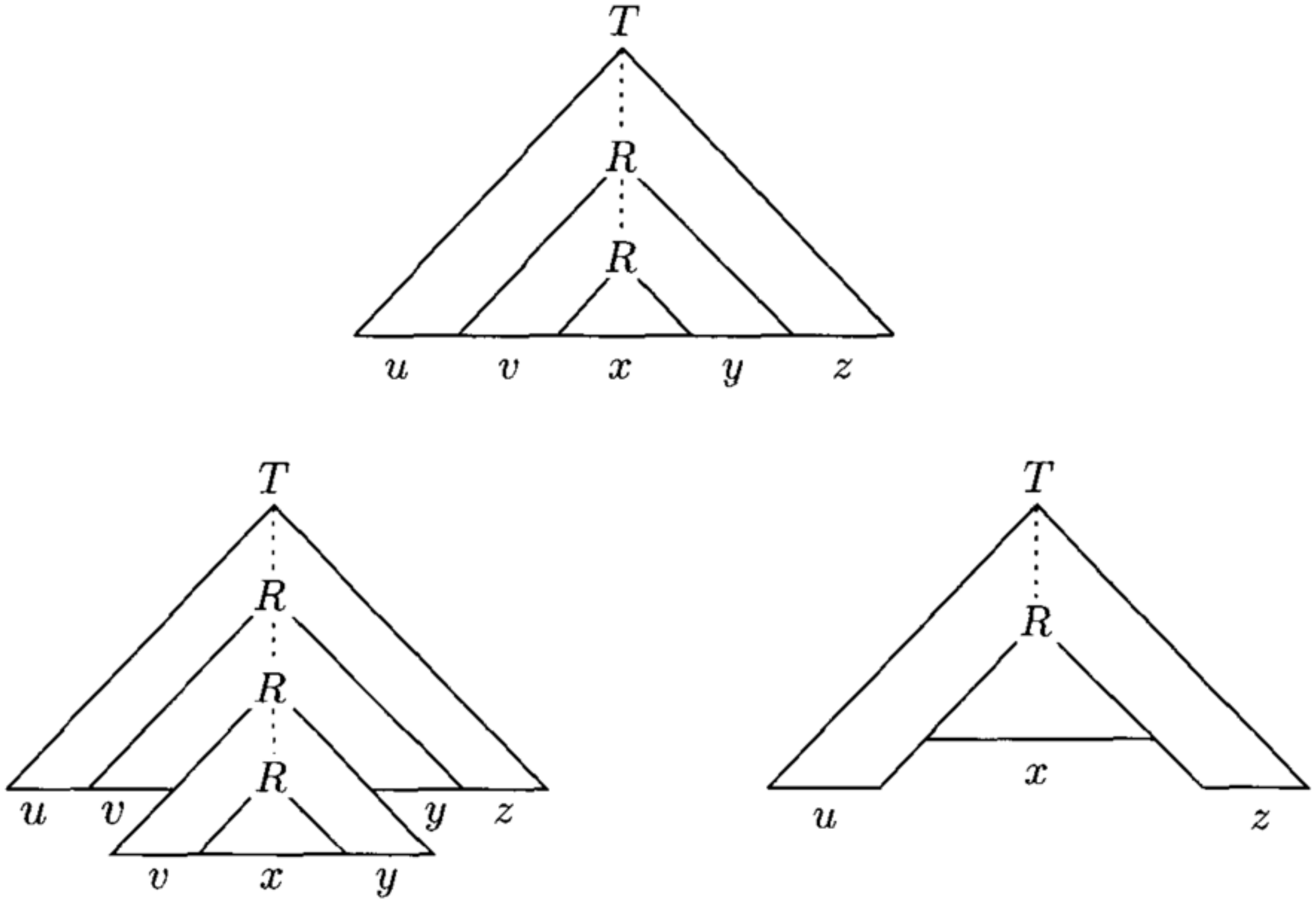



**FIGURE 2.35**
Surgery on parse trees

In order to get condition 3 we need to be sure that $vxy$ has length at most $p$. In the parse tree for $s$ the upper occurrence of $R$ generates $vxy$. We chose $R$ so that both occurrences fall within the bottom $|V| + 1$ variables on the path, and we chose the longest path in the parse tree, so the subtree where $R$ generates $vxy$ is at most $|V| + 1$ high. A tree of this height can generate a string of length at most $b^{|V|+1} = p$.

**THEOREM 2.34**

**Pumping lemma for context-free languages** If $A$ is a context-free language, then there is a number $p$ (the pumping length) where, if $s$ is any string in $A$ of length at least $p$, then $s$ may be divided into five pieces $s = uvxyz$ satisfying the conditions

1. for each $i \geq 0$, $uv^ixy^iz \in A$,
2. $|vy| > 0$, and
3. $|vxy| \leq p$.

**THEOREM 2.34**

**Pumping lemma for context-free languages** If $A$ is a context-free language, then there is a number $p$ (the pumping length) where, if $s$ is any string in $A$ of length at least $p$, then $s$ may be divided into five pieces $s = uvxyz$ satisfying the conditions

1. for each $i \geq 0$, $uv^ixy^iz \in A$,
2. $|vy| > 0$, and
3. $|vxy| \leq p$.

$A \in \mathbb{CFL} \implies$

$\exists p \forall s \in A$, $|s| \geq p$, $\exists uvxyz = s$ st 1,2,3=true.

**THEOREM 2.34**

**Pumping lemma for context-free languages** If $A$ is a context-free language, then there is a number $p$ (the pumping length) where, if $s$ is any string in $A$ of length at least $p$, then $s$ may be divided into five pieces $s = uvxyz$ satisfying the conditions

1. for each $i \geq 0$, $uv^ixy^iz \in A$,
2. $|vy| > 0$, and
3. $|vxy| \leq p$.

$A \in \mathbb{CFL} \implies$

$\exists p \forall s \in A$, $|s| \geq p$, $\exists uvxyz = s$ st 1,2,3=true.

$\forall p \exists s \in A$, $|s| \geq p$, $\forall uvxyz = s$ [1 or 2 or 3 = false].

$\implies A \notin \mathbb{CFL}$

**THEOREM 2.34**

**Pumping lemma for context-free languages** If $A$ is a context-free language, then there is a number $p$ (the pumping length) where, if $s$ is any string in $A$ of length at least $p$, then $s$ may be divided into five pieces $s = uvxyz$ satisfying the conditions

1. for each $i \geq 0$, $uv^ixy^iz \in A$,
2. $|vy| > 0$, and
3. $|vxy| \leq p$.

$A \in \mathbb{CFL} \implies$

$\exists p \forall s \in A$, $|s| \geq p$, $\exists uvxyz = s$ st 1,2,3=true.

$\forall p \exists s \in A$, $|s| \geq p$, $\forall uvxyz = s$ st 2,3=true [1=false].

$\implies A \notin \mathbb{CFL}$

**THEOREM 2.34**

**Pumping lemma for context-free languages** If $A$ is a context-free language, then there is a number $p$ (the pumping length) where, if $s$ is any string in $A$ of length at least $p$, then $s$ may be divided into five pieces $s = uvxyz$ satisfying the conditions

1. for each $i \geq 0$, $uv^ixy^iz \in A$,
2. $|vy| > 0$, and
3. $|vxy| \leq p$.

$A \in \mathbb{CFL} \implies$

$\exists p \forall s \in A$, $|s| \geq p$, $\exists uvxyz = s$ st 1,2,3=true.

$\forall p \exists s \in A$, $|s| \geq p$, $\forall uvxyz = s$ st 2,3=true [1=false].

$\implies A \notin \mathbb{CFL}$

$\forall p \exists s \in A$, $|s| \geq p$, $\forall uvxyz = s$ s.t. $|vy| > 0$, $|vxy| < p$,

then $\exists i \geq 0$ s.t. $s' = uv^ixy^iz \notin A$.

$\implies A \notin \mathbb{CFL}$

# NON-CFLs

**EXAMPLE 2.36**

Use the pumping lemma to show that the language $B = \{a^n b^n c^n | \, n \geq 0\}$ is not context free.

We assume that $B$ is a CFL and obtain a contradiction. Let $p$ be the pumping length for $B$ that is guaranteed to exist by the pumping lemma. Select the string $s = a^p b^p c^p$. Clearly $s$ is a member of $B$ and of length at least $p$. The pumping lemma states that $s$ can be pumped, but we show that it cannot. In other words, we show that no matter how we divide $s$ into $uvxyz$, one of the three conditions of the lemma is violated.

# NON-CFLs

First, condition 2 stipulates that either $v$ or $y$ is nonempty. Then we consider one of two cases, depending on whether substrings $v$ and $y$ contain more than one type of alphabet symbol.

1. When both $v$ and $y$ contain only one type of alphabet symbol, $v$ does not contain both a's and b's or both b's and c's, and the same holds for $y$. In this case the string $uv^2xy^2z$ cannot contain equal numbers of a's, b's, and c's. Therefore it cannot be a member of $B$. That violates condition 1 of the lemma and is thus a contradiction.
2. When either $v$ or $y$ contain more than one type of symbol $uv^2xy^2z$ may contain equal numbers of the three alphabet symbols but not in the correct order. Hence it cannot be a member of $B$ and a contradiction occurs.

# NON-CFLs

One of these cases must occur. Because both cases result in a contradiction, a contradiction is unavoidable. So the assumption that $B$ is a CFL must be false. Thus we have proved that $B$ is not a CFL.

# NON-CFLs

**EXAMPLE 2.37**

Let $C = \{a^i b^j c^k \mid 0 \leq i \leq j \leq k\}$. We use the pumping lemma to show that $C$ is not a CFL. This language is similar to language $B$ in Example 2.36, but proving that it is not context free is a bit more complicated.

Assume that $C$ is a CFL and obtain a contradiction. Let $p$ be the pumping length given by the pumping lemma. We use the string $s = a^p b^p c^p$ that we used earlier, but this time we must "pump down" as well as "pump up." Let $s = uvxyz$ and again consider the two cases that occurred in Example 2.36.

# NON-CFLs

$s = uvxyz$ and again consider the two cases that occurred in Example 2.36.

1. When both $v$ and $y$ contain only one type of alphabet symbol, $v$ does not contain both a's and b's or both b's and c's, and the same holds for $y$. Note that the reasoning used previously in case 1 no longer applies. The reason is that $C$ contains strings with unequal numbers of a's, b's, and c's as long as the numbers are not decreasing. We must analyze the situation more carefully to show that $s$ cannot be pumped. Observe that because $v$ and $y$ contain only one type of alphabet symbol, one of the symbols a, b, or c doesn't appear in $v$ or $y$. We further subdivide this case into three subcases according to which symbol does not appear.

# NON-CFLs

**a.** *The* a*'s do not appear*. Then we try pumping down to obtain the string $uv^0xy^0z = uxz$. That contains the same number of a's as $s$ does, but it contains fewer b's or fewer c's. Therefore it is not a member of $C$, and a contradiction occurs.

**b.** *The* b*'s do not appear.* Then either a's or c's must appear in $v$ or $y$ because both can't be the empty string. If a's appear, the string $uv^2xy^2z$ contains more a's than b's, so it is not in $C$. If c's appear, the string $uv^0xy^0z$ contains more b's than c's, so it is not in $C$. Either way a contradiction occurs.

**c.** *The* c*'s do not appear.* Then the string $uv^2xy^2z$ contains more a's or more b's than c's, so it is not in $C$, and a contradiction occurs.

# NON-CFLs

2. When either $v$ or $y$ contain more than one type of symbol, $uv^2xy^2z$ will not contain the symbols in the correct order. Hence it cannot be a member of $C$, and a contradiction occurs.

Thus we have shown that $s$ cannot be pumped in violation of the pumping lemma and that $C$ is not context free.

# NON-CFLs

**EXAMPLE 2.38**

Let $D = \{ww \mid w \in \{0,1\}^*\}$. Use the pumping lemma to show that $D$ is not a CFL. Assume that $D$ is a CFL and obtain a contradiction. Let $p$ be the pumping length given by the pumping lemma.

This time choosing string $s$ is less obvious. One possibility is the string $0^p10^p1$. It is a member of $D$ and has length greater than $p$, so it appears to be a good candidate. But this string *can* be pumped by dividing it as follows, so it is not adequate for our purposes.

$$\overbrace{\underbrace{000\cdots000}_{u}\;\underbrace{0}_{v}\;\underbrace{1}_{x}}^{0^p1}\;\overbrace{\underbrace{0}_{y}\;\underbrace{000\cdots0001}_{z}}^{0^p1}$$

# NON-CFLs

Let's try another candidate for $s$. Intuitively, the string $0^p1^p0^p1^p$ seems to capture more of the "essence" of the language $D$ than the previous candidate did. In fact, we can show that this string does work, as follows.

We show that the string $s = 0^p1^p0^p1^p$ cannot be pumped. This time we use condition 3 of the pumping lemma to restrict the way that $s$ can be divided. It says that we can pump $s$ by dividing $s = uvxyz$, where $|vxy| \leq p$.

# NON-CFLs

First, we show that the substring $vxy$ must straddle the midpoint of $s$. Otherwise, if the substring occurs only in the first half of $s$, pumping $s$ up to $uv^2xy^2z$ moves a 1 into the first position of the second half, and so it cannot be of the form $ww$. Similarly, if $vxy$ occurs in the second half of $s$, pumping $s$ up to $uv^2xy^2z$ moves a 0 into the last position of the first half, and so it cannot be of the form $ww$.

But if the substring $vxy$ straddles the midpoint of $s$, when we try to pump $s$ down to $uxz$ it has the form $0^p1^i0^j1^p$, where $i$ and $j$ cannot both be $p$. This string is not of the form $ww$. Thus $s$ cannot be pumped, and $D$ is not a CFL.

# Reductions (& Construction tools)

- CFLs are closed under union, concatenation and star. If there exists a CFL C s.t. either $A^*=A'$, $A \cup C=A'$, $A \circ C=A'$ (but not complement nor intersection) or any combinations of these operations then A′ is a CFL as long as A is.
- If A′ is NON-CFL then so is A.

# Reduction example

- Consider languages $D=\{ ww \mid w \in \{a,b\}^* \}$, $E=\{ ww \mid w \in \{a,b\}^* \text{ and } |w|<1000 \}$ and $F=D \setminus E$.

- Since $D=E \cup F$ and E is a CFL (a finite and regular language), and since D is a NON-CFL we conclude that

  $$F= \{ ww \mid w \in \{a,b\}^* \text{ and } |w|>999 \}$$

  is also a NON-CFL.

# Reduction example

[A]**2.18** **a.** Let $C$ be a context-free language and $R$ be a regular language. Prove that the language $C \cap R$ is context free.

**b.** Use part (a) to show that the language $A = \{w |\ w \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^*$ and contains equal numbers of a's, b's, and c's$\}$ is not a CFL.

*2.18* **(a)** Let $C$ be a context-free language and $R$ be a regular language. Let $P$ be the PDA that recognizes $C$, and $D$ be the DFA that recognizes $R$. If $Q$ is the set of states of $P$ and $Q'$ is the set of states of $D$, we construct a PDA $P'$ that recognizes $C \cap R$ with the set of states $Q \times Q'$. $P'$ will do what $P$ does and also keep track of the states of $D$. It accepts a string $w$ if and only if it stops at a state $q \in F_P \times F_D$, where $F_P$ is the set of accept states of $P$ and $F_D$ is the set of accept states of $D$. Since $C \cap R$ is recognized by $P'$, it is context free.

**(b)** Let $R$ be the regular language $\mathtt{a}^*\mathtt{b}^*\mathtt{c}^*$. If $A$ were a CFL then $A \cap R$ would be a CFL by part (a). However, $A \cap R = \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n |\ n \geq 0\}$, and Example 2.36 proves that $A \cap R$ is not context free. Thus $A$ is not a CFL.

# COMP-330
# Theory of Computation

Fall 2019 -- Prof. Claude Crépeau

# Lec. 13 :
# Pumping Lemma for CFLs