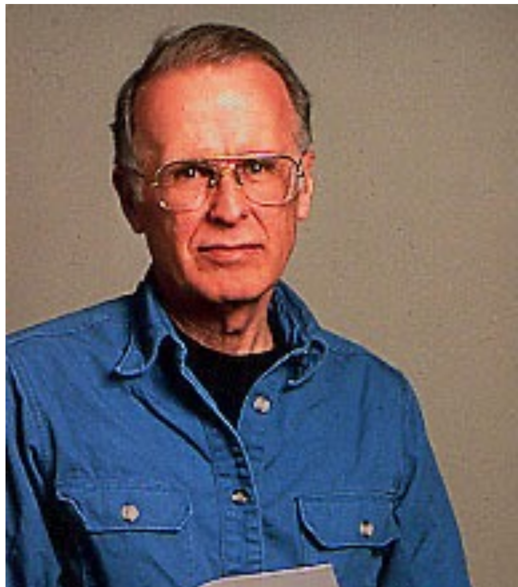
Theory of Computation Chapter 2

Context-Free Languages

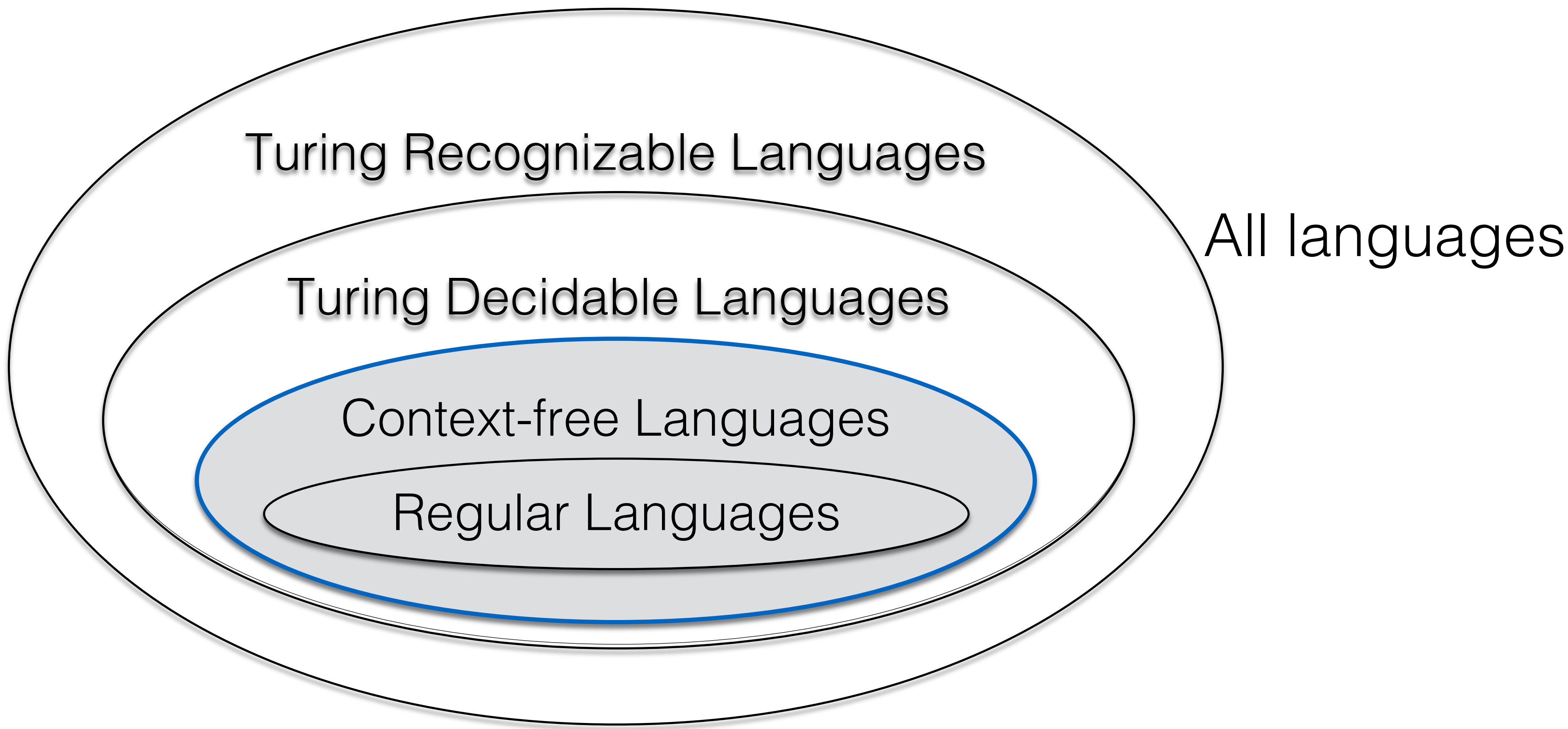
School of Engineering | Computer Science

John Backus
1924-2007

- Director of team that created Fortran, one of the first high-level programming languages (ever!) in 1957
- Fortran is still used today, in science/engineering fields
- Co-creator of Backus-Naur Form, which is used (today) to precisely describe the grammars of programming languages
- 1977 Turing Award winner



Context-Free Languages



Context-Free Languages

PDA = CFG = CNF

Closed under union, ∪, concatenation, ∘, and star, ∗.

VCU School of Engineering | Computer Science

Context-Free Languages

- Context-free languages are the next level up from regular languages

- They provide more power or expressiveness than regular languages since they have the ability to store information on a stack
- Most programming languages are Context-Free Languages

Context-Free Languages

- We describe context-free languages with context-free grammars (CFG)
- The grammar consists of a set of substitution rules of variables and terminals
- Variables are the symbols that help us derive the strings of the language, which consist of terminals
- Ex: $L(G_1) = \{0^n\#1^n \mid n \geq 0\}$
  - Grammar: $A \rightarrow 0A1$
    $A \rightarrow B$
    $B \rightarrow \#$

Context-Free Languages

- The grammar consists of a set of substitution rules of variables and terminals
- Ex: $L(G_1) = \{0^n\#1^n \mid n \geq 0\}$
  - Grammar: $A \rightarrow 0A1$
    $A \rightarrow B$
    $B \rightarrow \#$
    - Variables: A and B
    - Terminals: 0, 1, #
    - Variable A is the <u>start variable</u> since it starts the first rule
    - There are three grammar rules listed

VCU School of Engineering | Computer Science

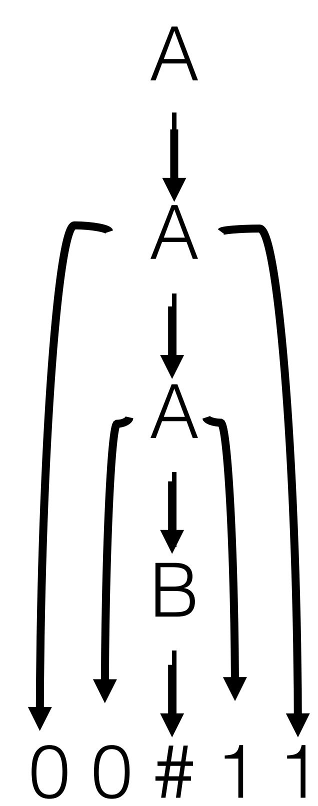
Context-Free Languages

- Ex: $L(G_1) = \{0^n\#1^n \mid n \geq 0\}$
  - Grammar: $A \rightarrow 0A1$
    $A \rightarrow B$
    $B \rightarrow \#$
  - To generate a string from this CFG:
    1. Start with the start variable (A)
    2. While variables are left, pick a variable and replace it with a substitution rule
    - Ex: $A \Rightarrow 0A1 \Rightarrow 0B1 \Rightarrow 0\#1$
    - Ex: $A \Rightarrow B \Rightarrow \#$
    - The sequence of substitutions to obtain a string is called a <u>derivation</u>.

VCU School of Engineering | Computer Science

Context-Free Languages

- We can also use a <u>parse tree</u> to represent the same information pictorially
- Ex: $L(G_1) = \{0^n\#1^n \mid n \geq 0\}$
  - Grammar: $A \rightarrow 0A1$
    $A \rightarrow B$
    $B \rightarrow \#$
    - Ex: $A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 00B11 \Rightarrow 00\#11$



Context-Free Languages

- English language example:
  - <sentence> → <noun-phrase><verb-phrase>
  - <noun-phrase>→ <complex-noun>|<complex-noun><prep-phrase>
  - <verb-phrase>→ <complex-verb>|<complex-verb><prep-phrase>
  - <prep-phrase> → <prep><complex-noun>
  - <complex-noun> → <article><noun>
  - <complex-verb> → <verb>|<verb><noun-phrase>
  - <article> → a | the
  - <noun> → boy | girl | flower
  - <verb> → touches | likes | sees
  - <prep> → with

The | symbol means you can choose one derivation or the other for that rule

Context-Free Languages

<sentence> → <noun-phrase><verb-phrase>
<noun-phrase>→ <complex-noun>| <complex-noun><prep-phrase>
<verb-phrase>→ <complex-verb>| <complex-verb><prep-phrase>
<prep-phrase> → <prep><complex-noun>
<complex-noun> → <article><noun>
<complex-verb> → <verb>|<verb><noun-phrase>
<article> → a | the
<noun> → boy | girl | flower
<verb> → touches | likes | sees
<prep> → with

- Example derivation:
  - <sentence> ⇒ <noun-phrase><verb-phrase>
    ⇒ <complex-noun><verb-phrase>
    ⇒ <article><noun>< verb-phrase >
    ⇒ A <noun>< verb-phrase >
    ⇒ A boy < verb-phrase >
    ⇒ A boy <complex-verb><prep-phrase>
    ⇒ A boy <verb><prep-phrase>
    ⇒ A boy touches <prep-phrase>
    ⇒ A boy touches <prep><complex-noun>
    ⇒ A boy touches with <complex-noun>
    ⇒ A boy touches with <article><noun>
    ⇒ A boy touches with the <noun>
    ⇒ A boy touches with the flower

Context-Free Grammar

- Formal Definition of the Context-Free Grammar (CFG)
  - A CFG is a 4-tuple (V, Σ, R, S) such that:
    1. V is a finite set of variables
    2. Σ is a finite set of terminals such that $V \cap \Sigma = \emptyset$
    3. R is a finite set of substitution rules, where a rule has a variable on the left and variables/terminals on the right
    4. $S \in V$ is the start variable

School of Engineering | Computer Science

Context-Free Terminology

- Context-Free Grammar (CFG) Terminology
  - $u \Longrightarrow v$ means u <u>yields</u> v
    - applying one substitution rule to u gives v
  - $u \Longrightarrow^* v$ means u <u>derives</u> v
    - Either:
      1. $u = v$, or
      2. there is a sequence of strings $u_1, u_2, \ldots, u_k$ exists for $k \geq 0$ such that $u \Longrightarrow u_1 \Longrightarrow u_2 \Longrightarrow \ldots \Longrightarrow u_k \Longrightarrow v$
  - The language of G, $L(G) = \{x \in \Sigma^* \mid S \Longrightarrow^* x\}$

VCU School of Engineering | Computer Science

Context-Free Grammar

- Context-Free Grammar (CFG) Example 2.3
  - G = ({S}, {a, b}, R, S) where the set of rules,

    R is: $S \rightarrow aSb \mid SS \mid \varepsilon$
  - L(G) = {set of all strings of the same number of a's and b's (where no prefix has more b's than a's)}
    - Ex: ε, ab, aabb, abab, aababb
  - Can think of a and b as (and), so L(G) = {set of all strings of balanced parentheses
    - Ex: from above - ε, (), (( )), ( )( ), (( )( ))
    - But not aabaabb or (( )(( )) or abba ())(

VCU School of Engineering | Computer Science

Context-Free Grammar

- Context-Free Grammar (CFG) Example 2.4
  - $G = (V, \Sigma, R, A)$ where $V = \{A, B, C\}$, $\Sigma = \{a, +, \times, (,)\}$
    - Rules R: $A \rightarrow A + B \mid B$
      $B \rightarrow B \times C \mid C$
      $C \rightarrow (A) \mid a$
    - String $a + a \times a$

VCU School of Engineering | Computer Science

Context-Free Grammar

- Context-Free Grammar (CFG) Example 2.4
  - $G = (V, \Sigma, R, A)$ where $V = \{A, B, C\}$, $\Sigma = \{a, +, \times, (,)\}$
    - Rules R: $A \rightarrow A + B \mid B$
      $B \rightarrow B \times C \mid C$
      $C \rightarrow (A) \mid a$
    - String $a + a \times a$

$$
\begin{aligned}
A &\Rightarrow A + B \\
&\Rightarrow A + B \times C \\
&\Rightarrow B + B \times C \\
&\Rightarrow C + B \times C \\
&\Rightarrow a + B \times C
\end{aligned}
\qquad
\begin{aligned}
&\Rightarrow a + B \times C \\
&\Rightarrow a + C \times C \\
&\Rightarrow a + a \times C \\
&\Rightarrow a + a \times a
\end{aligned}
$$

Context-Free Grammar

- Ex 2.4 cont.
  - String a + a × a (parse tree)

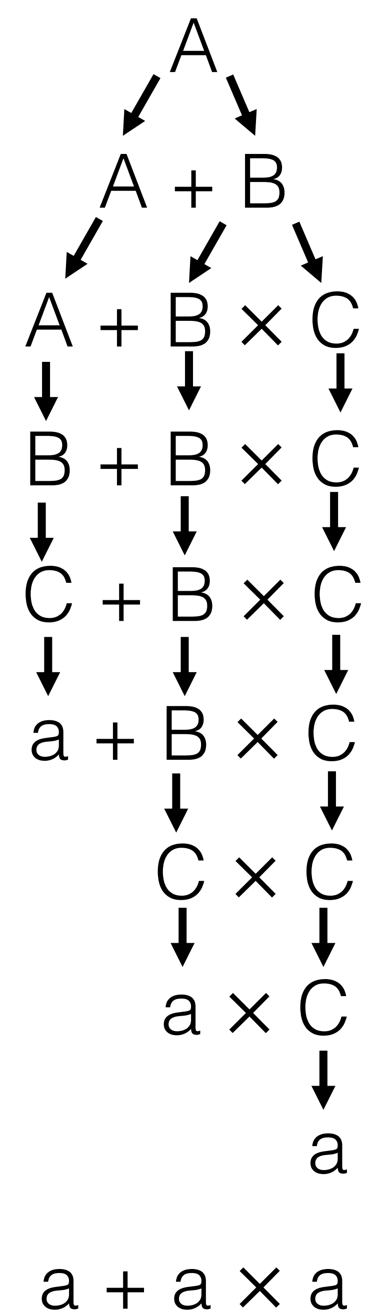
$A \rightarrow A + B \mid B$
$B \rightarrow B \times C \mid C$
$C \rightarrow (A) \mid a$

VCU School of Engineering | Computer Science

Context-Free Grammar

- Ex 2.4 cont.
  - String a + a × a (parse tree)

$A \rightarrow A + B \mid B$
$B \rightarrow B \times C \mid C$
$C \rightarrow (A) \mid a$



Context-Free Grammar

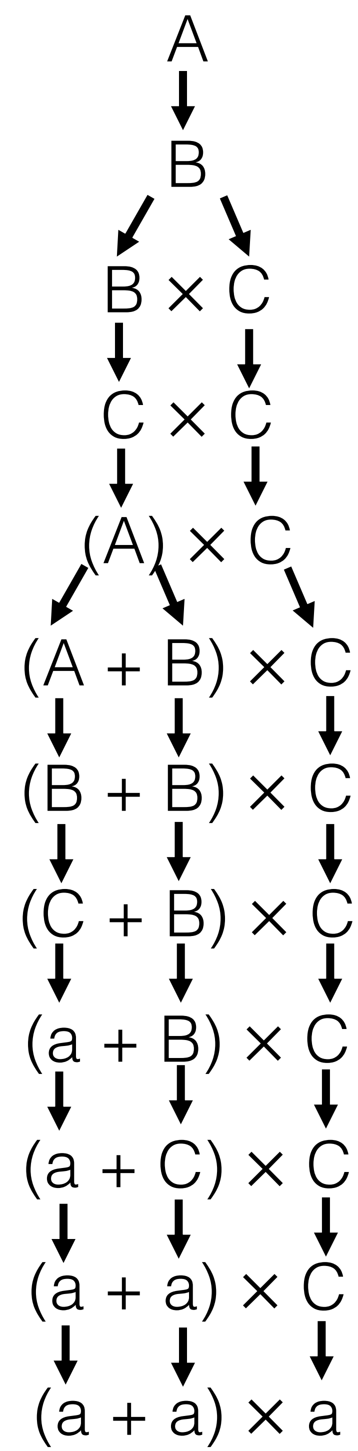
- Ex 2.4 cont.
  - String $(a + a) \times a$

$$A \rightarrow A + B \mid B$$
$$B \rightarrow B \times C \mid C$$
$$C \rightarrow (A) \mid a$$

School of Engineering | Computer Science

Context-Free Grammar

- Ex 2.4 cont.
    - String (a + a) × a
        - $$\begin{aligned} A &\Rightarrow B \\ &\Rightarrow B \times C \\ &\Rightarrow C \times C \\ &\Rightarrow (A) \times C \\ &\Rightarrow (A + B) \times C \\ &\Rightarrow (B + B) \times C \\ &\Rightarrow (C + B) \times C \\ &\Rightarrow (a + B) \times C \\ &\Rightarrow (a + C) \times C \\ &\Rightarrow (a + a) \times C \\ &\Rightarrow (a + a) \times a \end{aligned}$$

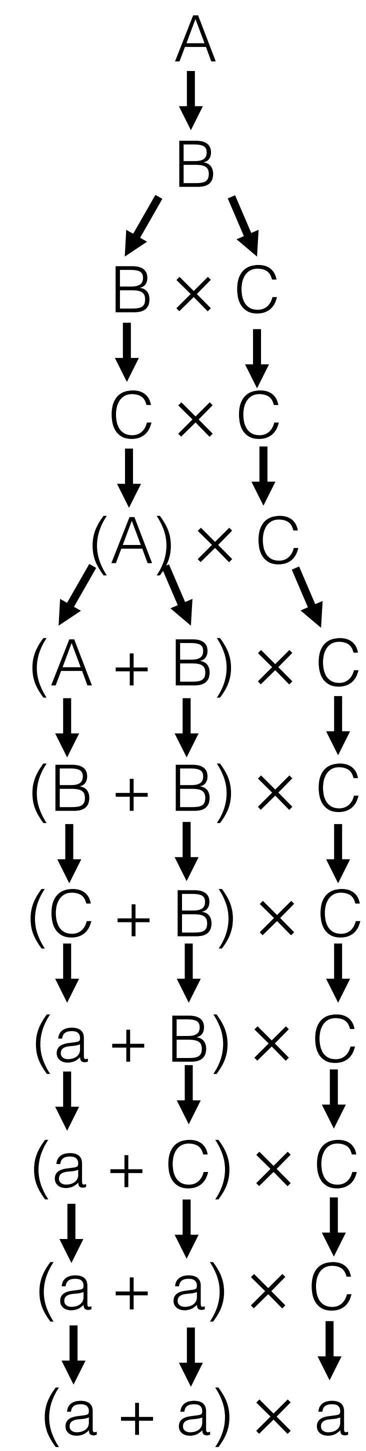


$A \rightarrow A + B \mid B$
$B \rightarrow B \times C \mid C$
$C \rightarrow (A) \mid a$

School of Engineering | Computer Science

Context-Free Grammar

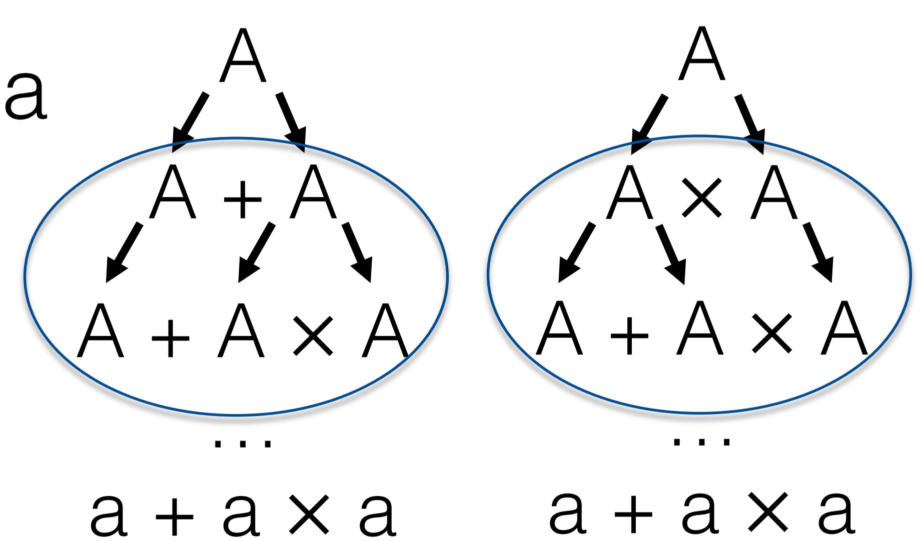
- Ex 2.4 cont.
  - $A \rightarrow A + B \mid B$
    $B \rightarrow B \times C \mid C$
    $C \rightarrow (A) \mid a$
  - There is just one way to derive the strings from this language
    - Ex: String $(a + a) \times a$ has only one possible parse tree, so does the string $a + a \times a$ or any other string from this language
  - The language is <u>unambiguous</u>



School of Engineering | Computer Science

Ambiguity in Grammars

- Definition 2.7:
  - A string w is ambiguously derived in CFG G if w has at least two different left-most derivations (where the left-most variable is always replaced first)
  - G is ambiguous if it generates some strings ambiguously
  - Note: some languages are inherently ambiguous
    - Ex: $A \rightarrow A + A \mid A \times A \mid A \mid a$
      - String: $a + a \times a$ has two different parse trees



Creating CFGs

- Give a Context-Free Grammar for the language L = {w | w is odd and the middle symbol is 0} where Σ = {0, 1}

Creating CFGs

- Give a Context-Free Grammar for the language L = {w | w is odd and the middle symbol is 0} where Σ = {0, 1}

$$S \rightarrow 0 \mid 0S0 \mid 1S1 \mid 0S1 \mid 1S0$$

School of Engineering | Computer Science

Creating CFGs

- Give a Context-Free Grammar for the language L = {w | w is even} where $\Sigma = \{0, 1\}$

Creating CFGs

- Give a Context-Free Grammar for the language L = {w | w is even} where $\Sigma = \{0, 1\}$

$$S \rightarrow \varepsilon \mid 0S0 \mid 1S1 \mid 0S1 \mid 1S0$$

Try It

1. Give a CFG for the language L = {w | w starts and ends with the same symbol} $\Sigma = \{0, 1\}$

2. Show the derivation or parse tree of the following string, 011001, using the grammar G:

$$S \rightarrow TSV \mid VST \mid \varepsilon$$

$$T \rightarrow 0$$

$$V \rightarrow 1$$

Is the grammar above ambiguous? Why or why not?

Try It

1. Give a CFG for the language L = {w | w starts and ends with the same symbol} where $\Sigma = \{0, 1\}$

$A \rightarrow 0B0 \mid 1B1 \mid 0 \mid 1$
$B \rightarrow 0B \mid 1B \mid \varepsilon$

Try It

2. Show the derivation or parse tree of the following string, 011001, using the grammar G:

$$S \rightarrow TSV \mid VST \mid \varepsilon$$
$$T \rightarrow 0$$
$$V \rightarrow 1$$

Is the grammar above ambiguous? Why or why not?

- No. There is only one possible parse tree for any string in the language.

S
↓
TSV
↓
0SV
↓
0VSTV
↓
01STV
↓
01VSTTV
↓
011STTV
↓
011εTTV
↓
011ε0TV
↓
011ε00V
↓
011ε001

011001

VCU School of Engineering | Computer Science

Theory of Computation
Chapter 2

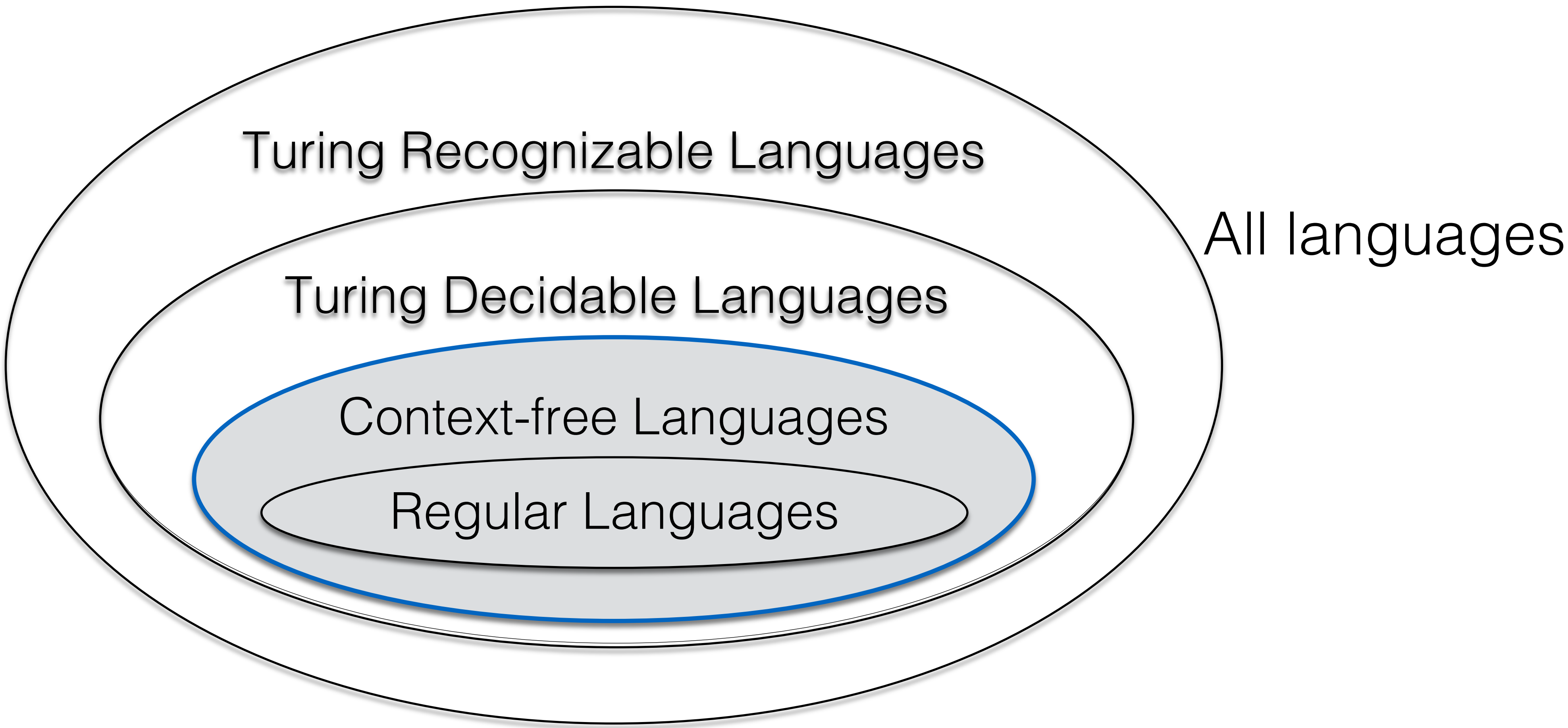
Properties of Context-Free Languages

School of Engineering | Computer Science

Noam Chomsky

- Sometimes called "the father of modern linguistics”
- Major figure in analytic philosophy and one of the founders of the field of cognitive science
- Wrote *Syntactic Structures* that revolutionized the scientific study of language and played a major role in remodeling the study of language
- Political activist and defender of free speech

Context-Free Languages



Context-Free Languages

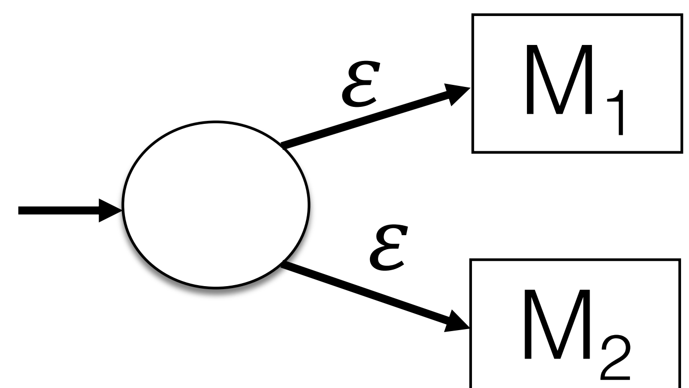
PDA = CFG = CNF

Closed under union, ∪, concatenation, ∘, and star, $*$.

VCU School of Engineering | Computer Science

Properties of Context-Free Languages

- Lemma: Context-free languages are closed under the union operation
- Proof: Let G_1 and G_2 be CFG's generating CFL's: $L(G_1)$ and $L(G_2)$ respectively. Let S_1 and S_2 be start variables for G_1 and G_2 respectively. Then the CFG for $L(G_1) \cup L(G_2)$ is:
  - $S \rightarrow S_1 \mid S_2$
    $S_1 \rightarrow \ldots$
    $S_2 \rightarrow \ldots$



M_1 and M_2 are the machines that represent G_1 and G_2 respectively

Create a new start state & variable

Properties of Context-Free Languages

- Lemma: Context-free languages are closed under the concatenation operation

- Proof: Let G_1 and G_2 be CFG's generating CFL's: $L(G_1)$ and $L(G_2)$ respectively. Let S_1 and S_2 be start variables for G_1 and G_2 respectively. Then the CFG for $L(G_1) \circ L(G_2)$ is:
  - $S \rightarrow S_1S_2$
    $S_1 \rightarrow \ldots$
    $S_2 \rightarrow \ldots$

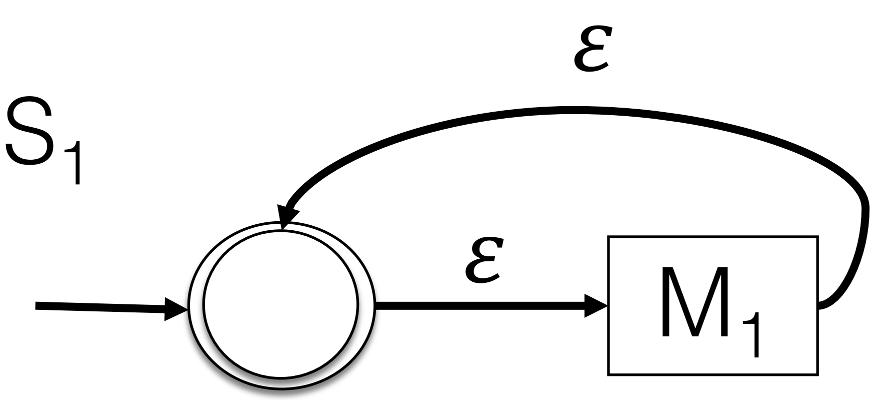
$\boxed{M_1} \xrightarrow{\varepsilon} \boxed{M_2}$

M₁'s accept states become regular states

M_1 and M_2 are the machines that represent G_1 and G_2 respectively

Properties of Context-Free Languages

- Lemma: Context-free languages are closed under the star operation
- Proof: Let G_1 be a CFG generating a CFL: $L(G_1)$. Let S_1 be the start variable for G_1. Then the CFG for $L(G_1)^*$ is:
  - $S \rightarrow \varepsilon \mid SS_1$
    $S_1 \rightarrow \ldots$



M1 is the machine that represents G_1

Create a new start state that is an accept state

School of Engineering | Computer Science

Comparison of Regular & Context-Free Languages

- Take a non-regular language L, can we make it a context-free language?
  - Ex: $L = \{0^n 1^n \mid n \geq 0\}$ We learned in Chapter 1 that L is not a regular language
    - We can describe it with productions: $S \rightarrow 0S1 \mid \varepsilon$
    - Since we can define the grammar, the language is context-free
- <u>Claim</u>: All regular languages are context-free languages

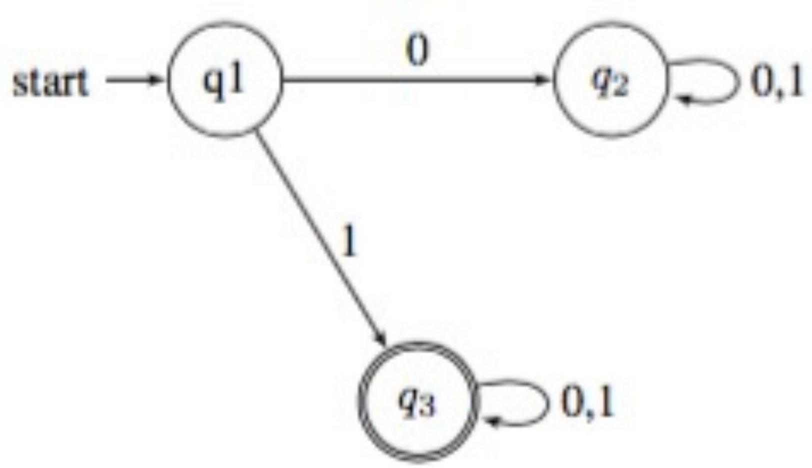
VCU School of Engineering | Computer Science

Comparison of Regular & Context-Free Languages

- Claim: All regular languages are context-free languages
- Proof: Let L be a regular language with a DFA M = (Q, Σ, δ, q_0, F)
  - We can create a grammar G by replacing the states in M with the variables in G
  - We construct the grammar as follows:
    1. For each state $q_i \in Q$, add variable R_i in G
    2. For each transition $\delta(q_i, a) = q_j$ for the DFA, add a substitution rule to G: $R_i \rightarrow aR_j$
    3. For any $q_i \in F$, add $R_i \rightarrow \varepsilon$
    4. Set R_0 in G to be the start variable q_0

Comparison of Regular & Context-Free Languages

- Claim: All regular languages are context-free languages
- Ex:

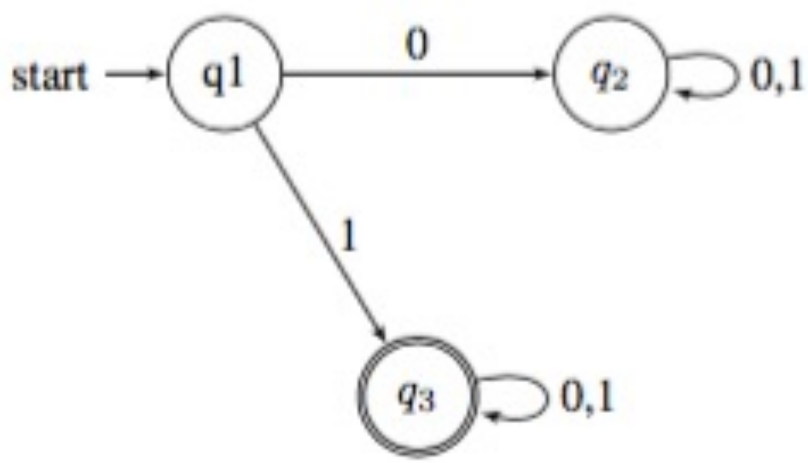


1. For each state $q_i \in Q$, add variable R_i in G
2. For each transition $\delta(q_i, a) = q_j$ for the DFA, add a substitution rule to G: $R_i \rightarrow aR_j$
3. For any $q_i \in F$, add $R_i \rightarrow \varepsilon$
4. Set R_0 in G to be the start variable q_0

- Create the CFG

Comparison of Regular & Context-Free Languages

- Claim: All regular languages are context-free languages
- Ex:



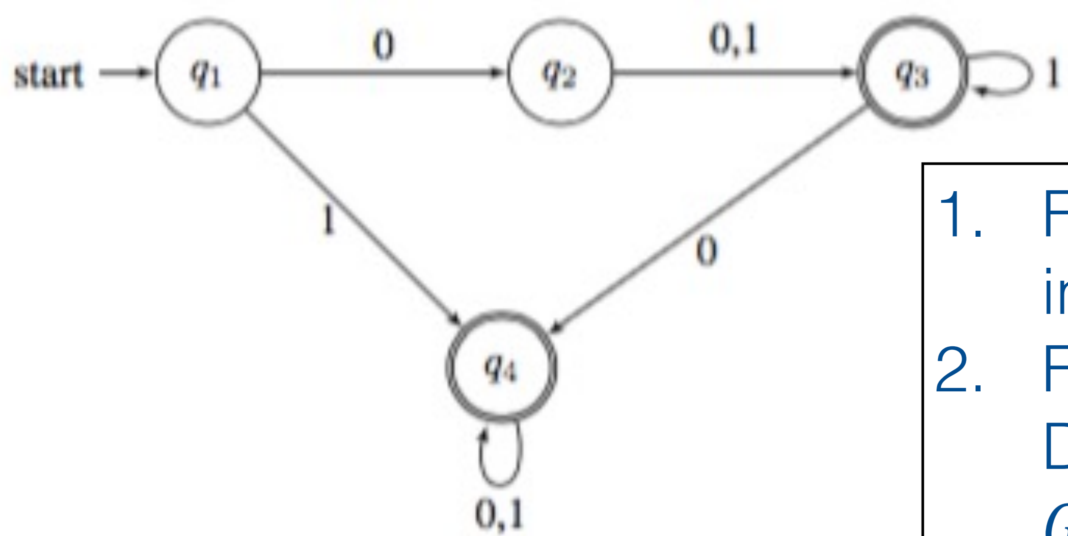
1. Variables: R_1, R_2, R_3
2. $R_1 \rightarrow 0R_2 \mid 1R_3$
   $R_2 \rightarrow 0R_2 \mid 1R_2$
   $R_3 \rightarrow 0R_3 \mid 1R_3 \mid \varepsilon$
4. R_1 is the start variable

The rules match the transitions, such as: on q_1 with a $\mathbf{0}$ move to q_2, so get $R_1 \rightarrow 0R_2$
Step 3 adds the ε rule to the variable that represents the accept state

School of Engineering | Computer Science

Comparison of Regular & Context-Free Languages

- Claim: All regular languages are context-free languages
- Ex:



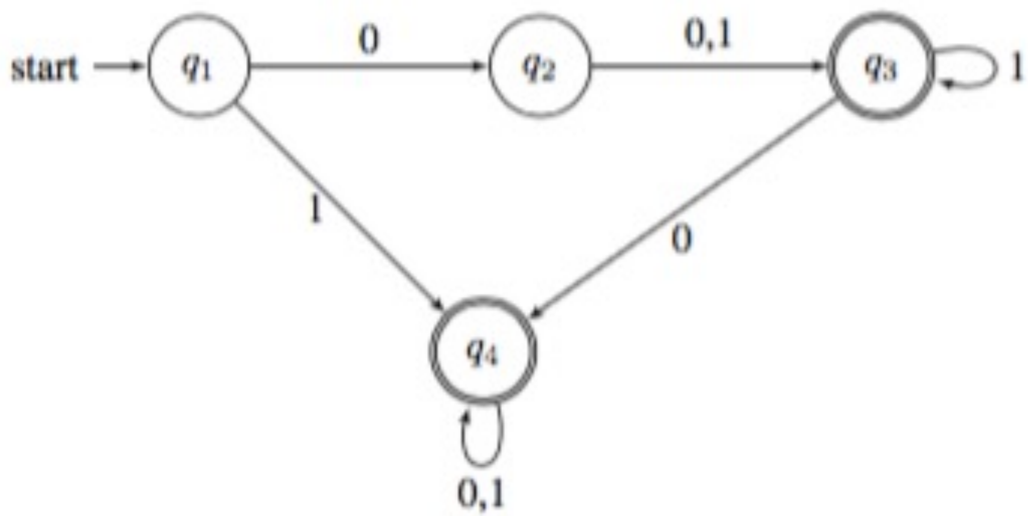
1. For each state $q_i \in Q$, add variable R_i in G
2. For each transition $\delta(q_i, a) = q_j$ for the DFA, add a substitution rule to G: $R_i \rightarrow aR_j$
3. For any $q_i \in F$, add $R_i \rightarrow \varepsilon$
4. Set R_0 in G to be the start variable q_0

- Create the CFG

School of Engineering | Computer Science

Comparison of Regular & Context-Free Languages

- Claim: All regular languages are context-free languages
- Ex:

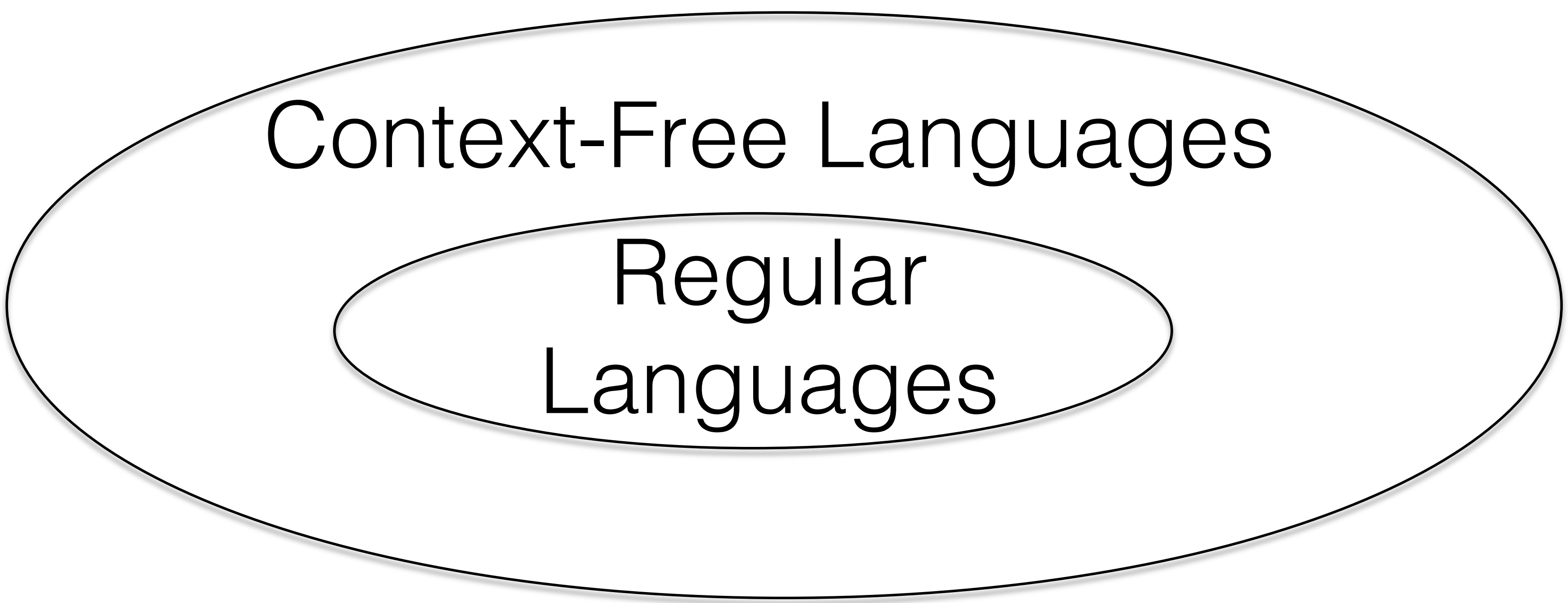


1. Variables: R_1, R_2, R_3, R_4
2. $R_1 \rightarrow 0R_2 \mid 1R_4$
   $R_2 \rightarrow 0R_3 \mid 1R_3$
   $R_3 \rightarrow 0R_4 \mid 1R_3 \mid \varepsilon$
   $R_4 \rightarrow 0R_4 \mid 1R_4 \mid \varepsilon$
4. R_1 is the start variable

School of Engineering | Computer Science

Comparison of Regular & Context-Free Languages

- All regular languages are context-free languages



VCU
School of Engineering | Computer Science

Chomsky Normal Form

- When working with CFG's, it is convenient to have them in simplified form
- A CFG is in Chomsky Normal form if every rule is of the form:
  - A → BC   or   A → a
  - Where a is a terminal and A, B, C are variables and B and C cannot be the start variable
  - The rule: S → ε is also allowed for the start variable S

VCU School of Engineering | Computer Science

Chomsky Normal Form

- <u>Theorem</u>: Any CFL is generated by a CFG in Chomsky Normal Form
  - Ex: $S \rightarrow ASA \mid aB$
    $A \rightarrow B \mid S$
    $B \rightarrow b \mid \varepsilon$
  - Need new start variable that is not also on the right-hand side, S_0

$S_0 \rightarrow S$
$S \rightarrow ASA \mid aB$
$A \rightarrow B \mid S$
$B \rightarrow b \mid \varepsilon$

All rules are in the form:
$A \rightarrow BC$
$A \rightarrow a$

VCU School of Engineering | Computer Science

Chomsky Normal Form

- Theorem: Any CFL is generated by a CFG in Chomsky Normal Form

Ex cont: $S_0 \rightarrow S$
$S \rightarrow ASA \mid aB$
$A \rightarrow B \mid S$
$B \rightarrow b \mid \varepsilon$

- Get rid of the rule $B \rightarrow \varepsilon$ (must add a rule for every B with the result of the rule $B \rightarrow \varepsilon$)

$S_0 \rightarrow S$
$S \rightarrow ASA \mid aB \mid \mathbf{a}$
$A \rightarrow B \mid S \mid \boldsymbol{\varepsilon}$
$B \rightarrow b$

All rules are in the form:
$A \rightarrow BC$
$A \rightarrow a$

VCU School of Engineering | Computer Science

Chomsky Normal Form

- Theorem: Any CFL is generated by a CFG in Chomsky Normal Form

Ex cont: $S_0 \rightarrow S$
$S \rightarrow ASA \mid aB \mid a$
$A \rightarrow B \mid S \mid \varepsilon$
$B \rightarrow b$

- Get rid of the rule $A \rightarrow \varepsilon$ (must add a rule for every A with the result of the rule $A \rightarrow \varepsilon$)

$S_0 \rightarrow S$
$S \rightarrow ASA \mid aB \mid a \mid \mathbf{SA} \mid \mathbf{AS} \mid \mathbf{S}$
$A \rightarrow B \mid S$
$B \rightarrow b$

All rules are in the form:
$A \rightarrow BC$
$A \rightarrow a$

Chomsky Normal Form

- Theorem: Any CFL is generated by a CFG in Chomsky Normal Form

Ex cont: $S_0 \rightarrow S$
$S \rightarrow ASA \mid aB \mid a \mid SA \mid AS \mid S$
$A \rightarrow B \mid S$
$B \rightarrow b$

- Get rid of the rules $S \rightarrow S$ and $S_0 \rightarrow S$

$S_0 \rightarrow \mathbf{ASA \mid aB \mid a \mid SA \mid AS}$
$S \rightarrow ASA \mid aB \mid a \mid SA \mid AS$
$A \rightarrow B \mid S$
$B \rightarrow b$

All rules are in the form:
$A \rightarrow BC$
$A \rightarrow a$

Chomsky Normal Form

- Theorem: Any CFL is generated by a CFG in Chomsky Normal Form

  Ex cont: $S_0 \rightarrow ASA \mid aB \mid a \mid SA \mid AS$
  $S \rightarrow ASA \mid aB \mid a \mid SA \mid AS$
  $A \rightarrow B \mid S$
  $B \rightarrow b$

  - Get rid of the rules $A \rightarrow B$ and $A \rightarrow S$

$S_0 \rightarrow ASA \mid aB \mid a \mid SA \mid AS$
$S \rightarrow ASA \mid aB \mid a \mid SA \mid AS$
$A \rightarrow \mathbf{b \mid ASA \mid aB \mid a \mid SA \mid AS}$
$B \rightarrow b$

All rules are in the form:
$A \rightarrow BC$
$A \rightarrow a$

Chomsky Normal Form

- Theorem: Any CFL is generated by a CFG in Chomsky Normal Form

  Ex cont: $S_0 \rightarrow ASA \mid aB \mid a \mid SA \mid AS$
  $S \rightarrow ASA \mid aB \mid a \mid SA \mid AS$
  $A \rightarrow b \mid ASA \mid aB \mid a \mid SA \mid AS$
  $B \rightarrow b$

  - Convert the rules to the form $A \rightarrow BC$ and $A \rightarrow a$ (must add new variables, T and U)

    - $S_0 \rightarrow \mathbf{AT} \mid \mathbf{UB} \mid a \mid SA \mid AS$
      $S \rightarrow \mathbf{AT} \mid \mathbf{UB} \mid a \mid SA \mid AS$
      $A \rightarrow b \mid \mathbf{AT} \mid \mathbf{UB} \mid a \mid SA \mid AS$
      $\mathbf{T \rightarrow SA}$
      $\mathbf{U \rightarrow a}$
      $B \rightarrow b$

All rules are in the form:
$A \rightarrow BC$
$A \rightarrow a$

Chomsky Normal Form

- Second Example to convert to Chomsky Normal Form:
  - R → XRX | S
    S → aTb | bTa
    T → XTX | X | ε
    X → a | b

  Make all rules in the form:
  A → BC
  A → a

  - What steps do we need to take?
    - New start variable that does not go to itself
    - Remove the empty string transitions (the new start state can go to the empty string)
    - Remove transitions of the form A → B
    - Add variables and rules to convert the rules to the correct form of A → BC or A → a

VCU School of Engineering | Computer Science

Chomsky Normal Form

- Second Example to convert to Chomsky Normal Form:
  - $R \rightarrow XRX \mid S$
    $S \rightarrow aTb \mid bTa$
    $T \rightarrow XTX \mid X \mid \varepsilon$
    $X \rightarrow a \mid b$

Make all rules in the form:
$A \rightarrow BC$
$A \rightarrow a$

- Need new start variable that is not also on the right-hand side, S_0
  - $S_0 \rightarrow R$
    $R \rightarrow XRX \mid S$
    $S \rightarrow aTb \mid bTa$
    $T \rightarrow XTX \mid X \mid \varepsilon$
    $X \rightarrow a \mid b$

Chomsky Normal Form

- Second Example to convert to Chomsky Normal Form:
  - $S_0 \rightarrow R$
    $R \rightarrow XRX \mid S$
    $S \rightarrow aTb \mid bTa$
    $T \rightarrow XTX \mid X \mid \varepsilon$
    $X \rightarrow a \mid b$

Make all rules in the form:
$A \rightarrow BC$
$A \rightarrow a$

- Remove the rule $T \rightarrow \varepsilon$ (must add a rule for every T with the result of the rule $T \rightarrow \varepsilon$)
  - $S_0 \rightarrow R$
    $R \rightarrow XRX \mid S$
    $S \rightarrow aTb \mid bTa \mid \mathbf{ab} \mid \mathbf{ba}$
    $T \rightarrow XTX \mid X \mid \mathbf{XX}$
    $X \rightarrow a \mid b$

VCU School of Engineering | Computer Science

Chomsky Normal Form

- Second Example to convert to Chomsky Normal Form:
  - $S_0 \rightarrow R$
    $R \rightarrow XRX \mid S$
    $S \rightarrow aTb \mid bTa \mid ab \mid ba$
    $T \rightarrow XTX \mid X \mid XX$
    $X \rightarrow a \mid b$

Make all rules in the form:
$A \rightarrow BC$
$A \rightarrow a$

- Remove the rules $S_0 \rightarrow R$ and $R \rightarrow S$
  - $S_0 \rightarrow$ **XRX | S**
    $R \rightarrow XRX \mid$ **aTb | bTa | ab | ba**
    $S \rightarrow aTb \mid bTa \mid ab \mid ba$
    $T \rightarrow XTX \mid X \mid XX$
    $X \rightarrow a \mid b$

Chomsky Normal Form

- Second Example to convert to Chomsky Normal Form:
  - $S_0 \rightarrow XRX \mid S$
    $R \rightarrow XRX \mid aTb \mid bTa \mid ab \mid ba$
    $S \rightarrow aTb \mid bTa \mid ab \mid ba$
    $T \rightarrow XTX \mid X \mid XX$
    $X \rightarrow a \mid b$

Make all rules in the form:
$A \rightarrow BC$
$A \rightarrow a$

- Remove the rules $S_0 \rightarrow S$ and $T \rightarrow X$
  - $S_0 \rightarrow XRX \mid \mathbf{aTb \mid bTa \mid ab \mid ba}$
    $R \rightarrow XRX \mid aTb \mid bTa \mid ab \mid ba$
    $S \rightarrow aTb \mid bTa \mid ab \mid ba$
    $T \rightarrow XTX \mid \mathbf{a \mid b} \mid XX$
    $X \rightarrow a \mid b$

VCU School of Engineering | Computer Science

Chomsky Normal Form

- Second Example to convert to Chomsky Normal Form:
  - $S_0 \rightarrow XRX \mid aTb \mid bTa \mid ab \mid ba$
    $R \rightarrow XRX \mid aTb \mid bTa \mid ab \mid ba$
    $S \rightarrow aTb \mid bTa \mid ab \mid ba$
    $T \rightarrow XTX \mid a \mid b \mid XX$
    $X \rightarrow a \mid b$

> Make all rules in the form:
> $A \rightarrow BC$
> $A \rightarrow a$

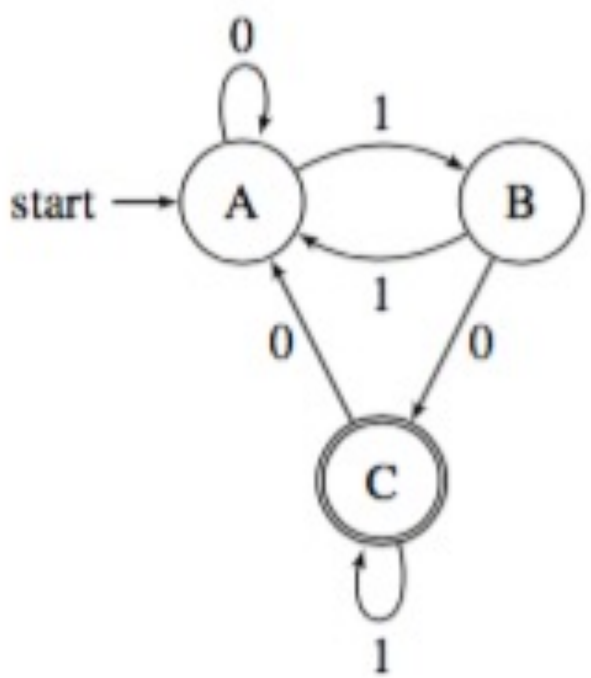
- Convert the rules to the form $A \rightarrow BC$ and $A \rightarrow a$ (must add new variables, Y, Z, W, V, B, and A)
  - $S_0 \rightarrow YX \mid WB \mid VA \mid AB \mid BA$
    $R \rightarrow YX \mid WB \mid VA \mid AB \mid BA$
    $S \rightarrow WB \mid VA \mid AB \mid BA$
    $T \rightarrow ZX \mid a \mid b \mid XX$
    $X \rightarrow a \mid b$
    $Y \rightarrow XR$
    $Z \rightarrow XT$

    $W \rightarrow AT$
    $V \rightarrow BT$
    $B \rightarrow b$
    $A \rightarrow a$

VCU School of Engineering | Computer Science

Try It

1. Generate the CFG from the DFA given below.



2. Convert the CFG in to Chomsky Normal Form:

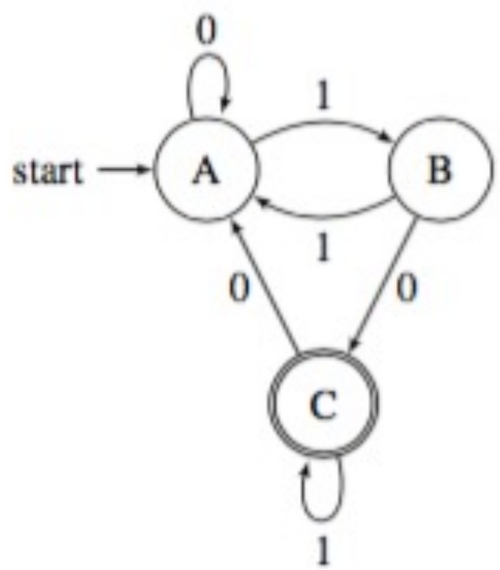
   $S \rightarrow TSV \mid VST \mid \varepsilon$

   $T \rightarrow 0$

   $V \rightarrow 1$

Try It

1. Generate the CFG from the DFA given below.



- Variables A, B, C, A is the start symbol
- Productions:

  A → 0A | 1B

  B → 0C | 1A

  C → 0A | 1C | ε (ε added since C is an accept state)

Try It

2. Convert the CFG in to Chomsky Normal Form:

   $S \rightarrow TSV \mid VST \mid \varepsilon$

   $T \rightarrow 0$

   $V \rightarrow 1$

- New start state S_0:

   $\mathbf{S_0 \rightarrow S}$

   $S \rightarrow TSV \mid VST \mid \varepsilon$

   $T \rightarrow 0$

   $V \rightarrow 1$

Try It

2. Convert the CFG in to Chomsky Normal Form:

   $S_0 \rightarrow S$

   $S \rightarrow TSV \mid VST \mid \varepsilon$

   $T \rightarrow 0$

   $V \rightarrow 1$

- Get rid of ε transition:

   $S_0 \rightarrow S \mid \varepsilon$

   $S \rightarrow TSV \mid VST \mid \mathbf{TV} \mid \mathbf{VT}$

   $T \rightarrow 0$

   $V \rightarrow 1$

Note that the rule $S_0 \rightarrow \varepsilon$ is acceptable.

VCU School of Engineering | Computer Science

Try It

2. Convert the CFG in to Chomsky Normal Form:

   $S_0 \rightarrow S \mid \varepsilon$

   $S \rightarrow TSV \mid VST \mid TV \mid VT$

   $T \rightarrow 0$

   $V \rightarrow 1$

- Get rid of $S_0 \rightarrow S$ transition:

   $S_0 \rightarrow \mathbf{TSV} \mid \mathbf{VST} \mid \mathbf{TV} \mid \mathbf{VT} \mid \varepsilon$

   $S \rightarrow TSV \mid VST \mid TV \mid VT$

   $T \rightarrow 0$

   $V \rightarrow 1$

VCU School of Engineering | Computer Science

Try It

2. Convert the CFG in to Chomsky Normal Form:

   $S_0 \rightarrow TSV \mid VST \mid TV \mid VT \mid \varepsilon$

   $S \rightarrow TSV \mid VST \mid TV \mid VT$

   $T \rightarrow 0$

   $V \rightarrow 1$

- Make all transitions in form of A → BC or A → a:

   $S_0 \rightarrow XV \mid YT \mid TV \mid VT \mid \varepsilon$

   $S \rightarrow XV \mid YT \mid TV \mid VT$

   $T \rightarrow 0$

   $V \rightarrow 1$

   $X \rightarrow TS$

   $Y \rightarrow VS$

Theory of Computation Chapter 2

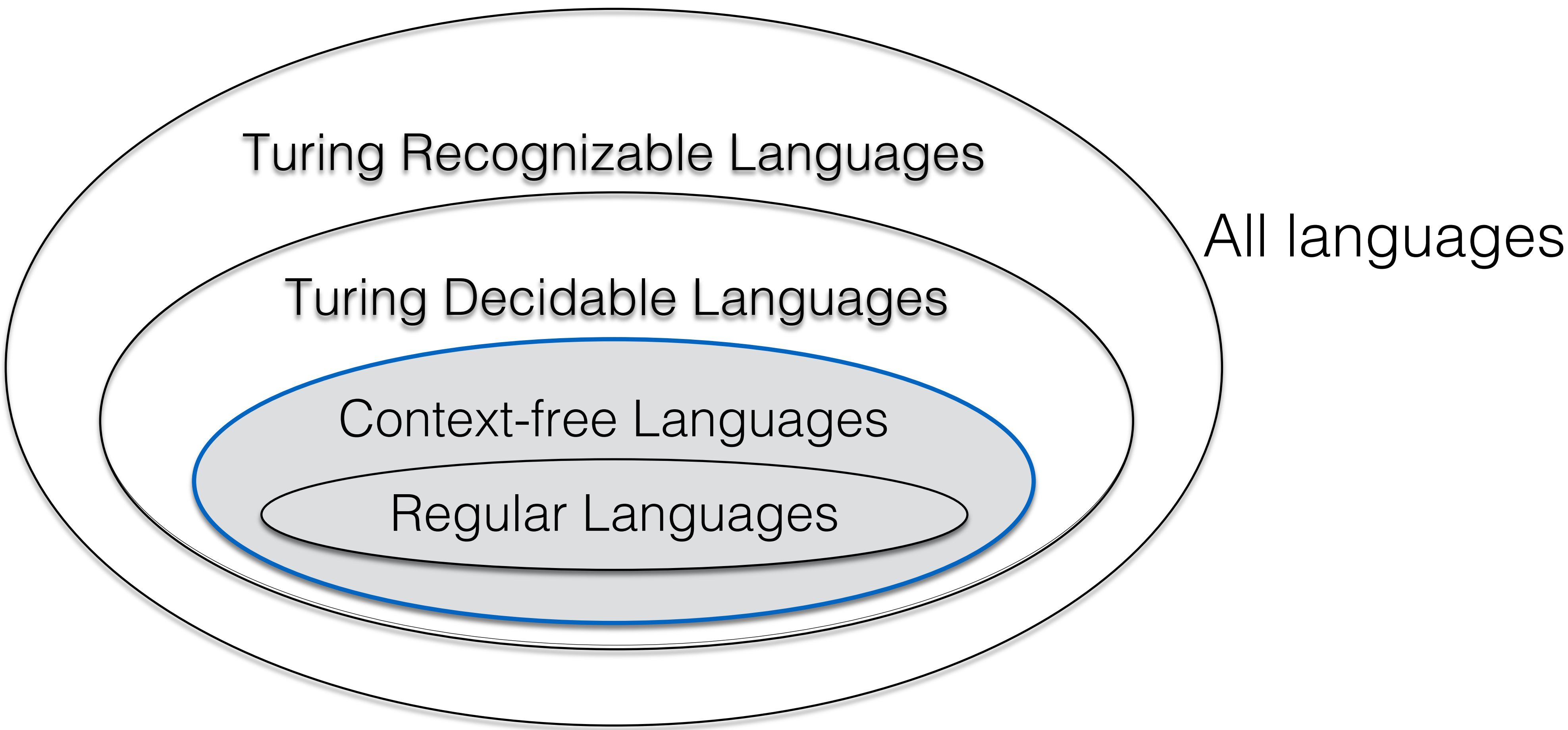
Pushdown Automata (PDA)

School of Engineering | Computer Science

J. Presper Eckert 1919-1995

- Co-designer of ENIAC, the first general purpose, electronic, digital computer
- Also, co-designer of UNIVAC I, the first commercial computer
- Electrical engineer

Context-Free Languages



Context-Free Languages

PDA = CFG = CNF

Closed under union, $\cup$, concatenation, $\circ$, and star, $*$.

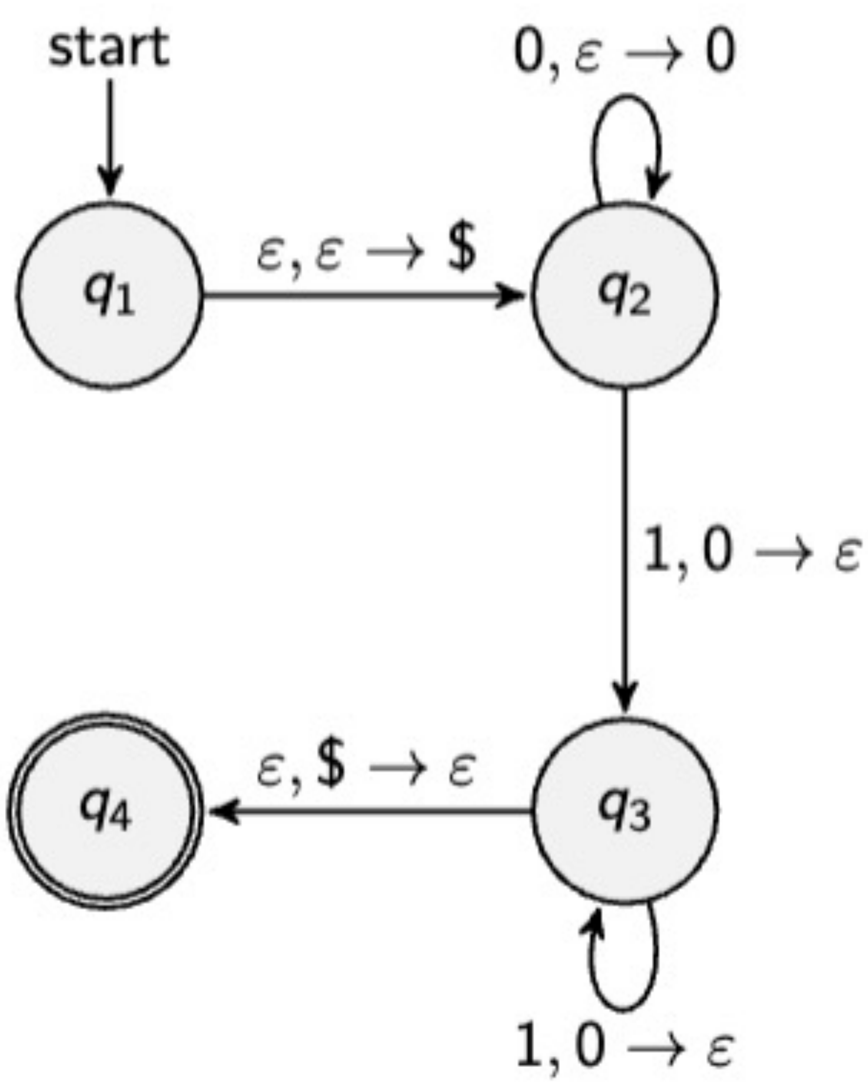
School of Engineering | Computer Science

Pushdown Automata

- The DFA's and NFA's that represent regular languages lack memory
- In contrast, Context-Free Languages are represented by Pushdown Automata (PDA), which contain a stack.
- They are essentially NFA's with a stack (LIFO data structure)

Pushdown Automata

- Ex: The PDA for L = $\{0^n 1^n \mid n \geq 0\}$ is:



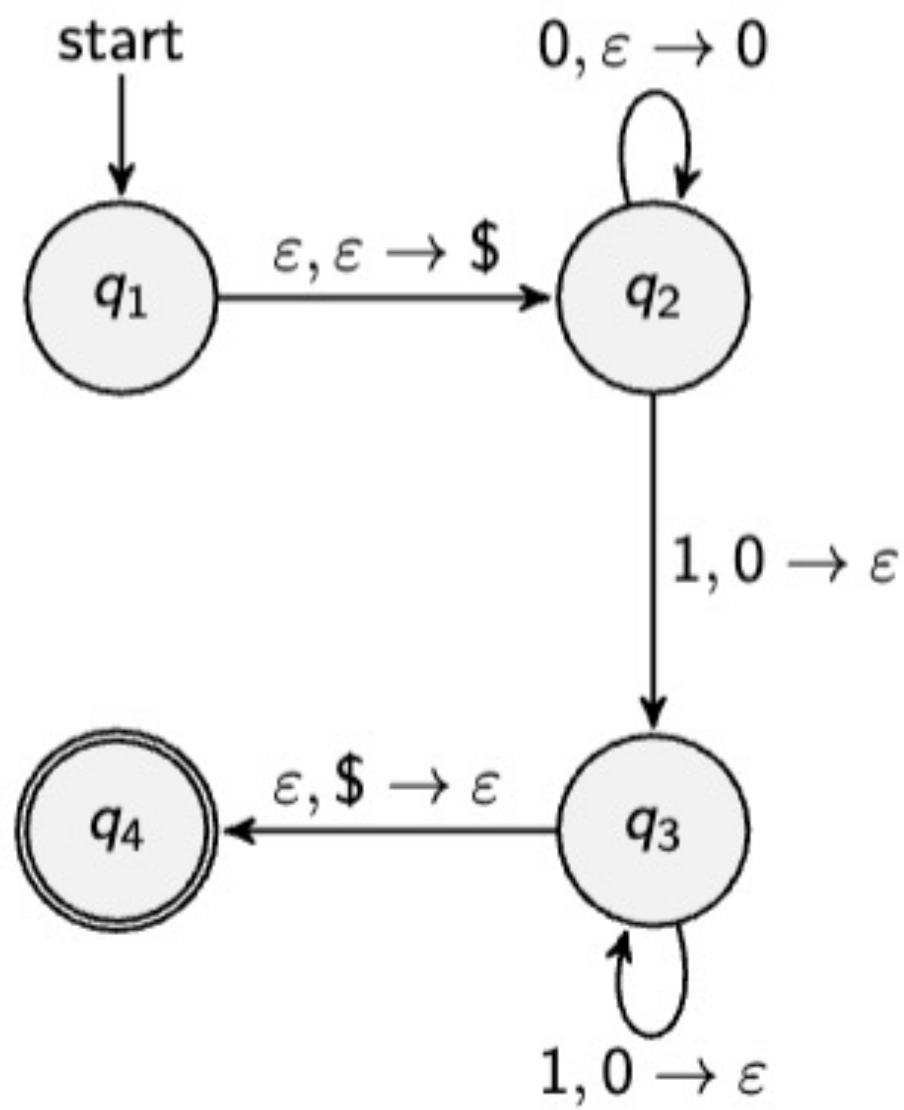
- The transition arrow can be read as: input symbol, pop from the stack → push onto the stack.
- So ε, ε → \$ means that the input symbol is the empty string, you are popping the empty string from the stack and pushing the \$, the start symbol onto the stack.

School of Engineering | Computer Science

Pushdown Automata

- Ex: The PDA for $L = \{0^n 1^n \mid n \geq 0\}$ is:
  - Idea:

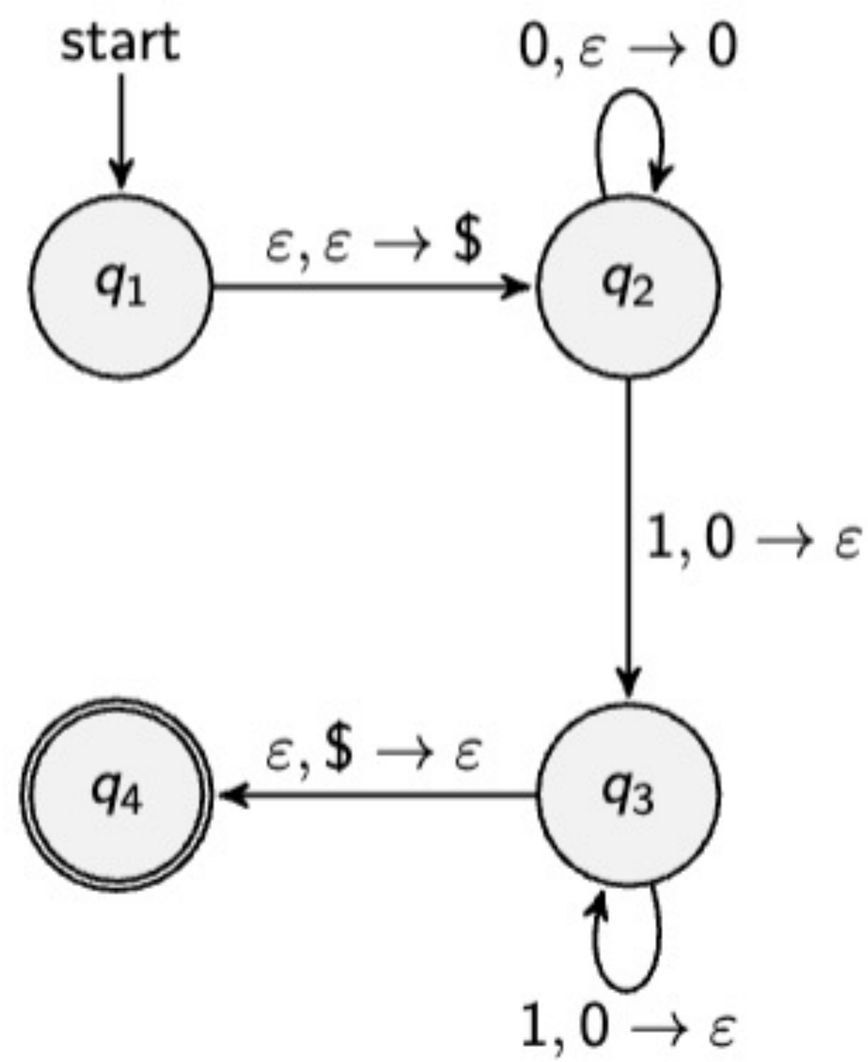
1. Push \$, start symbol, on stack
2. If read a 1 first reject
3. Each time you read a 0, push a 0 on the stack
4. Each time you read a 1, pop a 0 off the stack
5. Accept if all symbols have been read and the \$ is the last symbol popped from the stack



School of Engineering | Computer Science

Pushdown Automata

- Ex: The PDA for $L = \{0^n1^n \mid n \geq 0\}$ is:
  - Will this PDA accept 00011?
    - Push \$ on stack
    - Read 0, push 0 on stack
    - Read 0, push 0 on stack
    - Read 0, push 0 on stack
    - Read 1, pop 0 off stack
    - Read 1, pop 0 off stack
    - Read empty string, cannot pop \$ off stack since still have a 0 on the stack, reject.

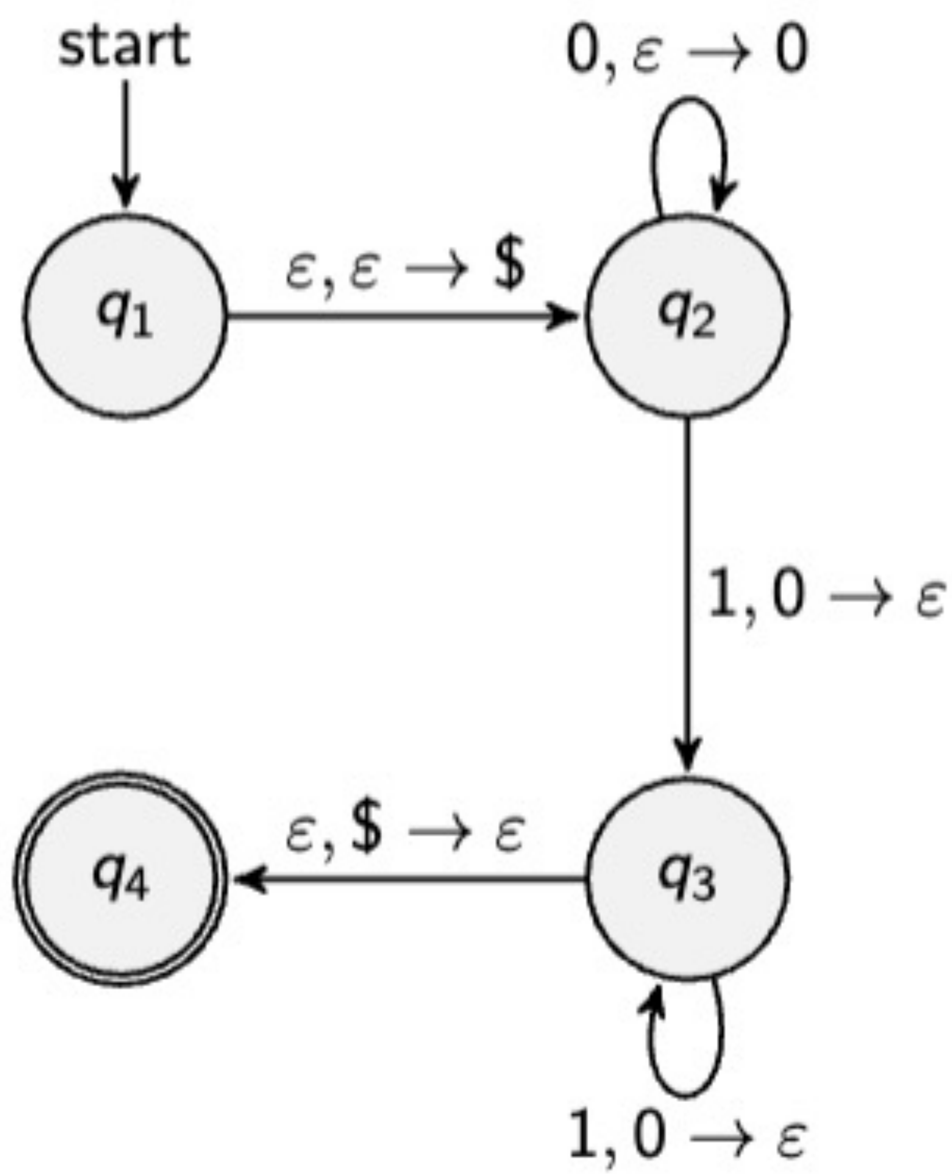


PDA Formal Definition

- <u>Formal Definition of PDA</u>: A PDA is a 6-tuple (Q, Σ, Γ, δ, q_0, F) such that:
  1. Q is a finite set of states
  2. Σ is the input alphabet
  3. Γ is the stack alphabet
  4. $\delta: Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow P(Q \times \Gamma_\varepsilon)$
  5. $q_0 \in Q$ is the start state
  6. $F \subseteq Q$ is the set of accept states

PDA Formal Definition

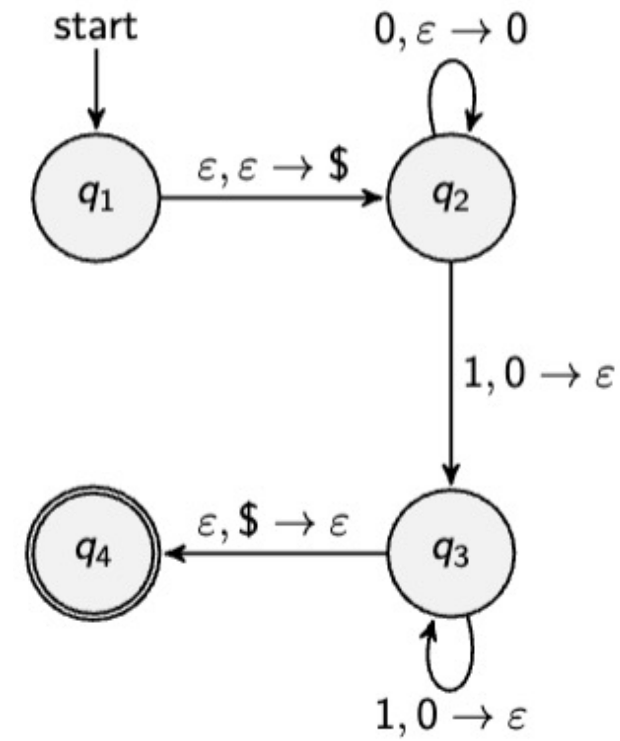
- Formal Definition of PDA Ex:
- Ex: PDA $L = \{0^n 1^n \mid n \geq 0\}$:
  - $Q =$
  - $\Sigma =$
  - $\Gamma =$
  - $q_0 =$
  - $F =$
  - $\delta =$



School of Engineering | Computer Science

PDA Formal Definition

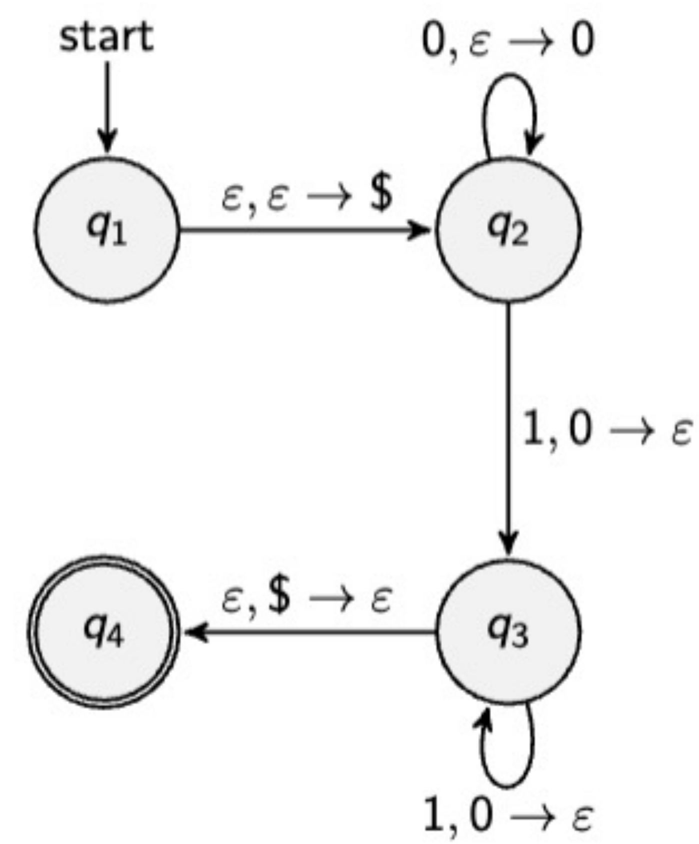
- <u>Formal Definition of PDA Ex:</u>
- Ex: PDA L = $\{0^n1^n \mid n \geq 0\}$:
  - $\delta =$



Input:	0			1			ε		
Pop off stack:	0	\$	ε	0	\$	ε	0	\$	ε
q_1									
q_2									
q_3									
q_4									

PDA Formal Definition

- <u>Formal Definition of PDA Ex</u>:
- Ex: PDA $L = \{0^n 1^n \mid n \geq 0\}$:
  - $Q = \{q_1, q_2, q_3, q_4\}$, $\Sigma = \{0, 1\}$,
  - $\Gamma = \{0, \$\}$, $q_0 = q_1$, $F = \{q_4\}$,
  - $\delta =$



Input:	0			1			ε		
Pop off stack:	0	\$	ε	0	\$	ε	0	\$	ε
q_1									$\{(q_2,\$)\}$
q_2			$\{(q_2, 0)\}$	$\{(q_3, \varepsilon)\}$					
q_3				$\{(q_3, \varepsilon)\}$				$\{(q_4, \varepsilon)\}$	
q_4									

$\{(q_3, \varepsilon)\}$ Means move to state, q_3, and push ε on the stack

Formal Definition of Acceptance

- A PDA M accepts a string w if w can be written as $w = w_1 w_2 \ldots wm$ for $m \geq 0$ where $w_i \in \Sigma_\varepsilon$, and there exists sequences of states $(r_0, r_1, \ldots, rm)$ and there exist strings $s_0, s_1, \ldots, sm \in \Gamma^*$ such that:
  1. $r_0 = q_0$ and $s_0 = \varepsilon$ (start on start state with empty stack)
  2. For $i = 0, \ldots, m-1$, have $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$, where $s_i = at$ and $s_{i+1} = bt$ for some $a, b \in \Gamma_\varepsilon$ and $t \in \Gamma^*$ (b is what is pushed on the stack, w_{i+1} is input, a is popped off the stack, s_i are the contents of the stack at state i)
  3. $r_m \in F$ (end in accept state)

VCU School of Engineering | Computer Science

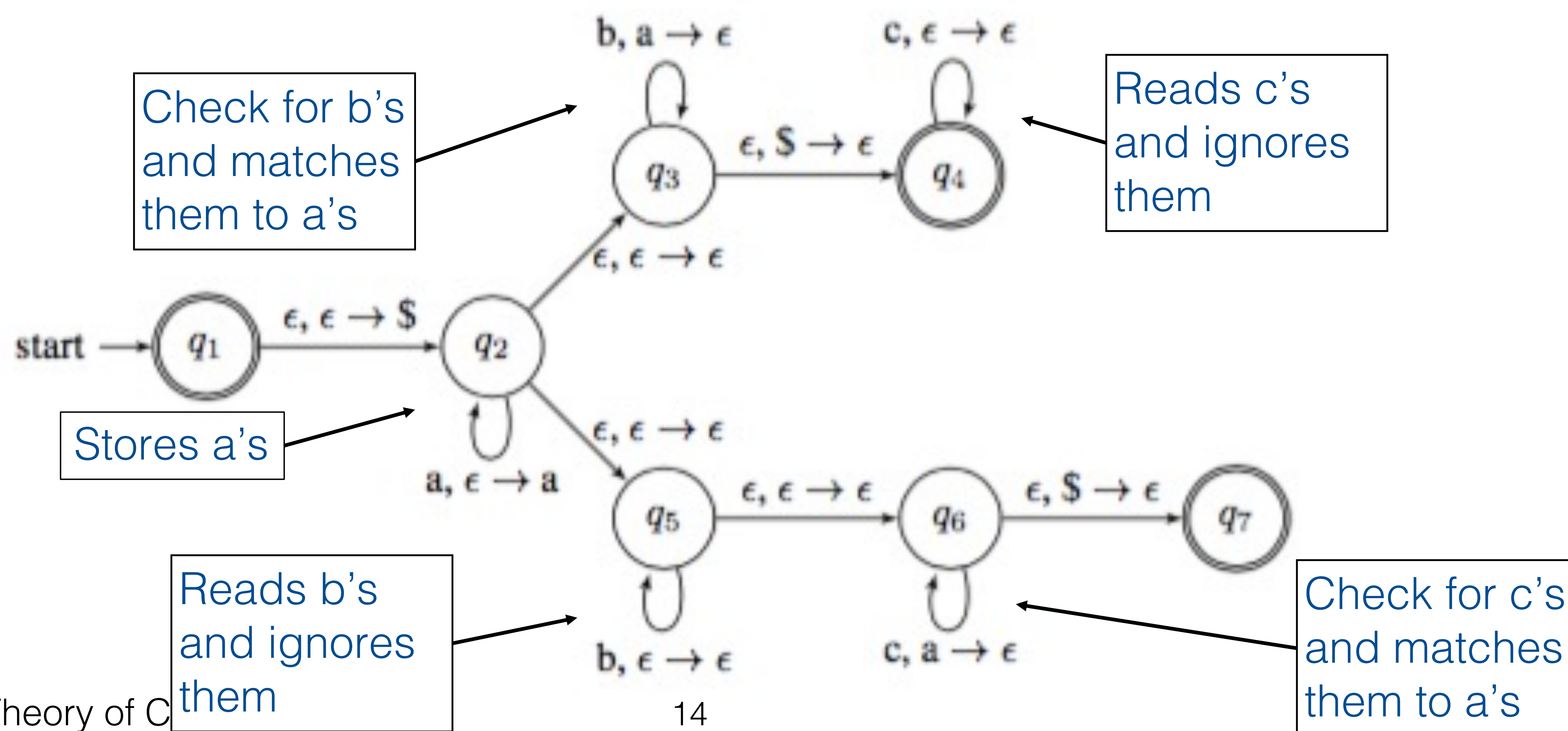
Creating a PDA for a Language

- Ex: What would the PDA for language $L = \{a^i b^j c^k \mid i, j, k \geq 0 \; and \; i = j \; or \; i = k\}$ be:
  - Ex: aabbccc, aabbbcc, aabb, …

VCU School of Engineering | Computer Science

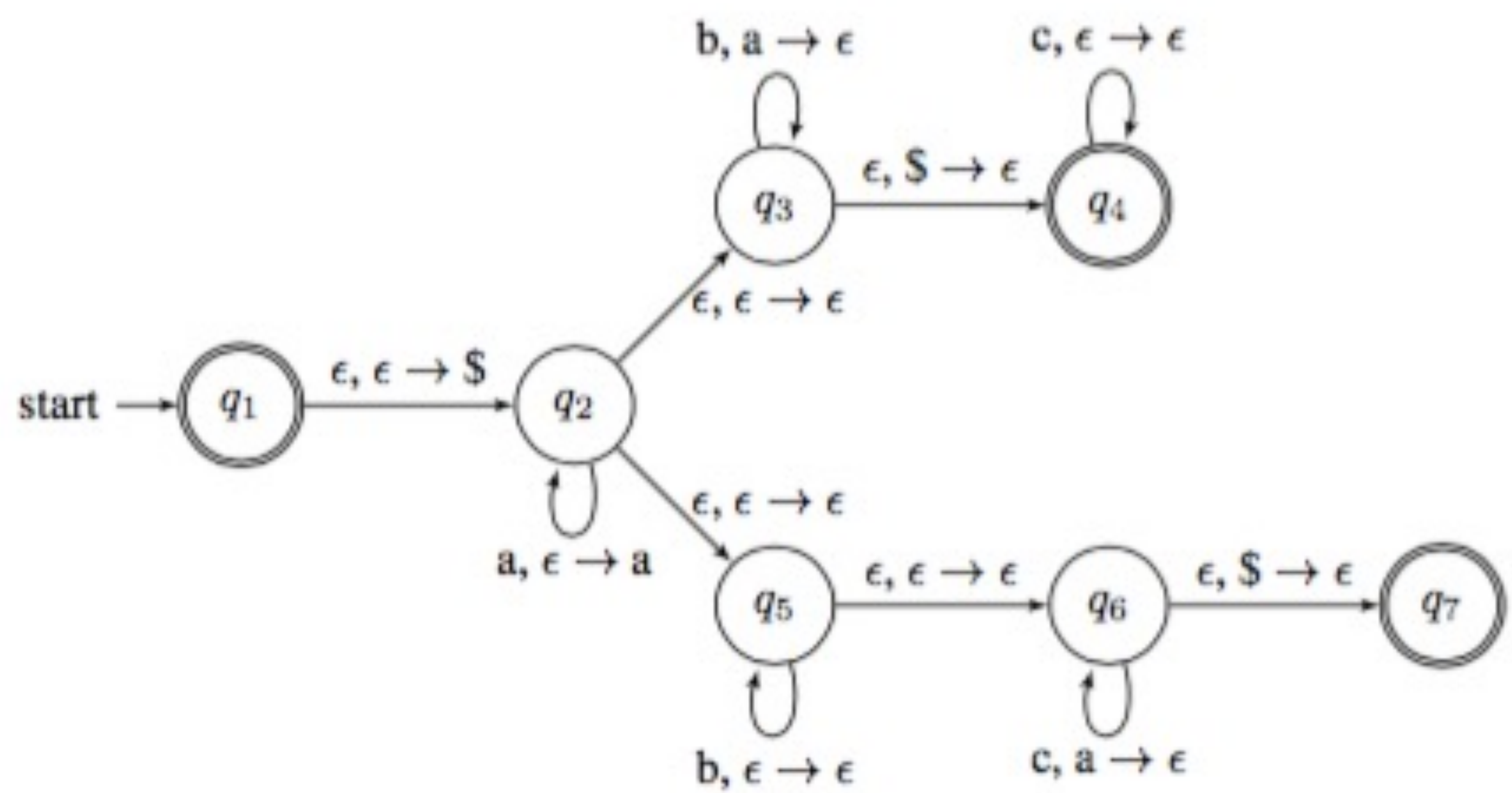
Creating a PDA for a Language

- Ex: What would the PDA for language $L = \{a^i b^j c^k \mid i, j, k \geq 0 \; and \; i = j \; or \; i = k\}$ be:
  - Ex: aabbccc, aabbbcc, aabb, …

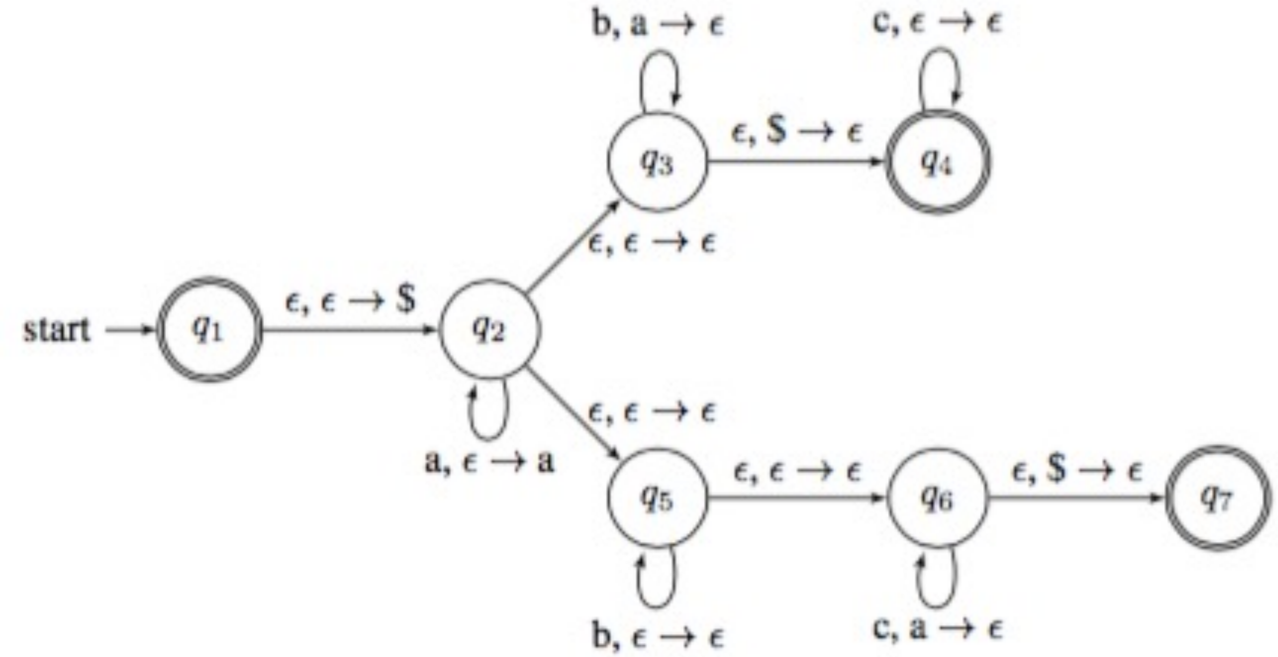


Formal Definition

- Ex: PDA $L = \{a^i b^j c^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } i = k\}$:
  - Q =
  - Σ =
  - Γ =
  - q_0 =
  - F =



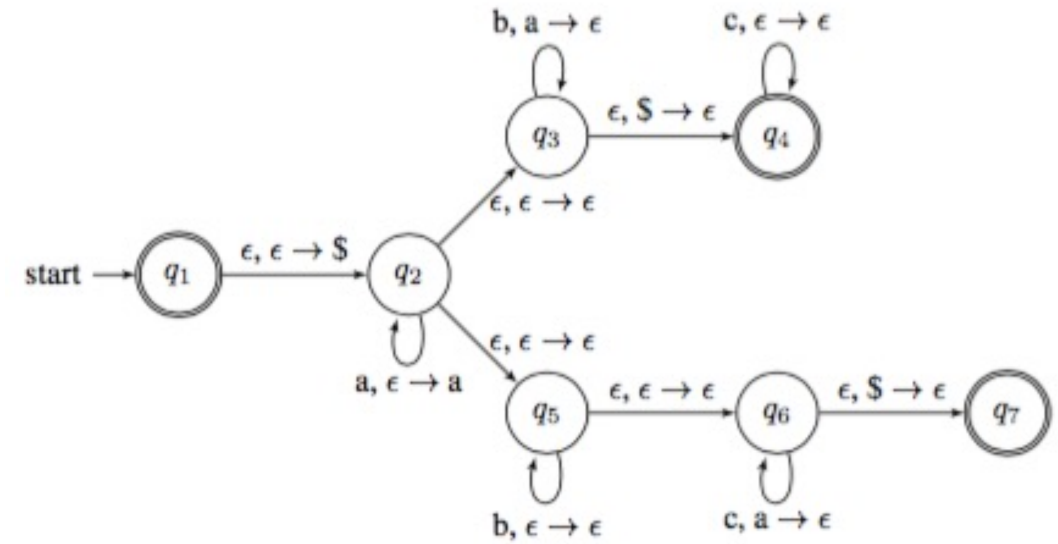
Formal Definition



- Ex: PDA
- $L = \{a^i b^j c^k \mid i, j, k \geq 0 \; and \; i = j \; or \; i = k\}$: $\delta =$

Input:	a			b			c			ε		
Pop off stack:	a	$	ε	a	$	ε	a	$	ε	0	$	ε
q_1												
q_2												
q_3												
q_4												
q_5												
q_6												
q_7												

Formal Definition



- Ex: PDA $L = \{a^i b^j c^k \mid i, j, k \geq 0 \; and \; i = j \; or \; i = k\}$:
  - $Q = \{q_1, q_2, q_3, q_4, q_5, q_6, q_7\}$, $\Sigma = \{a, b, c\}$, $\Gamma = \{a, \$\}$, $q_0 = q_1$, $F = \{q_1, q_4, q_7\}, \delta =$

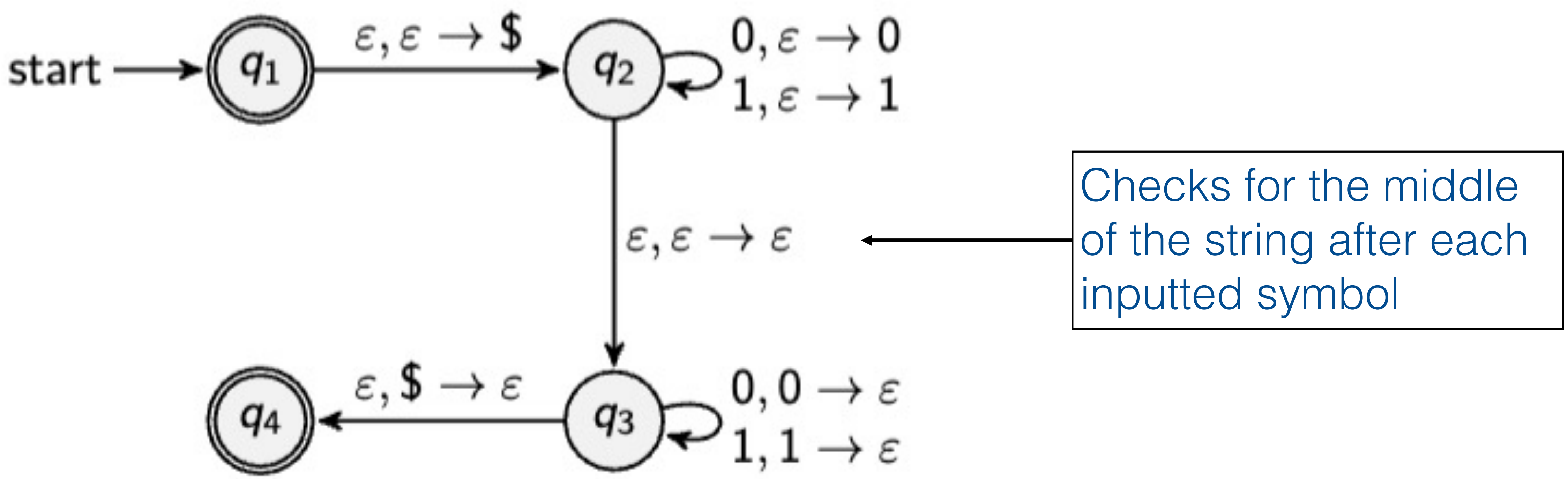
Input:	a			b			c			ε		
Pop off stack:	a	\$	ε	a	\$	ε	a	\$	ε	0	\$	ε
q_1												$\{(q_2, \$)\}$
q_2			$\{(q_2, a)\}$									$\{(q_3, \varepsilon)\}$ $\{(q_5, \varepsilon)\}$
q_3				$\{(q_3, \varepsilon)\}$							$\{(q_4, \varepsilon)\}$	
q_4									$\{(q_4, \varepsilon)\}$			
q_5						$\{(q_5, \varepsilon)\}$						$\{(q_6, \varepsilon)\}$
q_6							$\{(q_6, \varepsilon)\}$				$\{(q_7, \varepsilon)\}$	
q_7												

Creating a PDA for a Language

- Ex: What would the PDA for language $L = \{ww^R \mid w \in \{0, 1\}^*$, where w^R is the reverse of $w\}$ be:
  - Ex: w = car, w^R = rac

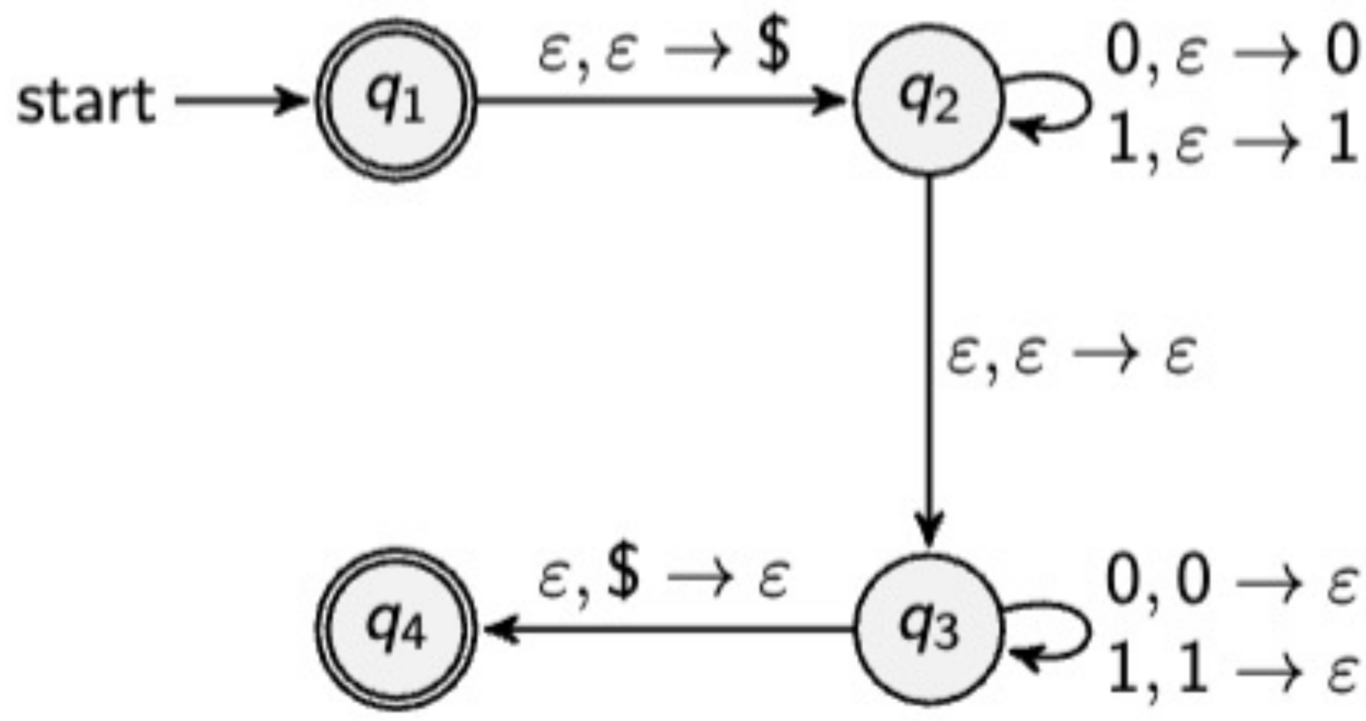
Creating a PDA for a Language

- Ex: What would the PDA for language $L = \{ww^R \mid w \in \{0, 1\}^*$, where w^R is the reverse of $w\}$ be:
  - Ex: w = car, w^R = rac

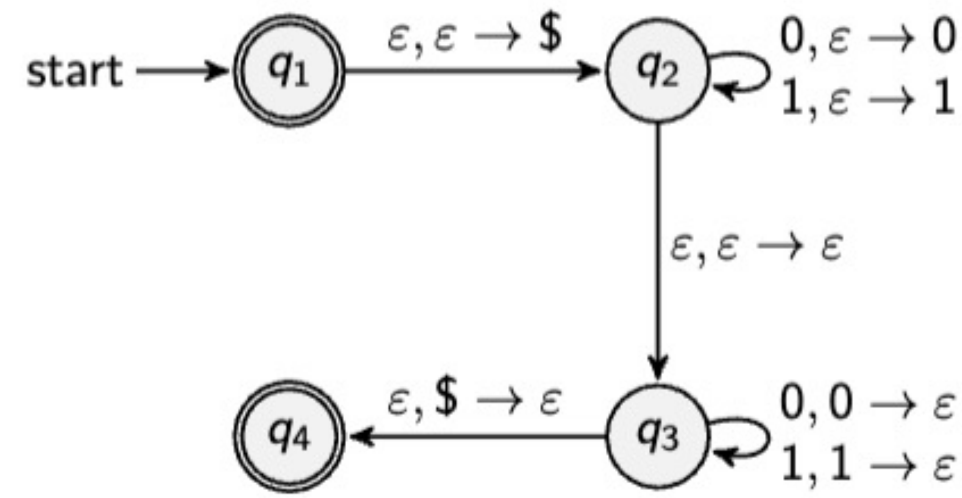


Formal Definition

- Ex: PDA L = L = {ww^R | w ∈ {0, 1}*, where w^R is the reverse of w}:
  - Q =
  - Σ =
  - Γ =
  - q_0 =
  - F =



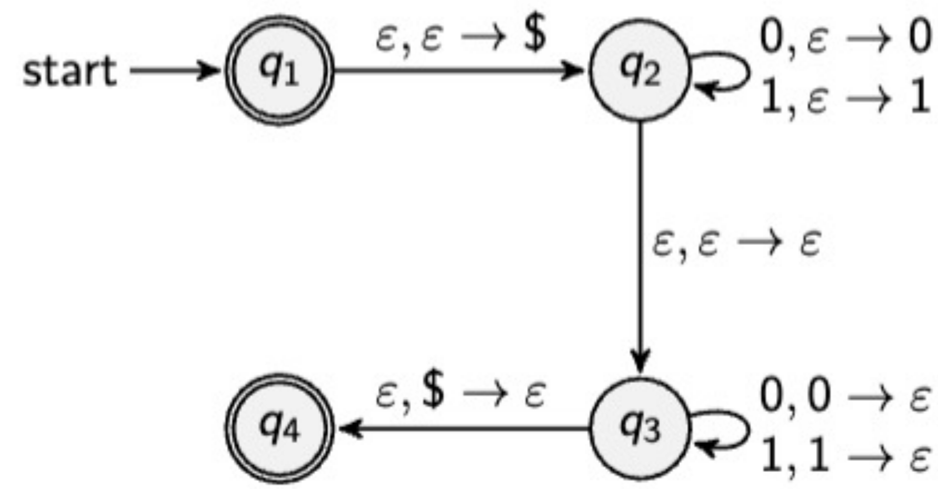
Formal Definition



- Ex: PDA L = L = $\{ww^R \mid w \in \{0, 1\}^*$, where w^R is the reverse of w$\}$:
  - $\delta =$

Input:	0				1				ε			
Pop off stack:	0	1	$	ε	0	1	$	ε	0	1	$	ε
q_1												
q_2												
q_3												
q_4												

VCU School of Engineering | Computer Science



Formal Definition

- Ex: PDA L = L = $\{ww^R \mid w \in \{0, 1\}^*$, where w^R is the reverse of $w\}$:
  - $Q = \{q_1, q_2, q_3, q_4\}$, $\Sigma = \{0, 1\}$, $\Gamma = \{0, 1, \$\}$, $q_0 = q_1$, $F = \{q_1, q_4\}$,

Input:	0				1				ε			
Pop off stack:	0	1	\$	ε	0	1	\$	ε	0	1	\$	ε
q_1												$\{(q_2, \$)\}$
q_2				$\{(q_2, 0)\}$				$\{(q_2, 1)\}$				$\{(q_3, \varepsilon)\}$
q_3	$\{(q_2, \varepsilon)\}$					$\{(q_3, \varepsilon)\}$					$\{(q_4, \varepsilon)\}$	
q_4												

VCU School of Engineering | Computer Science

Try It

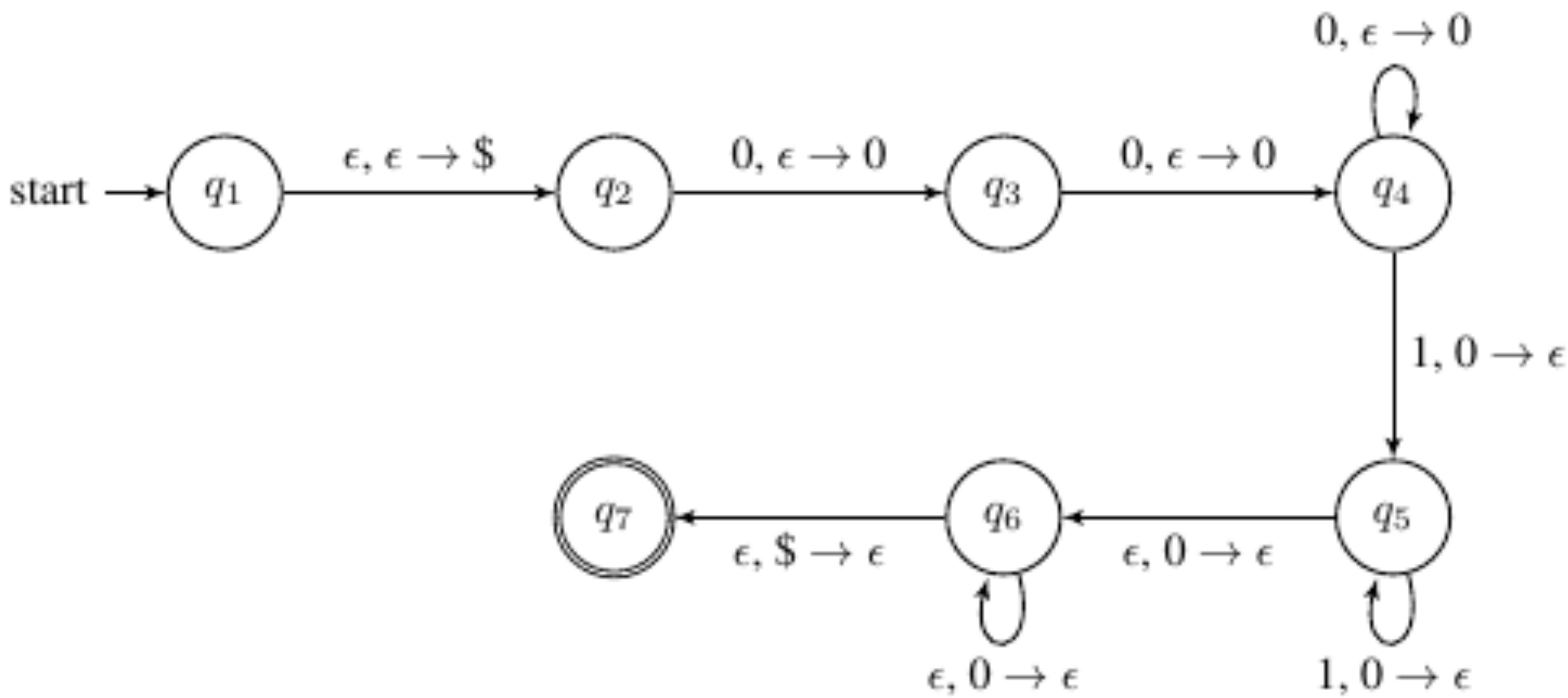
- Give state diagrams of PDAs that accepts $\{0^i 1^j \mid i \geq 2, j \geq 1, i > j\}$. (Modify the examples from this lecture.)
- Create a state diagram for a PDA recognizing the language as defined below:
  - $Q = \{q_0, q_1, q_2, q_3\}$
  - $\Sigma = \{a, b\}$
  - $\Gamma = \{a, \$\}$
  - $F = \{q_3\}$

$\delta =$

δ	a			b			ϵ		
pop	a	\$	ϵ	a	\$	ϵ	a	\$	ϵ
q_0									$(q_1, \$)$
q_1			(q_1, a)						(q_2, ϵ)
q_2				(q_2, ϵ)		(q_2, ϵ)		(q_3, ϵ)	
q_3									

Try It

- Give state diagrams of PDAs that accepts $\{0^i 1^j \mid i \geq 2, j \geq 1, i > j\}$. (Modify the examples from this lecture.)



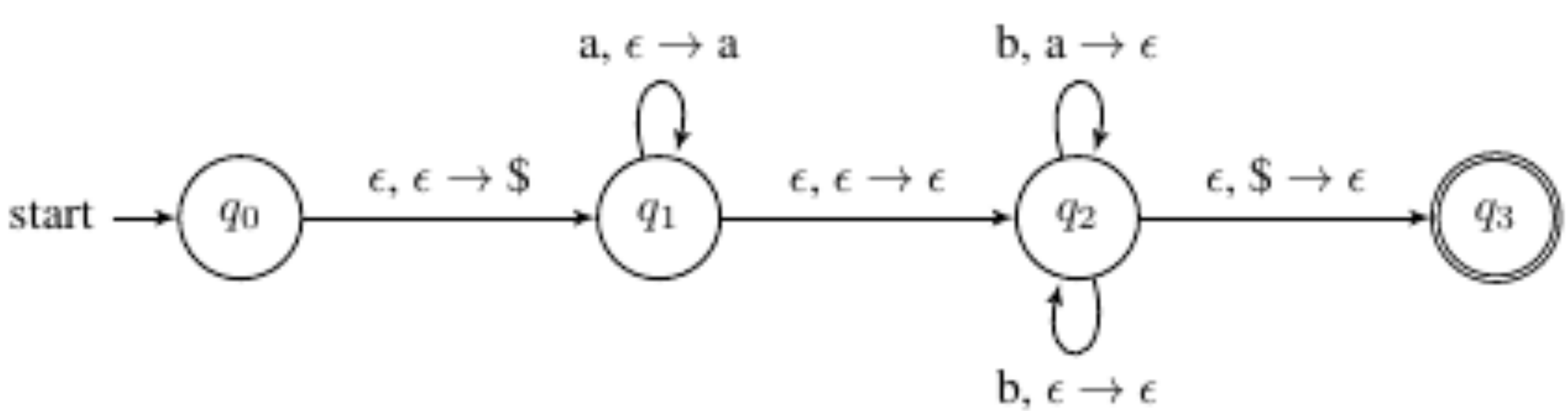
School of Engineering | Computer Science

Try It

- Create a state diagram for a PDA recognizing the language as defined below:
  - $Q = \{q_0, q_1, q_2, q_3\}$
  - $\Sigma = \{a, b\}$
  - $\Gamma = \{a, \$\}$
  - $F = \{q_3\}$

$\delta =$

δ	a			b			ϵ		
pop	a	\$	ϵ	a	\$	ϵ	a	\$	ϵ
q_0									$(q_1, \$)$
q_1			(q_1, a)						(q_2, ϵ)
q_2				(q_2, ϵ)		(q_2, ϵ)		(q_3, ϵ)	
q_3									



Language:
$\{a^i b^j | i \geq 0,$
$j \geq 0, i \leq j\}$

Theory of Computation
Chapter 2

Equivalence of PDAs and CFGs

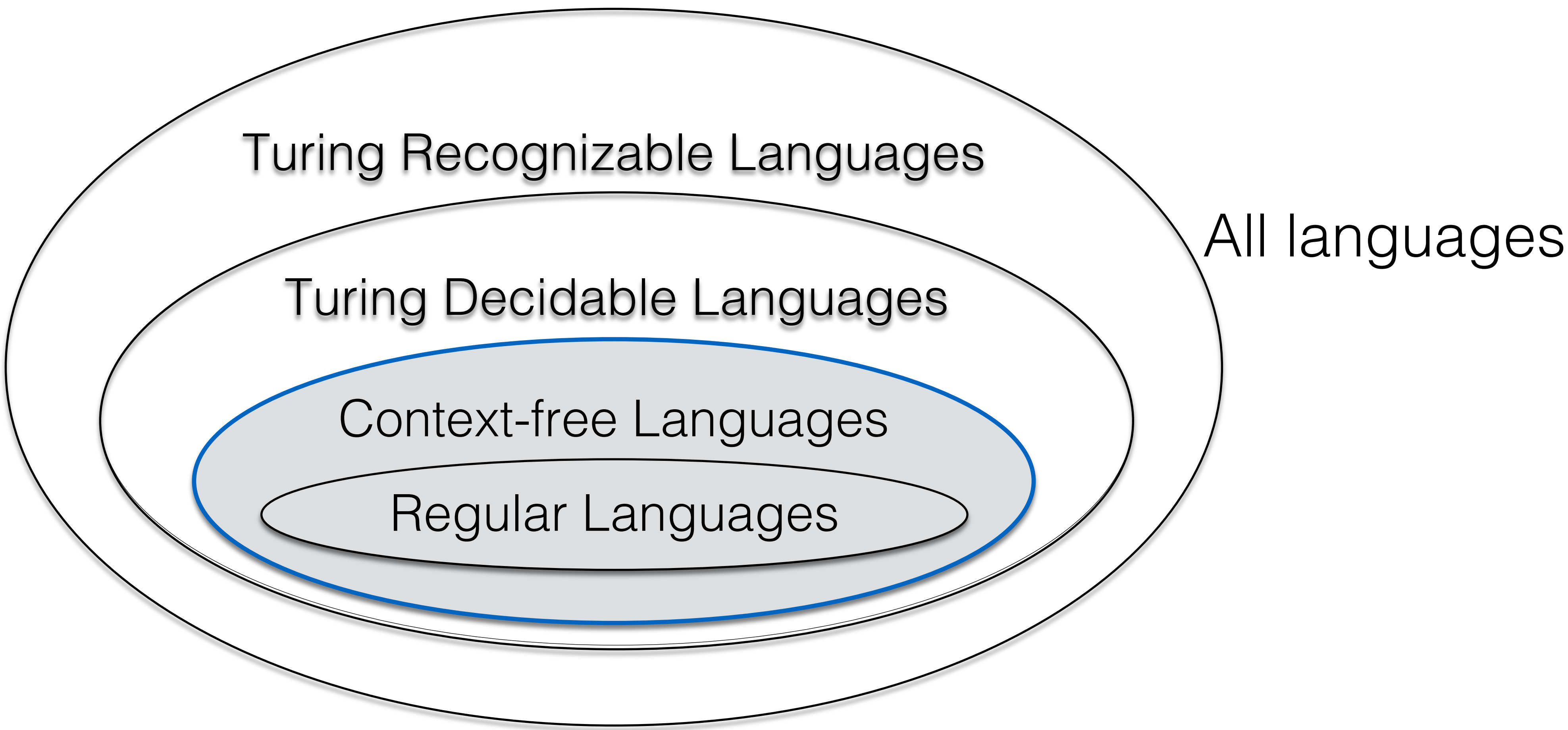
School of Engineering | Computer Science

Donald Knuth

- Author of multi-volume "The Art of Computer Programming" - *The* books of algorithms
  - Impact akin to the Bible's impact on Christianity, or Pride and Prejudice's impact on romance novels
- "Father" of the analysis of algorithms
- Turing Award, 1974
- Creator of TeX computer typesetting language
- Knuth used to pay a finder's fee of $2.56 for any typographical errors or mistakes discovered in his books, because "256 pennies is one hexadecimal dollar"

VCU School of Engineering | Computer Science

Context-Free Languages



Context-Free Languages

PDA = CFG = CNF

Closed under union, ∪, concatenation, ∘, and star, $*$.

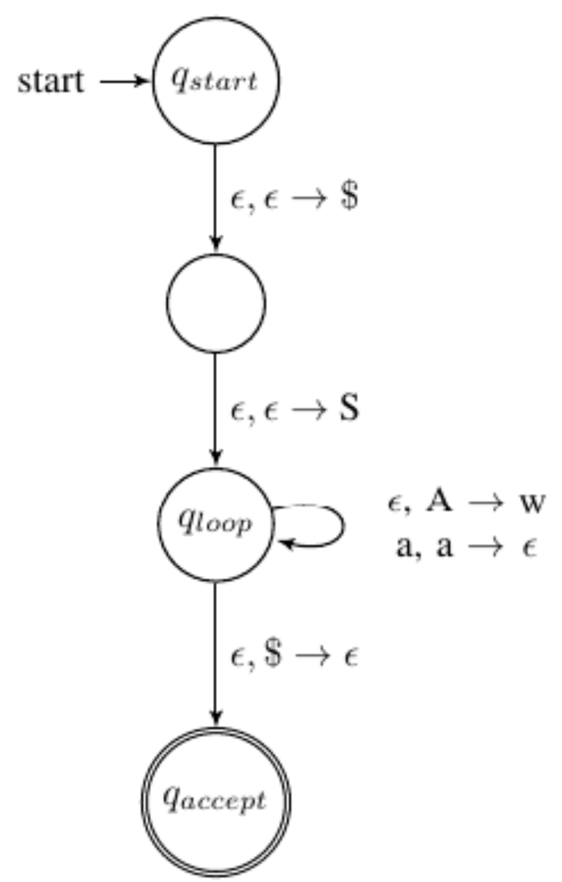
Equivalence of PDAs and CFGs

- Theorem 2.20: A language is context-free if and only if a PDA recognizes it (iff, so two directions)
- Forward: Lemma 2.21
  - Every context-free language has a pushdown automaton that recognizes it
  - Proof: Let L be a CFL with a CFG, G = (V, Σ, R, S). We construct a PDA M = (Q, Σ', Γ, δ, q_0, F) to recognize it.
  - In words:
    1. Design M to non-deterministically pick a derivation in G to follow to see if it derives the given input x
    2. Use a stack to store variables and terminals during the derivation. When a variable is popped, non-deterministically choose a substitution rule to push onto the stack.

VCU School of Engineering | Computer Science

Equivalence of PDAs and CFGs

- Theorem 2.20: A language is context-free if and only if a PDA recognizes it (iff, so two directions)
- Forward: Lemma 2.21- Construction of the PDA
  - Start with 3 states: q_{start}, q_{loop}, q_{accept}



$\varepsilon, \varepsilon \rightarrow \$$ is the starting transitions:
$\varepsilon, \varepsilon \rightarrow S$ pushes on the start variable
$\varepsilon, A \rightarrow w$ - is a transition inserted for the rules: $A \rightarrow w$ in G, where w is a string of terminals and variables
$a, a \rightarrow \varepsilon$ - is a transition inserted for each terminal a

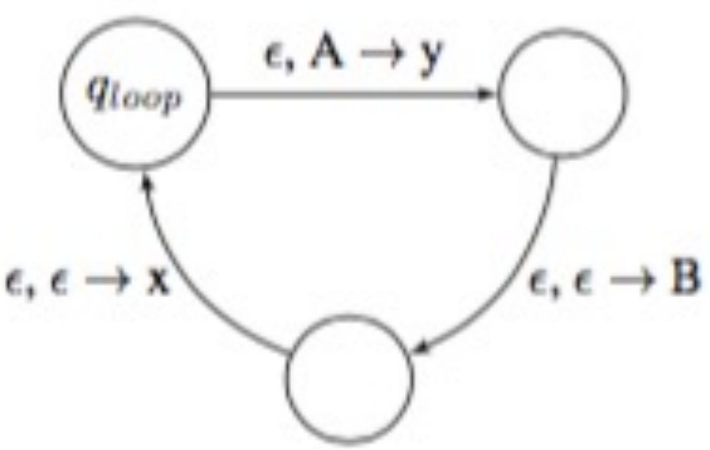
$\varepsilon, \$ \rightarrow \varepsilon$ is the last transition to the accept state

Equivalence of PDAs and CFGs

- Theorem 2.20: A language is context-free if and only if a PDA recognizes it (iff, so two directions)
- Forward: Lemma 2.21- Construction of the PDA
  - Two types of transitions are added to the loop state
    - ε, A → w - is a transition inserted for the rules: A → w in G, where w is a string of terminals and variables
    - a, a → ε - is a transition inserted for each terminal a
    - Ex: If there was a rule in the format of A → w: such as:

Stack	
	x
	B
A	y

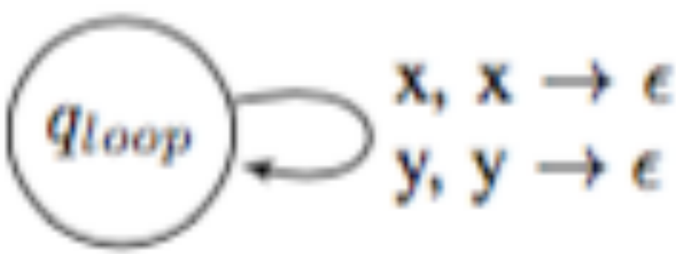
- A → xBy
- You would add the transition rules:
- ε, A → y ε, ε → B ε,ε → x in that order



CU School of Engineering | Computer Science

Equivalence of PDAs and CFGs

- Forward: Lemma 2.21- Construction of the PDA
  - Two types of transitions are added to the loop state
    - ε, A → w - is a transition inserted for the rules: A → w in G, where w is a string of terminals and variables
    - a, a → ε - is a transition inserted for each terminal a
    - Ex: If there was a rule in the format of A → w: such as:
      - A → xBy
      - You would add the transitions: ε, A → y; ε, ε → B; ε, ε → x in that order
      - You would also add the transitions: x, x → ε and y, y → ε for the terminals x and y



VCU School of Engineering | Computer Science

Equivalence of PDAs and CFGs

- Forward: Lemma 2.21- Construction of the PDA
  - $Q = \{q_{start}, q_{loop}, q_{accept}\} \cup E$ (E is the set of states needed to implement the ε, A → w transitions shown in the slides above)
  - $q_0 = q_{start}$
  - $F = q_{accept}$
  - $\Sigma' = \Sigma$
  - $\Gamma = \Sigma \cup V \cup \{\$\}$ (V = variables)
  - δ = the transitions from the rules as shown in the previous three slides

VCU School of Engineering | Computer Science

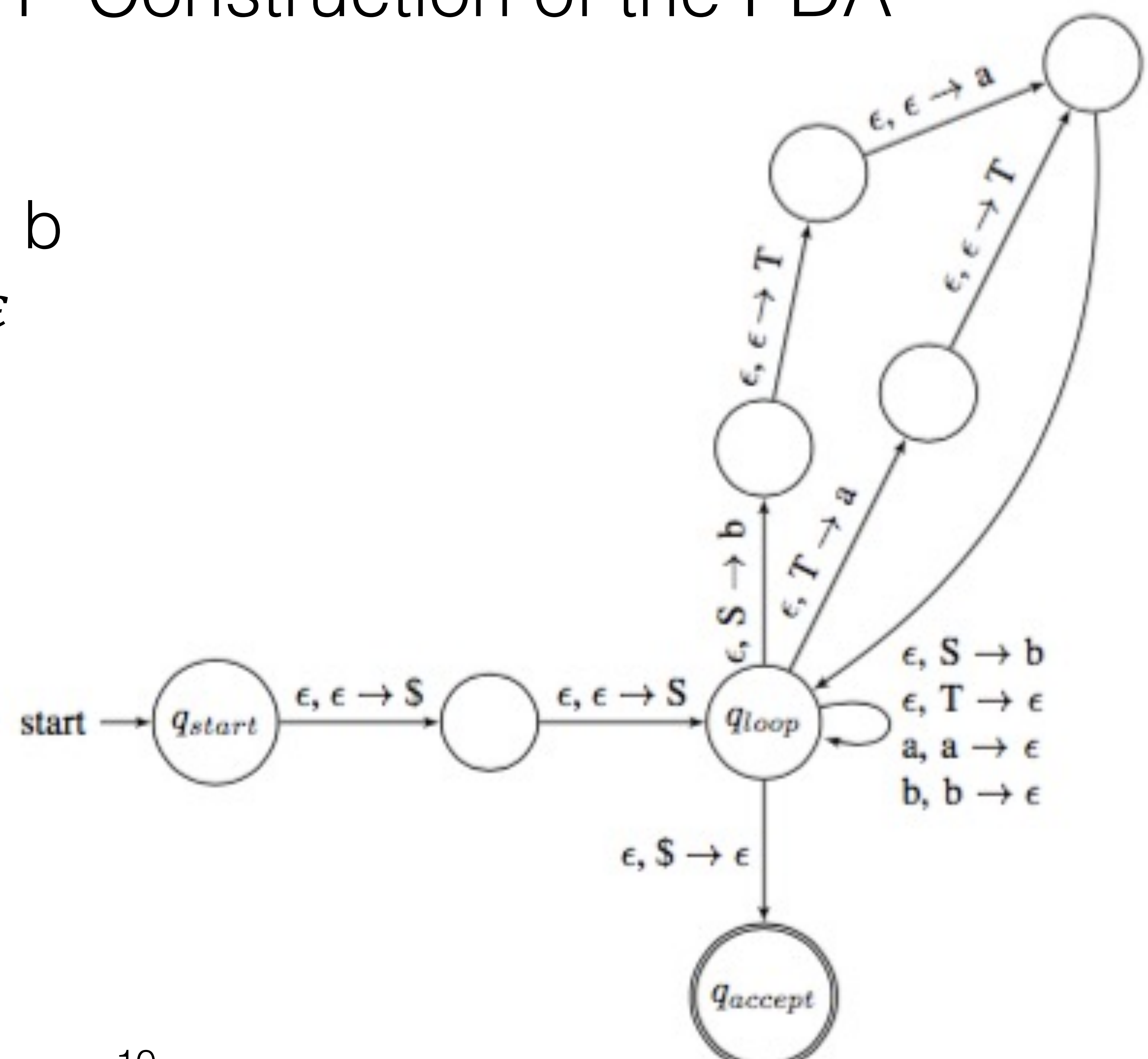
Equivalence of PDAs and CFGs

- <u>Forward</u>: Lemma 2.21- Construction of the PDA
  - Ex 2.25:
    - CFG G: $S \rightarrow aTb \mid b$
      $T \rightarrow Ta \mid \varepsilon$
    - PDA:

School of Engineering | Computer Science

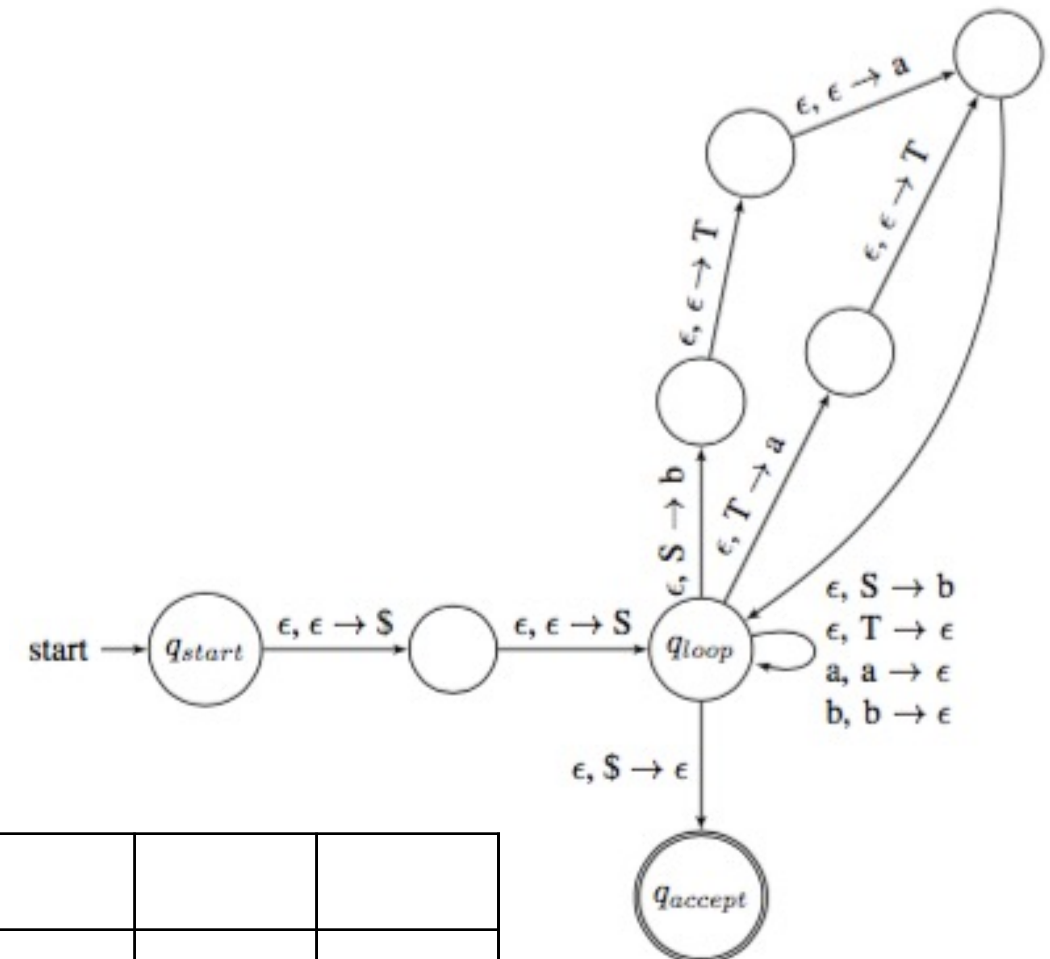
Equivalence of PDAs and CFGs

- Forward: Lemma 2.21- Construction of the PDA
  - Ex 2.25:
    - CFG G: S → aTb | b
      T → Ta | ε
    - PDA:



Equivalence of PDAs and CFGs

- <u>Forward</u>: Lemma 2.21- Construction of the PDA
  - Ex 2.25:
    - CFG G: S → aTb | b
      T → Ta | ε
    - PDA: (stack example below)
      - S → aTb → ab



					a				
				T	T	T			
		S	b	b	b	b	b		
stack	$	$	$	$	$	$	$	$	

VCU School of Engineering | Computer Science

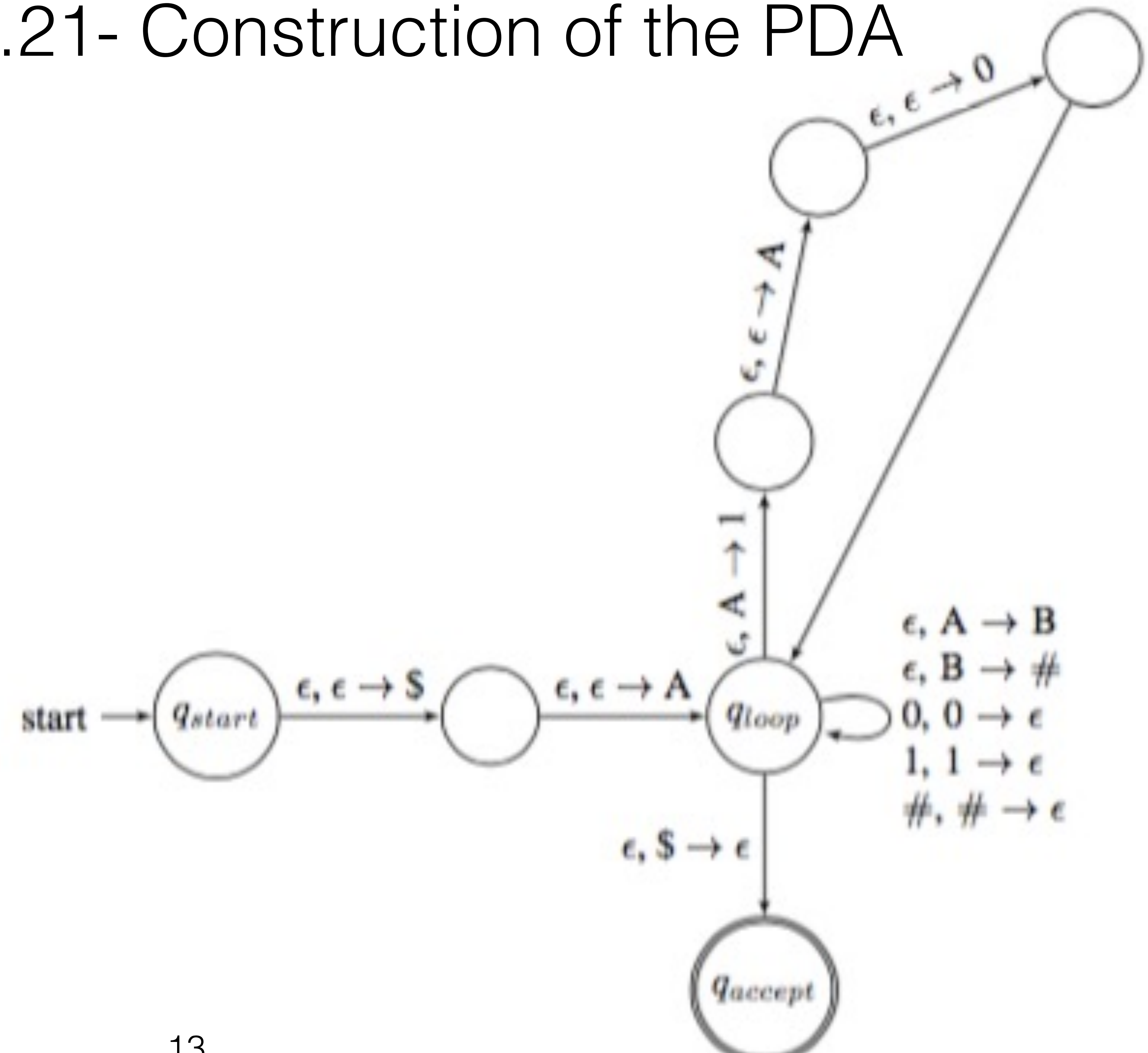
Equivalence of PDAs and CFGs

- Forward: Lemma 2.21- Construction of the PDA
  - Ex 2:
  - CFG G: $A \rightarrow 0A1$
    $A \rightarrow B$
    $B \rightarrow \#$
    - PDA:

VCU School of Engineering | Computer Science

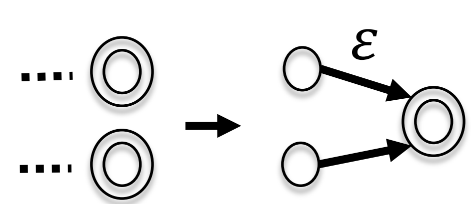
Equivalence of PDAs and CFGs

- Forward: Lemma 2.21- Construction of the PDA
  - Ex 2.25:
  - CFG G: A → 0A1
    A → B
    B → #
    - PDA:

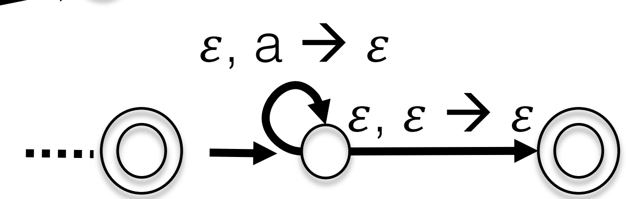


Equivalence of PDAs and CFGs

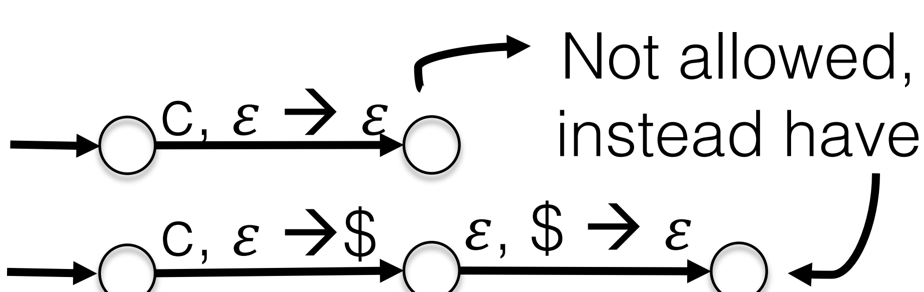
- Theorem 2.20: A language is context-free if and only if a PDA recognizes it (iff, so two directions)
- Backwards: Lemma 2.27 Every PDA has an equivalent CFG
  - Proof Idea: Let PDA P recognize L, construct a CFG G to generate L
  - Assume P Satisfies:
    1. P has a single accept state

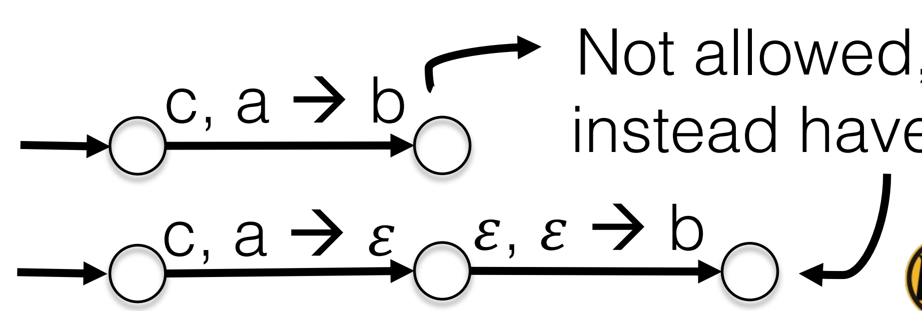


    2. P empties its stack before accepting



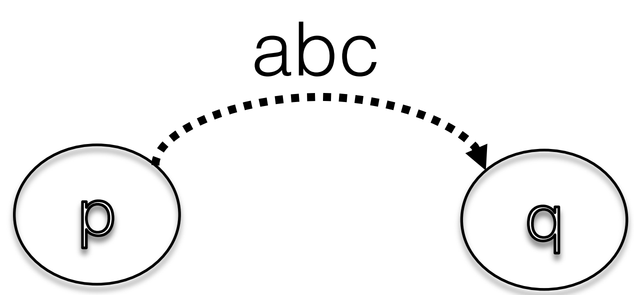
    3. Each transition of P either pushes a symbol on the stack or pops one off, but not both or neither





Equivalence of PDAs and CFGs

- Backwards: Lemma 2.27
  - Start with an empty stack on p, and end with an empty stack on q. Between these two states, design G so that A_{pq} generates all strings that take P from p to q.



  - In the CFG, "add rule" $R_p \rightarrow abc\ R_q$

VCU School of Engineering | Computer Science

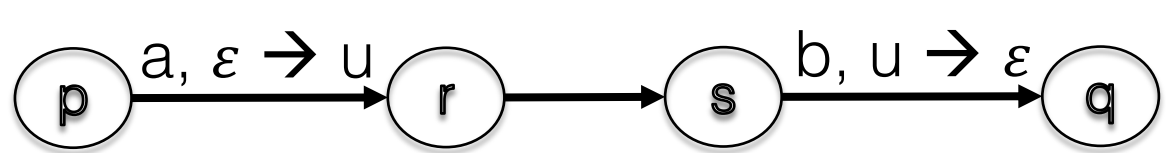
Equivalence of PDAs and CFGs

- Backwards: Lemma 2.27
  - Let PDA P = (Q, Σ, Γ, δ, q_0, {q_{accept}}), construct CFG G = (V, Σ' , R, S) as follows:
    - V = {A_{pq} | p, q ∈ Q}
    - $\Sigma' = \Sigma$
    - S (start variable) = A$q_0 q_{accept}$
    - R = Cont. →

VCU School of Engineering | Computer Science

Equivalence of PDAs and CFGs

- <u>Backwards</u>: Lemma 2.27
  - Let PDA P = (Q, Σ, Γ, δ, q_0, $\{q_{accept}\}$), construct CFG G = (V, Σ' , R, S) as follows:
    - R (substitution rules) as follows:
      1. For each p ∈ Q, add rule $A_{pp} \rightarrow \varepsilon$ to R
      2. For each p, q, r ∈ Q, add rule $A_{pq} \rightarrow A_{pr}A_{rq}$
      3. If P contains a path:

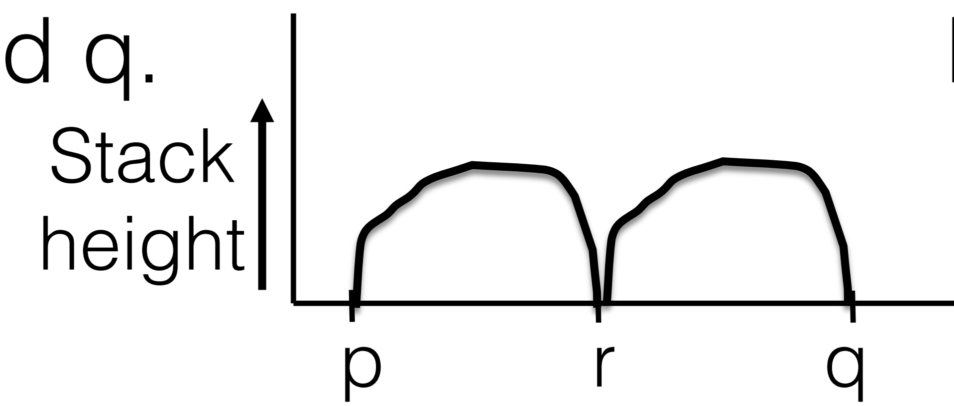


then add $A_{pq} \rightarrow aA_{rs}b$ to R

	u		u
	…	….	…
Stack at:	p		s

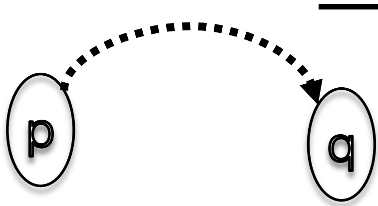
Equivalence of PDAs and CFGs

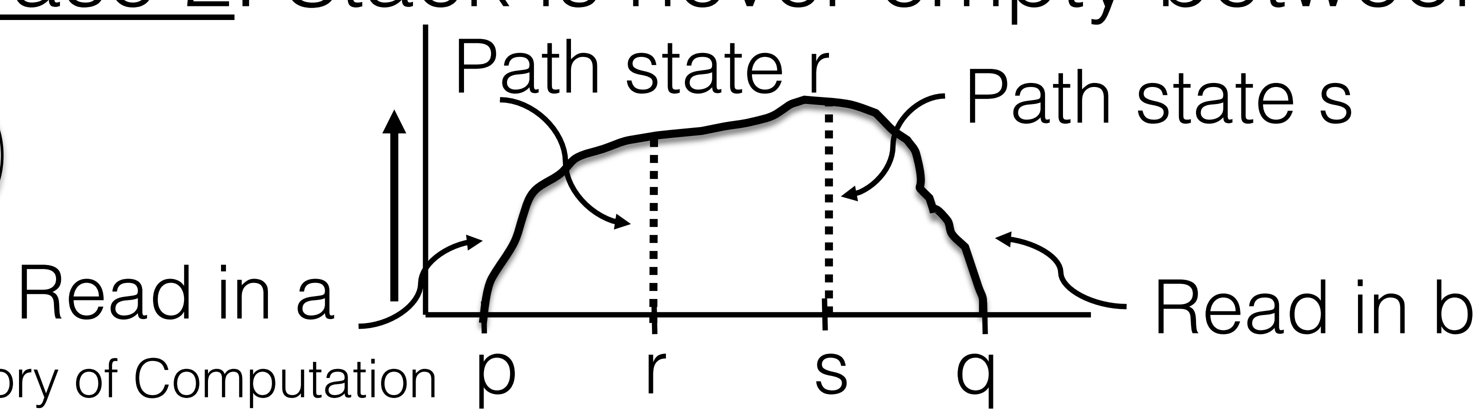
- Backwards: Lemma 2.27
  - Each A_{pq} corresponds to transitions in P from p to q which start and end with the empty stack.
  - Case 1: Stack is empty at some point r between p and q.



Rule in G: $A_{pr}A_{rq}$

  - Case 2: Stack is never empty between p and q.





Rule in G:

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

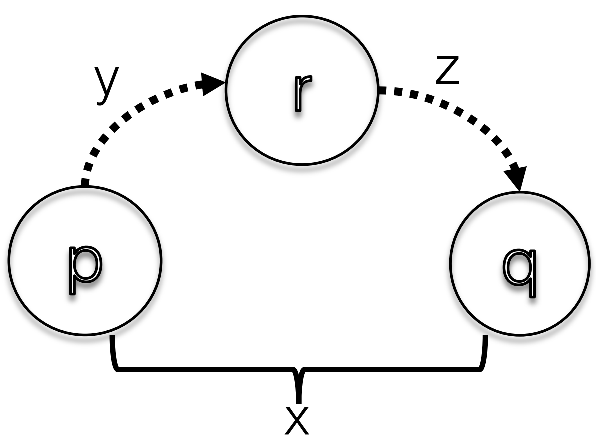
VCU School of Engineering | Computer Science

Equivalence of PDAs and CFGs

- <u>Claim 2.31</u>: If x can bring the PDA P from state p with an empty stack to q, then A_{pq} generates x
  $A_{pq} \Rightarrow^* x$
- <u>Proof by Induction</u>: (based on the number of steps in P from p to q on input x)
  - <u>Base Case</u>: Path takes 0 transitions, so $p = q \Rightarrow x = \varepsilon$
    - $A_{pp} \rightarrow \varepsilon$, so this case works ($A_{pp} \rightarrow \varepsilon$ is in CFG G)
  - <u>Inductive Hypothesis</u>: Assume true for paths of length at most k
  - <u>Inductive Step</u>: Prove for path of length k+1, cont.

Equivalence of PDAs and CFGs

- Claim 2.31:
- Proof by Induction: (based on the number of steps in P from p to q on input x)
  - Inductive Step: Prove for path of length k+1, cont.
    - Case 1: Stack becomes empty at some state $r \notin \{p, q\}$ during computation



Paths $p \rightarrow r$ and $r \rightarrow q$ have $\leq k$ steps.
This implies $A_{pr} \Rightarrow^* y$ and $A_{rq} \Rightarrow^* z$

$A_{pq} \rightarrow A_{pr}A_{rq} \Rightarrow yz = x$

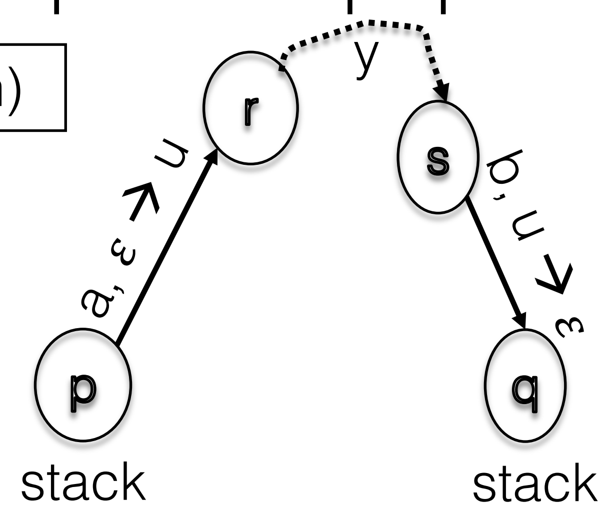
($A_{pq} \rightarrow A_{pr}A_{rq}$ is in CFG G)

School of Engineering | Computer Science

Equivalence of PDAs and CFGs

- Claim 2.31:
- Proof by Induction: (based on the number of steps in P from p to q on input x)
  - Inductive Step: Prove for path of length k+1, cont.
    - Case 2: Stack empty only at p and q, since push or pop at each step:

$\boxed{\delta(\text{state input pop}) \rightarrow (\text{state push})}$



1. First step has form: $\delta(p, a, \varepsilon) \rightarrow (r, u)$
2. Last step has form: $\delta(s, b, u) \rightarrow (q, \varepsilon)$

- Add rules: $A_{pq} \rightarrow aA_{rs}b$ to k
- Suppose x = ayb, since P goes from p to q, y takes P from r to s in k-1 steps ($A_{rs} \Rightarrow^* y$, so $A_{pq} \Rightarrow^* ayb = x$)
- ($A_{pq} \rightarrow aA_{rs}b$ is in CFG G)

VCU School of Engineering | Computer Science

Try It

- Construct the PDA from the given Grammar

CFG G: $S \rightarrow TS\#V \mid VST \mid \varepsilon$

$T \rightarrow 0$

$V \rightarrow 1$

Try It

- Construct the PDA from the given Grammar

  CFG G: S → TS#V |VST | ε

  T → 0

  V → 1