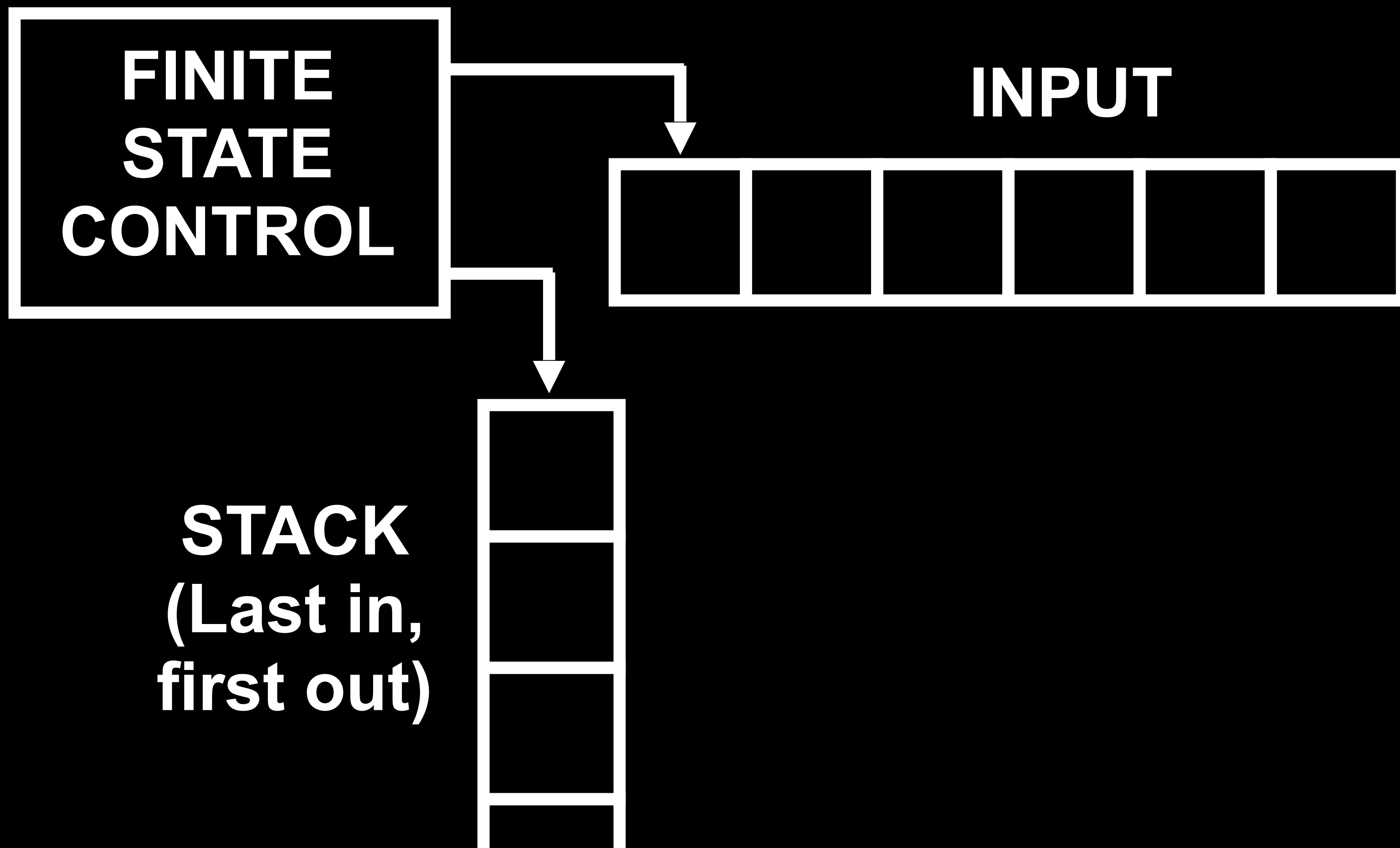
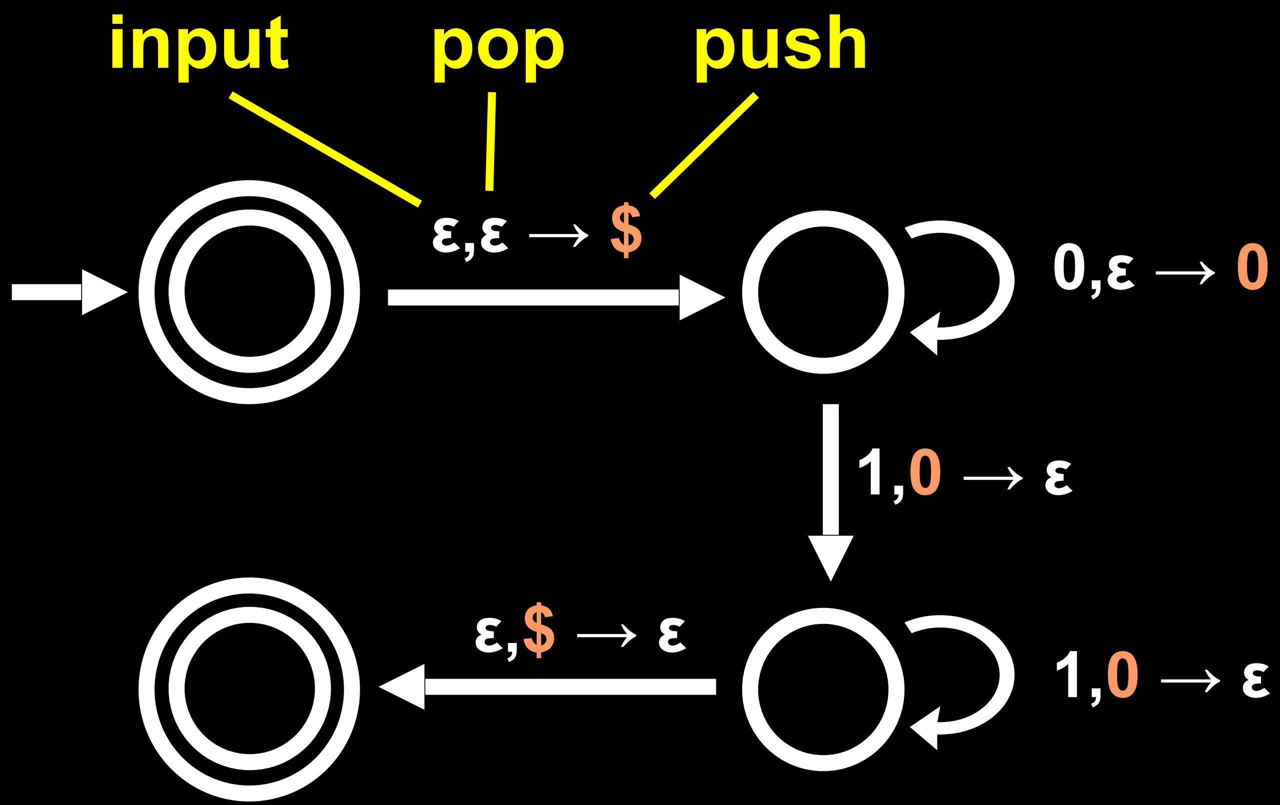
PUSHDOWN AUTOMATA (PDA)





PDA that recognizes $L = \{ 0^n1^n \mid n \geq 0 \}$

Definition: A (*non-deterministic*) PDA is a 6-tuple $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$, where:

Q is a finite set of states

Σ is the input alphabet

Γ is the stack alphabet

pop

push

$\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to 2^{Q \times \Gamma_\varepsilon}$

$q_0 \in Q$ is the start state

$F \subseteq Q$ is the set of accept states

$2^{Q \times \Gamma_\varepsilon}$ is the set of subsets of $Q \times \Gamma_\varepsilon$

$\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$, $\Gamma_\varepsilon = \Gamma \cup \{\varepsilon\}$

Let $w \in \Sigma^*$ and suppose w can be written as $w_1 \ldots w_n$ where $w_i \in \Sigma_\varepsilon$ (recall $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$)

Then P accepts w if there are

$r_0, r_1, \ldots, r_n \in Q$ and

$s_0, s_1, \ldots, s_n \in \Gamma^*$ (sequence of stacks) such that

1. $r_0 = q_0$ and $s_0 = \varepsilon$ (P starts in q_0 with empty stack)

2. For $i = 0, \ldots, n-1$:
$(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$, where $s_i = at$ and $s_{i+1} = bt$ for some $a, b \in \Gamma_\varepsilon$ and $t \in \Gamma^*$
(P moves correctly according to state, stack and symbol read)

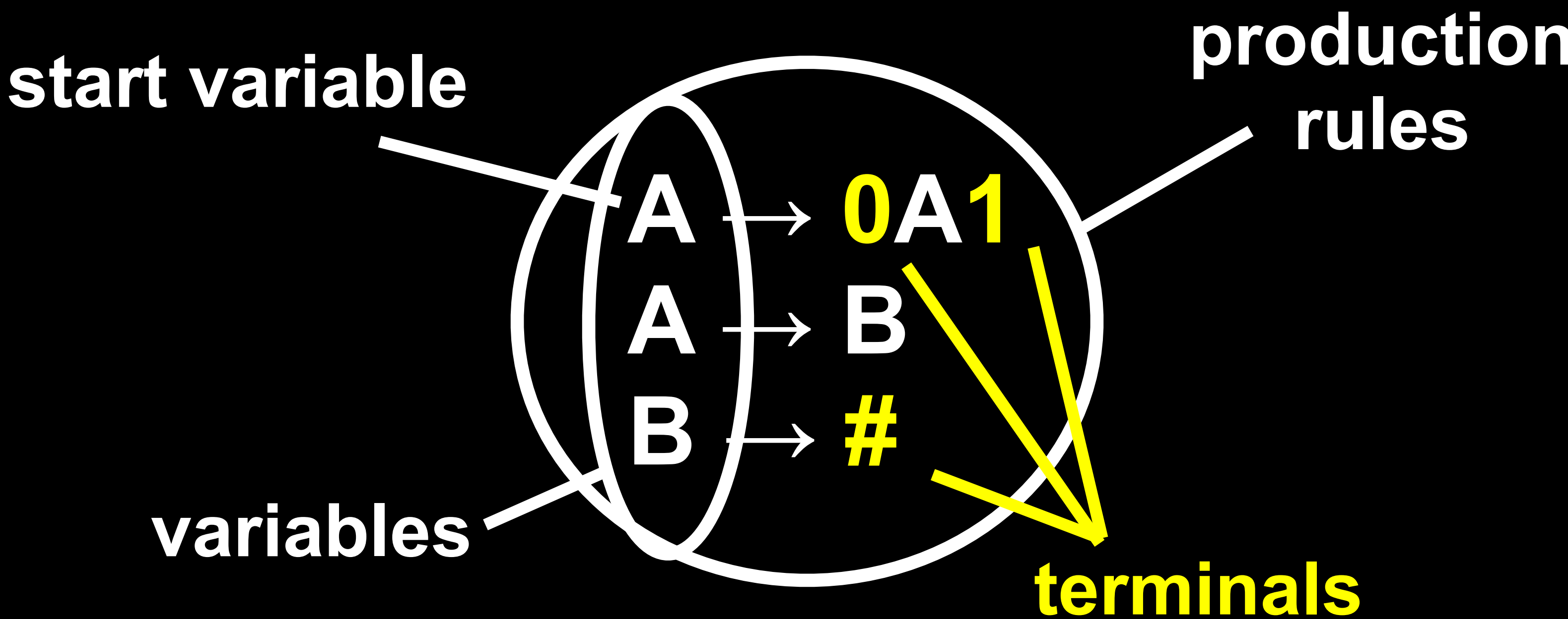
3. $r_n \in F$ (P is in an accept state at the end of its input)

CONTEXT-FREE GRAMMARS

production rules

$$A \rightarrow 0A1$$
$$A \rightarrow B$$
$$B \rightarrow \#$$

CONTEXT-FREE GRAMMARS



CONTEXT-FREE GRAMMARS

start variable

production rules

$$A \to 0A1$$
$$A \to B$$
$$B \to \#$$

variables

terminals

$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 00B11 \Rightarrow 00\#11$$

$\Rightarrow$ (yields)

Derivation

$$A \Rightarrow^* 00\#11$$

(derives)

CONTEXT-FREE GRAMMARS

start variable

production rules

$$A \rightarrow 0A1$$
$$A \rightarrow B$$
$$B \rightarrow \#$$

variables

terminals

$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 00B11 \Rightarrow 00\#11$$

$\Rightarrow$ (yields)

Derivation

$$A \Rightarrow^* 00\#11$$

(derives)

We say: **00#11** is *generated* by the Grammar

Non-deterministic

SNOOP'S GRAMMAR

(courtesy of Luis von Ahn)

<PHRASE> → <FILLER><PHRASE>

<PHRASE> → <START WORD><END WORD>DUDE

<FILLER> → LIKE

<FILLER> → UMM

<START WORD> → FO

<START WORD> → FA

<END WORD> → SHO

<END WORD> → SHAZZY

<END WORD> → SHEEZY

<END WORD> → SHIZZLE

SNOOP'S GRAMMAR

(courtesy of Luis von Ahn)

Generate:

Umm Like Umm Umm Fa Shizzle Dude

Fa Sho Dude

CONTEXT-FREE GRAMMARS

A context-free grammar (CFG) is a tuple
$G = (V, \Sigma, R, S)$, where:

V is a finite set of variables

Σ is a finite set of terminals (disjoint from V)

R is set of production rules of the form $A \rightarrow W$, where $A \in V$ and $W \in (V \cup \Sigma)^*$

$S \in V$ is the start variable

CONTEXT-FREE LANGUAGES

A context-free grammar (**CFG**) is a tuple
$G = (V, \Sigma, R, S)$, where:

V is a finite set of **variables**

Σ is a finite set of **terminals** (disjoint from V)

R is set of **production rules** of the form $A \to W$, where $A \in V$ and $W \in (V \cup \Sigma)^*$

$S \in V$ is the **start variable**

$L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$ Strings Generated by G

A Language **L is context-free** if there is a **CFG** that generates precisely the strings in **L**

CONTEXT-FREE LANGUAGES

A context-free grammar (CFG) is a tuple
$G = (V, \Sigma, R, S)$, where:

V is a finite set of variables

Σ is a finite set of terminals (disjoint from V)

R is set of production rules of the form $A \rightarrow W$, where $A \in V$ and $W \in (V \cup \Sigma)^*$

$S \in V$ is the start variable

$G = \{ \{S\}, \{0,1\}, R, S \}$ $\quad$ $R = \{ S \rightarrow 0S1, S \rightarrow \varepsilon \}$

$L(G) =$

CONTEXT-FREE LANGUAGES

A context-free grammar (CFG) is a tuple $G = (V, \Sigma, R, S)$, where:

V is a finite set of **variables**

Σ is a finite set **of terminals** (disjoint from V)

R is set of **production rules** of the form $A \to W$, where $A \in V$ and $W \in (V \cup \Sigma)^*$

$S \in V$ is the **start variable**

$G = \{ \{S\}, \{0,1\}, R, S \}$ $\quad$ $R = \{ S \to 0S1, S \to \varepsilon \}$

$L(G) = \{ 0^n 1^n \mid n \geq 0 \}$ Strings Generated by G

WRITE A CFG FOR EVEN-LENGTH PALINDROMES

$S \rightarrow \sigma S \sigma$ for all $\sigma \in \Sigma$

$S \rightarrow \varepsilon$

WRITE A CFG FOR THE EMPTY SET

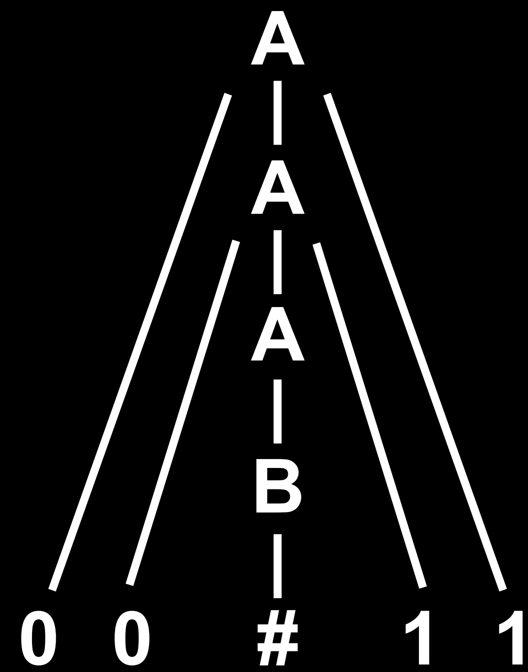
$$G = \{ \{S\}, \Sigma, \varnothing, S \}$$

PARSE TREES

$A \rightarrow 0A1$

$A \rightarrow B$

$B \rightarrow \#$



$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 00B11 \Rightarrow 00\#11$$

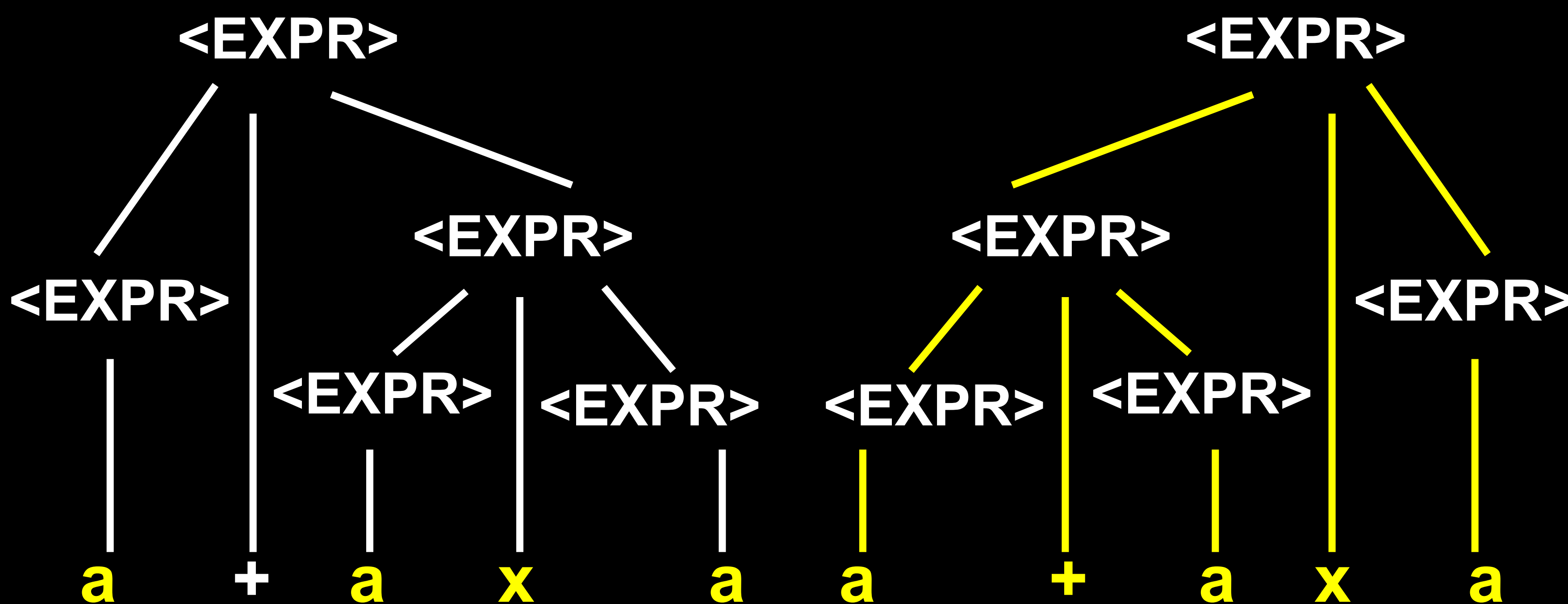
<EXPR> → <EXPR> + <EXPR>

<EXPR> → <EXPR> x <EXPR>

<EXPR> → (<EXPR>)

<EXPR> → a

Build a parse tree for a + a x a



Definition: **a string is derived ambiguously in a context-free grammar if it has more than one parse tree**

Definition: **a grammar is ambiguous if it generates some string ambiguously**

See G_4 for unambiguous standard arithmetic precedence

$L = \{ a^i b^j c^k \mid i, j, k \geq 0$ and $(i = j$ or $j = k) \}$ is *inherently ambiguous* (xtra credit)

Undecidable to tell if a language has unambiguous parse trees (**Post's problem**)

NOT REGULAR

$\Sigma = \{0, 1\}$, $L = \{ 0^n 1^n \mid n \geq 0 \}$

But L is **CONTEXT FREE**

$A \rightarrow 0A1$
$A \rightarrow \varepsilon$

WHAT ABOUT?

$\Sigma = \{0, 1\}$, $L_1 = \{ 0^n 1^n 0^m \mid m,n \geq 0 \}$

$\Sigma = \{0, 1\}$, $L_2 = \{ 0^n 1^m 0^n \mid m,n \geq 0 \}$

$\Sigma = \{0, 1\}$, $L_3 = \{ 0^m 1^n 0^n \mid m=n \geq 0 \}$

WHAT ABOUT?

$\Sigma = \{0, 1\}$, $L_1 = \{ 0^n 1^n 0^m \mid m, n \geq 0 \}$

$\Sigma = \{0, 1\}$, $L_2 = \{ 0^n 1^m 0^n \mid m, n \geq 0 \}$

$\Sigma = \{0, 1\}$, $L_3 = \{ 0^m 1^n 0^n \mid m=n \geq 0 \}$

WHAT ABOUT?

$\Sigma = \{0, 1\}$, $L_1 = \{ 0^n 1^n 0^m \mid m, n \geq 0 \}$

S -> AB

A -> 0A1 | **ε**

B -> 0B | **ε**

$\Sigma = \{0, 1\}$, $L_2 = \{ 0^n 1^m 0^n \mid m, n \geq 0 \}$

S -> 0S0 | A

A -> 1A | **ε**

$\Sigma = \{0, 1\}$, $L_3 = \{ 0^m 1^n 0^n \mid m=n \geq 0 \}$

THE PUMPING LEMMA FOR CFGs

Let L be a context-free language

Then there is a P such that
if $w \in L$ and $|w| \geq P$

then can write $w = uvxyz$, where:

1. $|vy| > 0$
2. $|vxy| \leq P$
3. For *every* $i \geq 0$, $uv^ixy^iz \in L$

WHAT ABOUT?

$\Sigma = \{0, 1\}$, $L_3 = \{ 0^m 1^n 0^n \mid m=n \geq 0 \}$

Choose $w = 0^P 1^P 0^P$.

By the Pumping Lemma, we can write
$w = uvxyz$ with $|vy| > 0$, $|vxy| \leq P$ such that pumping v together with y will produce another word in L_3

Since $|vxy| \leq P$, $vxy = 0^a 1^b$, or $vxy = 1^a 0^b$.

WHAT ABOUT?

$\Sigma = \{0, 1\}$, $L_3 = \{ 0^m 1^n 0^n \mid m=n \geq 0 \}$

Choose $w = 0^P 1^P 0^P$.

By the Pumping Lemma, we can write $w = uvxyz$ with $|vy| > 0$, $|vxy| \leq P$ such that pumping v together with y will produce another word in L_3

Since $|vxy| \leq P$, $vxy = 0^a 1^b$, or $vxy = 1^a 0^b$.

Pumping in the first case will unbalance with the 0's at the end; in the second case, will unbalance with the 0's at the beginning. Contradiction.

THE PUMPING LEMMA FOR CFGs

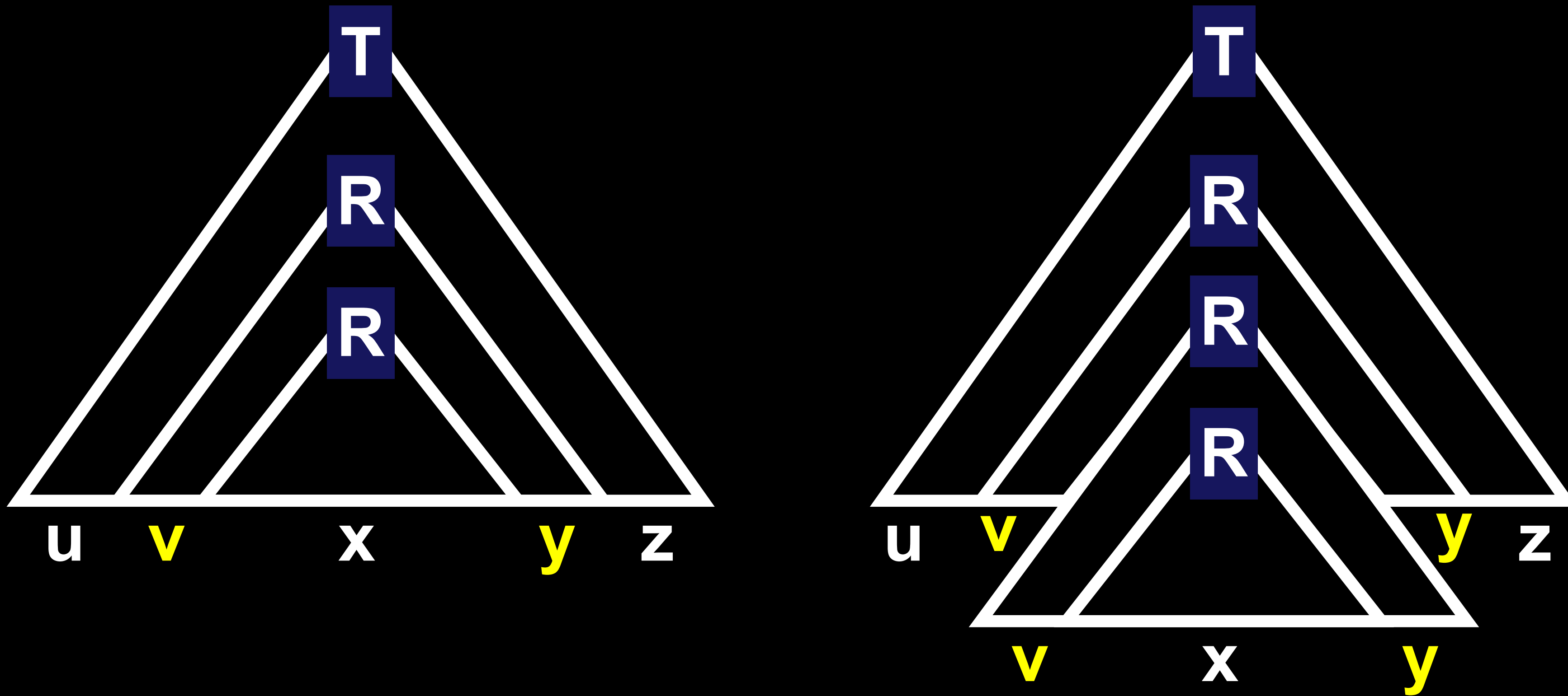
Let L be a context-free language

Then there is a P such that
if $w \in L$ and $|w| \geq P$

then can write $w = uvxyz$, where:

1. $|vy| > 0$
2. $|vxy| \leq P$
3. For *every* $i \geq 0$, $uv^ixy^iz \in L$

Idea of Proof: If w is long enough, then any parse tree for w must have a path that contains a variable more than once



Formal Proof:

Let b be the maximum number of symbols (length) on the right-hand side of any rule

If the height of a parse tree is h, the length of the string generated by that tree is at most: b^h

---

Let $|V|$ be the number of variables in G

Define $P = b^{|V|+1}$

Let w be a string of length at least P

Let T be a parse tree for w with a ***minimum*** number of **nodes.**

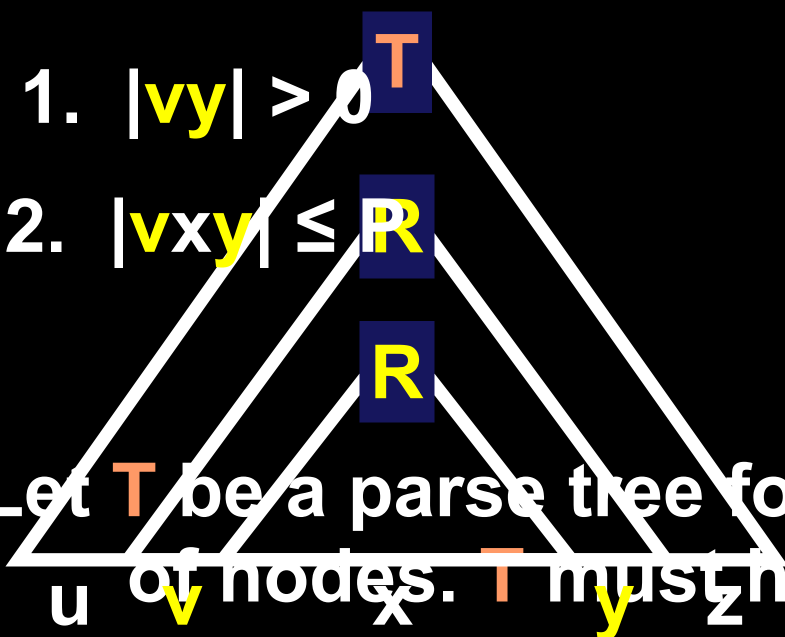
$$b^{|V|+1} = P \le |w| \le b^h$$

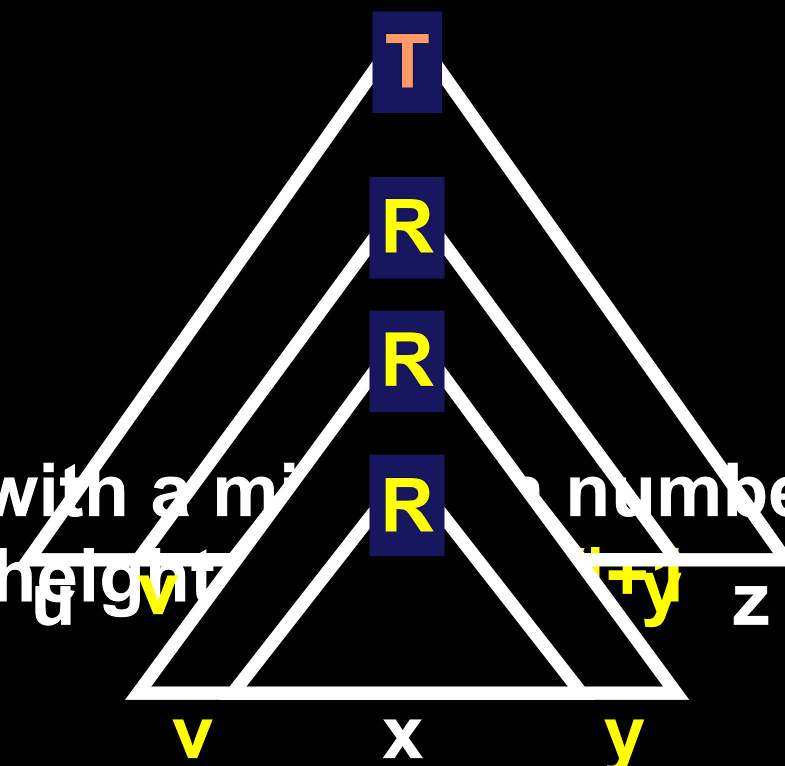
T must have height h at least $|V|+1$

The longest path in T must have ≥ |V|+1 variables

Select R to be a variable that repeats among the lowest |V|+1 variables (in the path)

1. |vy| > 0
2. |vxy| ≤ p



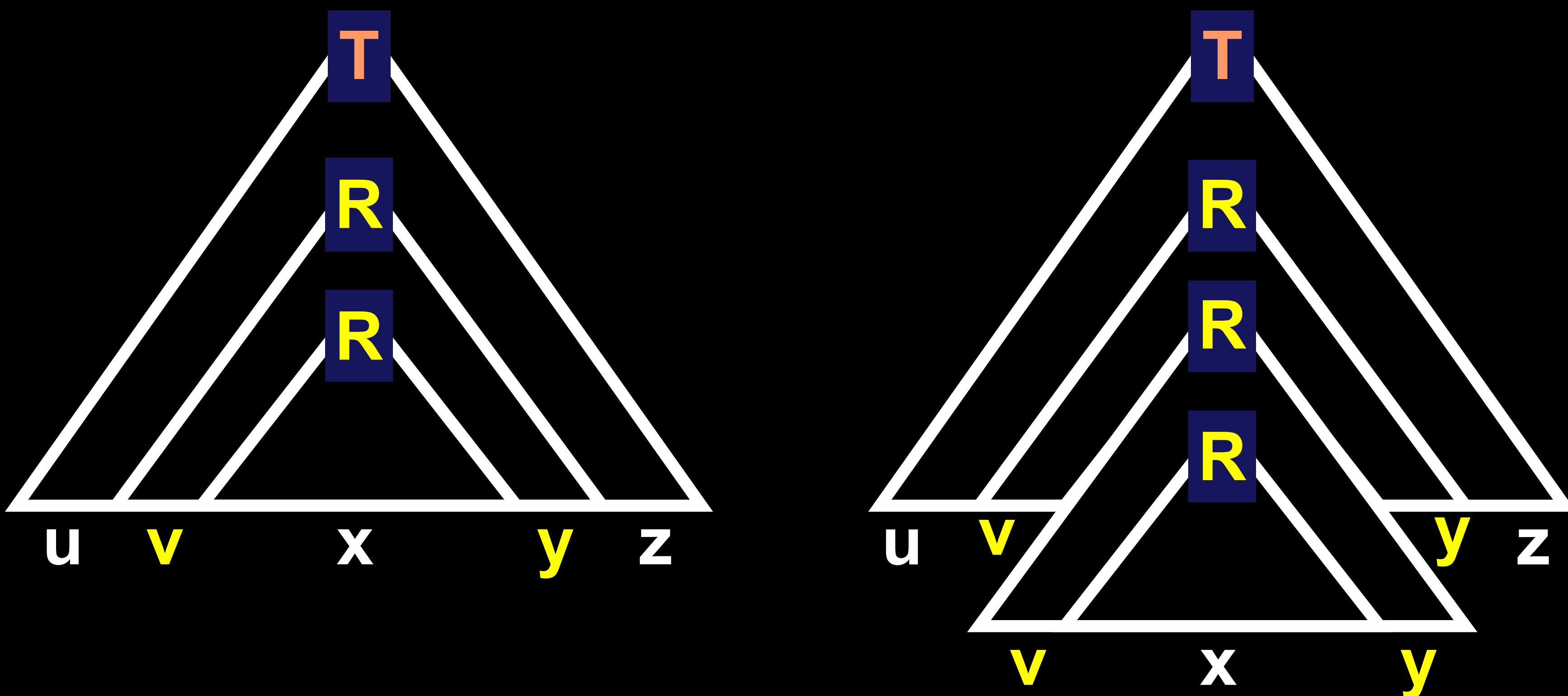


Let T be a parse tree for w with a mi number of nodes. T must have height

The longest path in T must have ≥ |V|+1 variables

Select R to be a variable in T that repeats, among the lowest |V|+1 variables in the tree

1. |vy| > 0 since T has minimun # nodes
2. |vxy| ≤ P since $|vxy| \leq b^{|V|+1} = P$



PDAs ARE EQUIVALENT TO CFGs

THURSDAY Jan 30

EQUIVALENCE OF CFGs and PDAs

A Language L is generated by a CFG

$\Leftrightarrow$

L is recognized by a PDA

A Language L is generated by a CFG

$\Leftrightarrow$

L is recognized by a PDA

Suppose L is generated by a CFG G = (V, Σ, R, S)

Construct P = (Q, Σ, Γ, δ, q, F) that recognizes L

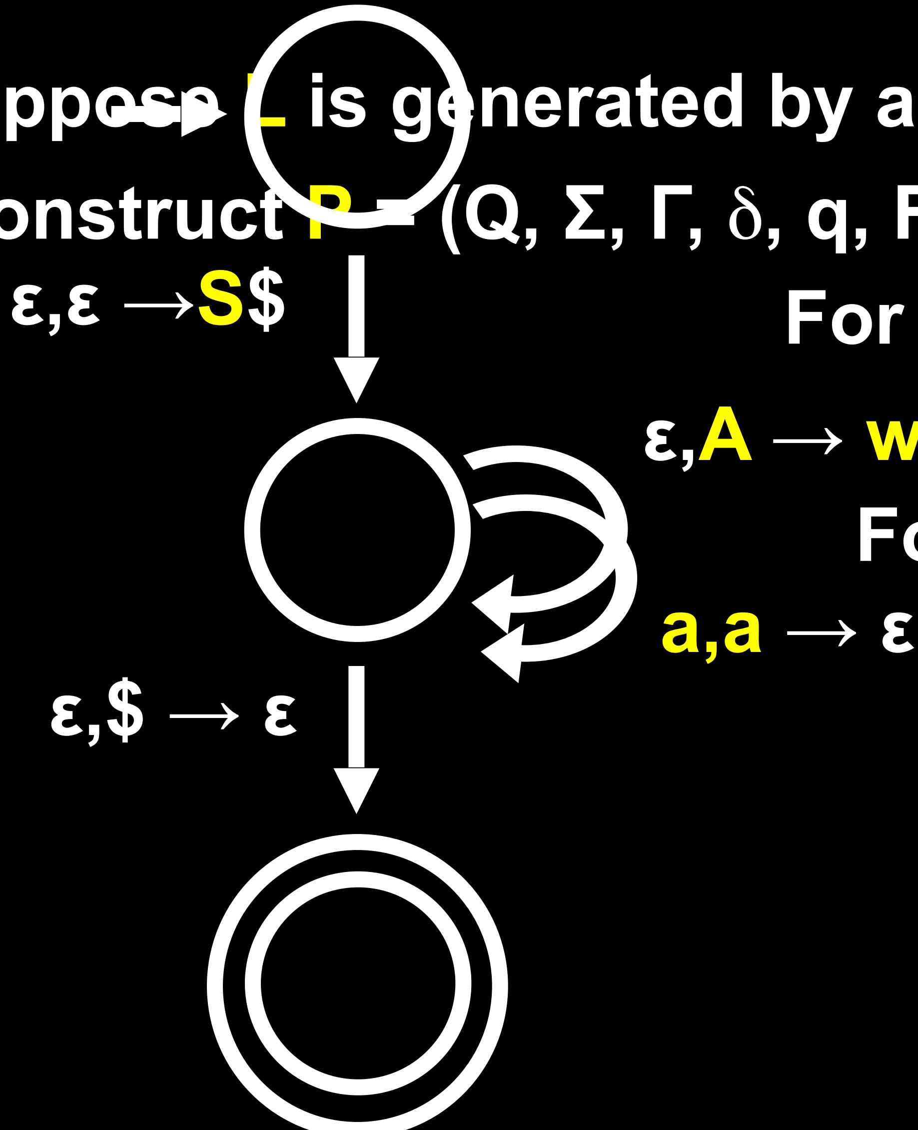
A Language L is generated by a CFG

$\Longrightarrow$

L is recognized by a PDA

Suppose **L** is generated by a CFG **G** = (V, Σ, R, S)

Construct **P** = (Q, Σ, Γ, δ, q, F) that recognizes **L**

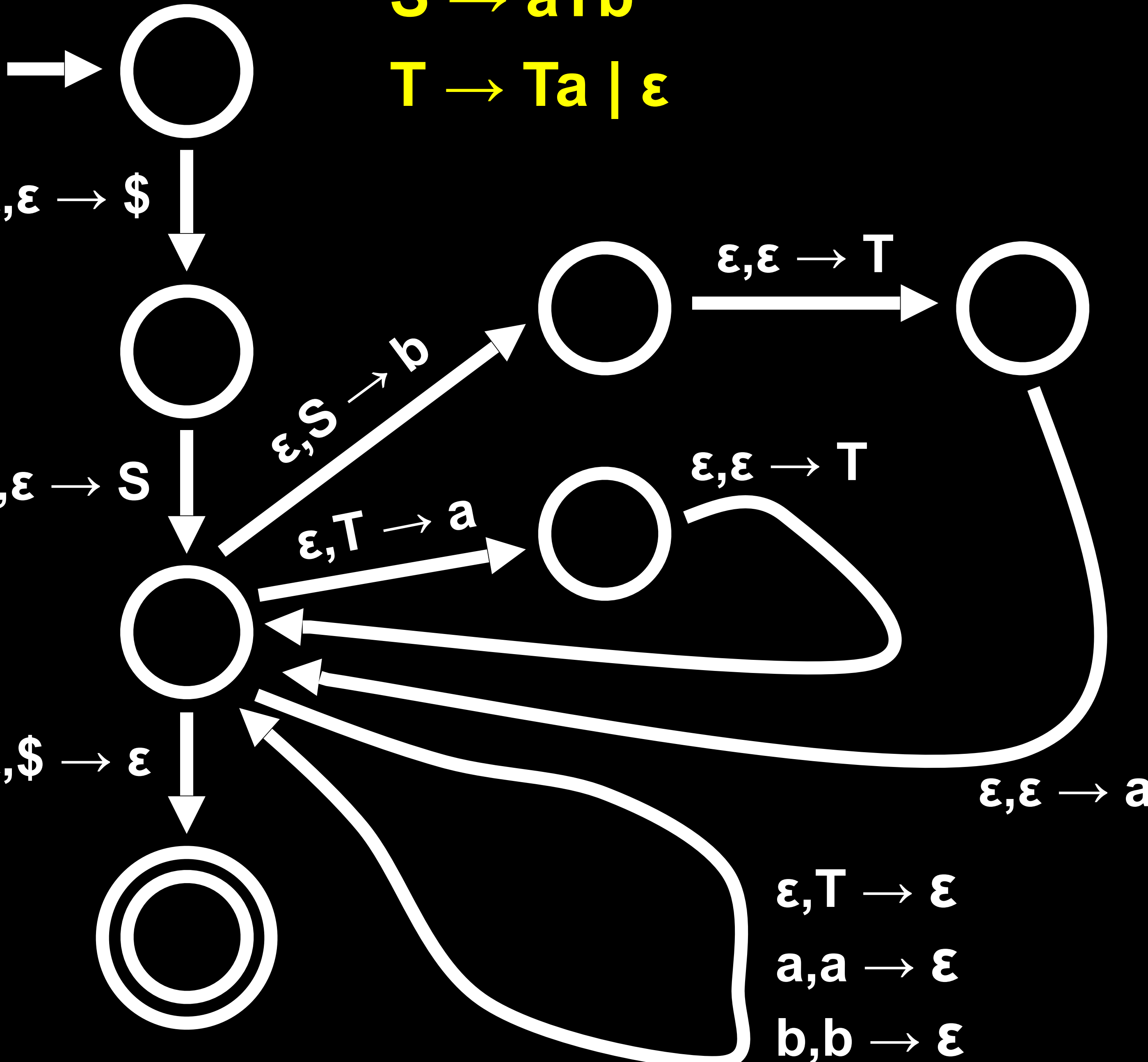


For each rule 'A → w' ∈ R:

For each terminal a ∈ Σ:

$S \rightarrow aTb$

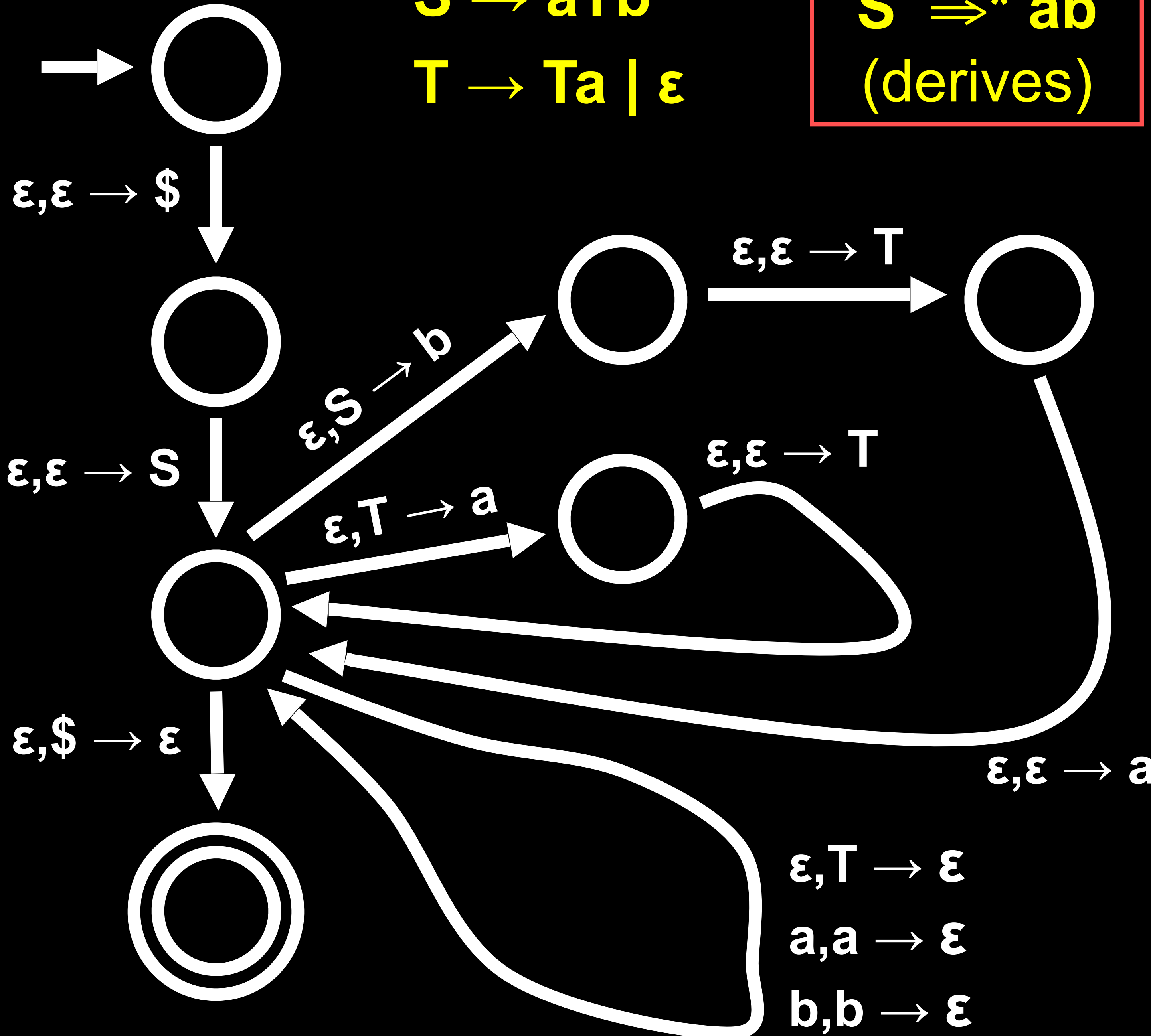
$T \rightarrow Ta \mid \varepsilon$



$S \rightarrow aTb$

$T \rightarrow Ta \mid \varepsilon$

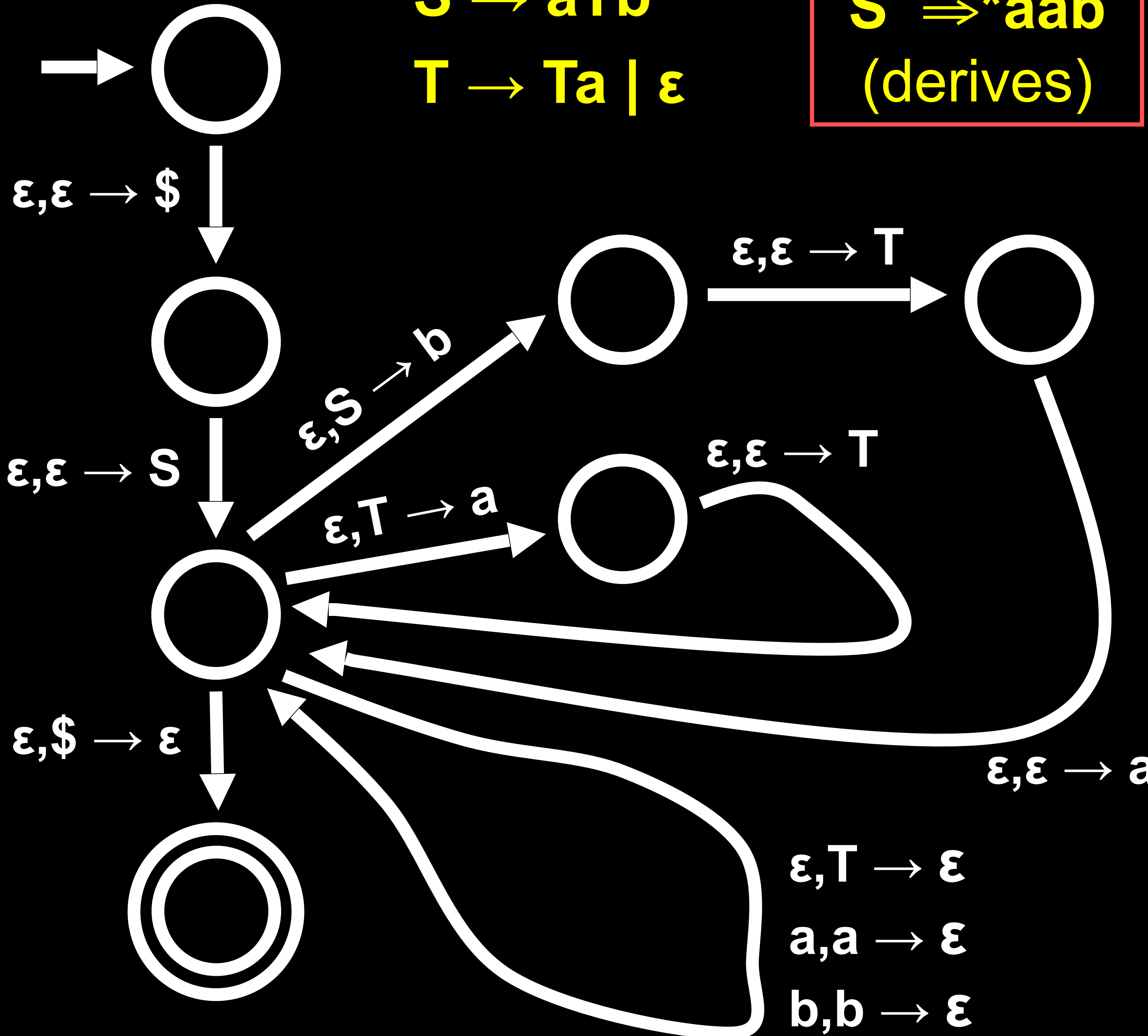
$S \Rightarrow^* ab$
(derives)



$S \rightarrow aTb$

$T \rightarrow Ta \mid \varepsilon$

$S \Rightarrow^* aab$
(derives)



Suppose L is generated by a CFG G = (V, Σ, R, S)

Describe P = (Q, Σ, Γ, δ, q, F) that recognizes L (via pseudocode):

(1) Push $ and then S on the stack

(2) Repeat the following steps forever:

(a) Suppose x is now on top of stack

(b) If x is a variable A, guess a rule for A and push yield into the stack and Go to (a).

(c) If x is a terminal, read next symbol from input and compare it to x. If they're different, *reject*. If same, pop x and Go to (a).

(d) If x is $: then *accept* iff no more input

A Language is generated by a CFG

$\Longrightarrow$

It is recognized by a PDA

A Language L is generated by a CFG <= L is recognized by a PDA

Given PDA $P = (Q, \Sigma, \Gamma, \delta, q, F)$

Construct a CFG $G = (V, \Sigma, R, S)$ such that $L(G) = L(P)$

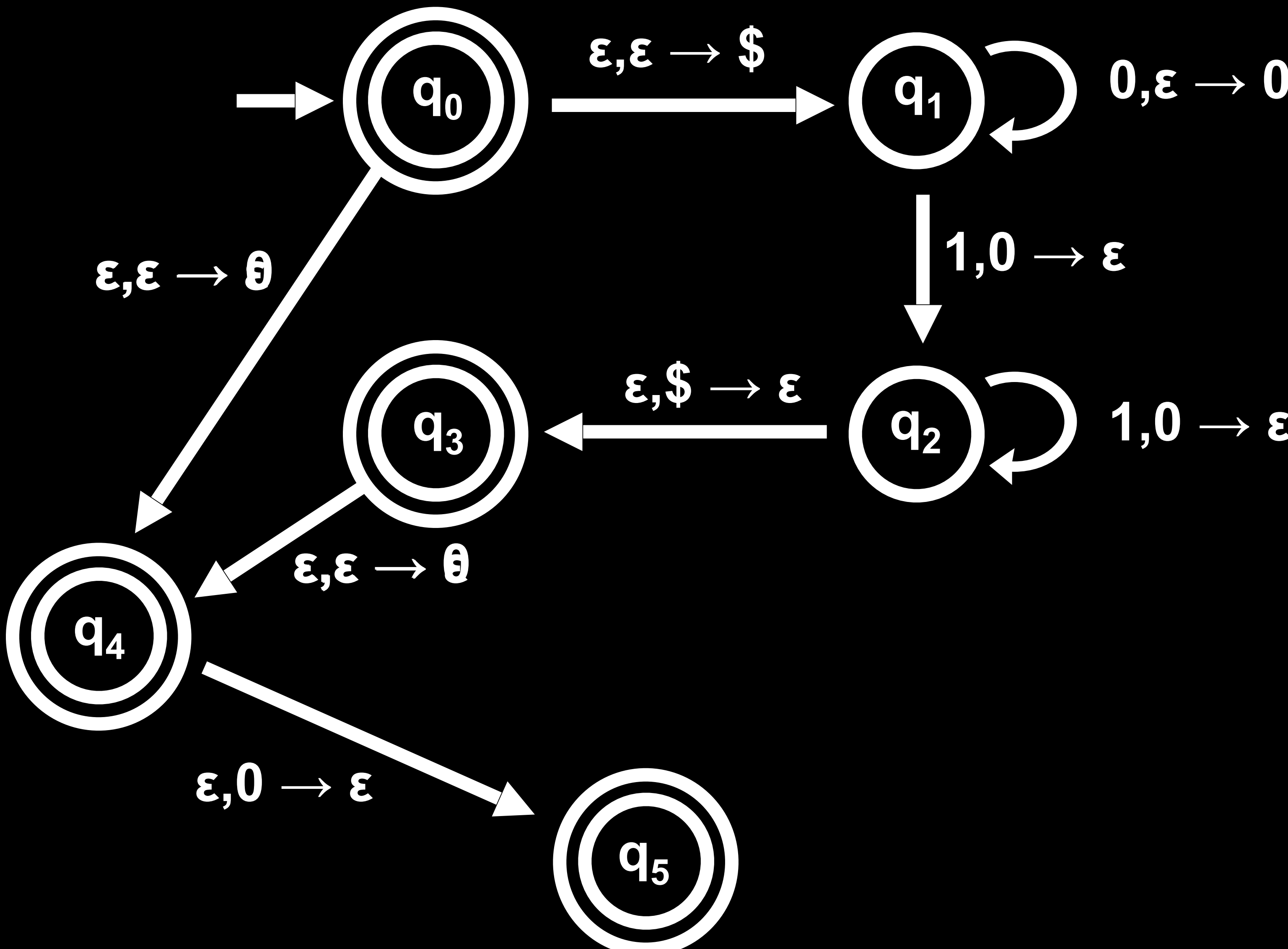
First, **simplify P** to have the following form:

(1) It has a unique accept state, q_{acc}

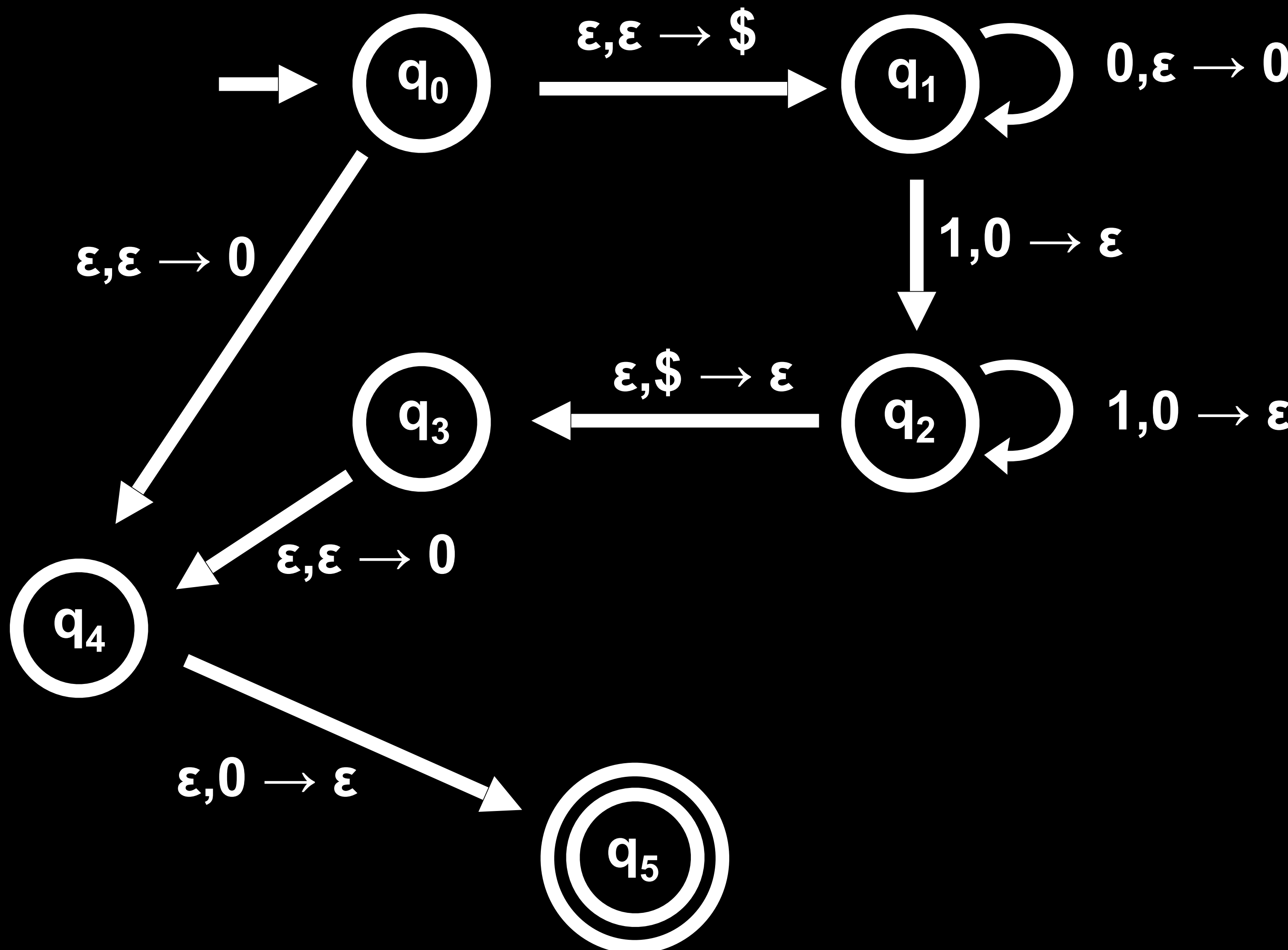
(2) It empties the stack before accepting

(3) Each transition either pushes a symbol or pops a symbol, but not both at the same time

SIMPLIFY



SIMPLIFY



q_0
$\varepsilon,\varepsilon \rightarrow \$$
q_1
$0,\varepsilon \rightarrow 0$
$\varepsilon,\varepsilon \rightarrow 0$
$1,0 \rightarrow \varepsilon$
$\varepsilon,\$ \rightarrow \varepsilon$
q_3
q_2
$1,0 \rightarrow \varepsilon$
$\varepsilon,\varepsilon \rightarrow 0$
q_4
$\varepsilon,0 \rightarrow \varepsilon$
q_5

Our task is to construct **Grammar G** to generate exactly the words that **PDA P** accepts.

Idea For Our Grammar G:

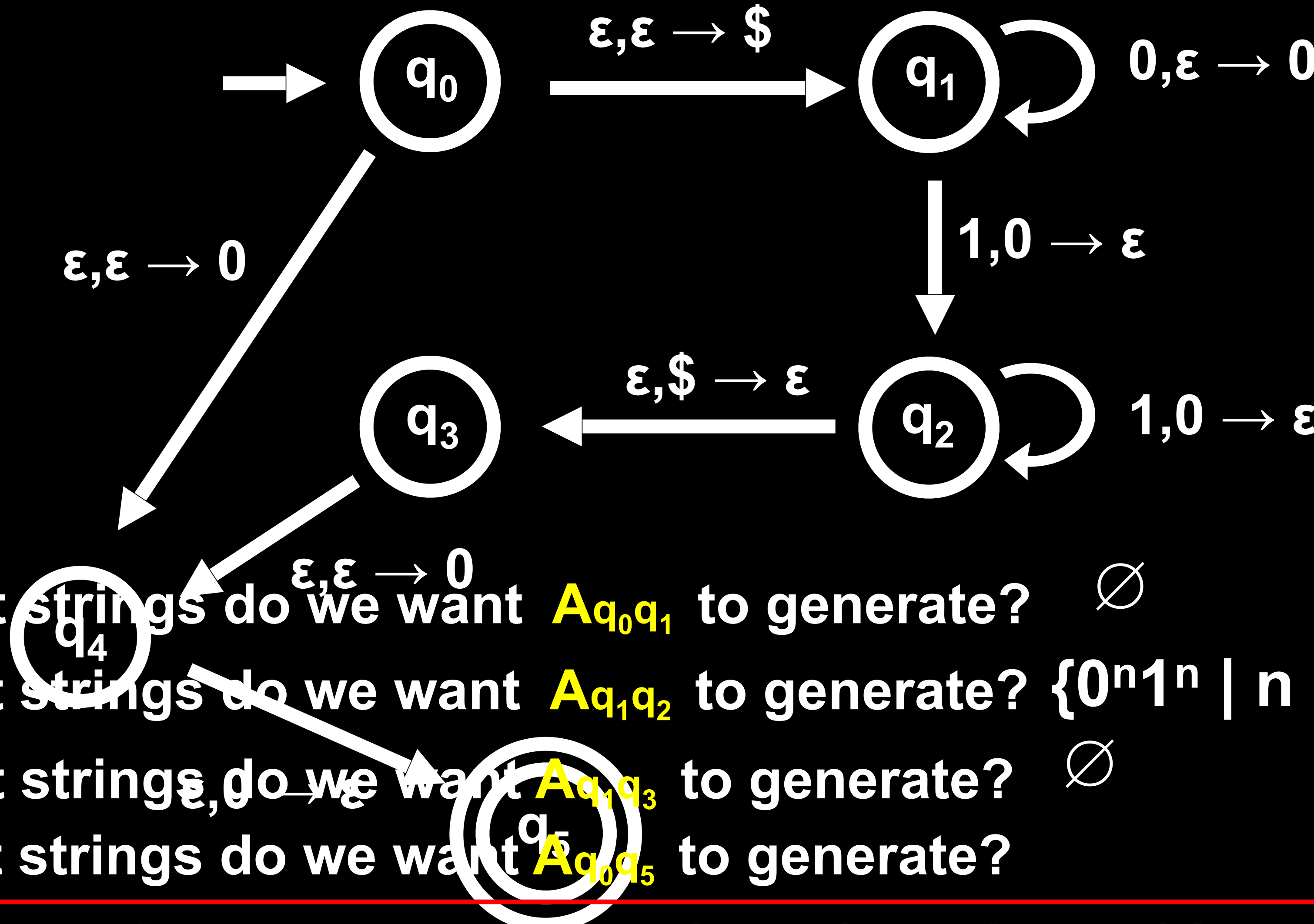
For every pair of states p and q in PDA P,

G will have a variable A_{pq} whose production rules will generate all strings x that can take:

P from p with an empty stack
to q with an empty stack

$$V = \{A_{pq} \mid p,q \in Q\}$$

$$S = A_{q_0 q_{acc}}$$



What strings do we want $A_{q_0q_1}$ to generate? $\varnothing$

What strings do we want $A_{q_1q_2}$ to generate? $\{0^n1^n \mid n > 0\}$

What strings do we want $A_{q_1q_3}$ to generate? $\varnothing$

What strings do we want $A_{q_0q_5}$ to generate?

WANT: A_{pq} to generate all strings that take **p** with an empty stack to **q** with empty stack

WANT: A_{pq} generates all strings that take p with an empty stack to q with empty stack

Let x be such a string

- P's first move on x must be a push (why?)
- P's last move on x must be a pop

Two possibilities:

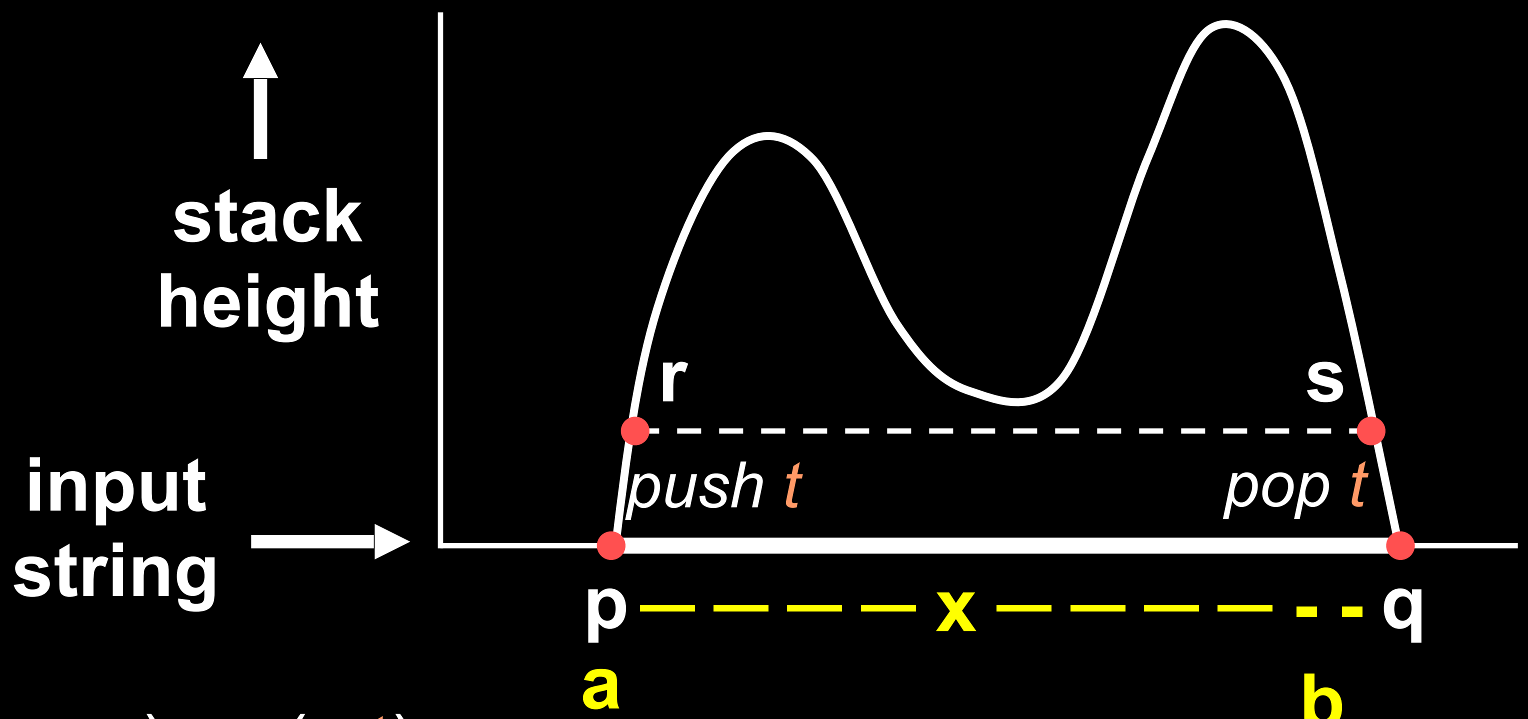
1. The symbol popped at the end is the one pushed at the beginning

2. The symbol popped at the end is not the one pushed at the beginning

(so P must empty stack somewhere in the middle, and then start pushing symbols on it again)

$x = ayb$ takes p with empty stack to q with empty stack

1. The symbol t popped at the end is exactly the one pushed at the beginning

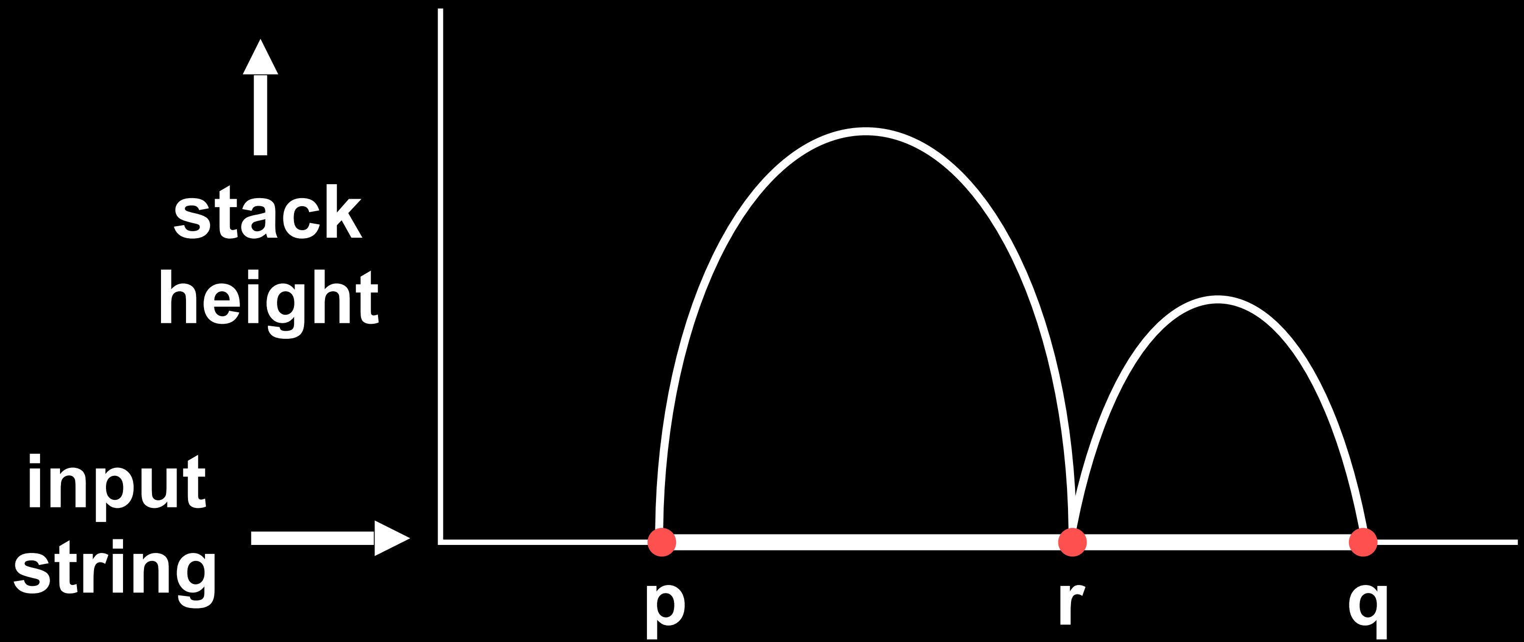


$\delta(p, a, \varepsilon) \rightarrow (r, t)$

$\delta(s, b, t) \rightarrow (q, \varepsilon)$

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

2. The symbol popped at the end is not the one pushed at the beginning



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

Formally:

$V = \{A_{pq} \mid p, q \in Q\}$

$S = A_{q_0 q_{acc}}$

For every $p, q, r, s \in Q$, $t \in \Gamma$ and $a, b \in \Sigma_\varepsilon$

If $(r, t) \in \delta(p, a, \varepsilon)$ and $(q, \varepsilon) \in \delta(s, b, t)$

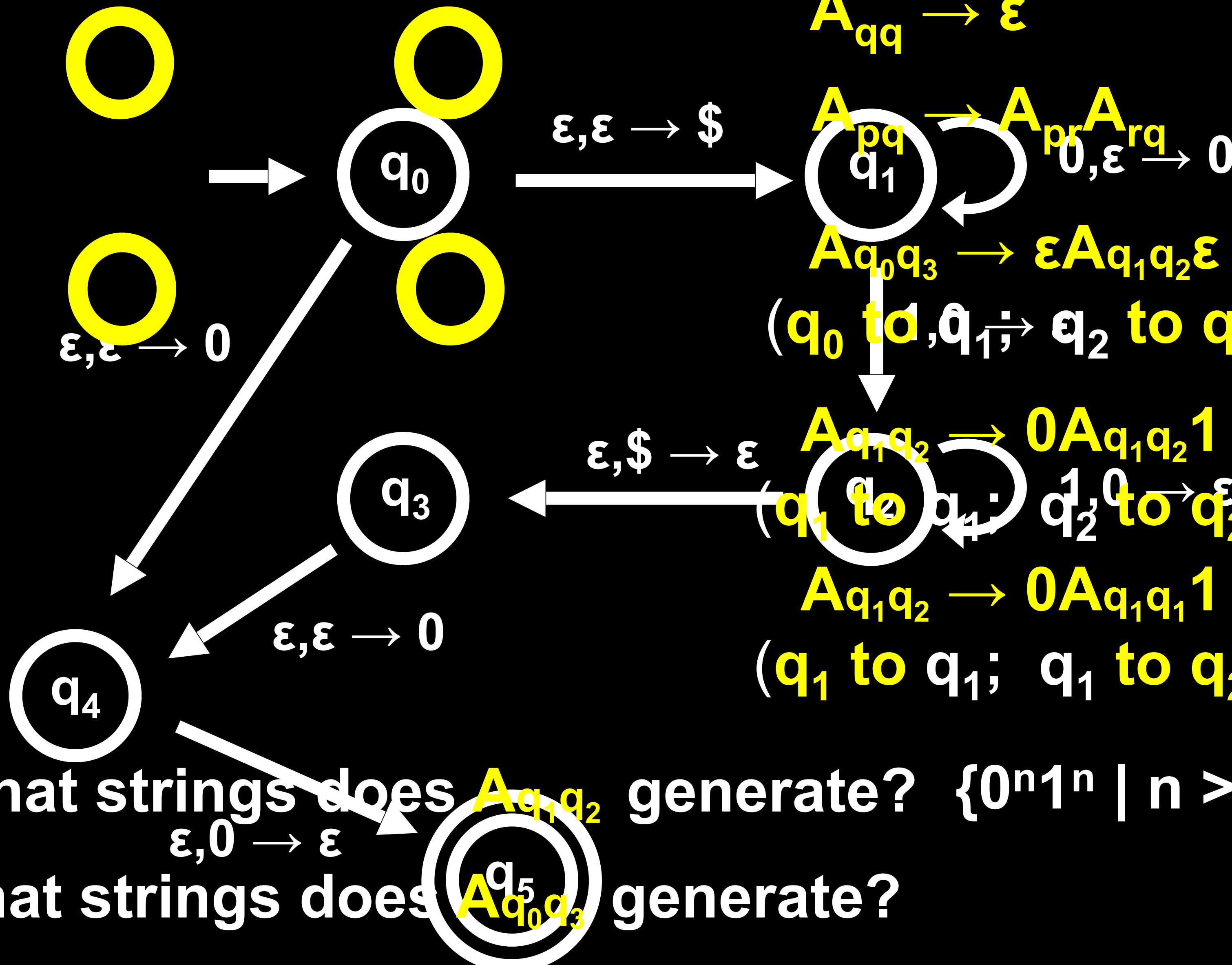
Then add the rule $A_{pq} \rightarrow aA_{rs}b$

For every $p, q, r \in Q$,

add the rule $A_{pq} \rightarrow A_{pr}A_{rq}$

For every $p \in Q$,

add the rule $A_{pp} \rightarrow \varepsilon$



$A_{qq} \to \varepsilon$

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

$A_{q_0q_3} \to \varepsilon A_{q_1q_2}\varepsilon$

(q_0 to q_1; q_2 to q_3)

$A_{q_1q_2} \to 0A_{q_1q_2}1$

(q_1 to q_1; q_2 to q_2)

$A_{q_1q_2} \to 0A_{q_1q_1}1$

(q_1 to q_1; q_1 to q_2)

What strings does $A_{q_1q_2}$ generate? $\{0^n1^n \mid n > 0\}$

What strings does $A_{q_0q_3}$ generate?

What strings does $A_{q_0q_5}$ generate?

Show, for all x,

A_{pq} generates x

$\Leftrightarrow$

x can bring P from p with an empty stack to q with an empty stack

Show, for all x, A_{pq} generates x

$$\Rightarrow$$

x can bring P from p with an empty stack to q with an empty stack

Proof (by induction on the number of steps in the derivation of x from A_{pq}):

Base Case: The derivation has 1 step: $A_{pp} \Rightarrow \varepsilon$

Inductive Step: Assume true for derivations of length ≤ k and prove true for derivations of length k+1:

$$A_{pq} \Rightarrow^{*} x \quad \text{in} \quad k+1 \text{ steps}$$

First step in derivation: $A_{pq} \rightarrow A_{pr}A_{rq}$ or $A_{pq} \rightarrow aA_{rs}b$

Show, for all x, A_{pq} generates x

$\Rightarrow$

x can bring P from p with an empty stack to q with an empty stack

Proof (by induction on the number of steps in the derivation of x from A_{pq}):

Inductive Step:

Assume true for derivations of length $\leq k$ and prove true for derivations of length $k+1$:

$$A_{pq} \Rightarrow^* x \quad \text{in} \quad k+1 \text{ steps}$$

First step in derivation: $A_{pq} \rightarrow A_{pr}A_{rq}$

Then, $x = yz$ with $A_{pr} \Rightarrow^* y$, $A_{rq} \Rightarrow^* z$

By IH, y can take p with empty stack to r with empty stack; similarly for z from r to q. So, …

Show, for all x, A_{pq} generates x

$\Rightarrow$

x can bring P from p with an empty stack
to q with an empty stack

Proof (by induction on the number of steps in the derivation of x from A_{pq}):

Inductive Step:

Assume true for derivations of length ≤ k and prove true for derivations of length k+1:

$$A_{pq} \Rightarrow^* x \quad \text{in} \quad k+1 \text{ steps}$$

First step in derivation: or $A_{pq} \rightarrow aA_{rs}b$

Then $x = ayb$ with $A_{rs} \Rightarrow^* y$.

By IH, y can take r with empty stack to s with empty stack

Show, for all x,

A_{pq} **generates x**

$\Rightarrow$

x can bring P from p with an empty stack to q with an empty stack

Proof (by induction on the number of steps in the derivation of x from A_{pq}):

Inductive Step:

Assume true for derivations of length ≤ k and prove true for derivations of length k+1:

$$A_{pq} \Rightarrow^* x \quad \text{in} \quad k+1 \text{ steps}$$

First step in derivation: **or** $A_{pq} \rightarrow aA_{rs}b$

By def of rules of G, $(r,t) \in \delta(p,a,\varepsilon)$ **and** $(q,\varepsilon) \in \delta(s,b,t)$

state push state alphabet pop

Show, for all x, A_{pq} generates x

$$\Rightarrow$$

x can bring P from p with an empty stack to q with an empty stack

Proof (by induction on the number of steps in the derivation of x from A_{pq}):

Inductive Step:

Assume true for derivations of length $\leq$ k and prove true for derivations of length k+1:

$$A_{pq} \Rightarrow^* x \quad \text{in} \quad k+1 \text{ steps}$$

First step in derivation: **or $A_{pq} \rightarrow aA_{rs}b$**

So if P starts in p then after reading a, it can go to r and push t.
By IH, y can bring P from r to s, with t at the top of the stack.
Then from s reading b, it can pop t and end in state q.

Show, for all x, A_{pq} generates x

$\Leftarrow$

x can bring P from p with an empty stack to q with an empty stack

Proof (by induction on the number of steps in the computation of P from p to q with empty stacks on input x):

Base Case: The computation has 0 steps

So it starts and ends in the same state.
The only string that can do that in 0 steps is ε.

Since $A_{pp} \rightarrow \varepsilon$ is a rule of G, $A_{pp} \Rightarrow^* \varepsilon$

Inductive Step:

Assume true for computations of length ≤ k, we'll prove true for computations of length k+1

Suppose that P has a computation where x brings p to q with empty stacks in k+1 steps

Two cases: (idea!)

1. The stack is empty only at the beginning and the end of this computation

2. The stack is empty somewhere in the middle of the computation

Inductive Step:

Assume true for computations of length ≤ k, we'll prove true for computations of length k+1

Suppose that P has a computation where x brings p to q with empty stacks in k+1 steps

Two cases: (idea!)

1. The stack is empty only at the beginning and the end of this computation

To Show: Can write x as ayb where $A_{rs} \Rightarrow^* y$ and $A_{pq} \rightarrow aA_{rs}b$ is a rule in G. So $A_{pq} \Rightarrow^* x$

2. The stack is empty somewhere in the middle of the computation

To Show: Can write x as yz where $A_{pr} \Rightarrow^* y$, $A_{rq} \Rightarrow^* z$ and $A_{pq} \rightarrow A_{pr}A_{rq}$ is a rule in G. So $A_{pq} \Rightarrow^* x$

Inductive Step:

1. The stack is empty *only* at the beginning and the end of this computation

To Show: Can write x as ayb where $A_{rs} \Rightarrow^* y$ and $A_{pq} \rightarrow aA_{rs}b$ is a rule in G. So $A_{pq} \Rightarrow^* x$

The symbol t pushed at the beginning must be the same symbol popped at the end. **why?**)

Let a be input symbol read at beginning, b read at end.

- So x = ayb, for some y.

Let r be the state after the first step, let s be the state before the last step.

- y can bring P from r with an empty stack to s with an empty stack. (why?) So by IH, $A_{rs} \Rightarrow^* y$.
- Also, $A_{pq} \rightarrow aA_{rs}b$ **must be a rule in G.** (**why?**)

Inductive Step:

2. The stack is empty somewhere in the middle of the computation

To Show: Can write x as yz where $A_{pr} \Rightarrow^* y$, $A_{rq} \Rightarrow^* z$ and $A_{pq} \rightarrow A_{pr}A_{rq}$ is a rule in G. So $A_{pq} \Rightarrow^* x$

Let r be a state in which the stack becomes empty in the middle.

Let y be the input read to that point, z be input read after. So, $x = yz$ where $|y|, |z| > 0$.

By IH, both $A_{pr} \Rightarrow^* y$, $A_{rq} \Rightarrow^* z$

By construction of G, $A_{pq} \rightarrow A_{pr}A_{rq}$ **is a rule in G**

A Language is generated by a CFG

$\Leftrightarrow$

It is recognized by a PDA

Corollary: Every regular language is context-free